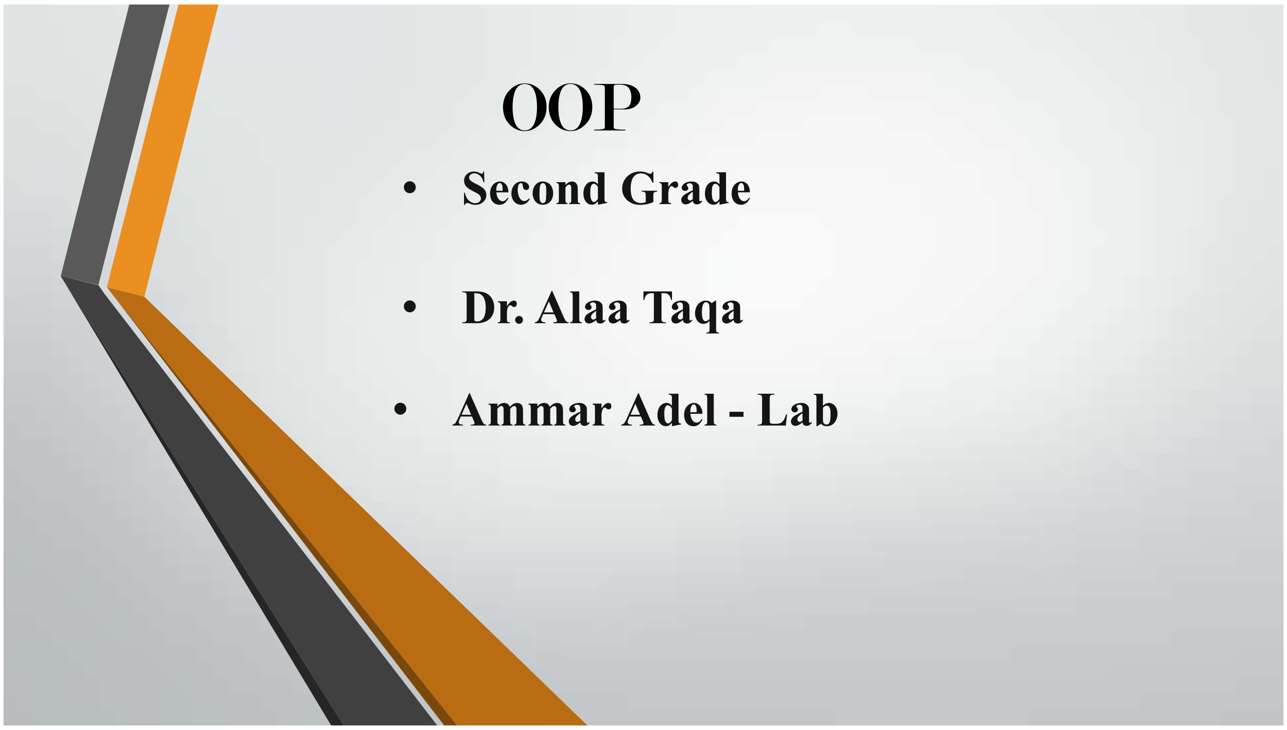
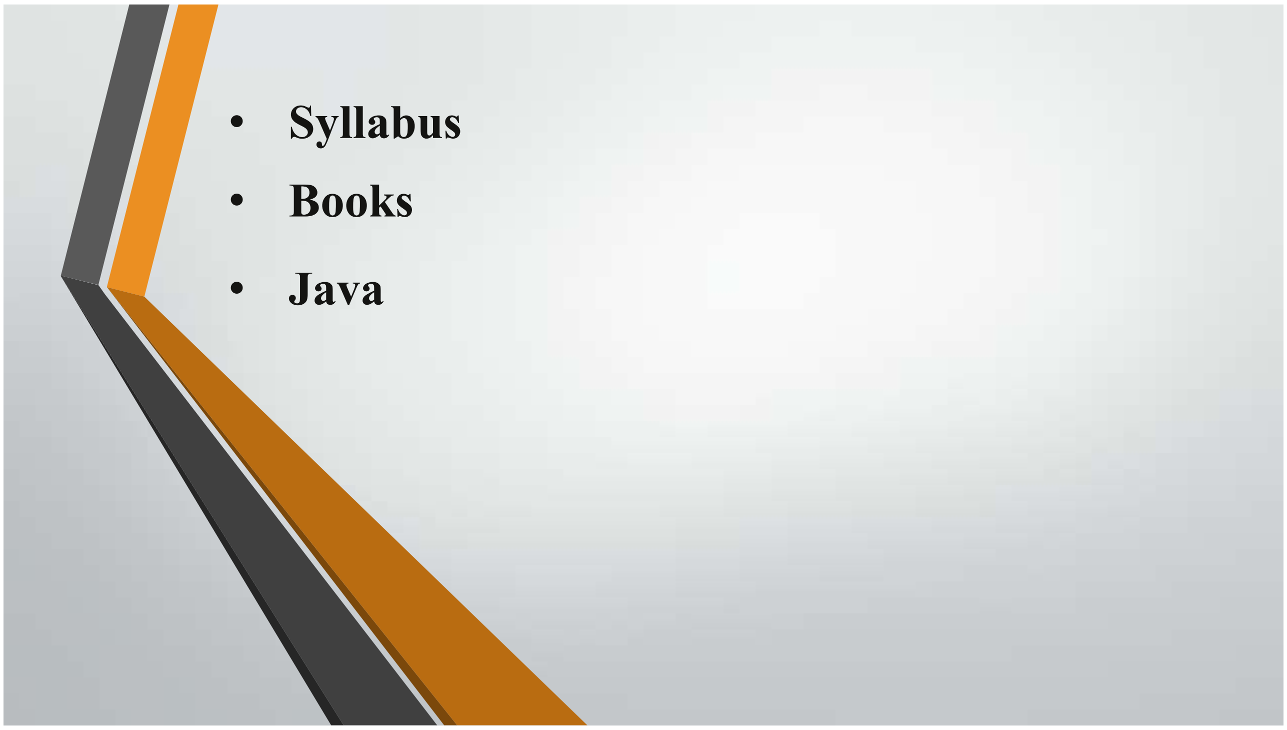


OOP

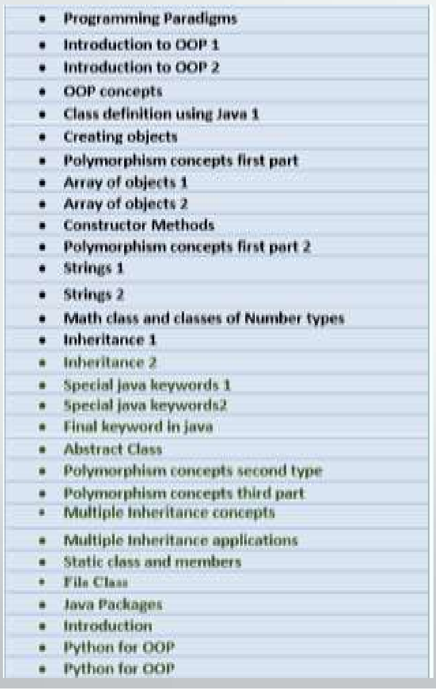
- Second Grade
- Dr. Alaa Taqa
- Ammar Adel - Lab



- Syllabus
- Books
- Java

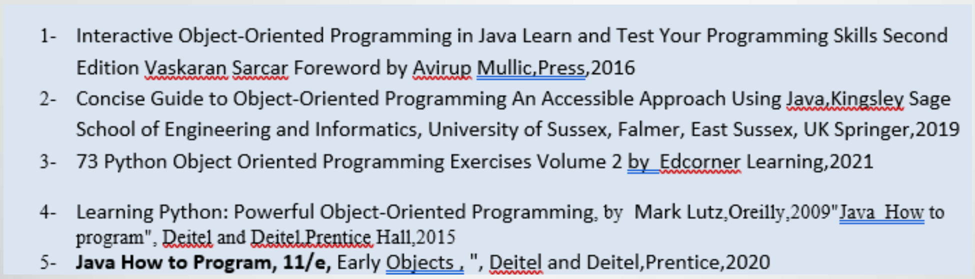
[redacted]

• Syllabus OOP



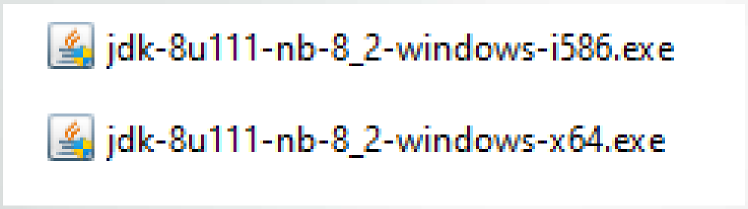
- Programming Paradigms
- Introduction to OOP 1
- Introduction to OOP 2
- OOP concepts
- Class definition using Java 1
- Creating objects
- Polymorphism concepts first part
- Array of objects 1
- Array of objects 2
- Constructor Methods
- Polymorphism concepts first part 2
- Strings 1
- Strings 2
- Math class and classes of Number types
- Inheritance 1
- Inheritance 2
- Special java keywords 1
- Special java keywords2
- Final keyword in java
- Abstract Class
- Polymorphism concepts second type
- Polymorphism concepts third part
- Multiple inheritance concepts
- Multiple inheritance applications
- Static class and members
- File Class
- Java Packages
- Introduction
- Python for OOP
- Python for OOP

• Books



1- Interactive Object-Oriented Programming in Java Learn and Test Your Programming Skills Second Edition Vaskaran Sarcar Foreword by Avirup Mullic,Press,2016
2- Concise Guide to Object-Oriented Programming An Accessible Approach Using Java,Kingsley Sage School of Engineering and Informatics, University of Sussex, Falmer, East Sussex, UK Springer,2019
3- 73 Python Object Oriented Programming Exercises Volume 2 by Edcorner Learning,2021
4- Learning Python: Powerful Object-Oriented Programming, by Mark Lutz,Oreilly,2009"Java How to program", Deitel and Deitel Prentice Hall,2015
5- **Java How to Program, 11/e,** Early Objects , ", Deitel and Deitel,Prentice,2020

- **Java Lab**



jdk-8u111-nb-8_2-windows-i586.exe

jdk-8u111-nb-8_2-windows-x64.exe

- **Programming Paradigms**

  - What is unstructured programming?
  - What is structured programming?
    - What is Procedural (Functional) Programming?
  - What is OOP Object Oriented Programming?

• Programming Paradigms

- Programming paradigm is a way to classify programming languages according to their style of programming and features they provide.

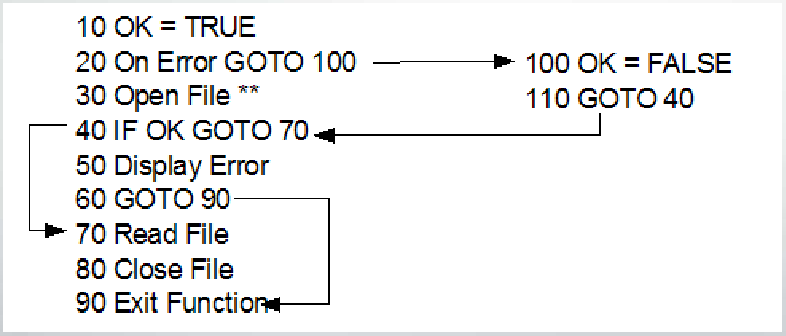
- There are several features that determine a programming paradigm such as modularity, objects, interrupts or events, control flow etc. A programming language can be single paradigm or multi-paradigm.

• Unstructured Programming

- Unstructured programming is the oldest paradigm and is still in practice.
- It mainly focuses on steps to be done and works on the logic of "*First do this then do that*".
- It defines a sequence of statements in order of which the operations must take place.
- In unstructured programming, the control flow is explicit and depend on collection of **GOTO** statements.
- unstructured programming lacks the support of modularity.

[redacted]

Example



```
10 OK = TRUE
20 On Error GOTO 100          100 OK = FALSE
30 Open File **               110 GOTO 40
40 IF OK GOTO 70
50 Display Error
60 GOTO 90
70 Read File
80 Close File
90 Exit Function
```

• Structured Programming

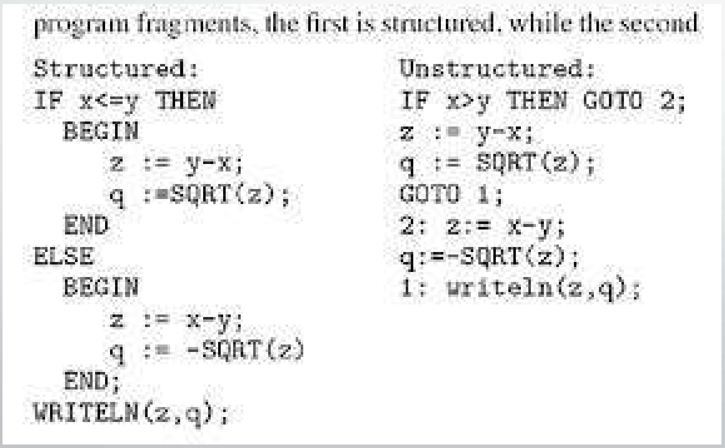
- It can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other.

- It doesn’t support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc.

- The instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are: C , C++ , Java and C#

[redacted]

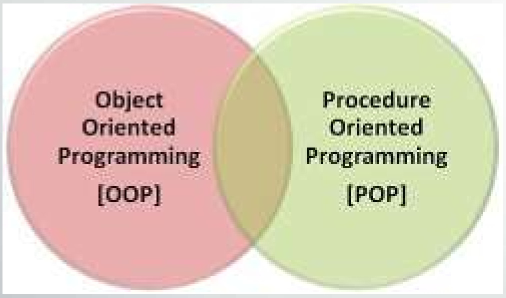
Example structured vs unstructured

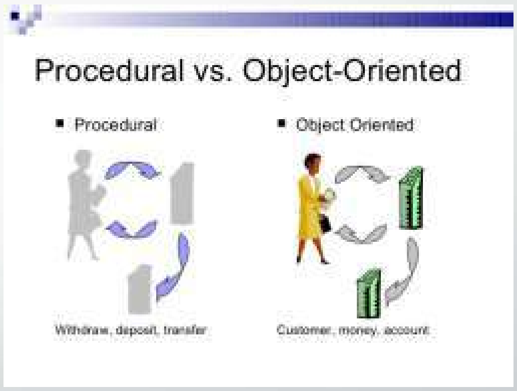

program fragments, the first is structured, while the second

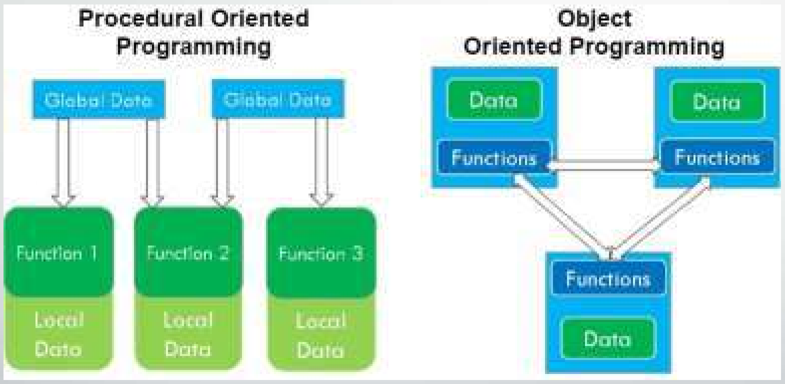
```
Structured:
IF x<=y THEN
  BEGIN
     z := y-x;
     q :=SQRT(z);
  END
ELSE
  BEGIN
     z := x-y;
     q := -SQRT(z)
  END;
WRITELN(z,q);
```

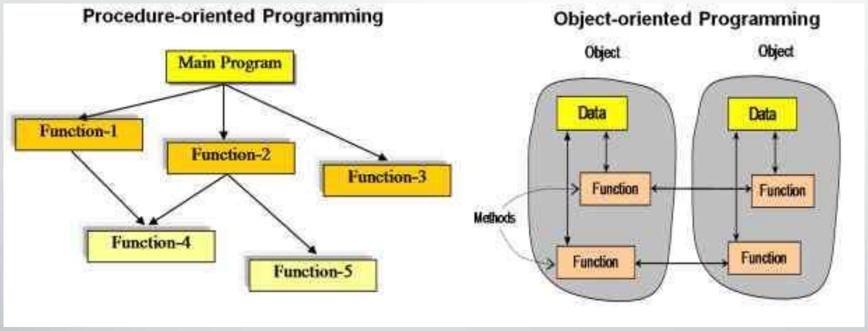
```
Unstructured:
IF x>y THEN GOTO 2;
z := y-x;
q := SQRT(z);
GOTO 1;
2: z:= x-y;
q:=-SQRT(z);
1: writeln(z,q);
```

The main two approaches are:


Object Oriented Programming [OOP]
Procedure Oriented Programming [POP]



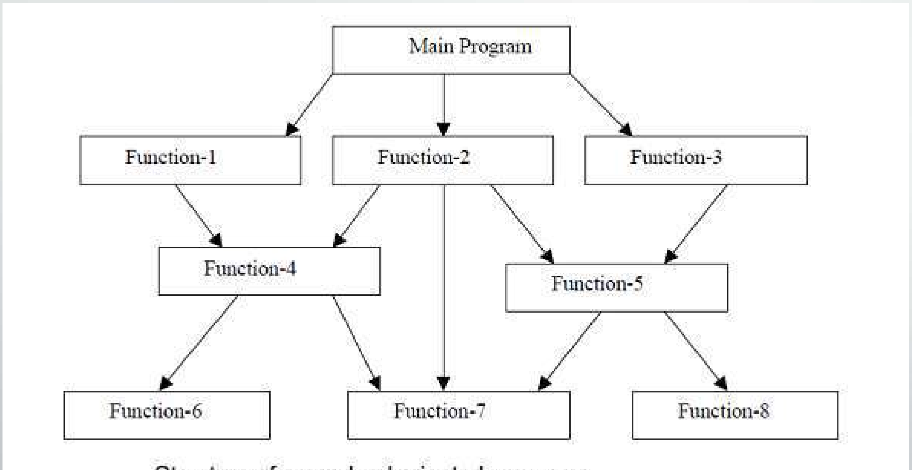

Procedural Oriented Programming
Global Data
Global Data
Function 1
Local Data
Function 2
Local Data
Function 3
Local Data
Object Oriented Programming
Data
Functions
Data
Functions
Functions
Data


Procedure-oriented Programming
Main Program
Function-1
Function-2
Function-3
Function-4
Function-5
Object-oriented Programming
Object
Object
Data
Function
Function
Data
Function
Function
Methods

[redacted]

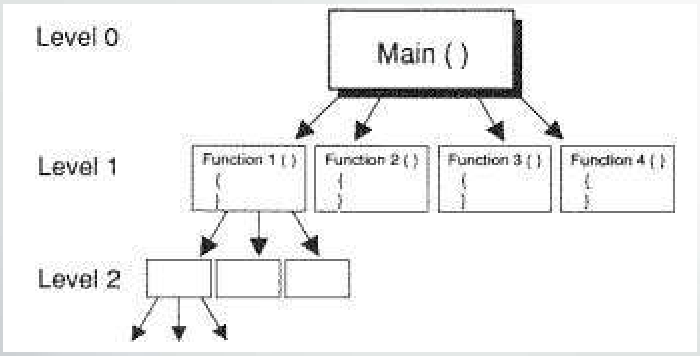
• Procedural /Functional Programming

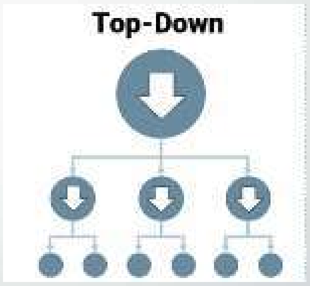
- Functional programming paradigm is completely different programming approach.
- Functional programming uses a combination of functions calls to drive the flow of the program.
- The result of a function becomes the input to another function.



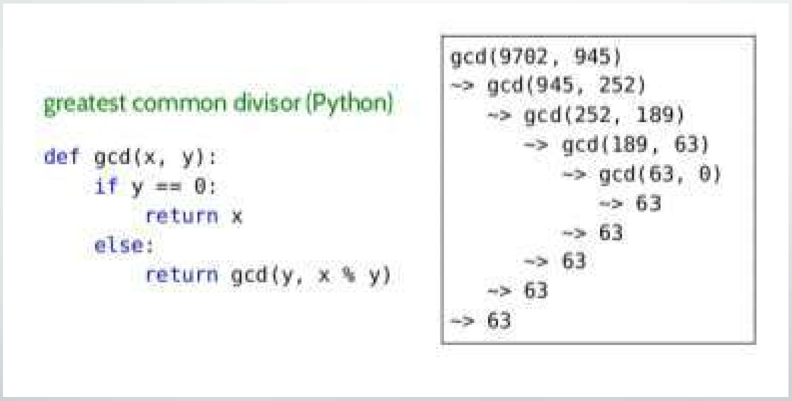
Structure of procedural oriented programs

[redacted]





Example

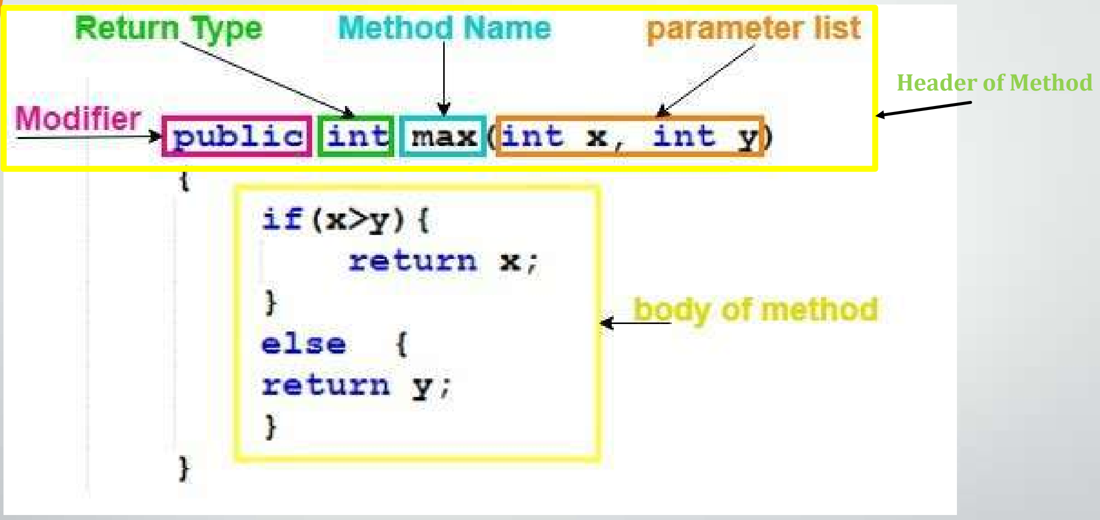


greatest common divisor (Python)

```
def gcd(x, y):
    if y == 0:
        return x
    else:
        return gcd(y, x % y)
```

```
gcd(9702, 945)
-> gcd(945, 252)
   -> gcd(252, 189)
      -> gcd(189, 63)
         -> gcd(63, 0)
            -> 63
         -> 63
      -> 63
   -> 63
-> 63
```

[redacted]



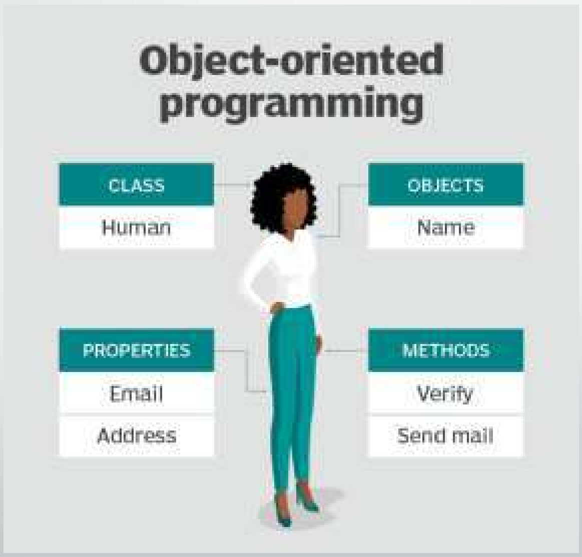
```
public int max(int x, int y)
{
    if(x>y){
        return x;
    }
    else   {
    return y;
    }
}
```

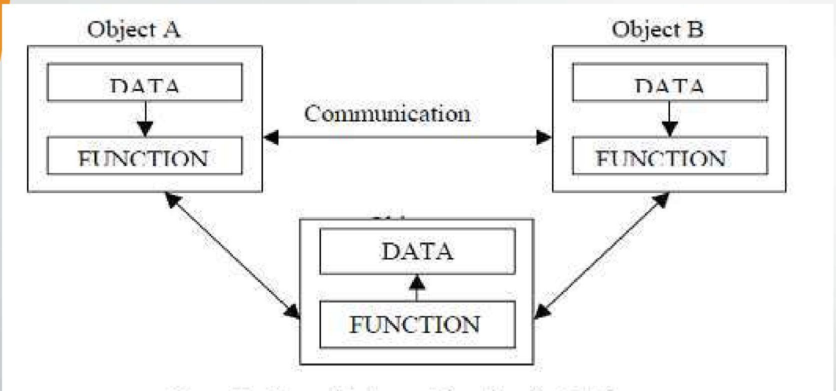
- OOP Object Oriented Programming

- Object Oriented Programming paradigm is widely practiced programming paradigm.
- It is based on the concept of objects. Objects are real world entity.
- Everything present around us is an object.
- Every object has two important property attribute (data) and behavior (function).

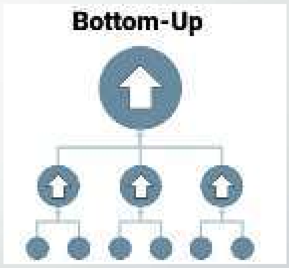
[redacted]

Example



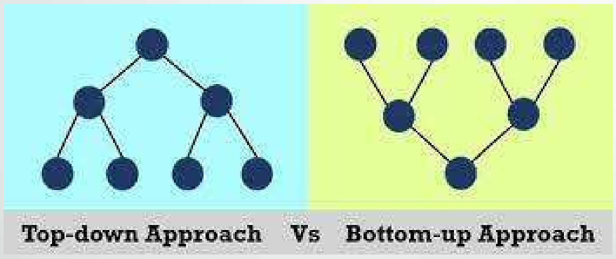


Organization of data and function in OOP



[redacted]

H.W.
What are the main differences of the following two approaches?

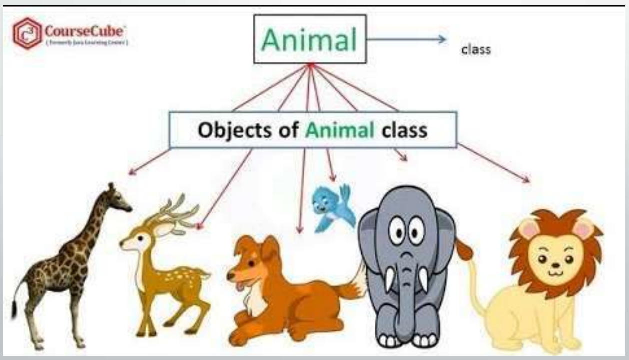


- OOP Concepts (مفاهيم)
  - What are the main concepts of OOP
    - Object and Class
    - State and behavior
    - Examples

[redacted]

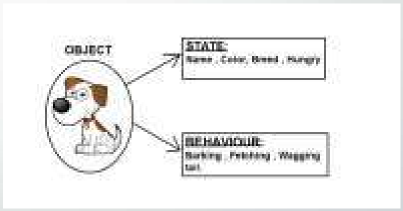
• Classes and Objects

- Classes and objects are the fundamental components of OOP's.
- Often there is a confusion between classes and objects. In this lecture, we try to tell you the difference between class and object.



• What is class ?

- A class is **an extensible program-code-template(blueprint prototype) for creating objects, providing initial values for state (member variables) and implementations of behavior** (member functions or methods). **It** defines the variables and the methods (functions) common to all objects of a certain kind.


OBJECT
STATE:
Name , Color, Breed , Hungry
BEHAVIOUR:
Barking , Fetching , Wagging tail

Syntax

```
class  class_name
{
  member variables;
  member methods or functions;
 }
```

Write examples of classes from Real World

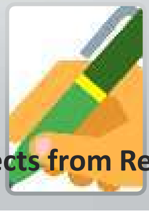
[redacted]

• What is an Object?

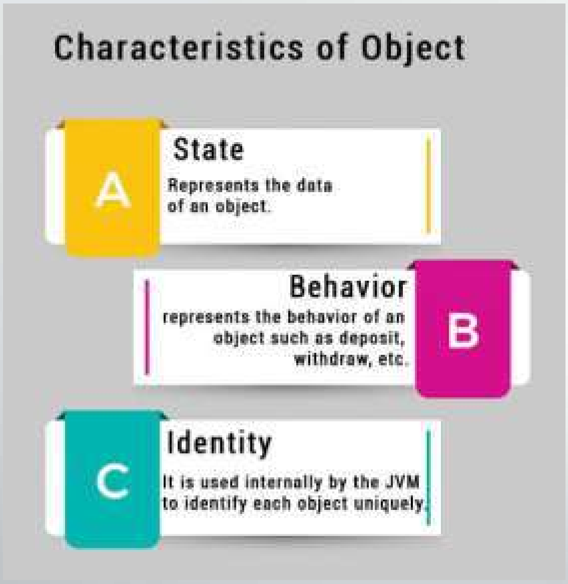
An object is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful. Object determines the behavior of the class. When you send a **message** (function call)to an object, you are asking the object to invoke or execute one of its methods. From a programming point of view, an object can be a data structure, a variable or a function. It has a memory location allocated.

Syntax

```
class_name   s = new class_name();
```



Write examples of objects from Real World

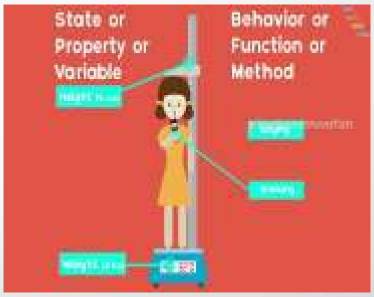


Characteristics of Object

A State
Represents the data of an object.

B Behavior
represents the behavior of an object such as deposit, withdraw, etc.

C Identity
It is used internally by the JVM to identify each object uniquely.

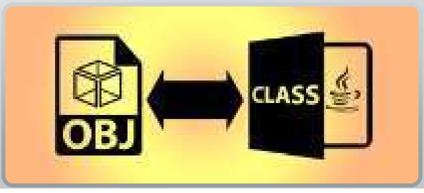


• What is the Difference Between Object & Class?

- A **class** is a **prototype** whereas an object is a instance of a class.

Let's see some real-life examples of class and object in java to understand the difference well:

Class: Human Object: Man, Woman
Class: Fruit Object: Apple, Banana, Mango, Guava wtc.
Class: Mobile phone Object: iPhone, Samsung, Moto
Class: Food Object: Pizza, Burger, Samosa

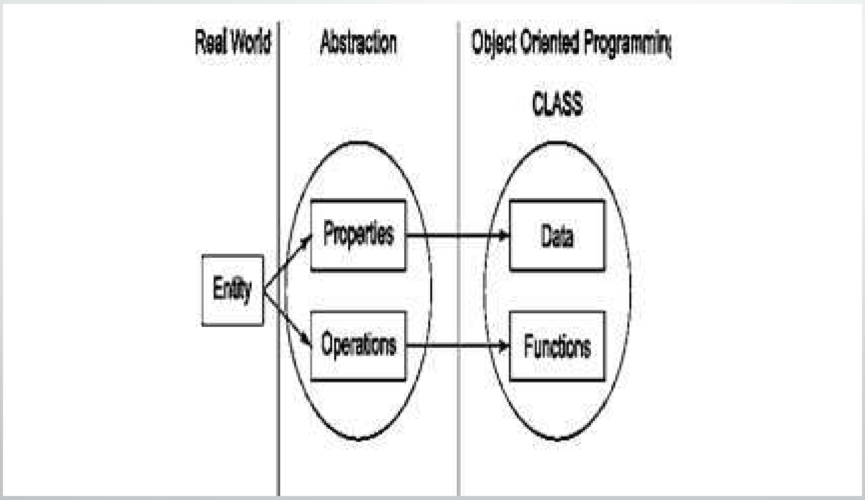


• The differences between object and class:

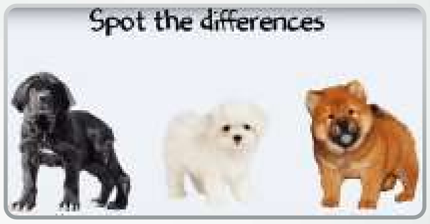
Object	Class
Object is an instance of a class.	Class is a blueprint or template from which objects are created.
Object is a real- world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a group of similar objects.
Object is a physical entity.	Class is a logical entity.
Object is created through new keyword mainly e.g. Student s1=new Student();	Class is declared using class keyword e.g. class Student{}
Object is created many times as per requirement.	Class is declared once.
Object allocates memory when it is created.	Class doesn't allocated memory when it is created.
There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only one way to define class in java using class keyword.

[redacted]

- **Different point of views**

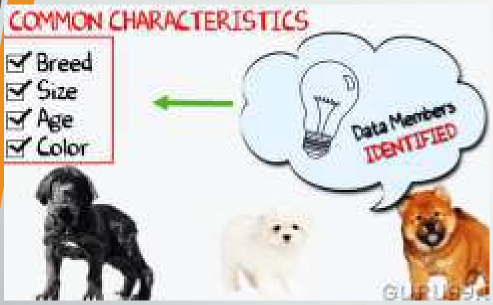


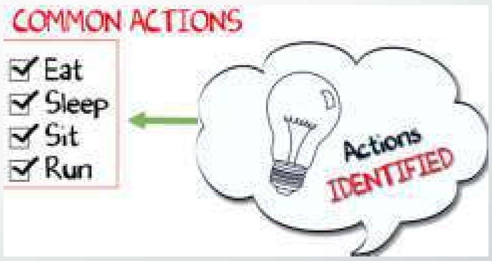
How to convert real life entity to object?



How to convert real life entities dogs into software objects?

[redacted]



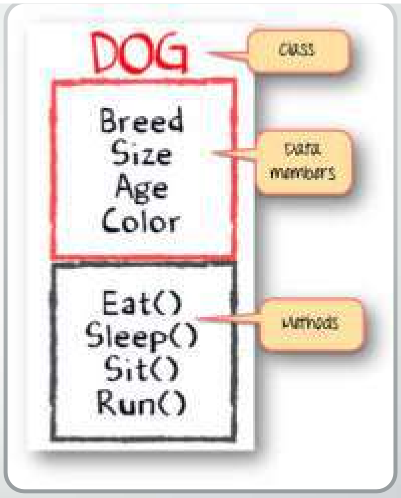


So far, we have defined following things,

Class - Dogs

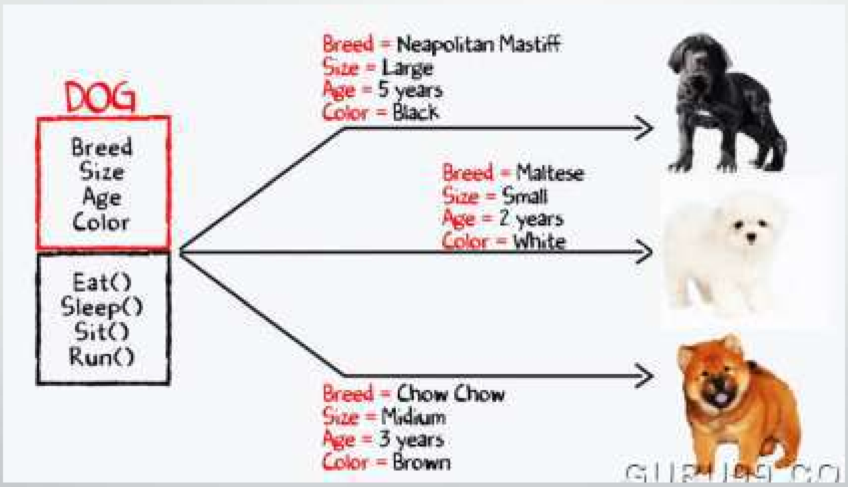
Data members - size, age, color, breed, etc.

Methods- eat, sleep, sit and run.

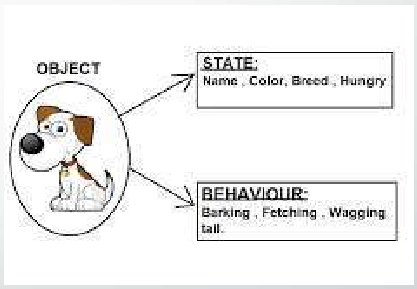


Now, for different values of data members (breed size, age, and color) in Java class, you will get different dog objects.

[redacted]



State and behavior are the basic properties of an Object

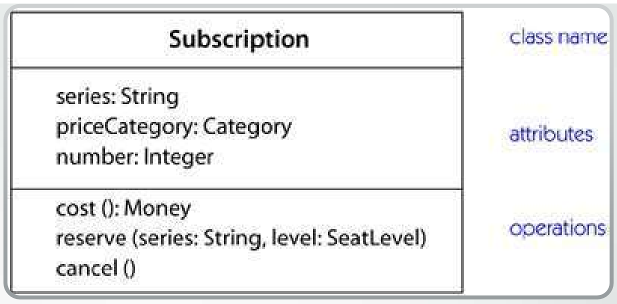

OBJECT
STATE:
Name , Color, Breed , Hungry
BEHAVIOUR:
Barking , Fetching , Wagging tail.

[redacted]

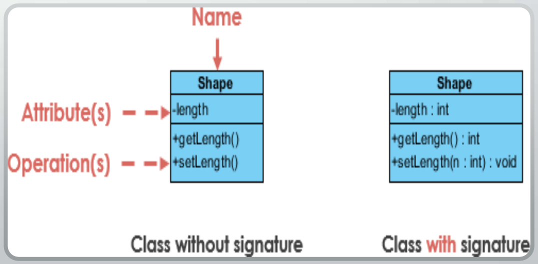
- **State** tells us about the type or the value of that object whereas **behavior** tells us about the operations or things that the object can perform.

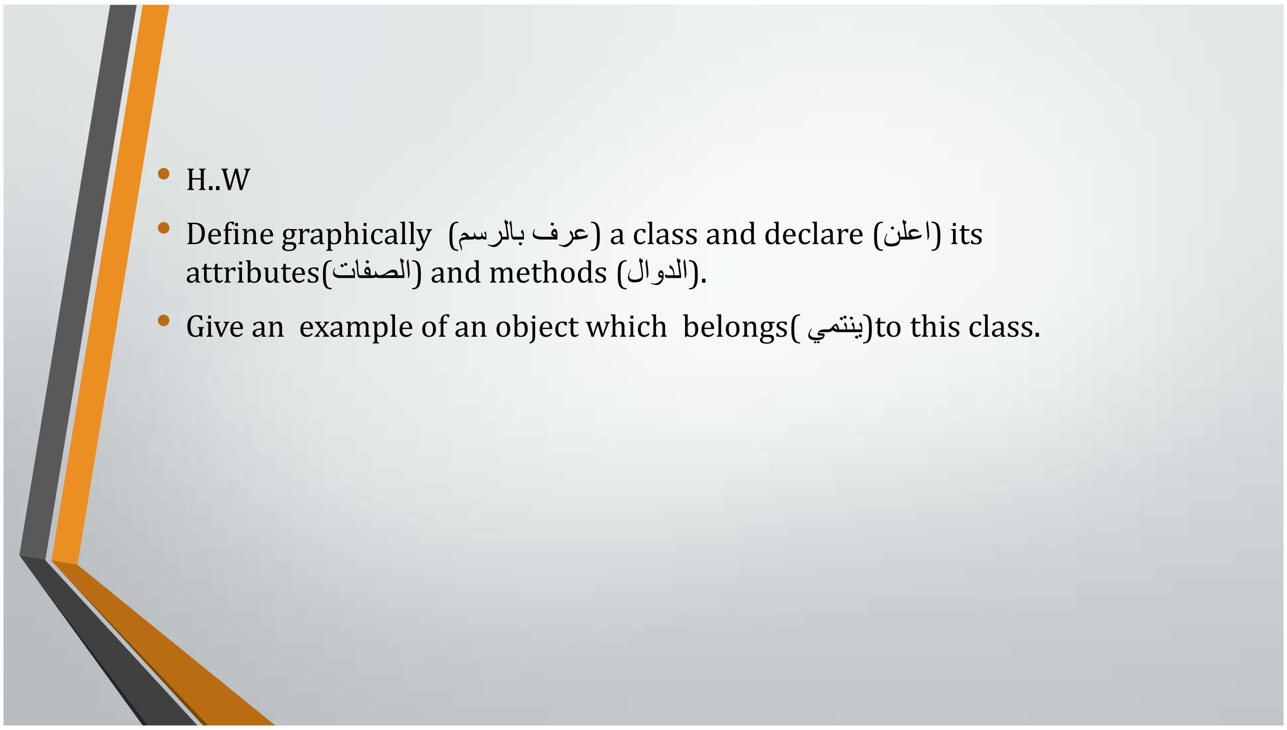
For example, let's say we have an Object called car, so car object will have color, engine type, wheels etc. as it's state, this car object can run at 180kmph, it can turn right and left, it can go back and forth, it can carry 4 people etc. These are its **behaviors.**

- In **object**-oriented programming, a **class** is a template **definition** of the method s and variable s in a particular kind of **object** . Thus, an **object** is a specific instance of a **class**; it contains real values instead of variables. The **class** is one of the **defining** ideas of **object**-oriented programming.

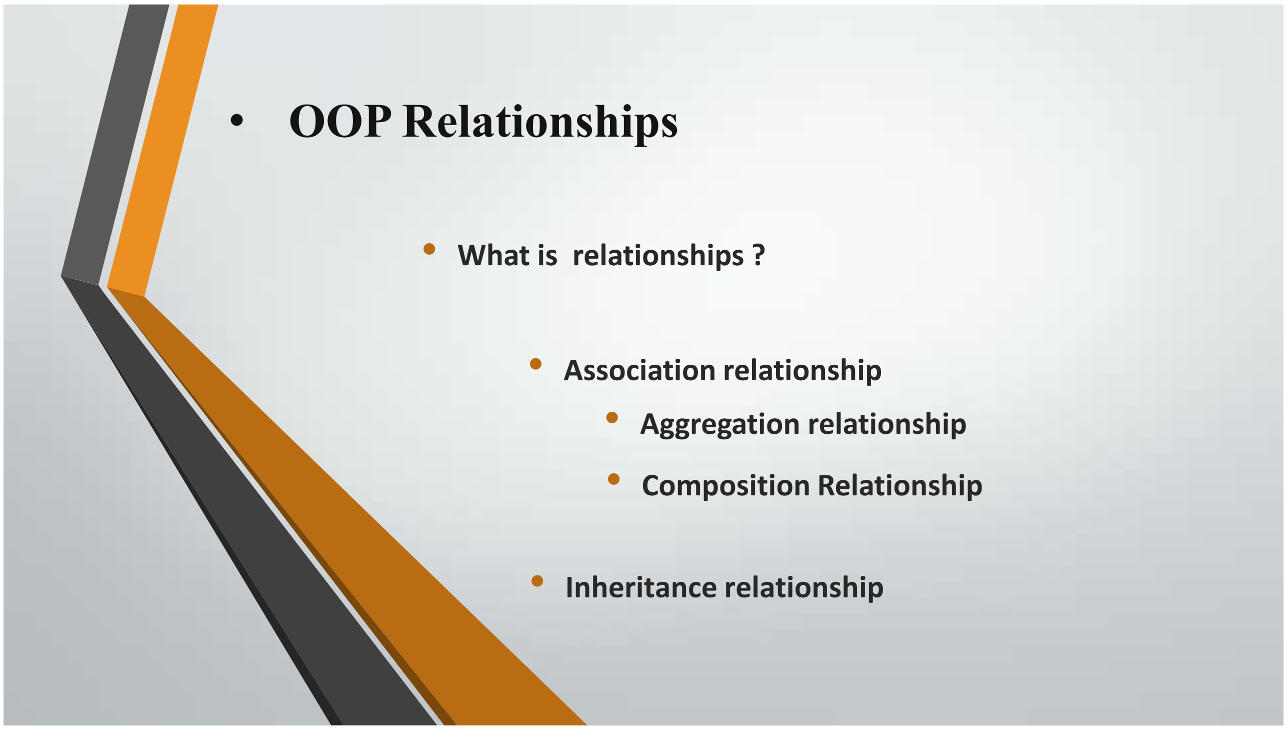


Class notation



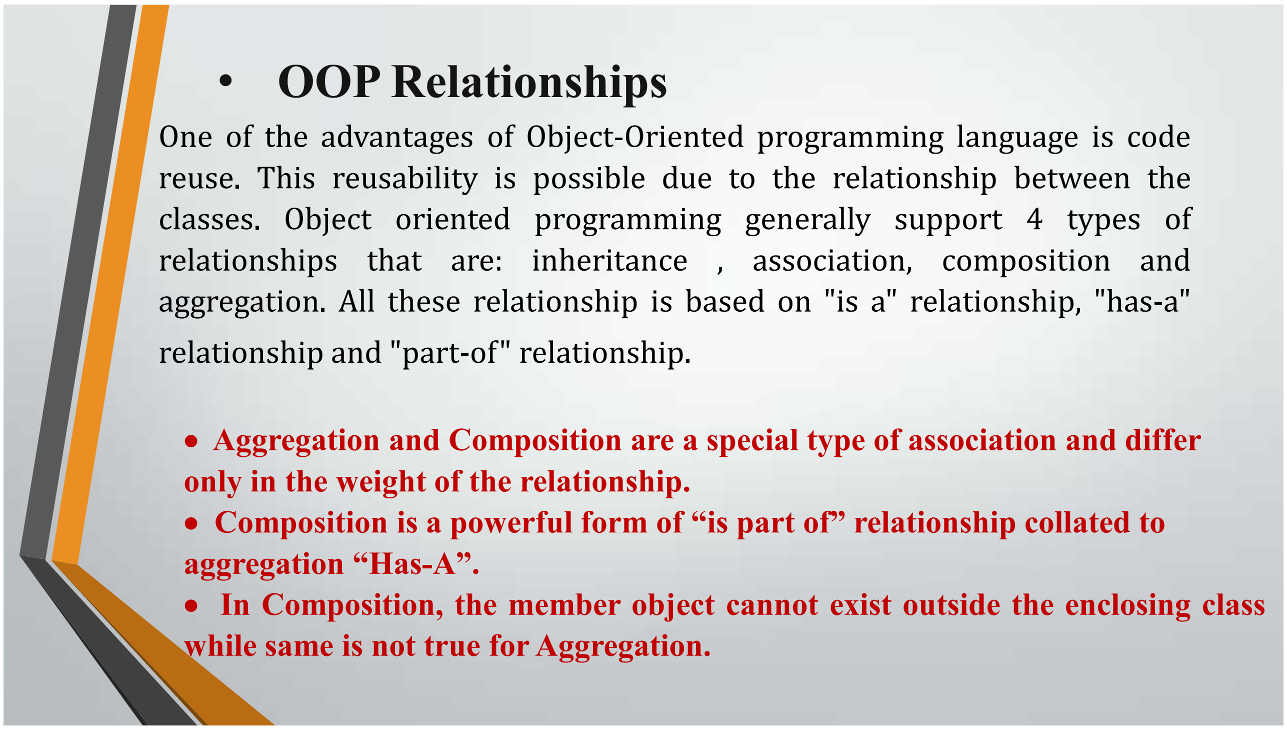


- H..W
- Define graphically (عرف بالرسم) a class and declare (اعلن) its attributes(الصفات) and methods (الدوال).
- Give an example of an object which belongs(ينتمي)to this class.



- **OOP Relationships**
  - What is relationships ?
    - Association relationship
      - Aggregation relationship
      - Composition Relationship
    - Inheritance relationship

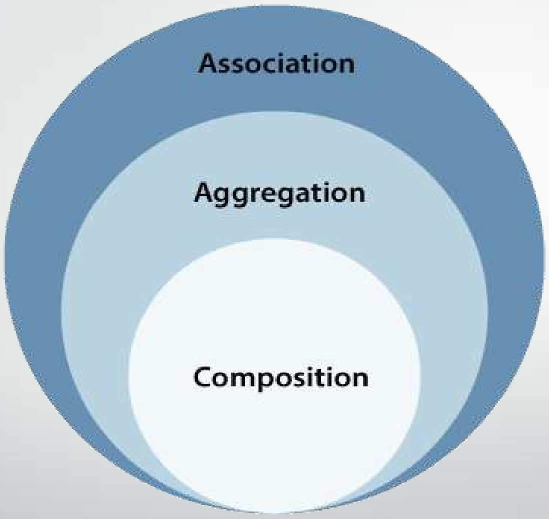
[redacted]



• OOP Relationships

One of the advantages of Object-Oriented programming language is code reuse. This reusability is possible due to the relationship between the classes. Object oriented programming generally support 4 types of relationships that are: inheritance , association, composition and aggregation. All these relationship is based on "is a" relationship, "has-a" relationship and "part-of" relationship.

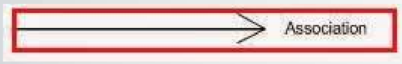
- **Aggregation and Composition are a special type of association and differ only in the weight of the relationship.**
- **Composition is a powerful form of "is part of" relationship collated to aggregation "Has-A".**
- **In Composition, the member object cannot exist outside the enclosing class while same is not true for Aggregation.**

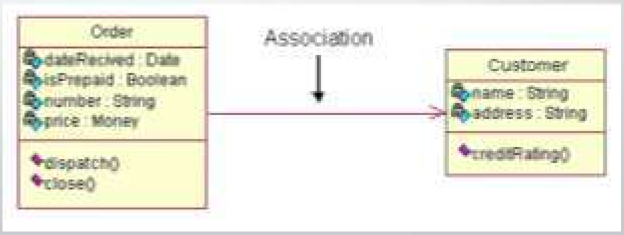


[redacted]

• Association in object oriented programming

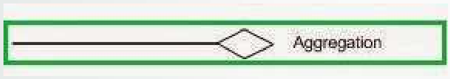
Association is a semantically weak relationship (a semantic dependency) between otherwise unrelated objects. An association is a “using” relationship between two or more objects in which the objects have their own lifetime and there is no owner.



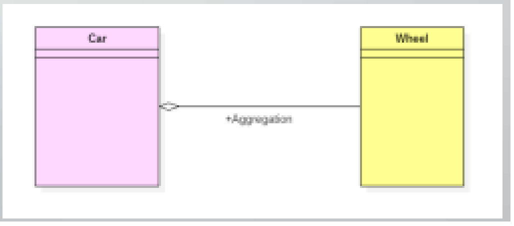


2- Aggregation (التجميع)

Aggregation is a special form of association. It is a relationship between two classes like association, however its a **directional** association, which means it is strictly a **one way association.** It represents a **HAS-A** relationship.



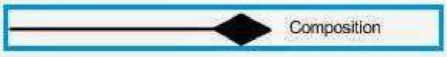
Example1
A car needs a wheel to function correctly, but a wheel doesn’t always need a car. It can also be used with the bike, bicycle, or any other vehicles but not a particular car. Here, the wheel object is meaningful even without the car object. Such type of relationship is called UML Aggregation relation.



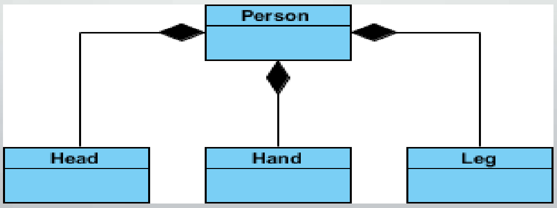
[redacted]

3- Composition (التكوين)

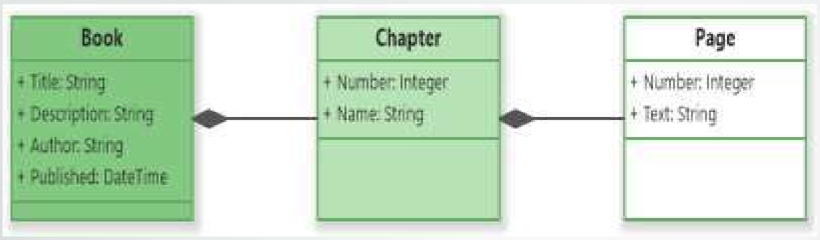
- Composition is a "part-of" relationship. In composition relationship both entities are interdependent of each other for example “heart is part of body. Heart is a part of each body and both are dependent on each other. (ownership).
- Composition implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate to a House.



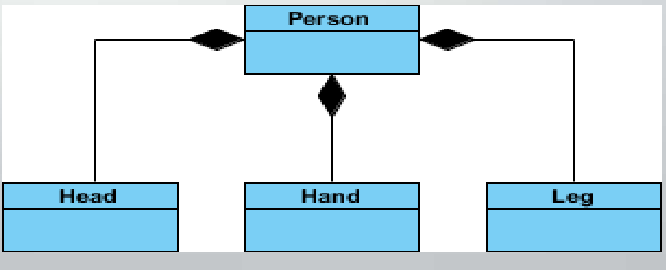
Example2



Example3

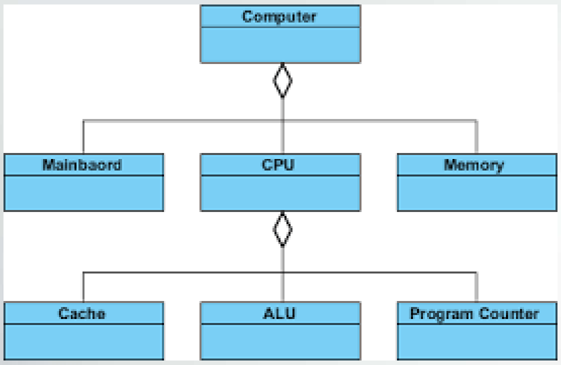


Example4

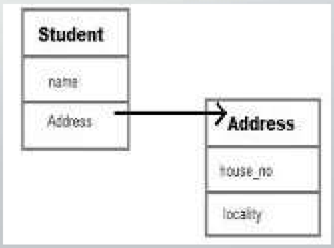


[redacted]

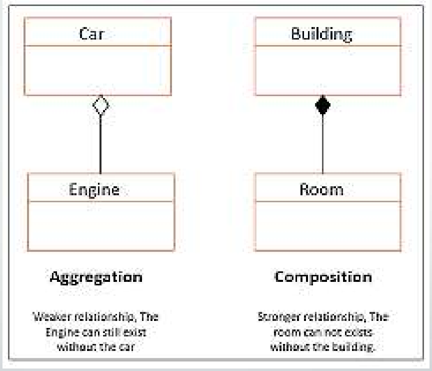
Example5



Example6



Example7



Aggregation

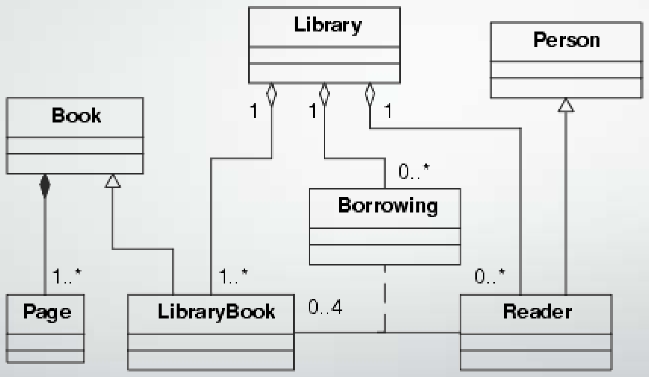
Weaker relationship, The Engine can still exist without the car

Composition

Stronger relationship, The room can not exists without the building.

[redacted]

Example8

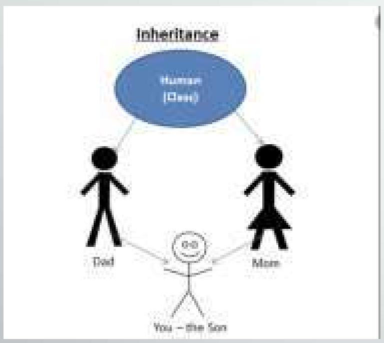


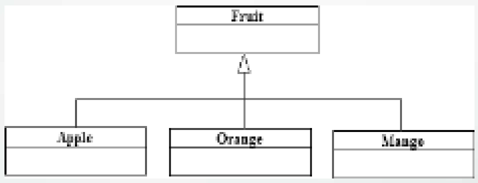
- **Inheritance (التوارث)**

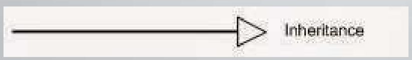
Inheritance is “IS-A” type of relationship. It means that it creates a new class by using existing class code. It is just like saying that “A is type of B”. For example:

Apple is a type of fruit, so **Apple is a fruit**. **Ferrari is a type of car,** so **Ferrari is a car**.

Reuse (reusability) إعادة الاستخدام واضحة جدا بالوراثة

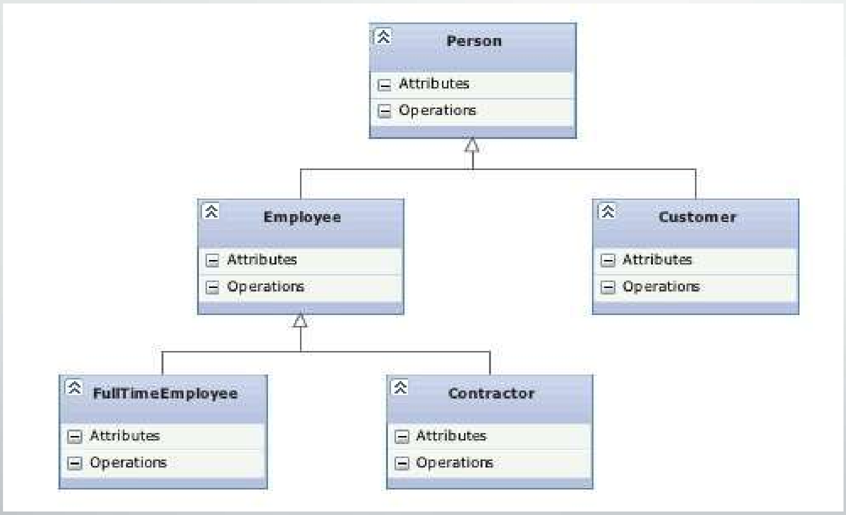






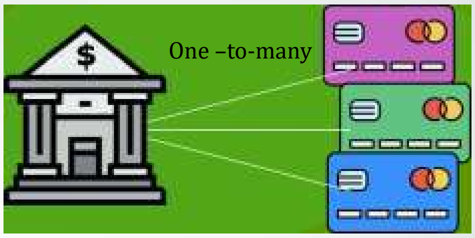
[redacted]

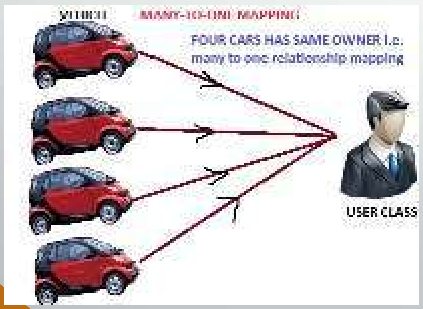
Example9

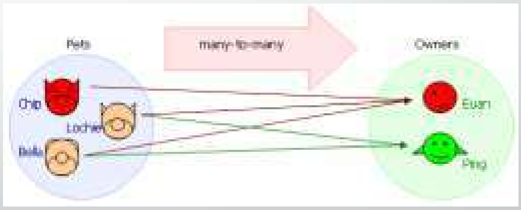


- **Relation Types**

ONE-TO-ONE RELATIONSHIP
PASSPORT
one passport
one person

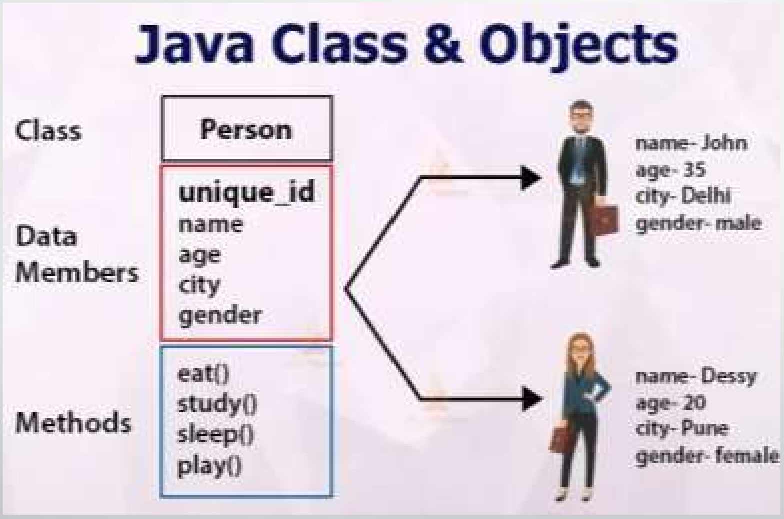



VEHICLE
MANY-TO-ONE MAPPING
FOUR CARS HAS SAME OWNER i.e. many to one relationship mapping
USER CLASS

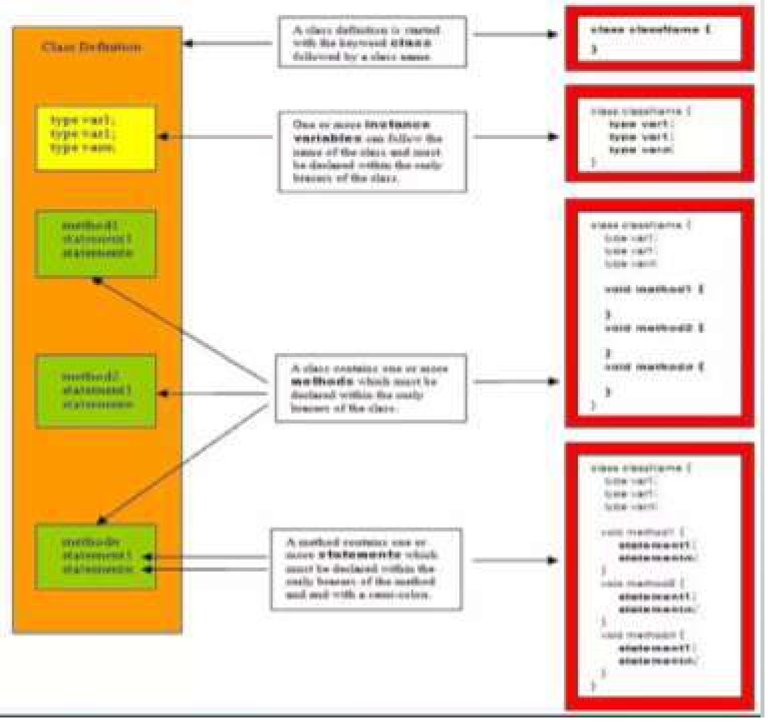


[redacted]

Example10



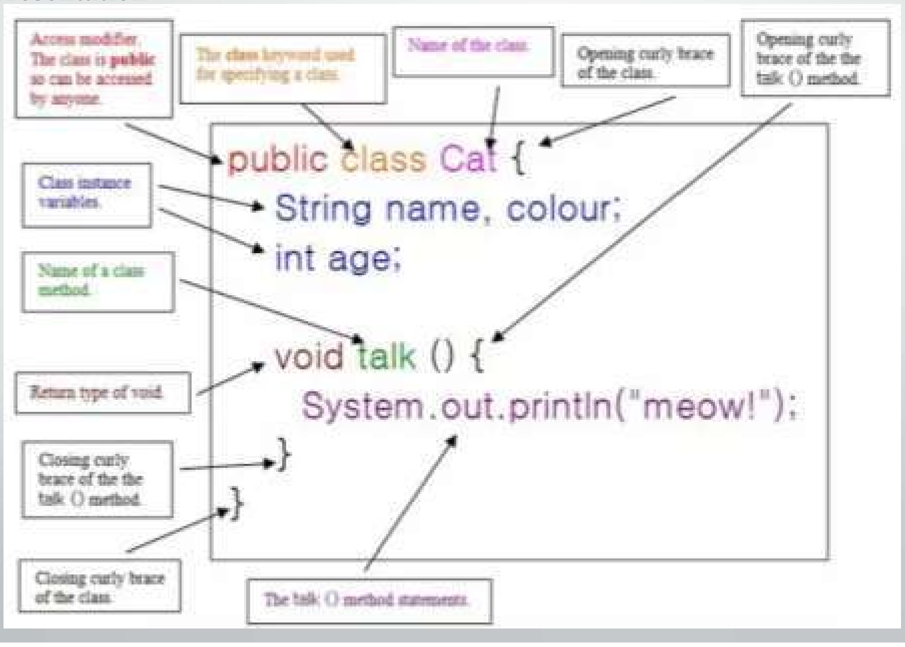
General form graphical representation



Class Definition
type var1;
type var1;
type varn;
method1
statement1
statementn
method2
statement1
statementn
methodn
statement1
statementn
A class definition is started with the keyword class followed by a class name.
One or more instance variables can follow the name of the class and must be declared within the curly braces of the class.
A class contains one or more methods which must be declared within the curly braces of the class.
A method contains one or more statements which must be declared within the curly braces of the method and end with a semi-colon.
class classname {
}
class classname {
type var1;
type var1;
type varn;
}
void method1 {
}
void method2 {
}
void methodn {
}
statement1;
statementn;

[redacted]

General form programming representation



Access modifier. The class is public so can be accessed by anyone.
The class keyword used for specifying a class.
Name of the class.
Opening curly brace of the class.
Opening curly brace of the the talk () method.
Class instance variables.
Name of a class method.
Return type of void.
Closing curly brace of the the talk () method.
Closing curly brace of the class.
The talk () method statements.

```
public class Cat {
  String name, colour;
  int age;

  void talk () {
    System.out.println("meow!");
  }
}
```

1. Defining a class

- Access Modifiers in Java
  - Declaration of variable members (Instance Variables)
    - Methods Types (kinds)
      - Special method main
        - Declaration of Methods
          - Creating an object
            - Message Components and Passing

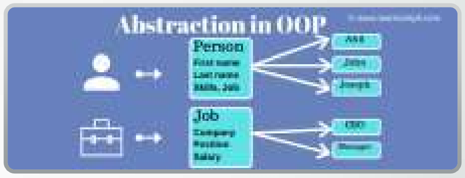
2. OOP Concepts (مفاهيم) Part2

- Encapsulation
  - Abstraction

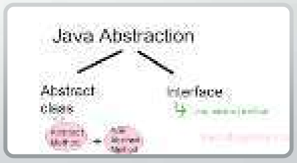
[redacted]

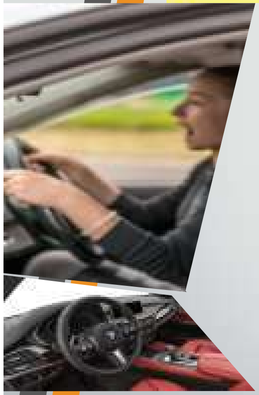
• What is Abstraction in OOP?

- Abstraction is selecting data from a larger pool to show only the relevant details to the object.
- التعامل مع أي شيء دون الخوض بكافة تفاصيله



- In Java, abstraction is accomplished using Abstract classes and interfaces. It is one of the most important concepts of OOPs.





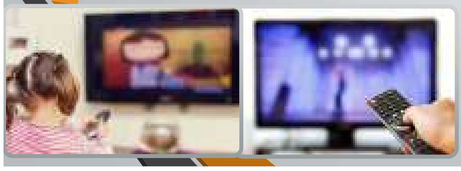
- The class in general is an **A**bstract **D**ata **T**ype (**ADT**).

- **Example1**

  Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc. in the car. This is what abstraction is..

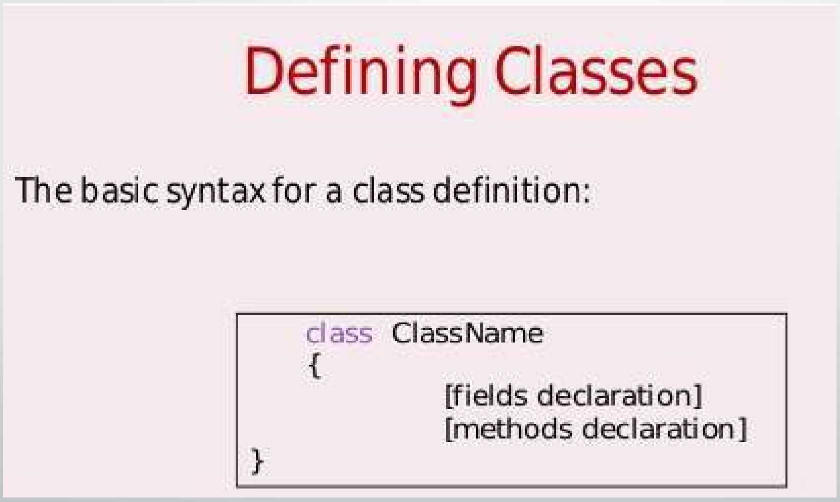
- **Example2**

  We all know how to turn the TV on, but we don’t need to know how it works to enjoy it.



[redacted]

• Defining a class

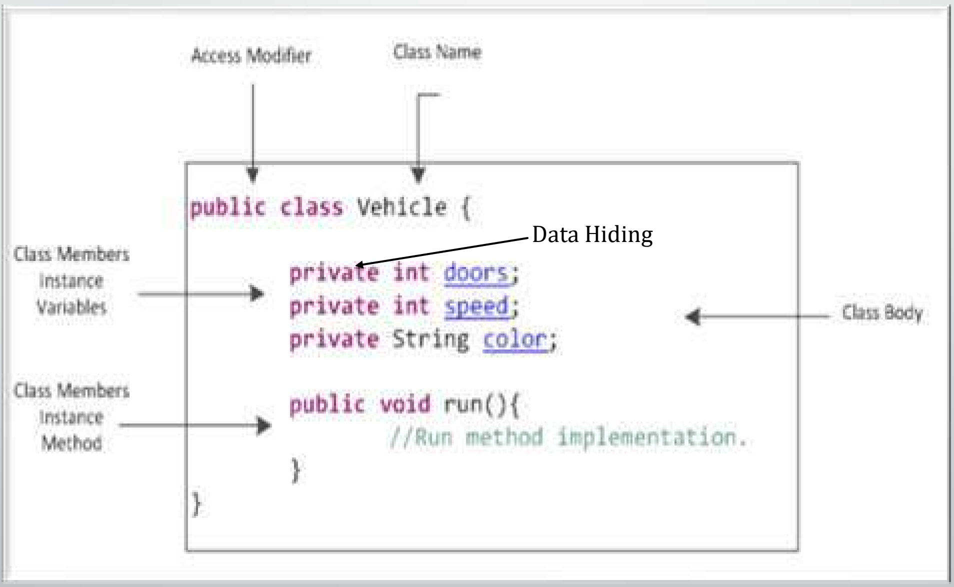


Defining Classes

The basic syntax for a class definition:

```
     class  ClassName
     {
                [fields declaration]
                [methods declaration]
}
```

Example 3

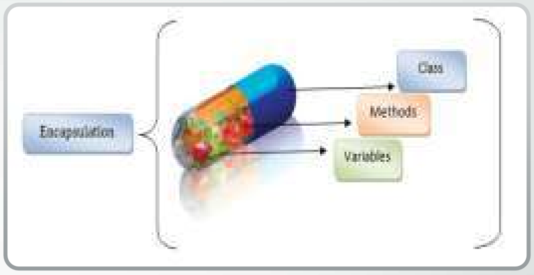

Access Modifier
Class Name
Data Hiding
Class Members Instance Variables
Class Body
Class Members Instance Method

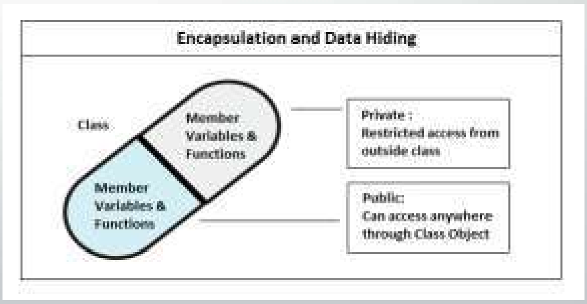
```
public class Vehicle {

        private int doors;
        private int speed;
        private String color;

        public void run(){
                //Run method implementation.
        }
}
```

[redacted]

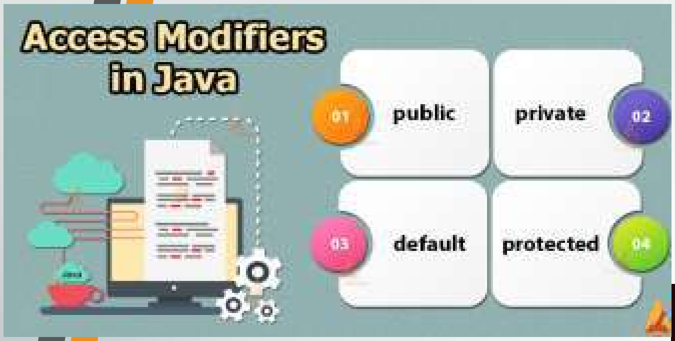
Encapsulation

- In encapsulation, the variables of a class will be hidden (private) from other classes and can be accessed only through the methods of their current class. If they need to be accessed from outside a class then the method should be public.

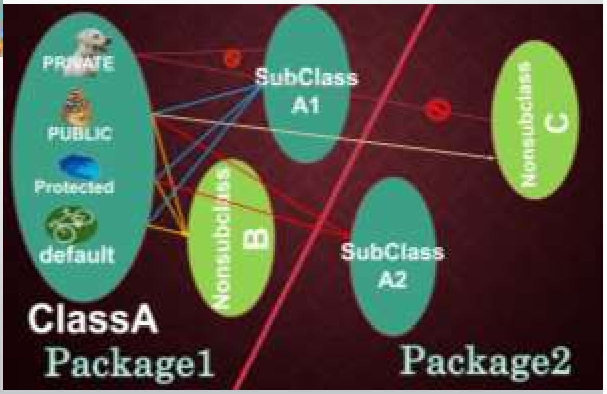



Encapsulation and Data Hiding
Class
Member Variables & Functions
Member Variables & Functions
Private : Restricted access from outside class
Public: Can access anywhere through Class Object

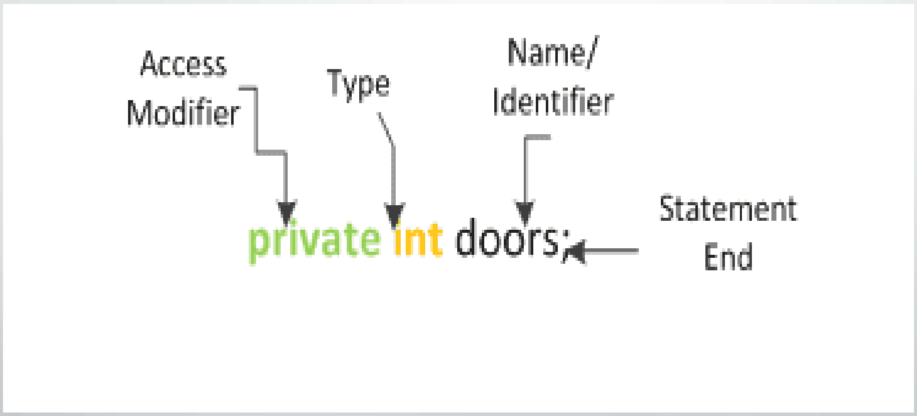
Abstraction .. Encapsulation



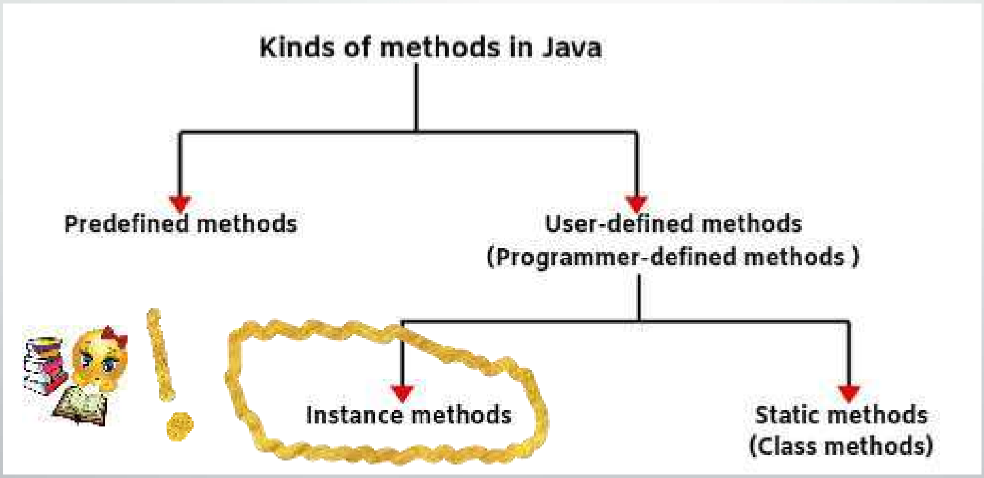
- **Access Modifiers in Java**



- **Declaration of variable members (Instance Variables)**

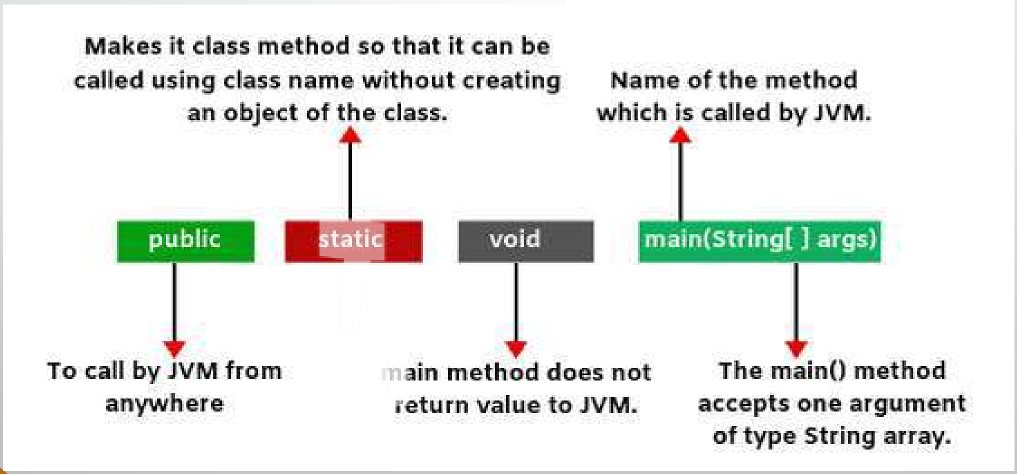


- **Methods Types (kinds)**


Kinds of methods in Java
Predefined methods
User-defined methods (Programmer-defined methods)
Instance methods
Static methods (Class methods)

- **Special method main**

It is a java main method

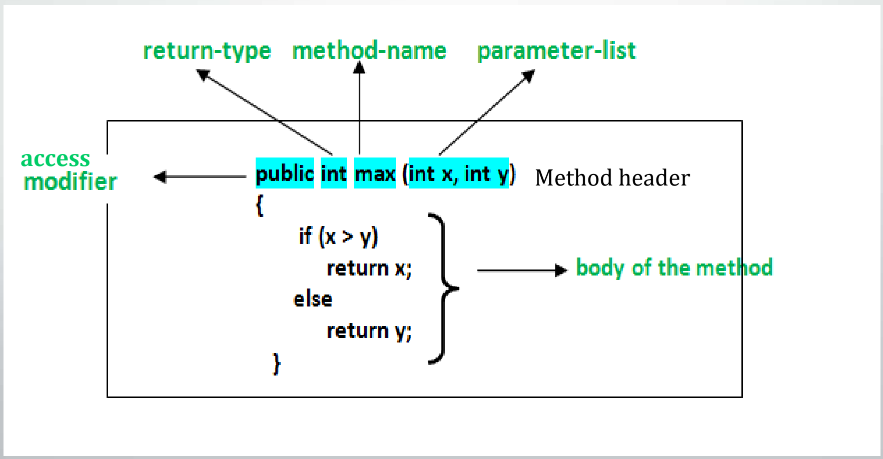

Makes it class method so that it can be called using class name without creating an object of the class.
Name of the method which is called by JVM.
public
static
void
main(String[ ] args)
To call by JVM from anywhere
main method does not return value to JVM.
The main() method accepts one argument of type String array.

[redacted]

• Declaration of Methods

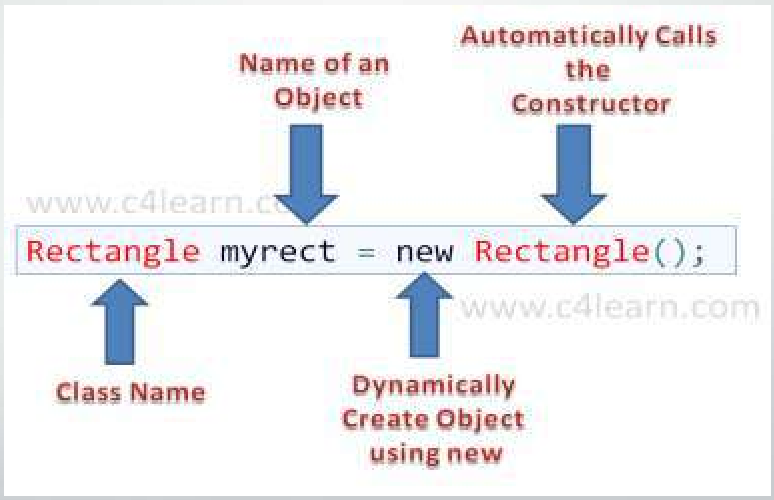
•A **method** is a block of code which only runs when it is called. You can pass data, known as parameters, into a **method**. **Methods** are used to perform certain actions, and they are also known as functions.

Example 4



```
public int max (int x, int y)
{
      if (x > y)
          return x;
      else
          return y;
  }
```

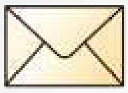
• Creating an object

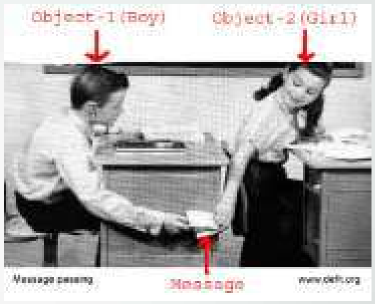

Name of an Object
Automatically Calls the Constructor
www.c4learn.co
www.c4learn.com
Class Name
Dynamically Create Object using new

```
Rectangle myrect = new Rectangle();
```

[redacted]

- **Message Passing**

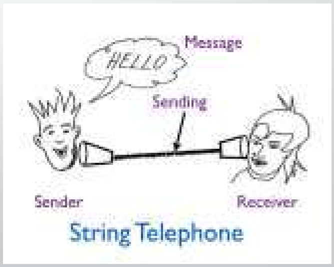




- **Message Components**

A message is composed of three components:

1- An object identifier.
2- A method name.
3- Arguments (Parameters)



Example 5

What exactly does the statement mean in code?

```
public class A {                          // starting of class definition
   public void methodA()    {   }       // this is a class body (definition)
}                                   // Ending of class definition

public class B {
   public void methodB(int z)   {    }    // this is a class body (definition)
}
public class C {
  public static void main(String [] args)    {
                        A a=new A();            // a is an object and  A is an abstract data type ADT
                        B b=new B();
      a.methodA( );       b.methodB(7);    // message passing

     }}
1     2          3         1     2          3
```

[redacted]

Example 6

Define a class called Time which has three integer members :

s for seconds

m for minuets

h for hour

Solution (الحل)

```
public class Time{
private int s,m,h;
public void set(){---}
public void print(){…}
}
```

Example 7

Define a class called Date which has three instance

d for day

m for month

y for year

Solution (الحل)

```
public class Date{
private int d;
private int m;
private int y;
public void set(){---}
public void print(){…}
}
```

Write either true or false and why?

👎 *private int d, m;*
private int y

private int d,m, y; 👍

👎 *private int d; private int m; int y;*

👎 *private int d; m; int y;*

👎 *private int d, private int m, private int y;*

[redacted]

- Examples for defining a class
  - Examples for creating an object
- behavior and state
- defining more than one object
- defining array of objects

- **Example 0**

let us consider the example of Date (التاريخ).

A Date consists of three integer members and two member methods:

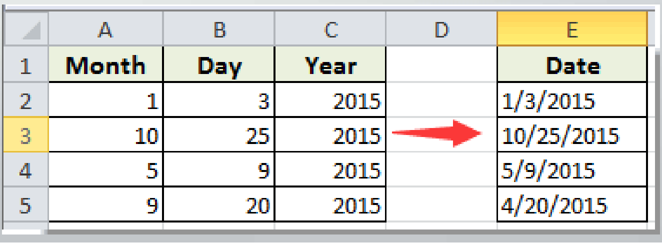
d is for day

m is for month

y is for year

set method ; it is used for stetting date

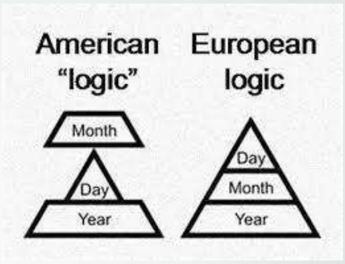
print method; it is used for printing date



	A	B	C	D	E
1	Month	Day	Year		Date
2	1	3	2015		1/3/2015
3	10	25	2015	→	10/25/2015
4	5	9	2015		5/9/2015
5	9	20	2015		4/20/2015

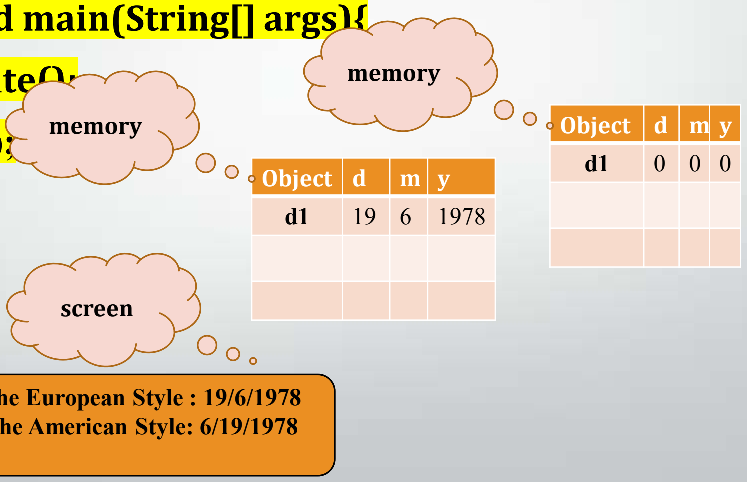
Define a java class for Date class.

```
public class Date{
private int d,m,y;
public void set (int dd,int mm,int yy){
d=dd; m=mm; y=yy;}
public void print_a(){
System.out.println(“The American Style: ”+ m+”/”+d+”/”+y);}
public void print_e(){
System.out.println(“The European Style :“+d+”/”+m+”/”+y);}
}
```



Use the Date class for creating one date object called d1 and set the date to 19-6-1978.Print the date in American style and European style.

```
public class Main{
public static void main(String[] args){
Date d1=new Date();
d1.set(19,6,1978);
d1.print_e();
d1.print_a();
}
}
```



Object	d	m	y
d1	0	0	0

Object	d	m	y
d1	19	6	1978

```
The European Style : 19/6/1978
The American Style: 6/19/1978
```

[redacted]

Use the Date class for creating two date objects called d1 and d2 and set the date to 19/6/1978 and 5/7/1974 respectively. Print the date in American style and European style.

```
public class Main{
Public static void main(String[] args){
Date d1=new Date();
d1.set(19,6,1978);
d1.print_e();
d1.print_a();
Date d2=new Dat
d2.set(5,7,1974);
d2.print_e();
d2.print_a(); } }
```

memory

Objec t	d	m	y
d1	0	0	0
d2	0	0	0

memory

Object	d	m	y
d1	19	6	1978
d2	5	7	1974

screen

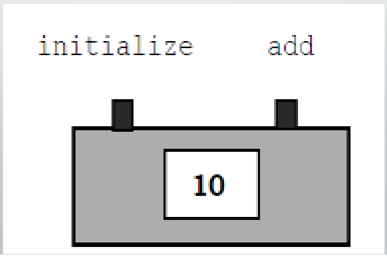
```
The European Style : 19/6/1978
The American Style: 6/19/1978
The European Style : 5/7/1974
The American Style: 7/5/1974
```

- **Example 1**

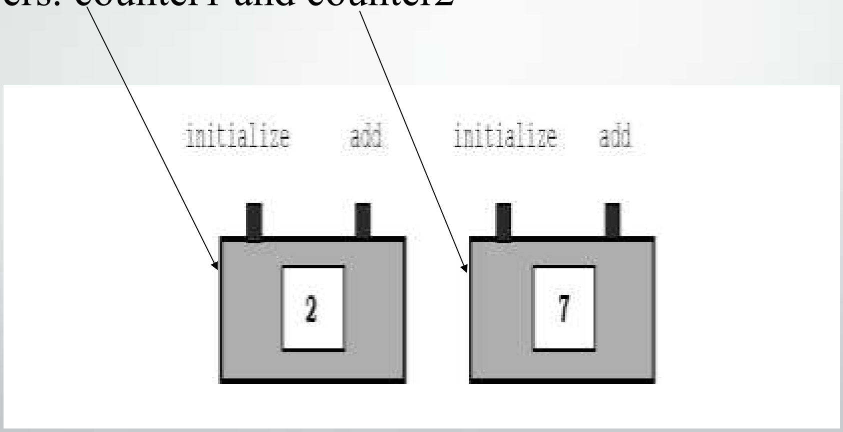
let us consider the example of counters (المعدادات).

A counter is a device that keeps account of the number of times an event has occurred. It has two buttons: an initialize button (زر القيمة الابتدائية) that resets the counter to 0, and

an add button (زر الاضافة) that adds 1 to its present number as shown in the following figure, It has a counter with a number 10.



The next figure shows two more counters with 2 and 7 numbers. counter1 and counter2

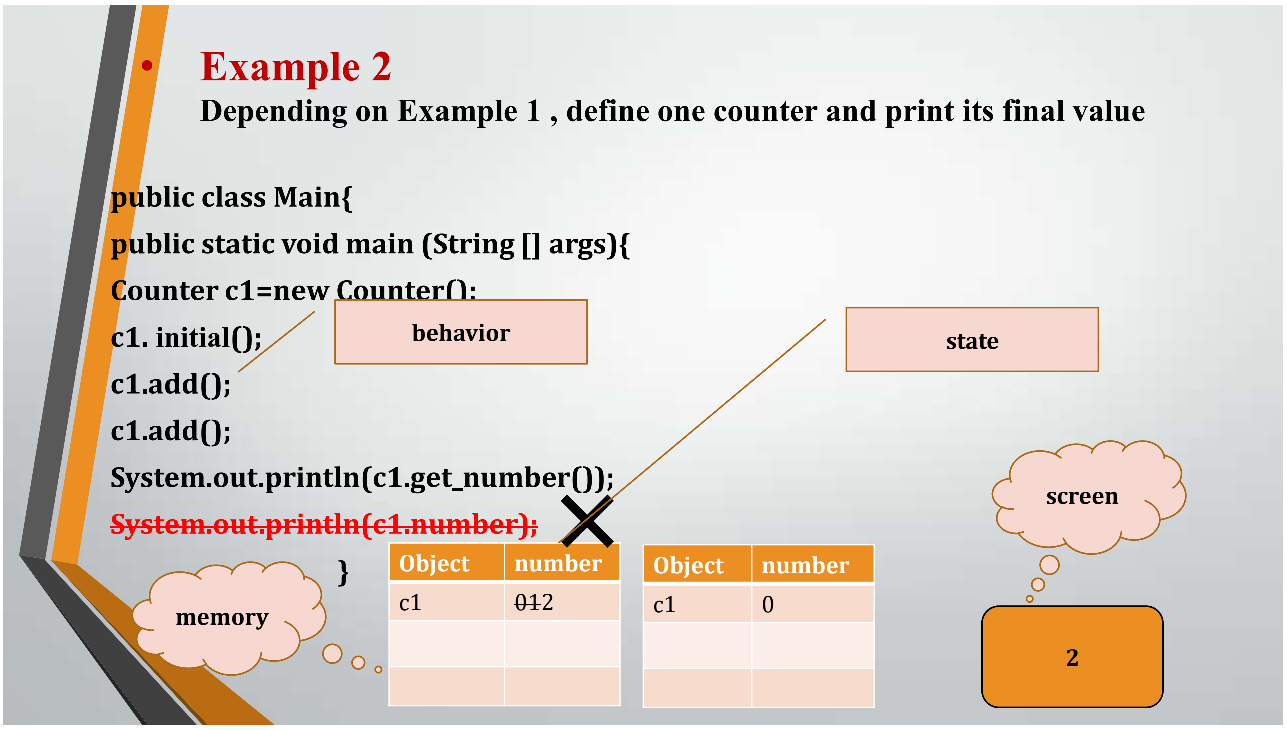


```
public class Counter{
// instance variable
private int number;
// method to increment counter by 1
public void add( ){
number=number+1;}
// method to initialize counter by 0
public void initial( ){
number=0;}
// method to return counter number
public int get_number( ){
return(number);}
}
```

The file is Counter.java
After compilation correctly,
The file is Counter.class

You should note that the order of methods within a class is not important but the order of calling them is very important.

[redacted]



Example 2

Depending on Example 1 , define one counter and print its final value

```
public class Main{
public static void main (String [] args){
Counter c1=new Counter();
c1. initial();
c1.add();
c1.add();
System.out.println(c1.get_number());
```

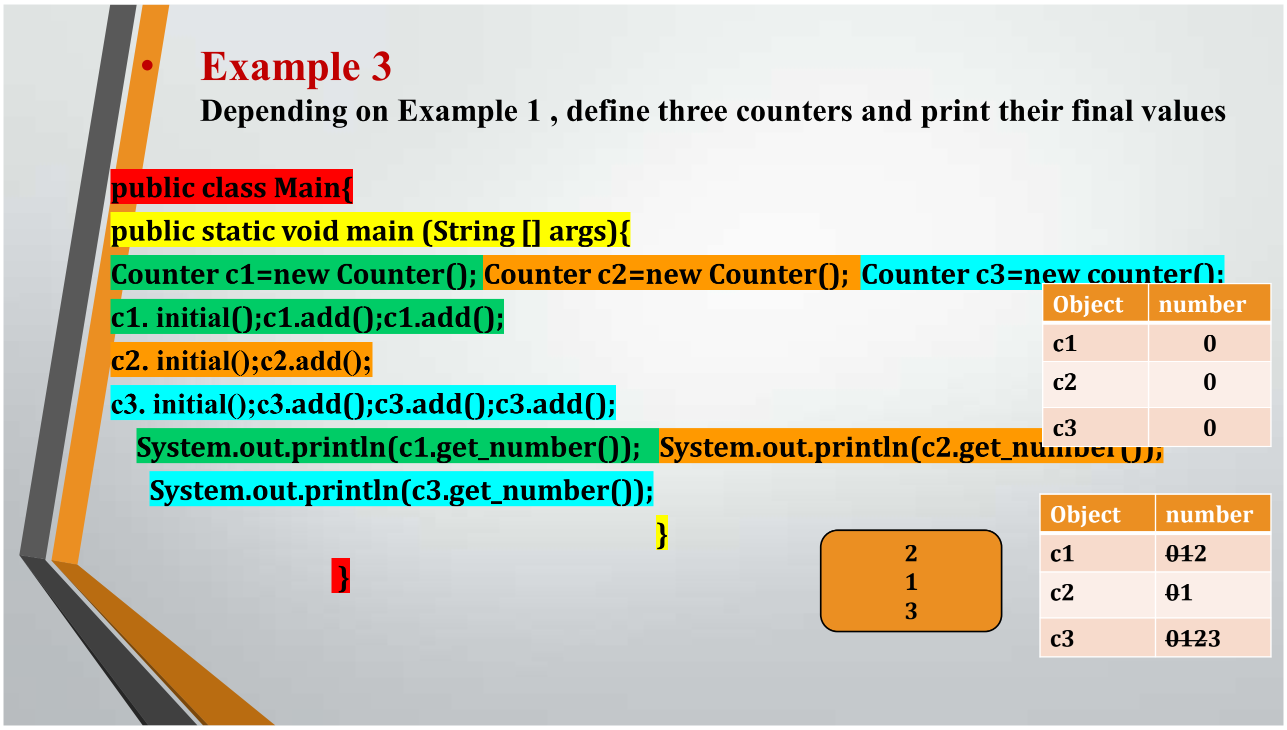
~~System.out.println(c1.number);~~

}

behavior
state
screen
memory
2

Object	number
c1	~~0~~~~1~~2

Object	number
c1	0



Example 3

Depending on Example 1 , define three counters and print their final values

```
public class Main{
public static void main (String [] args){
Counter c1=new Counter(); Counter c2=new Counter(); Counter c3=new counter();
c1. initial();c1.add();c1.add();
c2. initial();c2.add();
c3. initial();c3.add();c3.add();c3.add();
  System.out.println(c1.get_number()); System.out.println(c2.get_number());
   System.out.println(c3.get_number());
                                                   }
                              }
```

Object	number
c1	0
c2	0
c3	0

Object	number
c1	~~01~~2
c2	~~0~~1
c3	~~012~~3

[redacted]

Example 4

Depending on Example 1 , define three counters and print the summation of their values

```
public class Main{
public static void main (String [] args){
Counter c1=new Counter(); Counter c2=new Counter(); Counter c3=new counter();
c1. initial()c1.add();c1.add();
c2. initial();c2.add();
c3. initial();c3.add();c3.add();c3.add();
 int sum;
sum=c1.get_number()+c2.get_number()+c3.get_number();
System.out.println(sum);                                    }
                  }
```

Object	number
c1	0
c2	0
c3	0

Object	number
c1	~~01~~2
c2	~~0~~1
c3	~~012~~3

6

Example 5

Depending on Example 1 , define two counters and print the value of greater one.

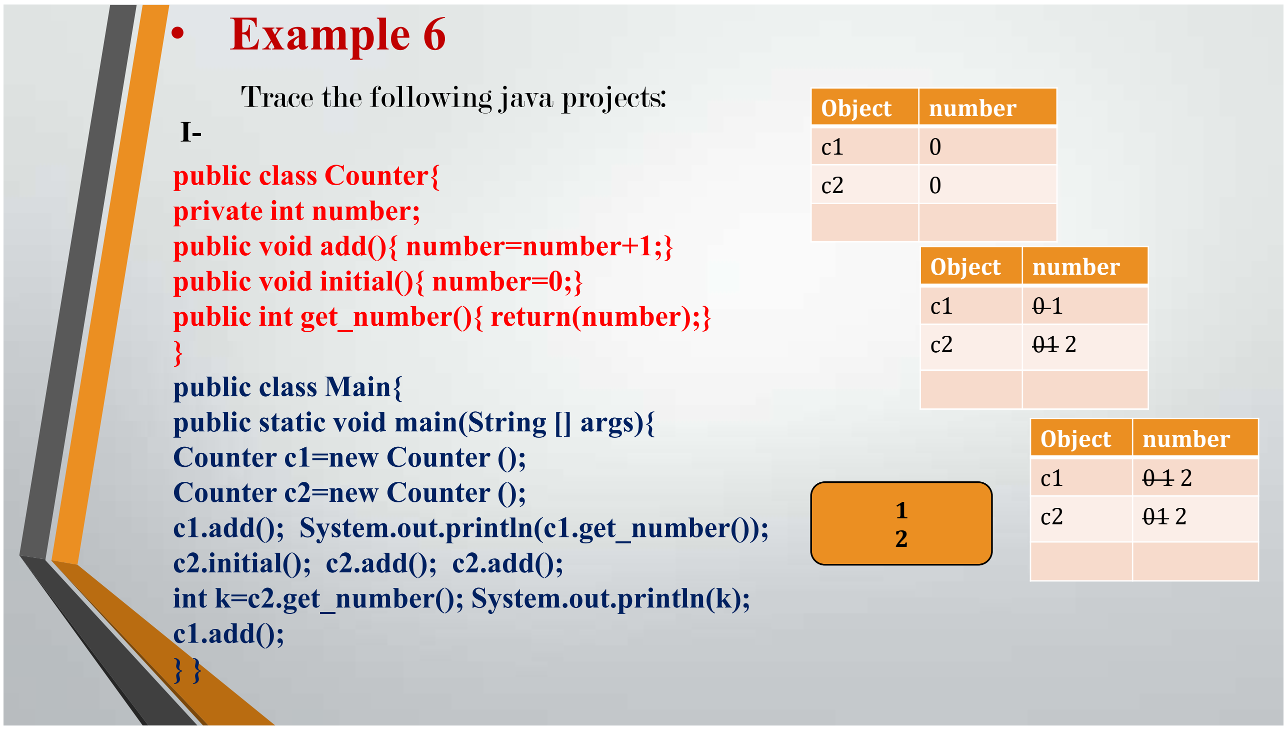
```
public class Main{
public static void main (String [] args){
Counter c1=new Counter(); Counter c2=new Counter();
c1. initial()c1.add();c1.add();c1.add();c1.add();
c2. initial();c2.add();c2.add();
 if(c1.get_number()>c2.get_number())  System.out.println(c1.get_number());
else   System.out.println(c2.get_number());
}
                  }
```

Object	number
c1	0
c2	0

Object	number
c1	~~0123~~ 4
c2	~~01~~ 2

4

[redacted]



Example 6

Trace the following java projects:

I-

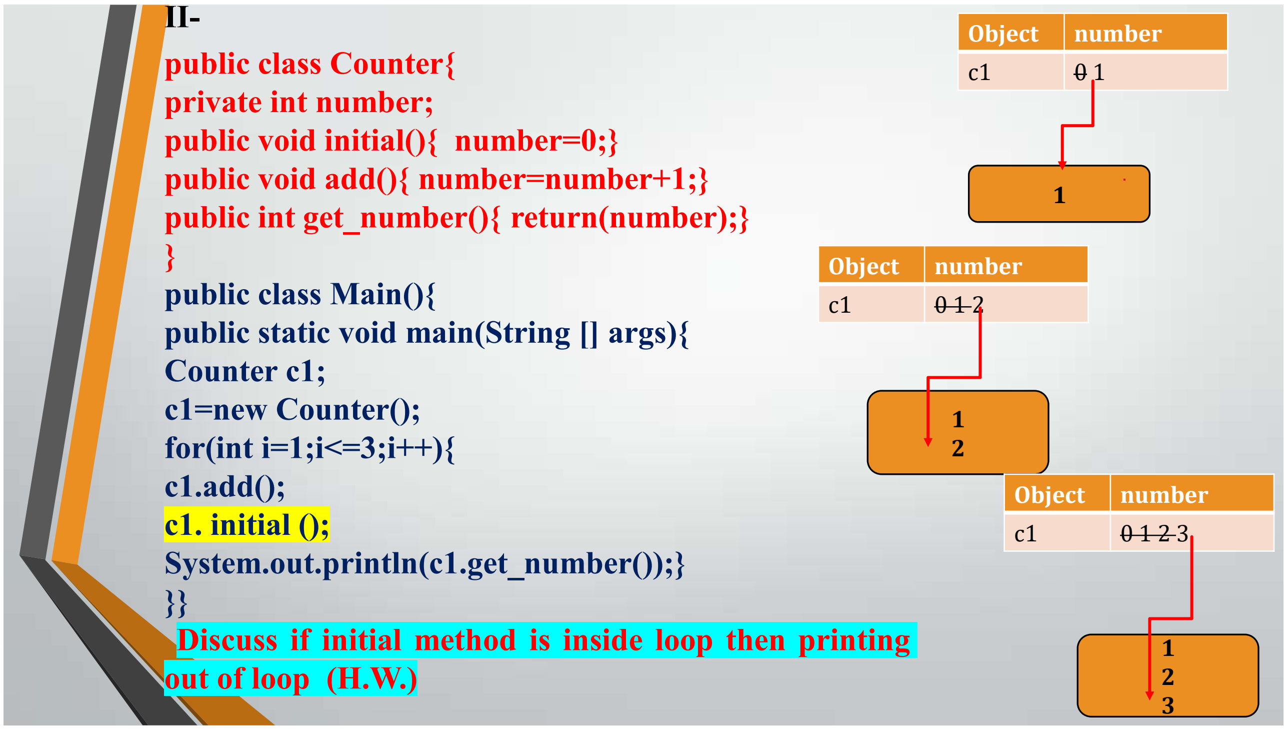
```
public class Counter{
private int number;
public void add(){ number=number+1;}
public void initial(){ number=0;}
public int get_number(){ return(number);}
}
public class Main{
public static void main(String [] args){
Counter c1=new Counter ();
Counter c2=new Counter ();
c1.add();  System.out.println(c1.get_number());
c2.initial();  c2.add();  c2.add();
int k=c2.get_number(); System.out.println(k);
c1.add();
} }
```

Object	number
c1	0
c2	0

Object	number
c1	~~0~~ 1
c2	~~01~~ 2

```
1
2
```

Object	number
c1	~~0 1~~ 2
c2	~~01~~ 2



II-

```
public class Counter{
private int number;
public void initial(){  number=0;}
public void add(){ number=number+1;}
public int get_number(){ return(number);}
}
public class Main(){
public static void main(String [] args){
Counter c1;
c1=new Counter();
for(int i=1;i<=3;i++){
c1.add();
c1. initial ();
System.out.println(c1.get_number());}
}}
```

Discuss if initial method is inside loop then printing out of loop (H.W.)

Object	number
c1	~~0~~ 1

```
1
```

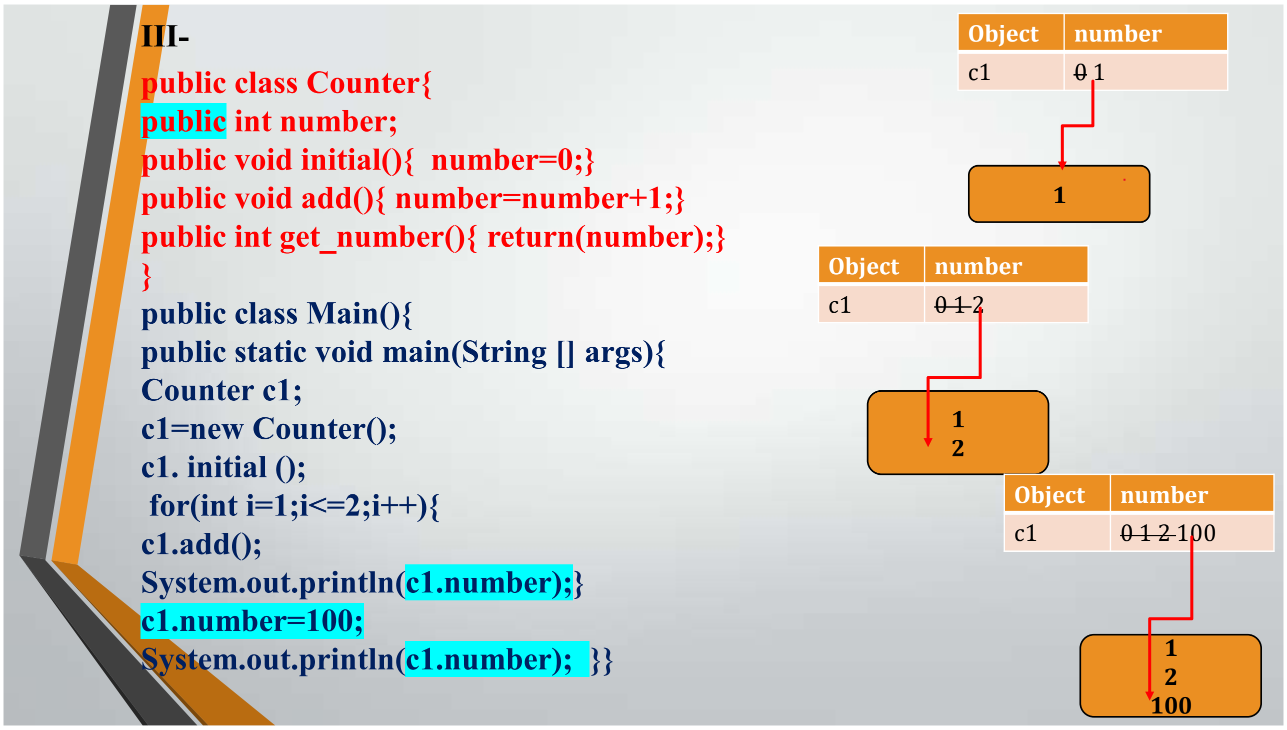
Object	number
c1	~~0 1~~ 2

```
1
2
```

Object	number
c1	~~0 1 2~~ 3

```
1
2
3
```

[redacted]



III-

```
public class Counter{
public int number;
public void initial(){  number=0;}
public void add(){ number=number+1;}
public int get_number(){ return(number);}
}
public class Main(){
public static void main(String [] args){
Counter c1;
c1=new Counter();
c1. initial ();
 for(int i=1;i<=2;i++){
c1.add();
System.out.println(c1.number);}
c1.number=100;
System.out.println(c1.number);  }}
```

Object	number
c1	~~0~~ 1

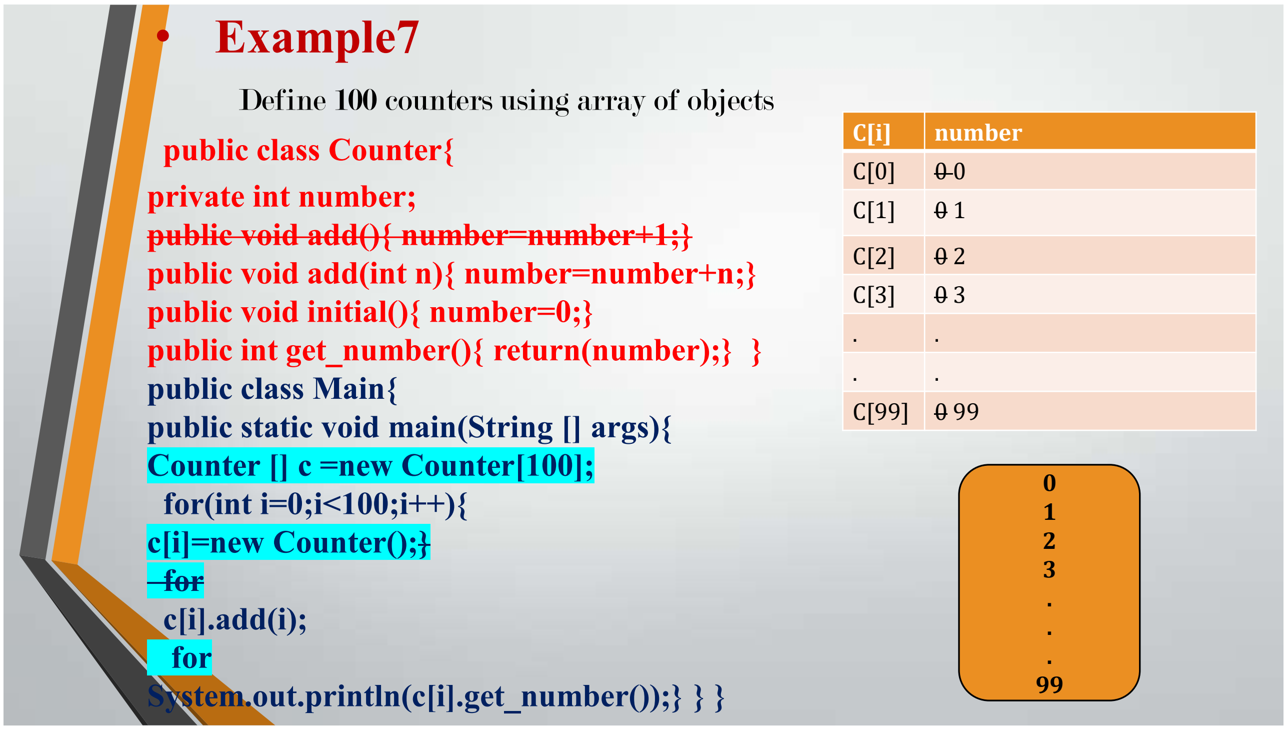
1

Object	number
c1	~~0 1~~ 2

1
2

Object	number
c1	~~0 1 2~~ 100

1
2
100



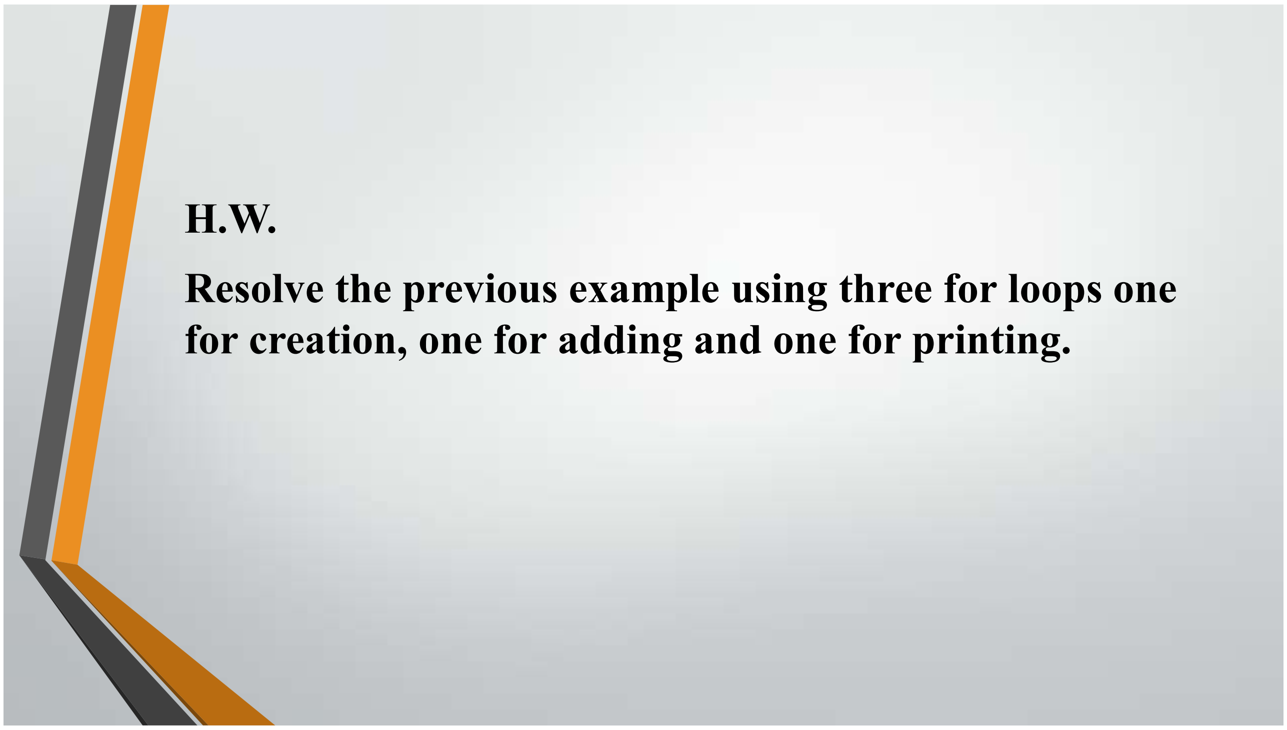
Example7

Define 100 counters using array of objects

```
 public class Counter{
private int number;
~~public void add(){ number=number+1;}~~
public void add(int n){ number=number+n;}
public void initial(){ number=0;}
public int get_number(){ return(number);}  }
public class Main{
public static void main(String [] args){
Counter [] c =new Counter[100];
  for(int i=0;i<100;i++){
c[i]=new Counter();~~}~~
~~ for~~
  c[i].add(i);
  for
System.out.println(c[i].get_number());} } }
```

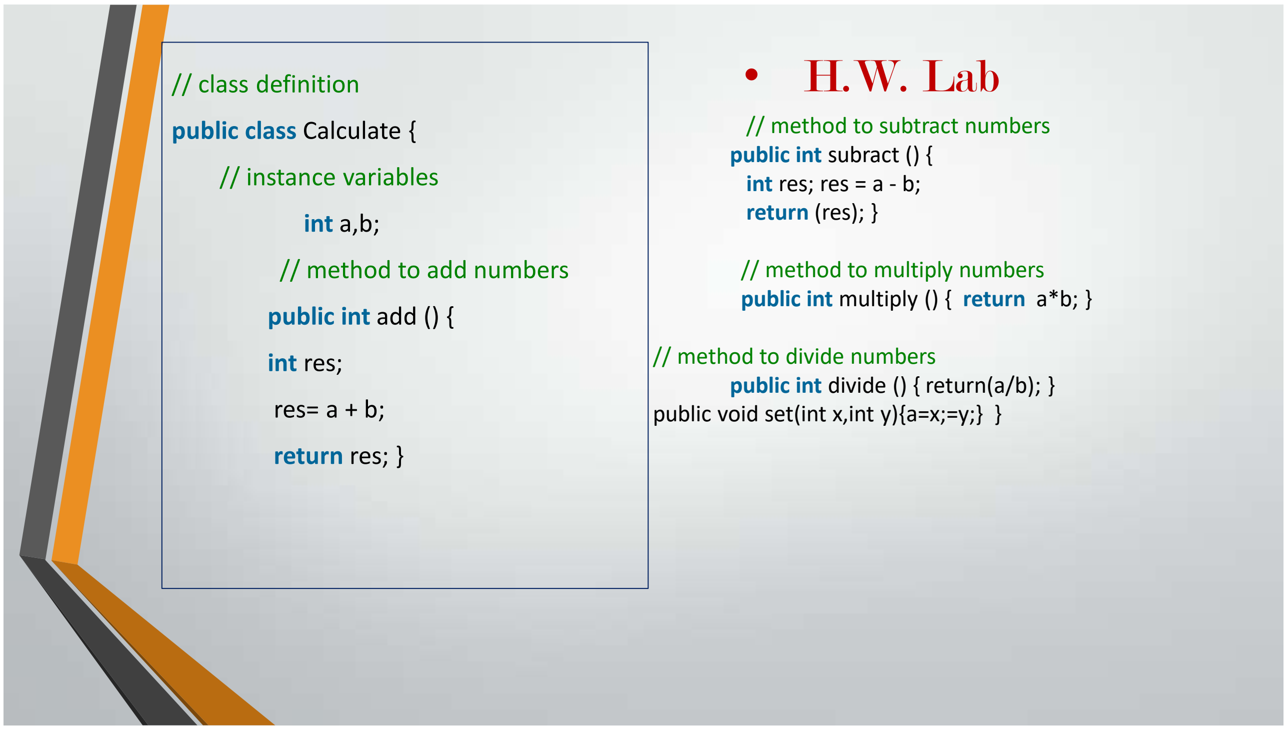
C[i]	number
C[0]	~~0~~ 0
C[1]	~~0~~ 1
C[2]	~~0~~ 2
C[3]	~~0~~ 3
.	.
.	.
C[99]	~~0~~ 99

0
1
2
3
.
.
.
99



H.W.

Resolve the previous example using three for loops one for creation, one for adding and one for printing.



```
// class definition
public class Calculate {
      // instance variables
            int a,b;
            // method to add numbers
          public int add () {
          int res;
           res= a + b;
           return res; }
```

- H.W. Lab

```
      // method to subtract numbers
    public int subract () {
      int res; res = a - b;
      return (res); }

      // method to multiply numbers
      public int multiply () {  return  a*b; }

// method to divide numbers
          public int divide () { return(a/b); }
public void set(int x,int y){a=x;=y;}  }
```

[redacted]

• H.W. Lab

```
public class Main {
public static void main(String[] args) {
    // creating object of Class
    Calculate c1 = new Calculate();  c1.set(49,4);
         // calling the methods of Calculate class
System.out.println("Addition is :" + c1.add());
System.out.println("Subtraction is :" + c1.subtract());
System.out.println("Multiplication is :" + c1.multiply());
System.out.println("Division is :" + c1.divide());
 }   }
```

• Polymorphism I (*Static Polymorphism*)-Method overloading

- What is method overloading?
- What are method overloading types?
- How is method overloading implemented in Java?

- Examples

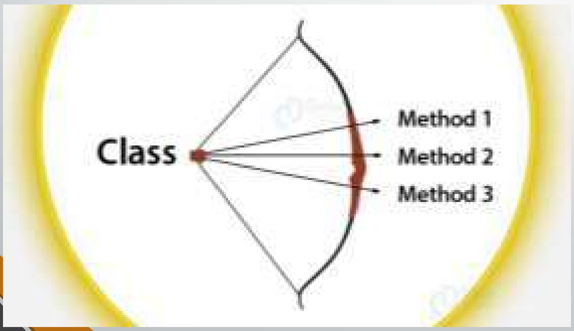
- Applying practical examples using NetBeans

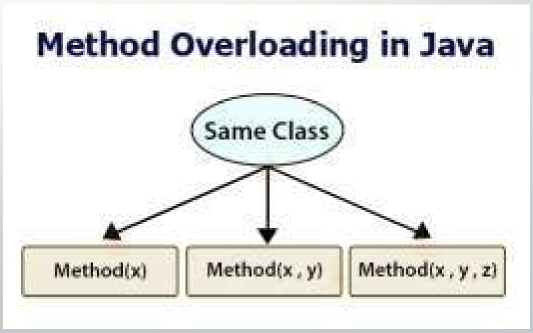
[redacted]

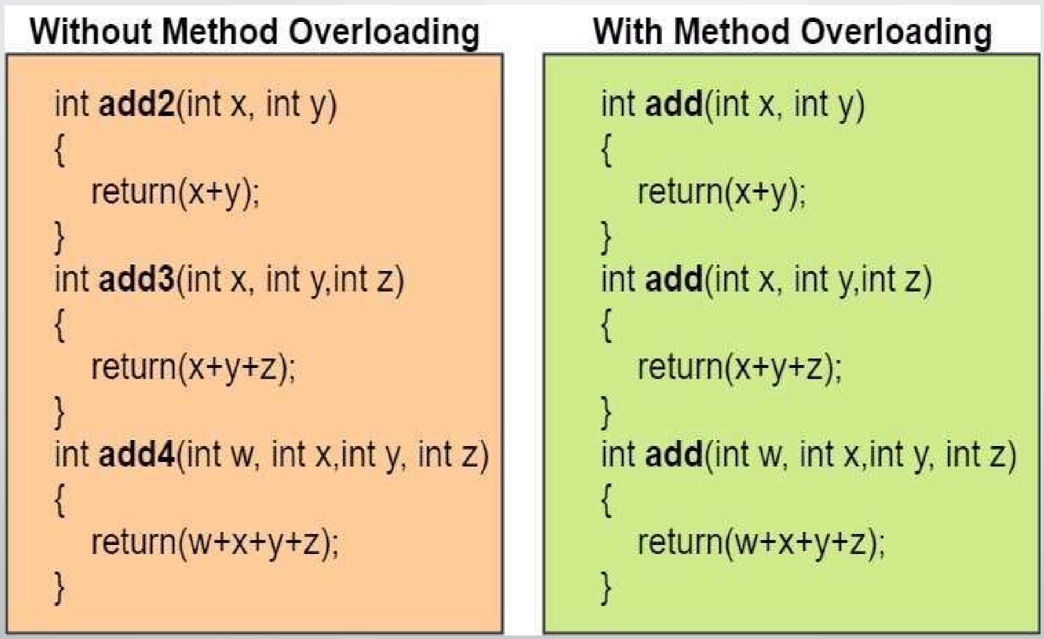
• Method Overloading

Method overloading is a feature that allows a class to have more than one **method** having the same name, if their argument lists are different.

In order to overload a method, the argument lists of the methods must differ in either of these:







Without Method Overloading

```
int add2(int x, int y)
{
   return(x+y);
}
int add3(int x, int y,int z)
{
   return(x+y+z);
}
int add4(int w, int x,int y, int z)
{
   return(w+x+y+z);
}
```

With Method Overloading

```
int add(int x, int y)
{
   return(x+y);
}
int add(int x, int y,int z)
{
   return(x+y+z);
}
int add(int w, int x,int y, int z)
{
   return(w+x+y+z);
}
```

Four ways to overload a method:

In order to overload a method in valid way , the argument lists of the methods must differ in either of these:

1. Number of Parameters

For example:

```
public int add(int a, int b) {----------}
public int add(int a, int b, int c) {------}
add(4,3); add(3,4,5);
```

2. Data type of Parameters

For example:

```
public int add(int a, int b) {-----}

public int add(int x, float y){-----}
```

~~public int add(int a, float b){-----}~~

```
add(5,7);
    add(5,7.6f);
```

3. Sequence of Data type of Parameters

For example:

```
public int add(int a, float b) {-----}
public int add(float a , int b){-----}
```
~~public int add(float b , int a){-----}~~

```
add(3,3.4f);
add(5.4f,9);
```

4. Number and data type of Parameters

For example:

```
public int  add(int a , int b) {-----}
public int  add(int a, float b, int c) {-----}

public  int add(int a, float b, float c){----}
```
~~public int add(int x,flaot y, float z){---}~~

```
add(5,3.4f,5.6f);
```

[redacted]

Invalid case of method overloading:

When I say argument list

I am not talking about return type of the method,
for example:
if two methods have same name,
same parameters and have different return type,
hen this is not a valid method overloading example.
his will throw compilation error.

```
public int add(int a, int b) {-----}
public float add(int a, int b) {-----}
public double add(int b, int a) {-----}
add(4,5);
```

Method overloading is an example of ***Static Polymorphism***.

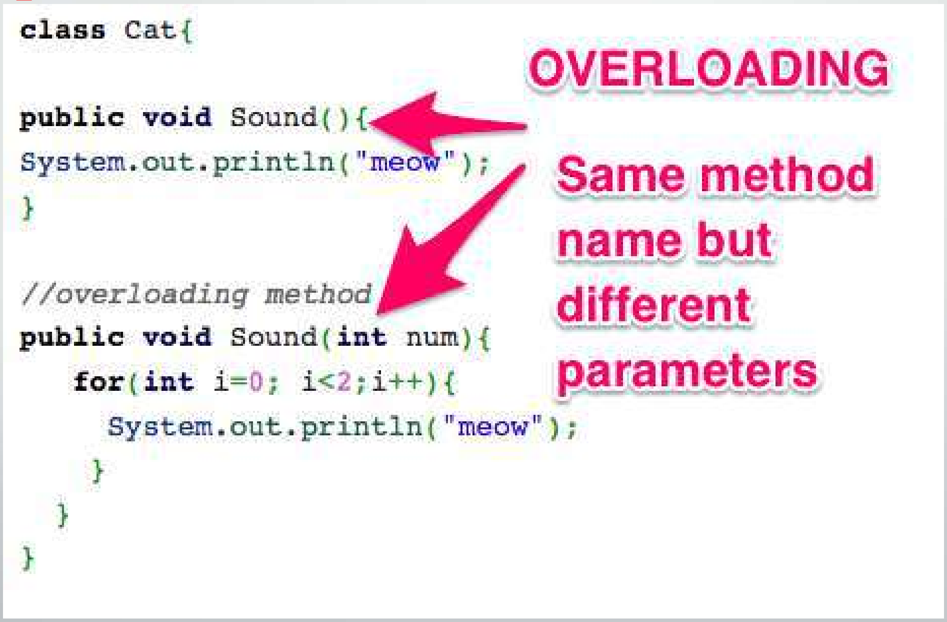
Points to Note (ملاحظات)

1. Static Polymorphism is also known as compile time binding or early binding.

2. Static binding happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

[redacted]

Example 1

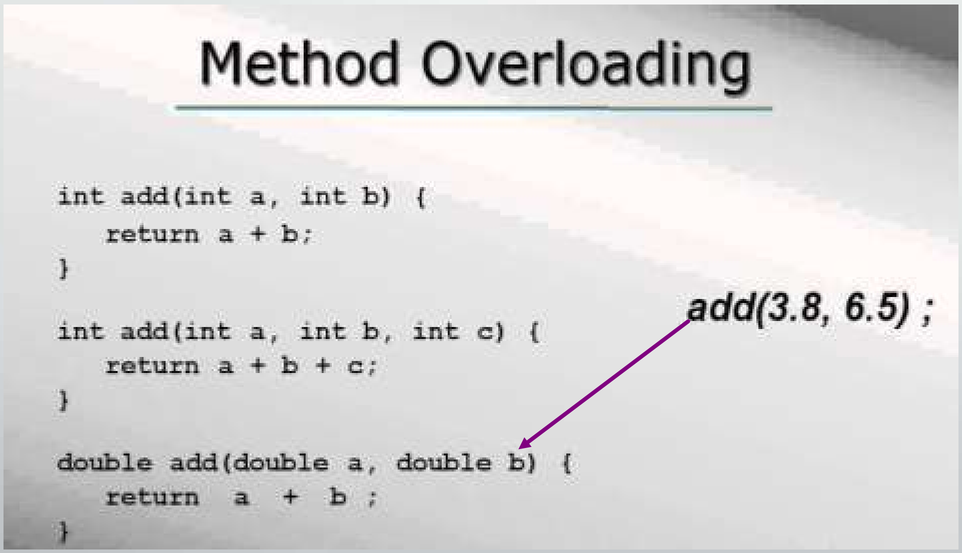


```
class Cat{

public void Sound(){
System.out.println("meow");
}

//overloading method
public void Sound(int num){
    for(int i=0; i<2;i++){
      System.out.println("meow");
    }
  }
}
```

OVERLOADING

Same method name but different parameters

Example 2

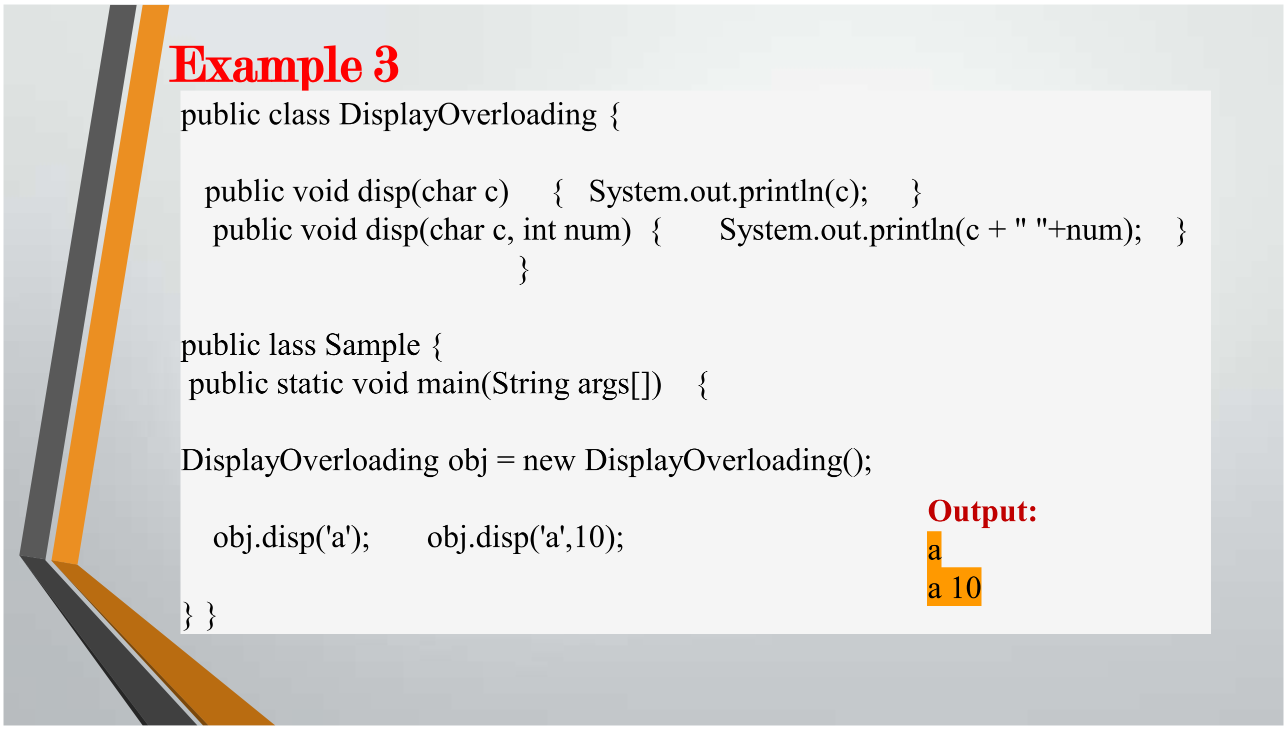


Method Overloading

```
int add(int a, int b) {
   return a + b;
}

int add(int a, int b, int c) {
   return a + b + c;
}

double add(double a, double b) {
   return  a  +  b ;
}
```

add(3.8, 6.5) ;

[redacted]

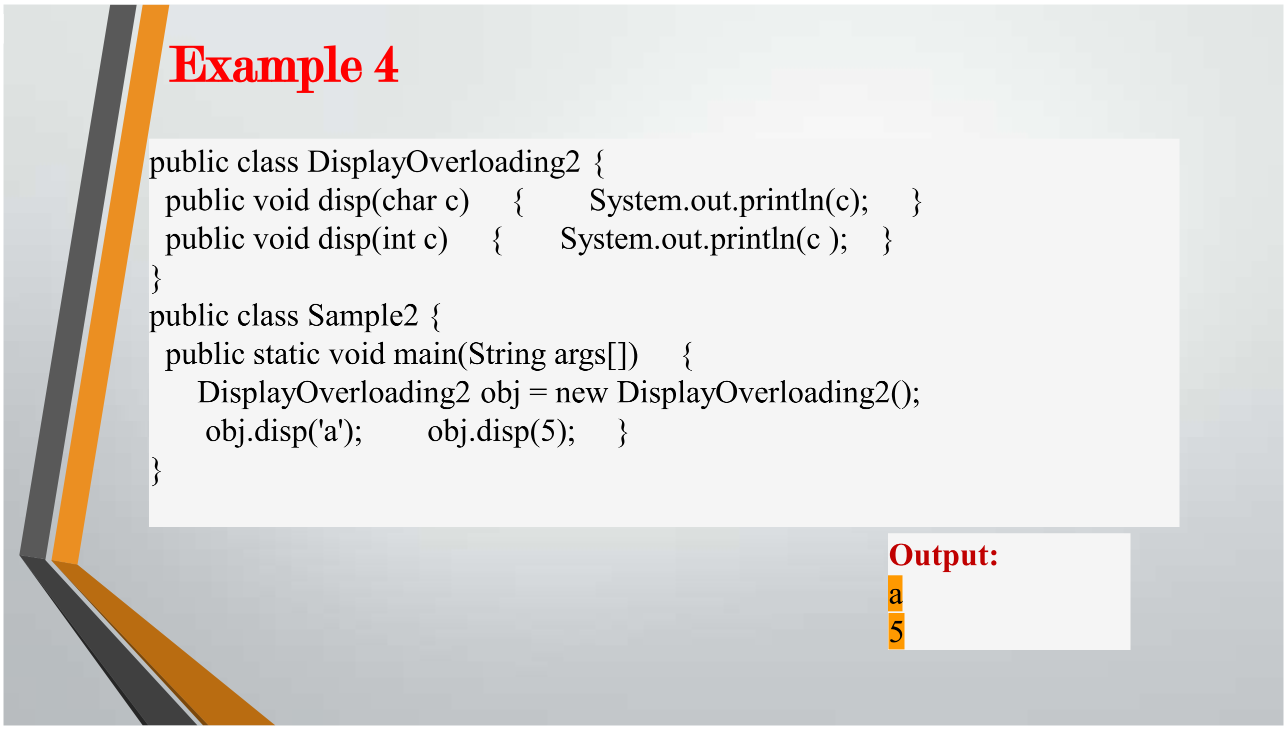


Example 3

```
public class DisplayOverloading {

  public void disp(char c)     {   System.out.println(c);     }
   public void disp(char c, int num)  {       System.out.println(c + " "+num);    }
                                       }

public lass Sample {
 public static void main(String args[])    {

DisplayOverloading obj = new DisplayOverloading();

   obj.disp('a');       obj.disp('a',10);

} }
```

Output:

```
a
a 10
```

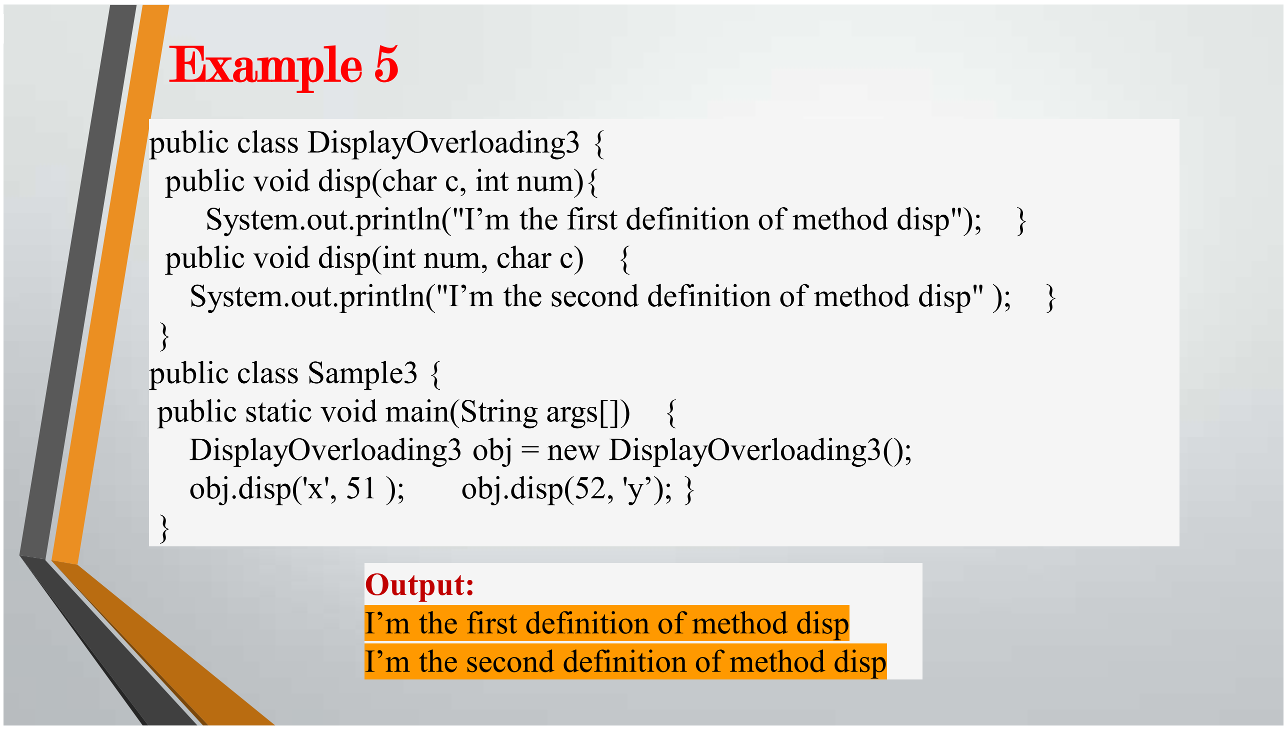


Example 4

```
public class DisplayOverloading2 {
  public void disp(char c)     {       System.out.println(c);     }
  public void disp(int c)     {      System.out.println(c );    }
}
public class Sample2 {
  public static void main(String args[])     {
      DisplayOverloading2 obj = new DisplayOverloading2();
       obj.disp('a');        obj.disp(5);     }
}
```

Output:

```
a
5
```

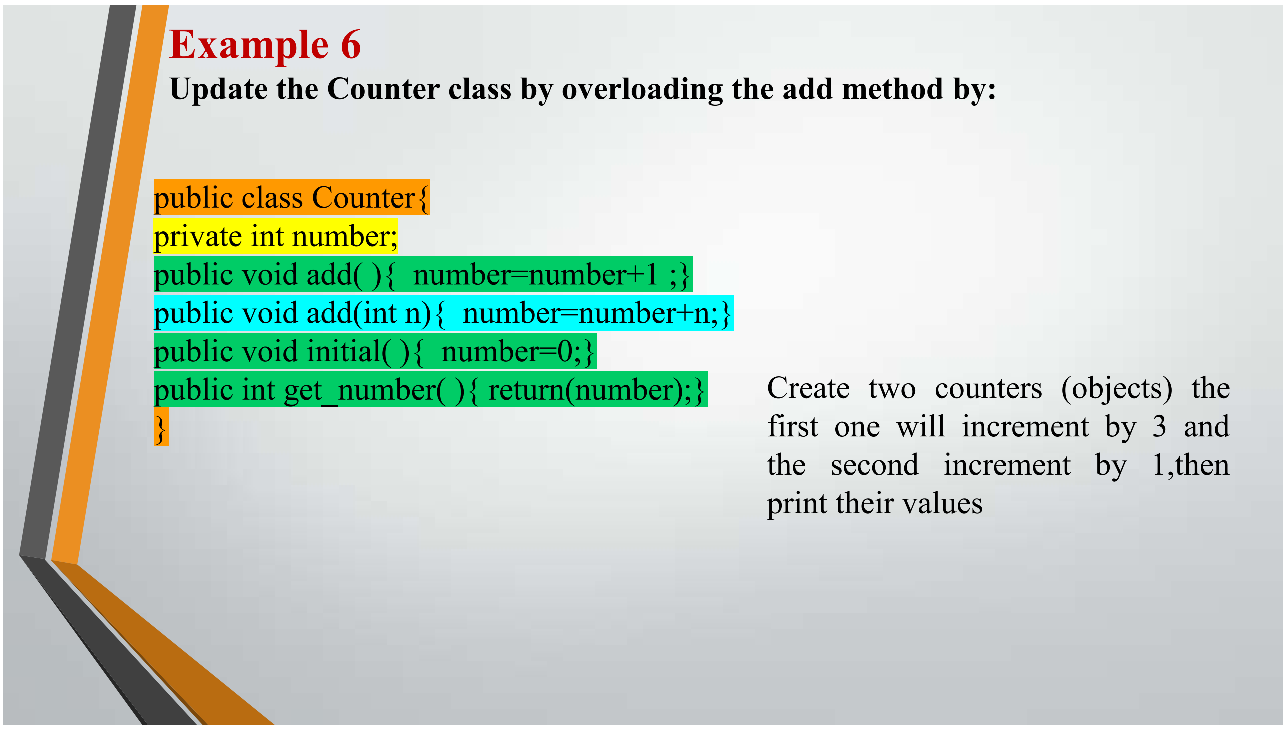


Example 5

```
public class DisplayOverloading3 {
  public void disp(char c, int num){
       System.out.println("I'm the first definition of method disp");    }
  public void disp(int num, char c)    {
     System.out.println("I'm the second definition of method disp" );    }
 }
public class Sample3 {
 public static void main(String args[])    {
     DisplayOverloading3 obj = new DisplayOverloading3();
     obj.disp('x', 51 );       obj.disp(52, 'y'); }
 }
```

Output:

```
I'm the first definition of method disp
I'm the second definition of method disp
```



Example 6

Update the Counter class by overloading the add method by:

```
public class Counter{
private int number;
public void add( ){  number=number+1 ;}
public void add(int n){  number=number+n;}
public void initial( ){  number=0;}
public int get_number( ){ return(number);}
}
```

Create two counters (objects) the first one will increment by 3 and the second increment by 1,then print their values

[redacted]

```
public class Main{
public static void main (String [] args){
Counter c1=new Counter(); Counter c2=new Counter();
c1. initial();c1.add(3);  …… c1.add();c1.add();c1.add();
c2. initial();c2.add();        c2.add(1);
System.out.println(c1.get_Number());
System.out.println(c2.get_Number());
}
}
```

Object	number
c1	0
c2	0

Object	number
c1	~~0~~ 3
c2	~~0~~ 1

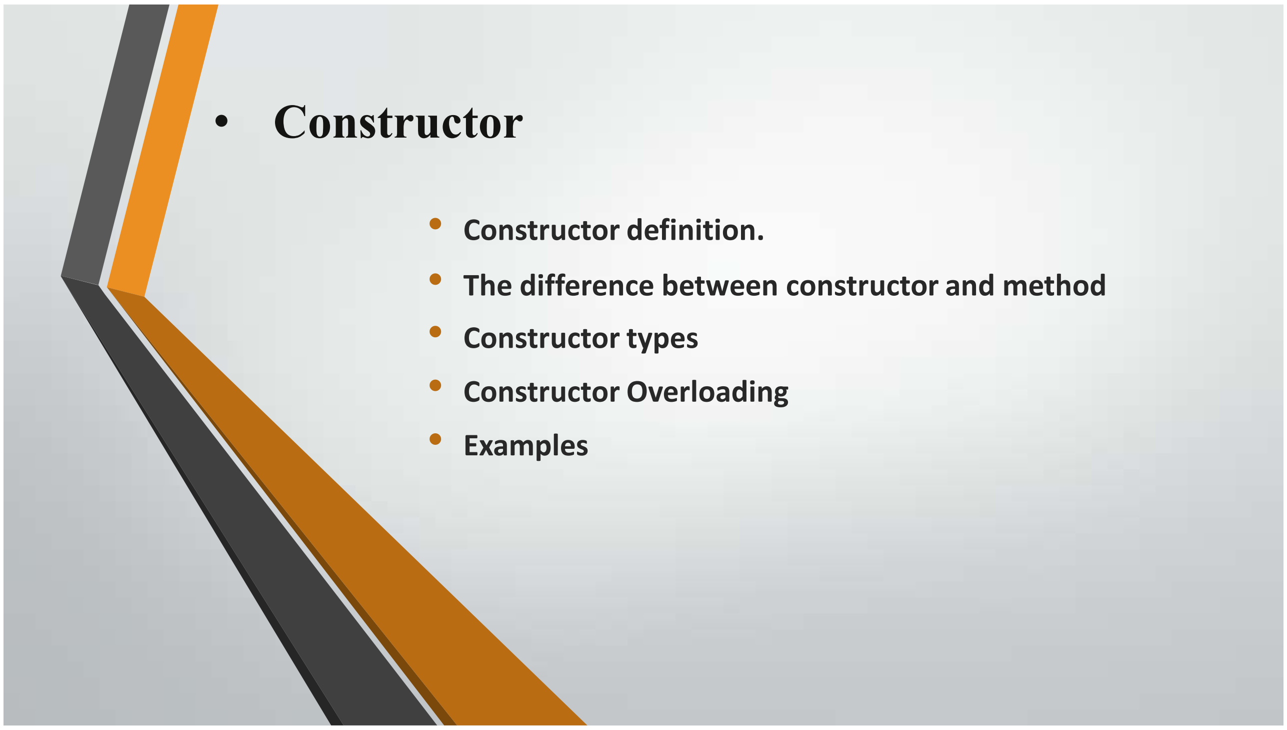
3
1

H.W. LAB

Define a class called B_Date which has 3 integer instance variables and two methods set and print, the set method has been overloaded:

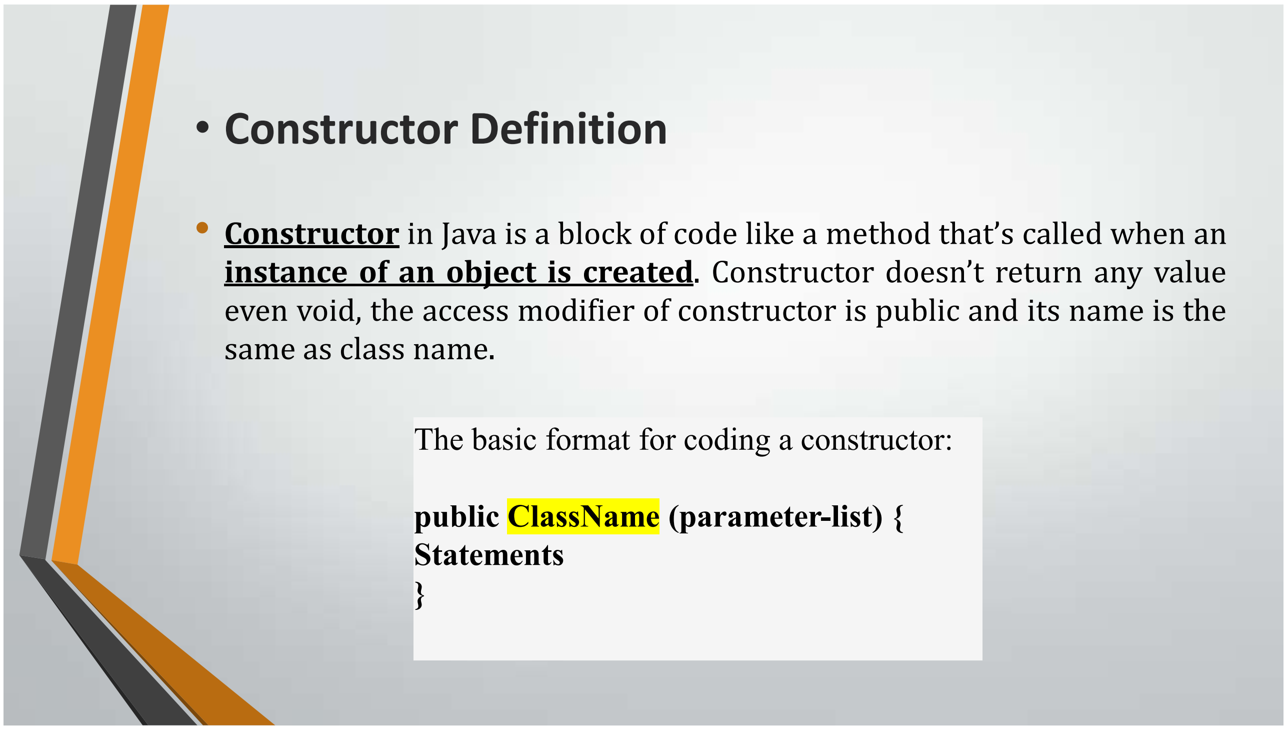
set( )
set(int, int ,int)

Use the above class to create 2 objects and print the details of the older one according to its birth date year.



• Constructor

- Constructor definition.
- The difference between constructor and method
- Constructor types
- Constructor Overloading
- Examples



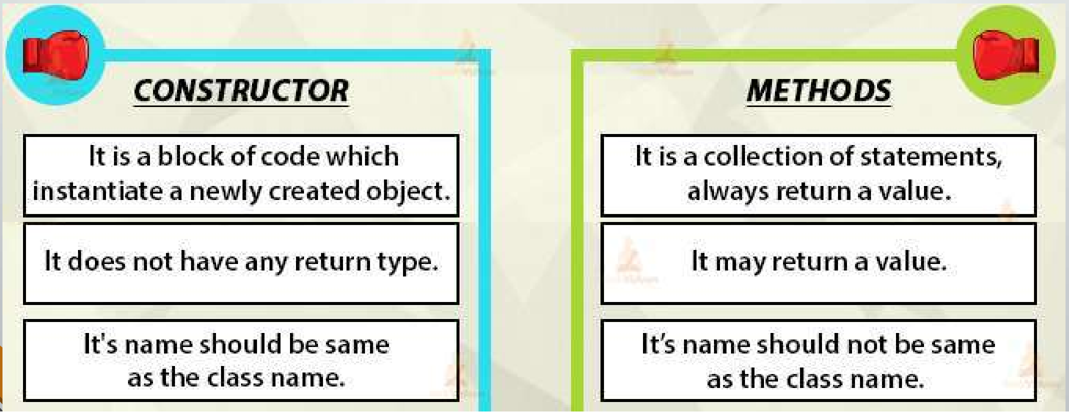
• Constructor Definition

- **<u>Constructor</u>** in Java is a block of code like a method that's called when an **<u>instance of an object is created</u>**. Constructor doesn't return any value even void, the access modifier of constructor is public and its name is the same as class name.

The basic format for coding a constructor:

```
public ClassName (parameter-list) {
Statements
}
```

[redacted]



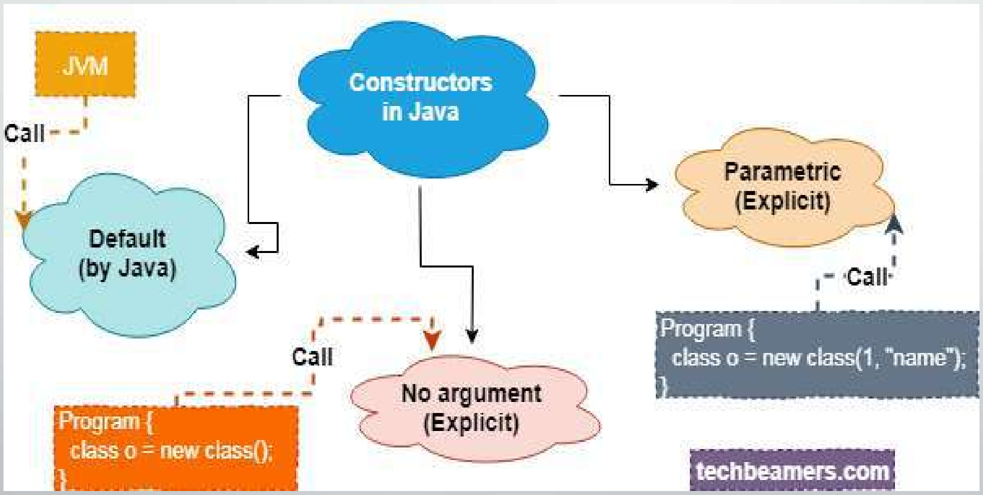
CONSTRUCTOR	METHODS
It is a block of code which instantiate a newly created object.	It is a collection of statements, always return a value.
It does not have any return type.	It may return a value.
It's name should be same as the class name.	It's name should not be same as the class name.

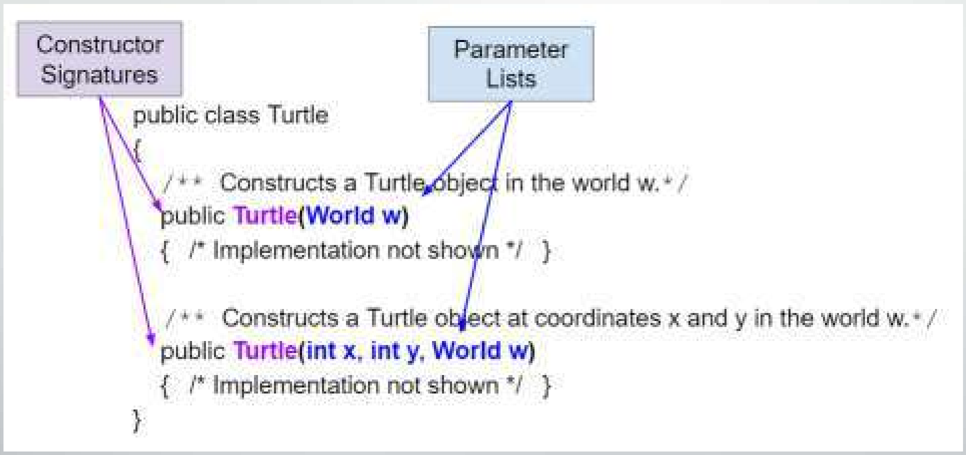
- The difference between constructor and method

Java Constructor	Java Method
A constructor is used only to initialize the state of an object, this means that it allows to provide initial values for class fields when the object is created.	A method is used to expose the behavior of an object and may be for initializing the state of an object
The constructor is invoked implicitly. Which is called automatically when a new instance of an object is created.	The method is invoked explicitly (not automatically by passing a message)
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructors are not considered members of a class.	The methods are members of a class.

[redacted]

• Constructor Types



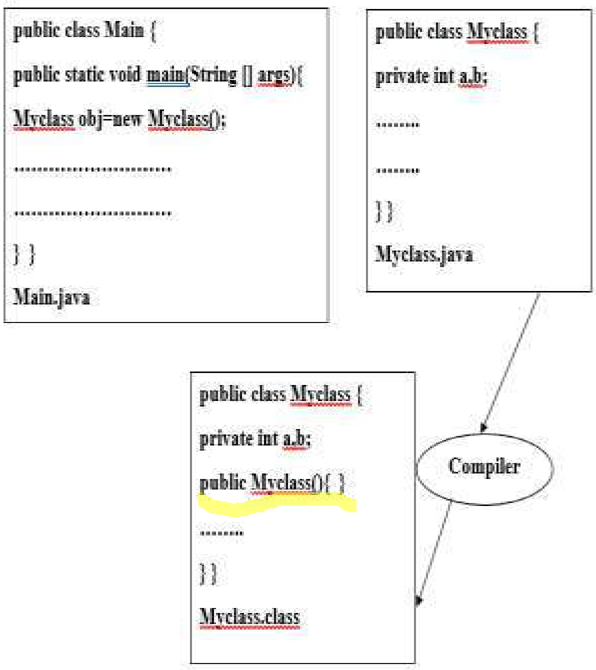


```
public class Turtle
{
   /** Constructs a Turtle object in the world w.*/
   public Turtle(World w)
   {  /* Implementation not shown */  }

   /** Constructs a Turtle object at coordinates x and y in the world w.*/
   public Turtle(int x, int y, World w)
   {  /* Implementation not shown */  }
}
```

[redacted]

1- Default constructor

- In computer programming languages, the term **default constructor** can refer to a **constructor** that is automatically generated by the compiler in the absence of any programmer-defined **constructors** which is usually a **nullary constructor**.
- All Java classes have at least one constructor even if we don't explicitly define one. The compiler automatically provides a public no-argument constructor for any class without constructors.

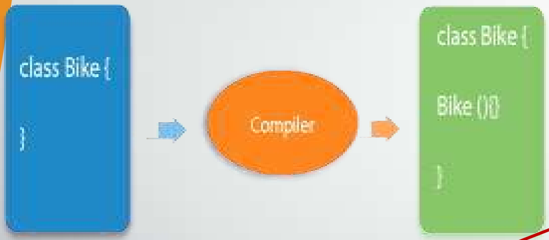


```
public class Main {
public static void main(String [] args){
Myclass obj=new Myclass();
..............................
..............................
} }
Main.java
```

```
public class Myclass {
private int a,b;
........
........
}}
Myclass.java
```

```
public class Myclass {
private int a,b;
public Myclass(){ }
........
}}
Myclass.class
```

Example 1:



```
public class Bike{
 public void print(){
System.out.println("Bike is created");
}}
public class Main{
public static void main(String args[]){
//calling a default constructor
Bike b=new Bike();  b.print();
}  }
```

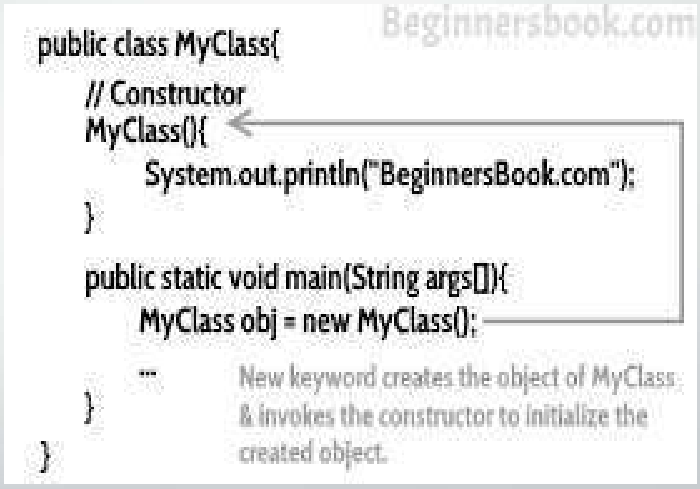
The Output is:
Bike is created

Calling a Constructor: You call a constructor when you create a new instance of the class containing Bike b=new Bike();

[redacted]

2- no-arg constructor

- Constructor with no arguments is known as no-arg constructor. The signature is same as default constructor; however, body can have any code unlike default constructor where the body of the constructor is empty.



```
public class MyClass{
    // Constructor
    MyClass(){
        System.out.println("BeginnersBook.com");
    }

    public static void main(String args[]){
        MyClass obj = new MyClass();
        ...
    }
}
```

New keyword creates the object of MyClass & invokes the constructor to initialize the created object.

Example 2:

```
public class Student{
private int id;
private boolean state;
public Student(){ id=10; state=true;}
 public void display(){
System.out.println(id+"   "+state);}  }
 public class Main{
        public static void main(String args[]){
Student s1=new Student();
Student s2=new Student();
s1.display();   s2.display();   }   }
```

The output is :

```
10   true
10   true
```

Calling a Constructor: You call a constructor when you create a new instance of the class containing

Student s1=new Student();

Student s2=new Student();

[redacted]

- **<u>H.W. what will be the output when there is no constructor in class Student?</u>**

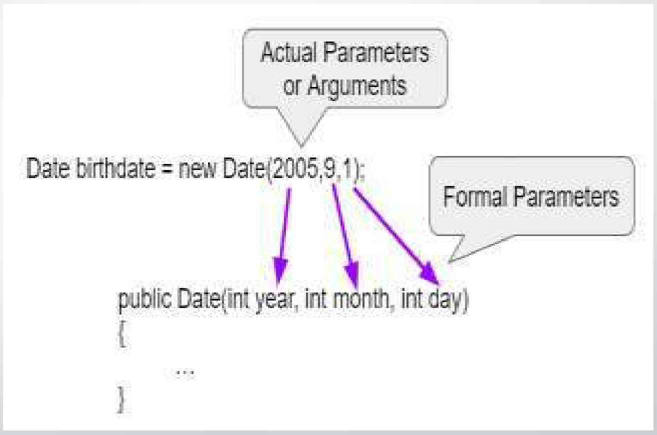
- **Rule: If there is no constructor in a class, compiler automatically creates a default constructor.**

When you are not creating any constructor , the compiler provides you a default constructor.

3- Parameterized Constructor

- Constructor with arguments (or you can say parameters) is known as Parameterized constructor.

The parameterized constructor is used to provide different values to the fields of objects.



```
Date birthdate = new Date(2005,9,1);

public Date(int year, int month, int day)
{
        ...
}
```

[redacted]

Example 3:

```
public class Student{
  private  int id;
   private boolean state;
   public  Student(int i,boolean b){id = i;     state = b; }
   public void display( ){System.out.println(id+"  "+state); }  }

 public class Main{
   public static void main(String args[]){
   Student s1 = new Student(111,true);
   Student s2 = new Student(222,false);
   s1.display();      s2.display();     }  }
```

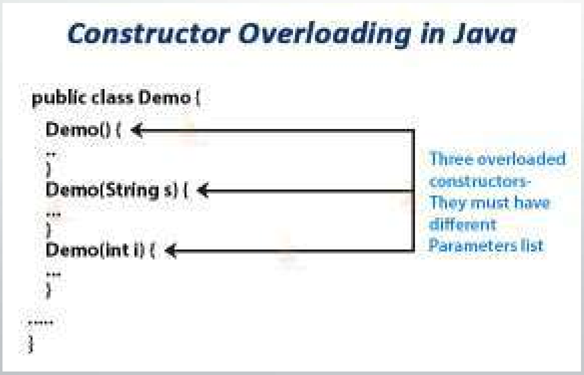
The Output is:
```
111 true
222 false
```

Calling a Constructor: You call a constructor when you create a new instance of the class containing

Student s1 = new Student(111,true);
Student s2 = new Student(222,false);

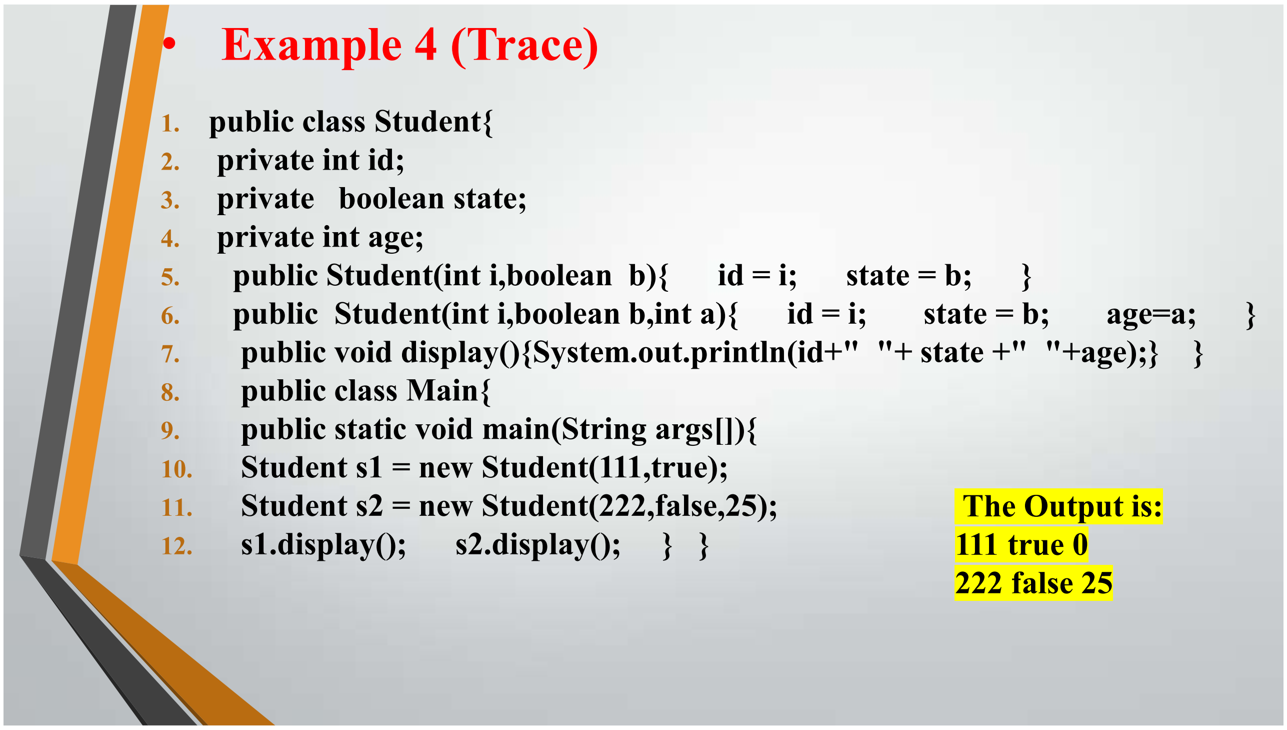
Constructor Overloading

As like of method overloading, Constructors also can be overloaded. The same constructor declared with different parameters in the same class is known as constructor overloading. Compiler differentiates which constructor is to be called depending upon the number of parameters and their sequence of data types


Constructor Overloading in Java

```
public class Demo {
  Demo() {
  ..
  }
  Demo(String s) {
  ...
  }
  Demo(int i) {
  ...
  }
.....
}
```

Three overloaded constructors-
They must have different Parameters list

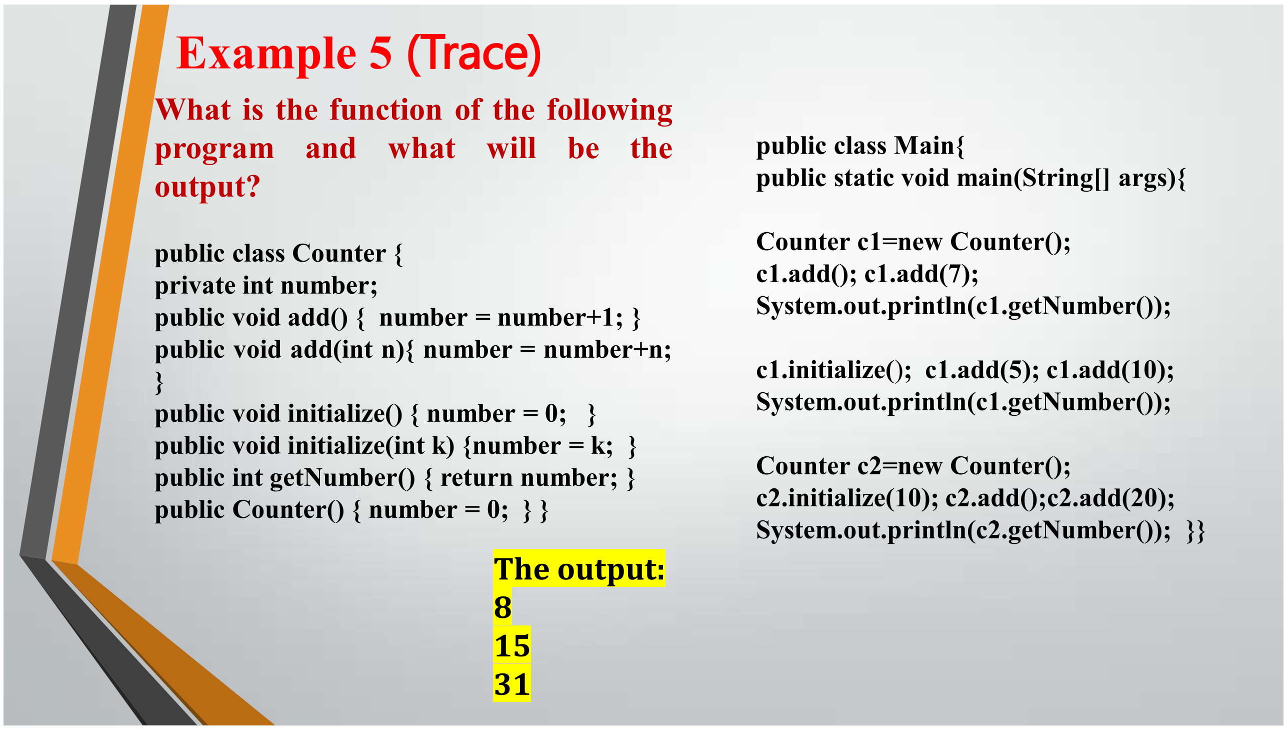


Example 4 (Trace)

```
public class Student{
 private int id;
 private   boolean state;
 private int age;
   public Student(int i,boolean  b){      id = i;      state = b;      }
   public  Student(int i,boolean b,int a){      id = i;       state = b;       age=a;      }
    public void display(){System.out.println(id+"  "+ state +"  "+age);}    }
    public class Main{
    public static void main(String args[]){
    Student s1 = new Student(111,true);
    Student s2 = new Student(222,false,25);
    s1.display();      s2.display();     }   }
```

The Output is:

```
111 true 0
222 false 25
```



Example 5 (Trace)

What is the function of the following program and what will be the output?

```
public class Counter {
private int number;
public void add() {  number = number+1; }
public void add(int n){ number = number+n;
}
public void initialize() { number = 0;   }
public void initialize(int k) {number = k;  }
public int getNumber() { return number; }
public Counter() { number = 0;  } }
```

```
public class Main{
public static void main(String[] args){

Counter c1=new Counter();
c1.add(); c1.add(7);
System.out.println(c1.getNumber());

c1.initialize();  c1.add(5); c1.add(10);
System.out.println(c1.getNumber());

Counter c2=new Counter();
c2.initialize(10); c2.add();c2.add(20);
System.out.println(c2.getNumber());  }}
```

The output:

```
8
15
31
```

H.W. Lab

Complete the following program to create three counters each of which uses a different constructor.

```
public class Counter {

private int number, reused;

public void add() { number = number+1; }

public void initialize() { number = 0;  reused = reused+1; }

public int getNumber() { return number; }

public int getReused() { return reused; }

public Counter() { number = 0; reused = 0; }

Counter(int x) { number = x; reused = 0; }

Counter(int x, int y) { number = x; reused = y; }

Counter(float z) { number = (int) z; reused = 0; }
}
```

- **Advanced Examples**
  - Taking advanced examples

[redacted]

• Example1

Define a class called Rectangle that has the following members:
Two integer attributes (variable members, attribute members) and two methods (method members) , area which returns the rectangle area while set_data is used to set the width and height.
Use this class to print:

1. one rectangle area
2. two rectangle areas using two objects
3. three rectangle areas using one object
4. rectangles areas' (use array)

Rectangle
private int w,h
void set_data(int , int) **int area( )**

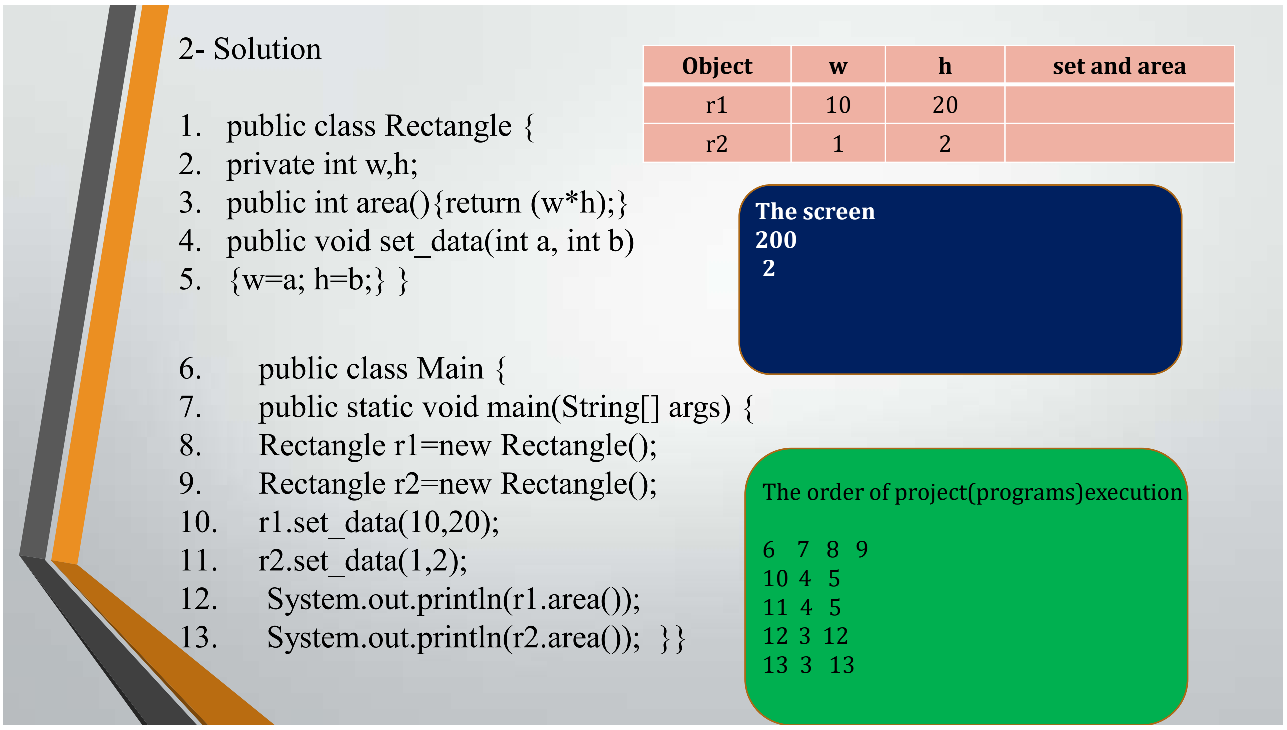
1- Solution:

Object	w	h	set and area
r1	10	20	

```
public class Rectangle {
private int w,h;
public int area(){return (w*h);}
public void set_data(int a, int b){w=a;
h=b;}
}

public class Main {
   public static void main(String[] args) {
      Rectangle r1=new Rectangle();
       r1.set_data(10,20);
      System.out.println(r1.area());  }}
```

```
200
```

[redacted]



2- Solution

Object	w	h	set and area
r1	10	20	
r2	1	2	

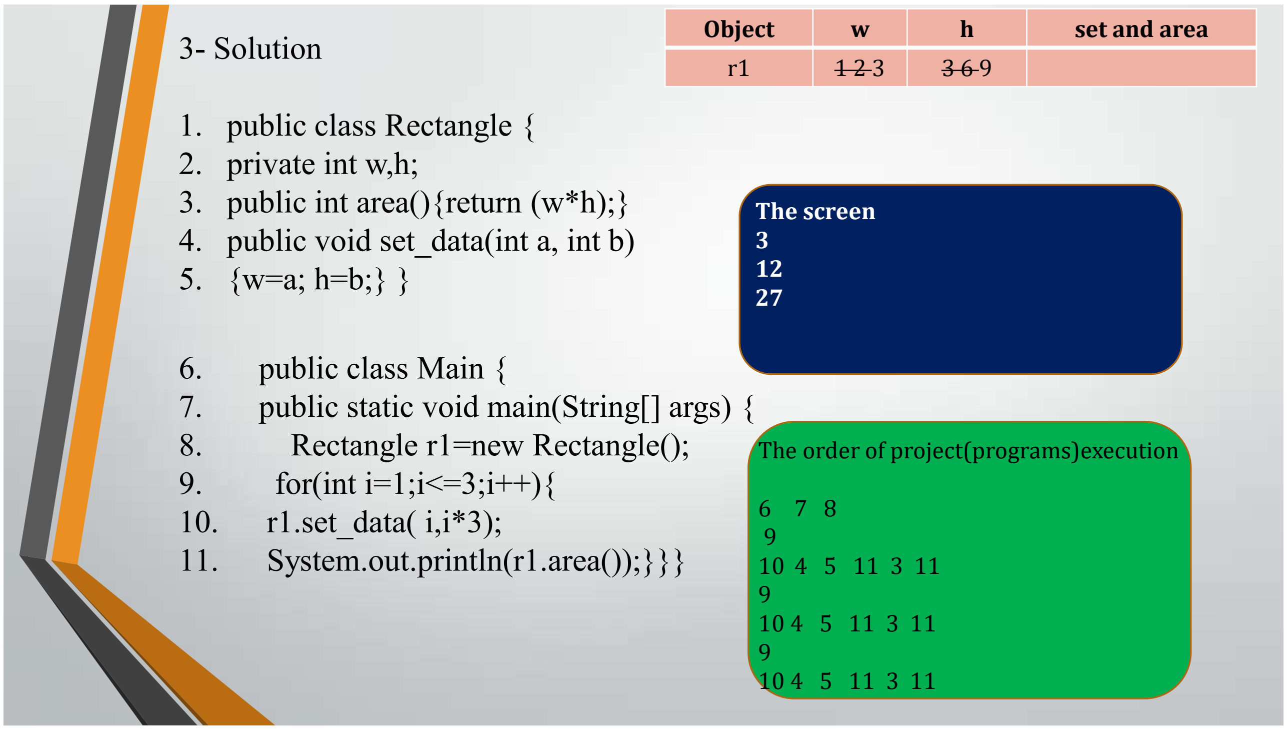
```
1.   public class Rectangle {
2.   private int w,h;
3.   public int area(){return (w*h);}
4.   public void set_data(int a, int b)
5.   {w=a; h=b;} }

6.       public class Main {
7.       public static void main(String[] args) {
8.       Rectangle r1=new Rectangle();
9.       Rectangle r2=new Rectangle();
10.      r1.set_data(10,20);
11.      r2.set_data(1,2);
12.       System.out.println(r1.area());
13.       System.out.println(r2.area());  }}
```

The screen

```
200
 2
```

The order of project(programs)execution

```
6  7  8  9
10  4  5
11  4  5
12  3  12
13  3  13
```



3- Solution

Object	w	h	set and area
r1	~~1 2~~ 3	~~3 6~~ 9	

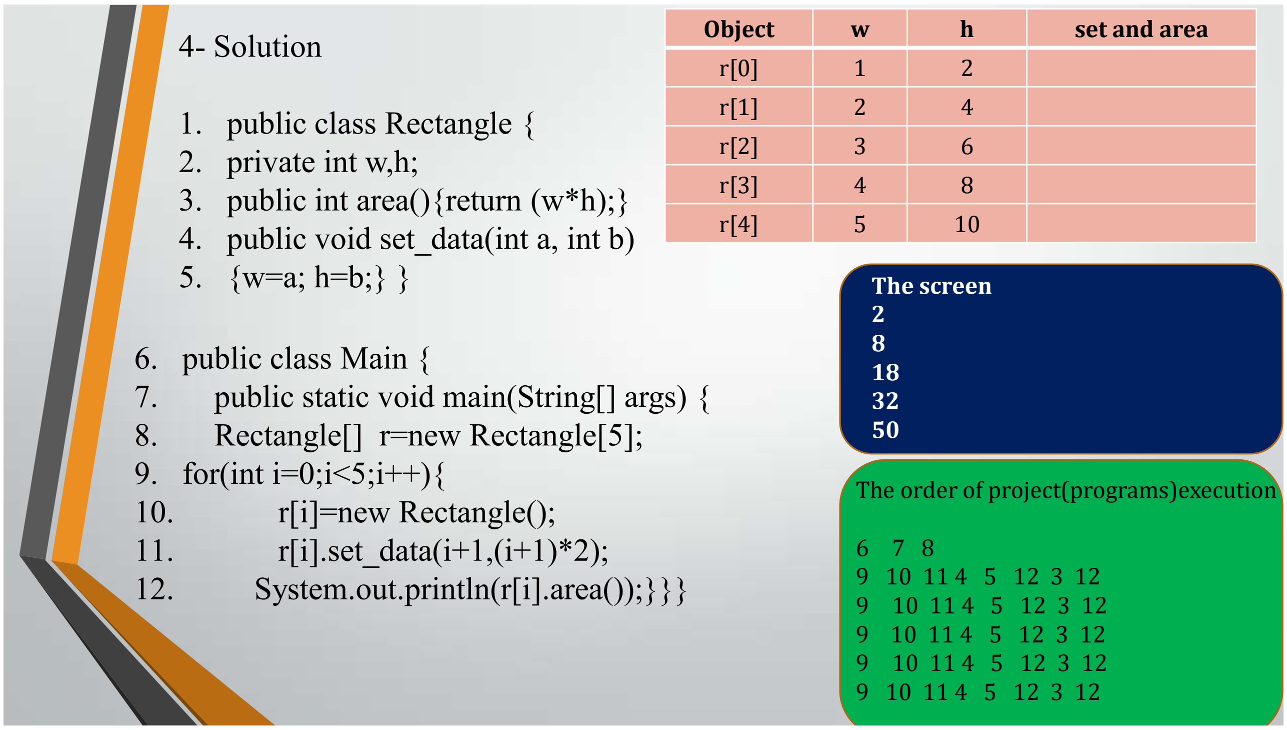
```
1.   public class Rectangle {
2.   private int w,h;
3.   public int area(){return (w*h);}
4.   public void set_data(int a, int b)
5.   {w=a; h=b;} }

6.       public class Main {
7.       public static void main(String[] args) {
8.           Rectangle r1=new Rectangle();
9.         for(int i=1;i<=3;i++){
10.       r1.set_data( i,i*3);
11.       System.out.println(r1.area());}}}
```

The screen

```
3
12
27
```

The order of project(programs)execution

```
6   7   8
 9
10  4   5   11  3  11
9
10 4   5   11  3  11
9
10 4   5   11  3  11
```


4- Solution

```
1.  public class Rectangle {
2.  private int w,h;
3.  public int area(){return (w*h);}
4.  public void set_data(int a, int b)
5.  {w=a; h=b;} }

6.  public class Main {
7.      public static void main(String[] args) {
8.      Rectangle[]  r=new Rectangle[5];
9.  for(int i=0;i<5;i++){
10.             r[i]=new Rectangle();
11.             r[i].set_data(i+1,(i+1)*2);
12.          System.out.println(r[i].area());}}}
```

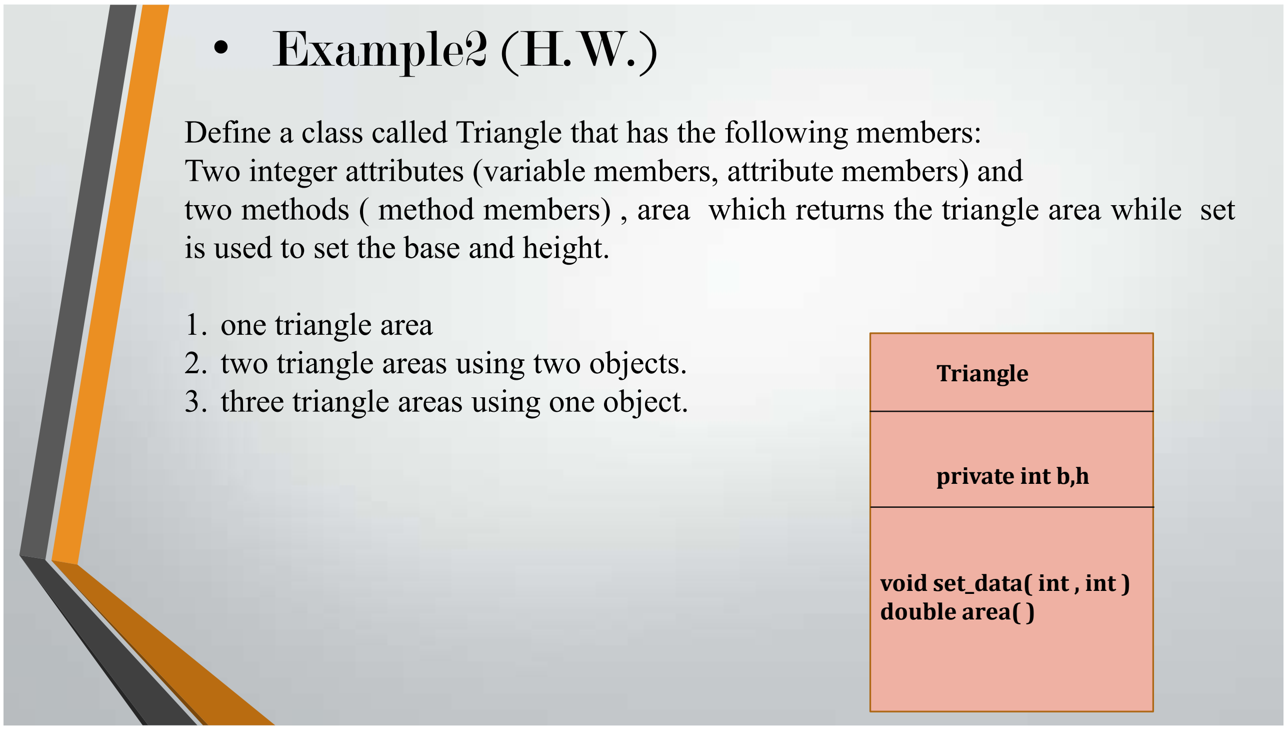
Object	w	h	set and area
r[0]	1	2	
r[1]	2	4	
r[2]	3	6	
r[3]	4	8	
r[4]	5	10	

The screen

```
2
8
18
32
50
```

The order of project(programs)execution

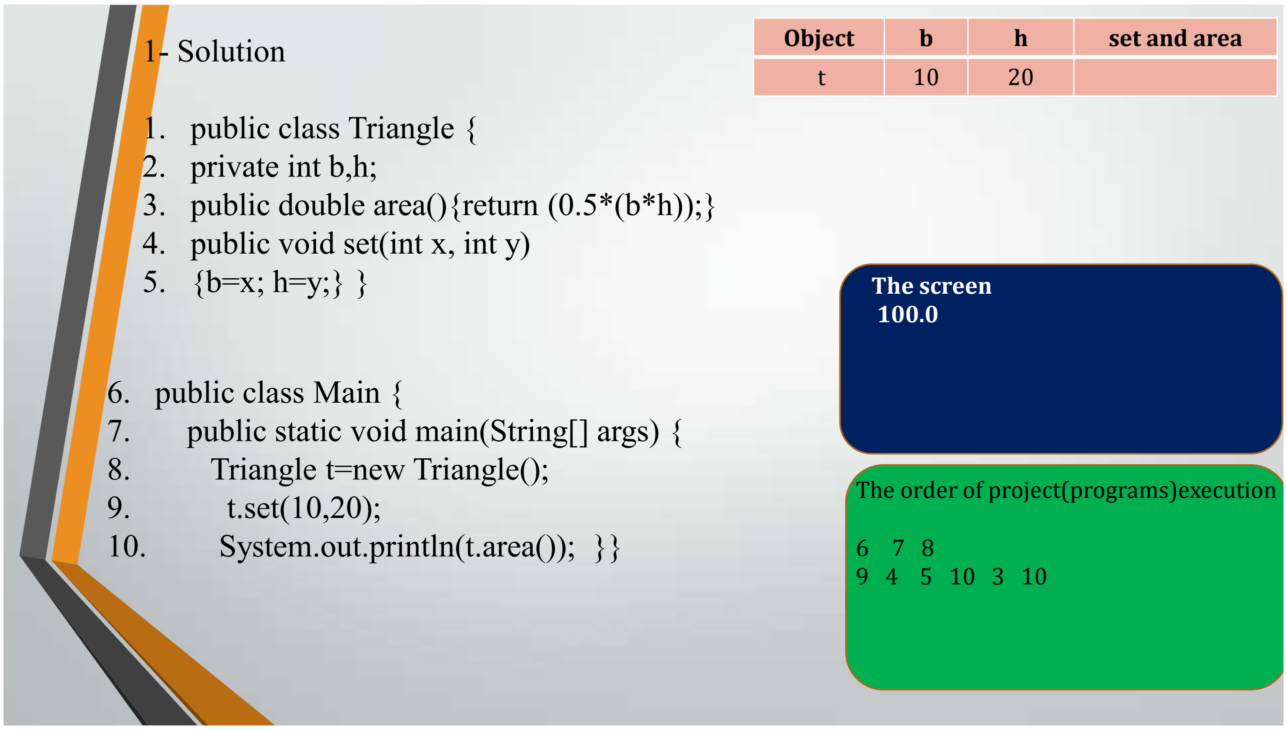
```
6   7  8
9  10  11 4  5  12  3  12
9   10  11 4  5  12  3  12
9   10  11 4  5  12  3  12
9   10  11 4  5  12  3  12
9  10  11 4  5  12  3  12
```


• Example2 (H.W.)

Define a class called Triangle that has the following members:
Two integer attributes (variable members, attribute members) and
two methods (method members) , area which returns the triangle area while set is used to set the base and height.

1. one triangle area
2. two triangle areas using two objects.
3. three triangle areas using one object.

Triangle
private int b,h
void set_data(int , int) **double area( )**



1- Solution

```
1.   public class Triangle {
2.   private int b,h;
3.   public double area(){return (0.5*(b*h));}
4.   public void set(int x, int y)
5.   {b=x; h=y;} }

6.   public class Main {
7.       public static void main(String[] args) {
8.          Triangle t=new Triangle();
9.            t.set(10,20);
10.          System.out.println(t.area());  }}
```

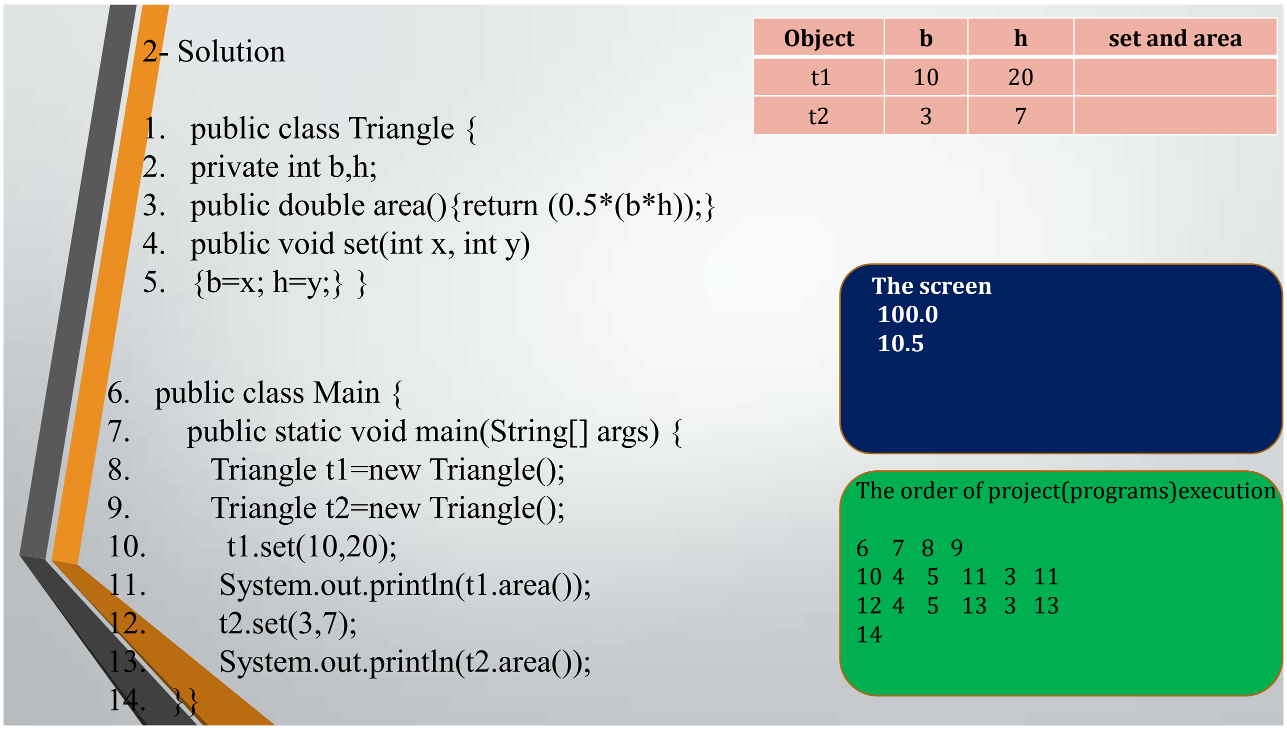
Object	b	h	set and area
t	10	20	

The screen

```
100.0
```

The order of project(programs)execution

```
6   7  8
9  4   5  10  3  10
```



2- Solution

```
1.   public class Triangle {
2.   private int b,h;
3.   public double area(){return (0.5*(b*h));}
4.   public void set(int x, int y)
5.   {b=x; h=y;} }

6.   public class Main {
7.       public static void main(String[] args) {
8.          Triangle t1=new Triangle();
9.          Triangle t2=new Triangle();
10.           t1.set(10,20);
11.          System.out.println(t1.area());
12.          t2.set(3,7);
13.          System.out.println(t2.area());
14.    }}
```

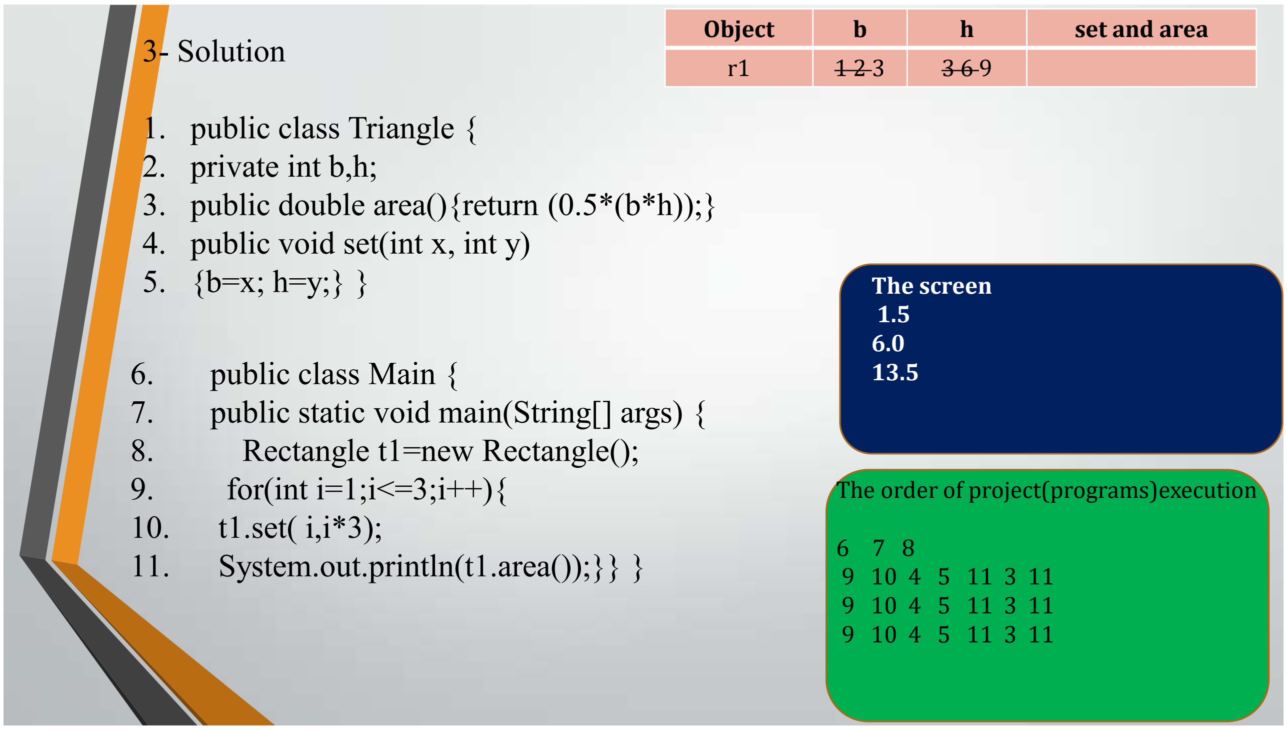
Object	b	h	set and area
t1	10	20	
t2	3	7	

The screen

```
100.0
10.5
```

The order of project(programs)execution

```
6   7  8  9
10  4   5   11  3  11
12  4   5   13  3  13
14
```



3- Solution

Object	b	h	set and area
r1	~~1 2~~ 3	~~3 6~~ 9	

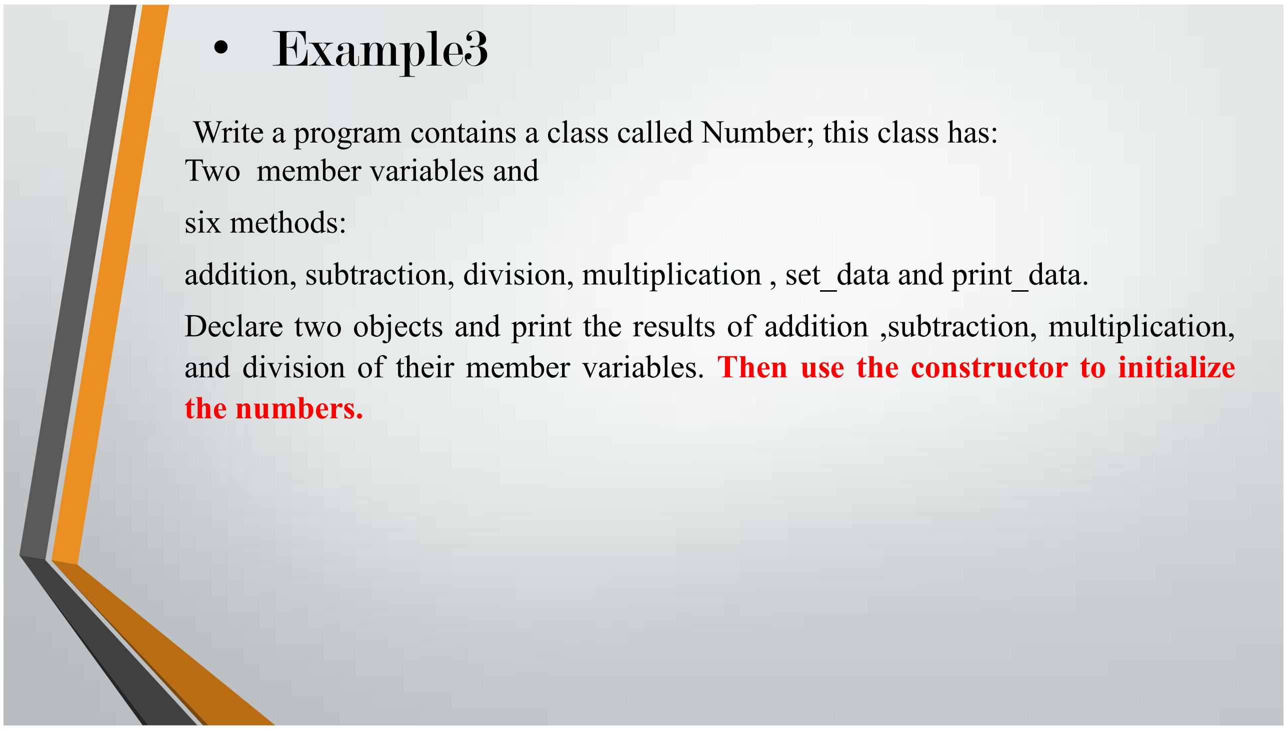
```
1.   public class Triangle {
2.   private int b,h;
3.   public double area(){return (0.5*(b*h));}
4.   public void set(int x, int y)
5.   {b=x; h=y;} }


6.      public class Main {
7.      public static void main(String[] args) {
8.          Rectangle t1=new Rectangle();
9.        for(int i=1;i<=3;i++){
10.      t1.set( i,i*3);
11.      System.out.println(t1.area());}} }
```

The screen

```
 1.5
6.0
13.5
```

The order of project(programs)execution

```
6   7  8
 9  10  4  5  11  3  11
 9  10  4  5  11  3  11
 9  10  4  5  11  3  11
```



- Example3

Write a program contains a class called Number; this class has:
Two member variables and

six methods:

addition, subtraction, division, multiplication , set_data and print_data.

Declare two objects and print the results of addition ,subtraction, multiplication, and division of their member variables. **Then use the constructor to initialize the numbers.**

[redacted]

Solution

```
1.  public class Number {
2.  private int x,y;
3.  public int add(){return (x+y);}
4.  public int sub(){return (x-y);}
5.  public int mul(){return (x*y);}
6.  public int dv(){return (x/y);}
7.  public void set_data(int a, int b){x=a; y=b; }
8.  public void print_data(){
9.  System.out.println(“x= “+x);
10. System.out.println(“y= “+y); }}
```

```
11. public class Main {
12.     public static void main(String[] args) {
13.  Number n=new Number();
14. n.set_data(100,20);
15. n.print_data();
16. System.out.println(“Add”+n.add());
17. System.out.println(“Sub”+n.sub());
18. System.out.println(“Mul”+n.mul());
19. System.out.println(“Div”+n.dv());}}
```

Object	x	y	set,add,....print
n	100	20	

The screen

```
 x=100
Y=20
Add 120
Sub 80
Mul 2000
Div 5
```

The order of project(programs)execution
11 12 13 14 7 15 8 9 10
16 3 16
17 4 17
18 5 18
19 6 19

```
public class Number {
private int x,y;

public Number(int n1,int n2){  x=n1; y=n2; }

public  int add(){ return(x+y); }

public  int sub(){return (x-y);}

public int mul(){ return x*y;}

public int div(){ return x/y;}

public void print(){
System.out.println("x="+x+" y="+y);}
 }
```

```
public class Main {

    public static void main(String[] args) {

        Number n;
        n=new Number(50,100);
        n.print();
        System.out.println(n.add());
        System.out.println(n.sub());
        System.out.println(n.mul());
        System.out.println(n.div());

} }
```

H.W.
Update the above program to print the summation of 5 integer numbers which are between -100 and 100.

The output will be:

??????H.W.??????

[redacted]

- Example4

Define a class which describes any item in supermarket. This class contains
two private variables (price and number) and
two public functions:
print_data() which used to print the data stored in private variables and
set_data() is used to store data in private variables.

1- Define two items in your supermarket and print the prices and number of them.
2- Define 100 items in your supermarket and print the prices and number of them.
3- Define two items in your supermarket and print the summation of their prices. (you need to add get_data method to return the price).

1- Solution

```
1.  public class Item {
2.  private int number;
3.  private double price;
4.  public void set_data(int n, double p){
5.  number=n;price=p; }
6.  public void print_data(){
7.  System.out.println("Number= "+number);
8.  System.out.println("Price= "+price+"$"); }}
```

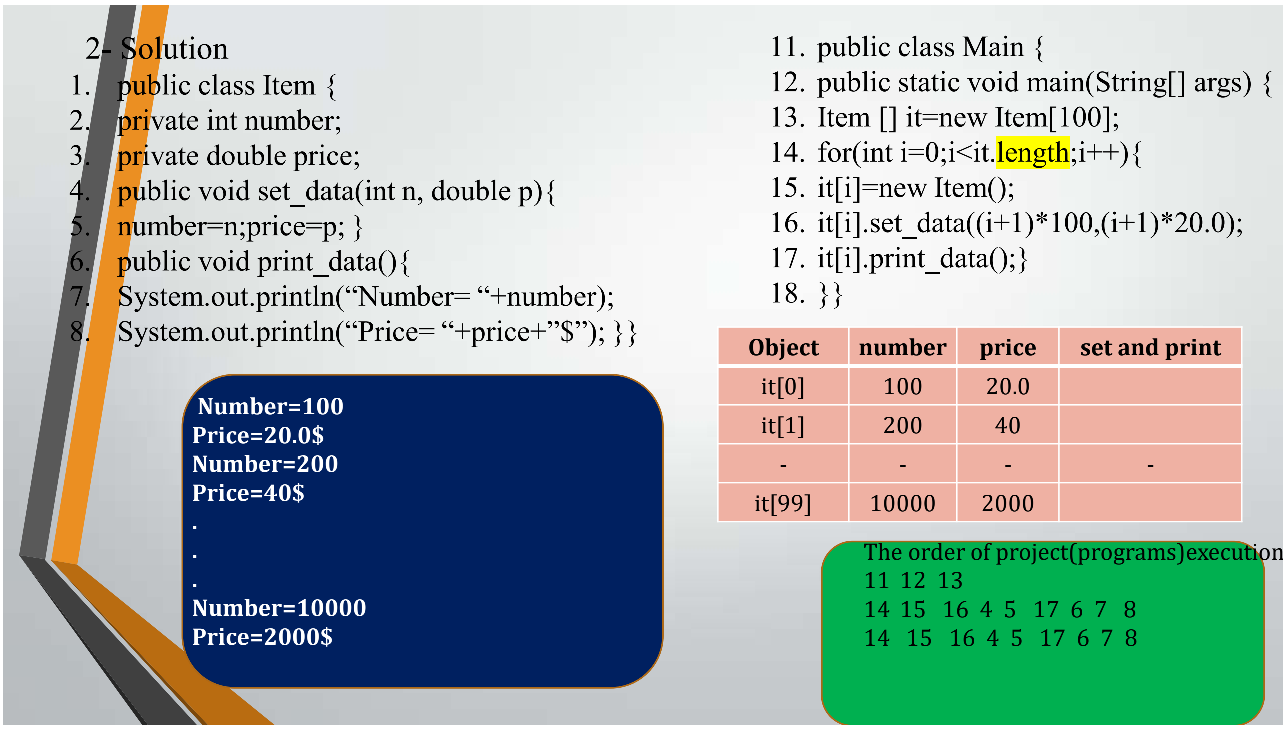
```
11. public class Main {
12. public static void main(String[] args) {
13. Item it1=new Item();
14. Item it2=new Item();
15. it1.set_data(100,20.0);
16. it2.set_data(200,123.5);
17. it1.print_data();
18. it2.print_data();
19. }}
```

Object	number	price	set and print
It1	100	20.0	
it2	200	123.5	

```
Number=100
Price=20.0$
Number=200
Price=123.5$
```

The order of project(programs)execution
11 12 13 14 15 4 5 16 4 5
17 6 7 8 18 6 7 8 19

[redacted]



2- Solution

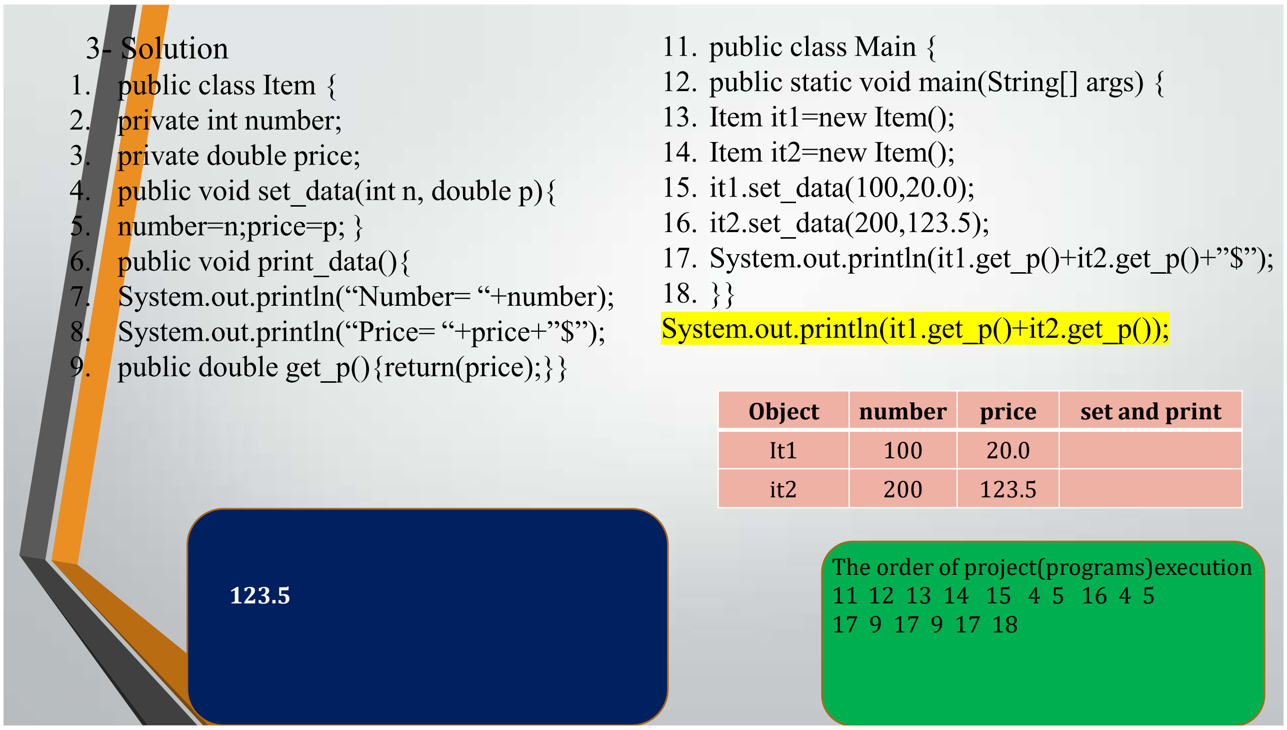
```
1. public class Item {
2. private int number;
3. private double price;
4. public void set_data(int n, double p){
5. number=n;price=p; }
6. public void print_data(){
7. System.out.println("Number= "+number);
8. System.out.println("Price= "+price+"$"); }}
```

```
11. public class Main {
12. public static void main(String[] args) {
13. Item [] it=new Item[100];
14. for(int i=0;i<it.length;i++){
15. it[i]=new Item();
16. it[i].set_data((i+1)*100,(i+1)*20.0);
17. it[i].print_data();}
18. }}
```

Object	number	price	set and print
it[0]	100	20.0	
it[1]	200	40	
-	-	-	-
it[99]	10000	2000	

```
Number=100
Price=20.0$
Number=200
Price=40$
.
.
.
Number=10000
Price=2000$
```

The order of project(programs)execution
11 12 13
14 15 16 4 5 17 6 7 8
14 15 16 4 5 17 6 7 8



3- Solution

```
1. public class Item {
2. private int number;
3. private double price;
4. public void set_data(int n, double p){
5. number=n;price=p; }
6. public void print_data(){
7. System.out.println("Number= "+number);
8. System.out.println("Price= "+price+"$");
9. public double get_p(){return(price);}}
```

```
11. public class Main {
12. public static void main(String[] args) {
13. Item it1=new Item();
14. Item it2=new Item();
15. it1.set_data(100,20.0);
16. it2.set_data(200,123.5);
17. System.out.println(it1.get_p()+it2.get_p()+"$");
18. }}
System.out.println(it1.get_p()+it2.get_p());
```

Object	number	price	set and print
It1	100	20.0	
it2	200	123.5	

```
123.5
```

The order of project(programs)execution
11 12 13 14 15 4 5 16 4 5
17 9 17 9 17 18

[redacted]

Example5 (Employee class)

Create a class called Employee that includes three pieces of information as instance variables:
an Employee number (type integer),
an Employee step (type integer) and
a monthly salary (double).
Your class should have a constructor that initializes the three instance variables.
Provide set and get methods for each instance variable. If the monthly salary is not positive, set it to 0.0.
Write a test application named EmployeeTest that demonstrates class Employee's capabilities. Create two Employee objects and display each object's yearly salary. Then give each Employee a 10% raise (زيادة) and display each Employee's yearly salary again.

```
public class Employee {

private int number;
private int step;
private double salary;

public Employee(int no,int st, double sal) {
number=no;
step=st;
if (sal > 0.0)     salary=sal;
else               salary=0.0; }

public void setNo(int no){ number=no; }
public void setSt(int st){ step=st; }
public void setSalary(double sal){
if (sal > 0.0)   salary = sal;
else  salary = 0.0; }

public int getNo(){   return (number);    }
public int getSt(){  return (step); }
public double getSalary(){ return salary; }
}
```

[redacted]

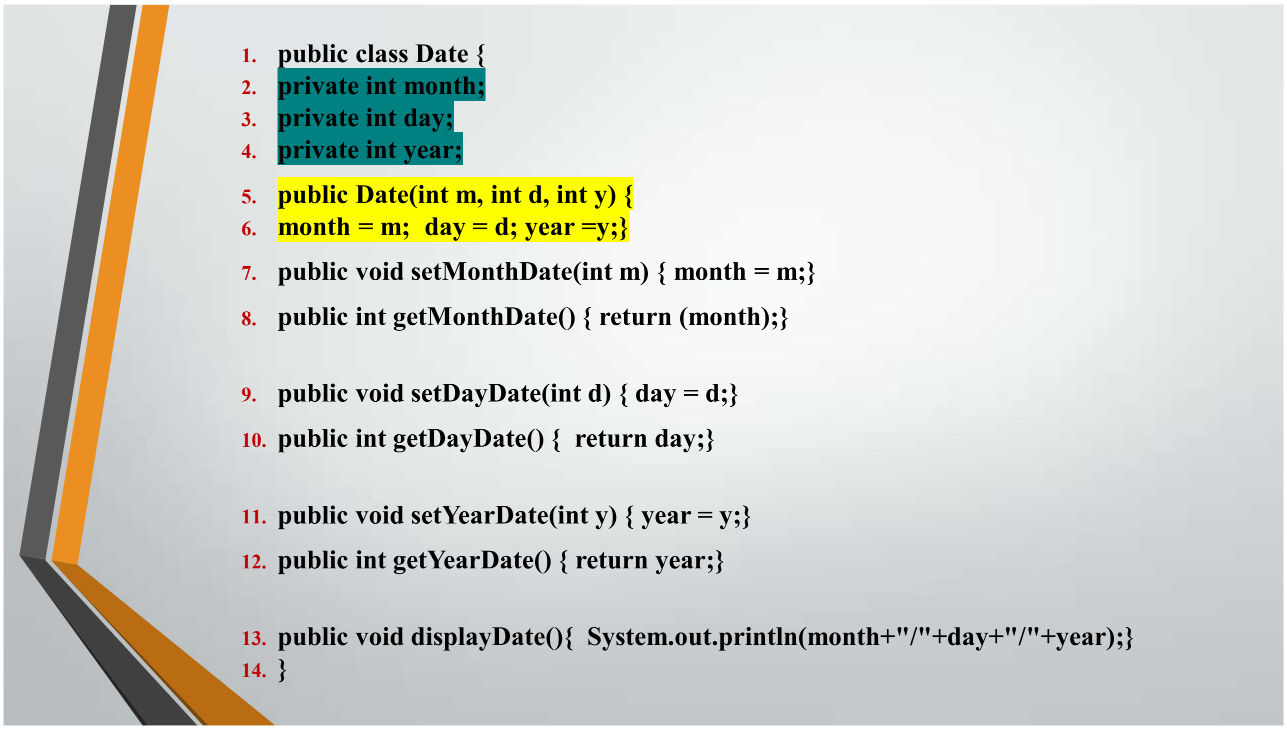
```
public class EmployeeTest {
public static void main (String args[]){

Employee e1=new Employee (1,3,20000.00);
Employee e2=new Employee (2,5,50000.00);

System.out.println("1) "+e1.getNo()+“ "+e1.getSt()+“ "+e1.getSalary()*12);
System.out.println("2) "+e2.getNo()+e2.getSt()+e2.getSalary()*12);

e1.setSalary(0.1*e1.getSalary()+e1.getSalary());
e2.setSalary(0.1*e2.getSalary()+e2.getSalary());

System.out.println("3) "+e1.getNo()+”  “ +e1.getSt()+” “+e1.getSalary()*12);
System.out.println("4) "+e2.getNo()+” “+ e2.getSt()+” “+e2.getSalary()*12);
                                         }
                              }
```

The output is the following:

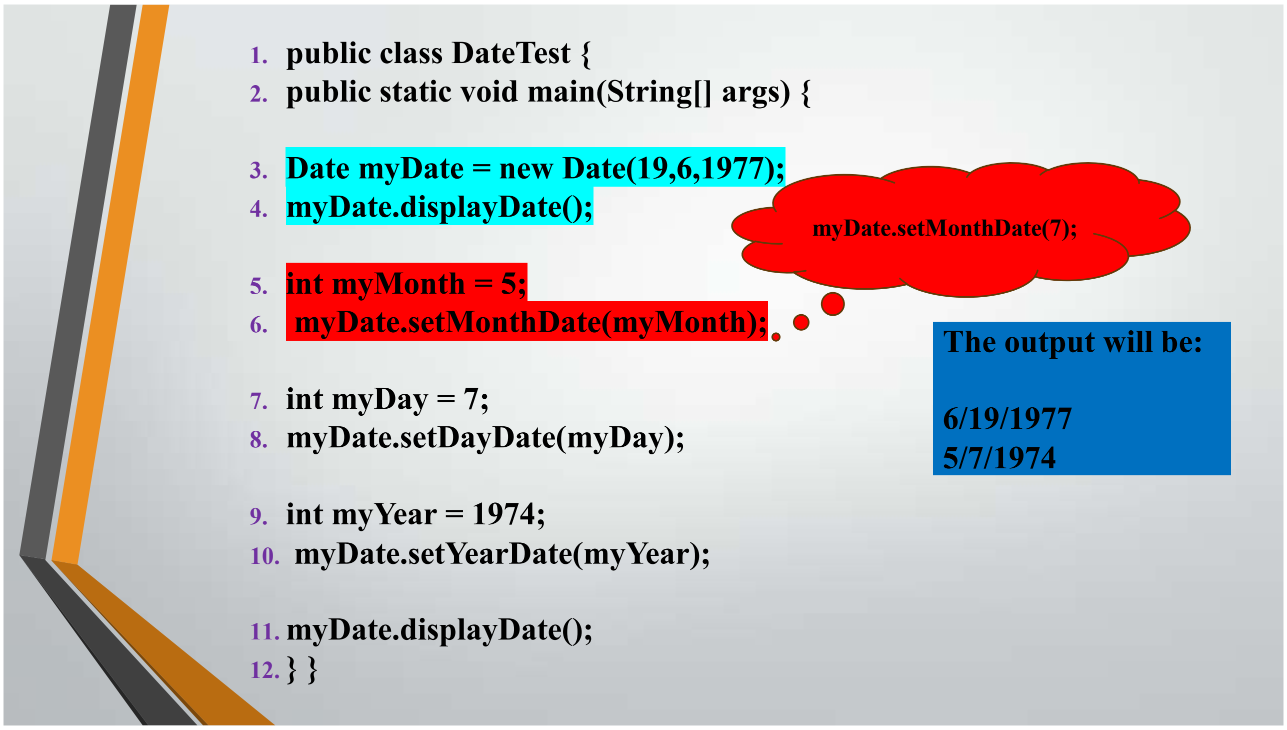
```
1) 1 3 240000.0
2) 2 5 600000.0
3) 1 3 264000.0
4) 2 5 660000.0
```

- **Example6 (Date class)**

Create a class called Date that includes three pieces of information as instance variables:
A month (type int), a day (type int) and a year (type int).
Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct.
Provide a set and a get method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/).
Write a test application named DateTest that creates a date object and print the full date.
Change the date to another one and print it again.



```
public class Date {
private int month;
private int day;
private int year;
public Date(int m, int d, int y) {
month = m;  day = d; year =y;}
public void setMonthDate(int m) { month = m;}
public int getMonthDate() { return (month);}

public void setDayDate(int d) { day = d;}
public int getDayDate() {  return day;}

public void setYearDate(int y) { year = y;}
public int getYearDate() { return year;}

public void displayDate(){  System.out.println(month+"/"+day+"/"+year);}
}
```



```
public class DateTest {
public static void main(String[] args) {

Date myDate = new Date(19,6,1977);
myDate.displayDate();

int myMonth = 5;
 myDate.setMonthDate(myMonth);

int myDay = 7;
myDate.setDayDate(myDay);

int myYear = 1974;
 myDate.setYearDate(myYear);

myDate.displayDate();
} }
```

myDate.setMonthDate(7);

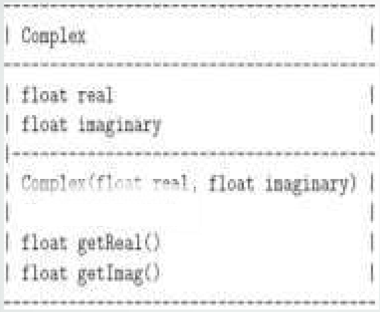
The output will be:

```
6/19/1977
5/7/1974
```

[redacted]

Example7 (Implementing a complex number operations)

Following is an example class diagram for what is expected to be a representation of Complex numbers



```
----------------------------------------
| Complex                              |
----------------------------------------
| float real                           |
| float imaginary                      |
|---------------------------------------
| Complex(float real, float imaginary) |
|                                      |
| float getReal()                      |
| float getImag()                      |
----------------------------------------
```

Use the above class for :

1-adding two complex number **$(a + b\,i) + (c + d\,i) = (a + c) + (b + d)\,i$**

2-subtracting two complex number $(a + b\,i) - (c + d\,i) = (a - b) + (b - d)\,i$

3-multiplying two complex number $(a + b\,i)(c + d\,i) = (a\,c - b\,d) + (a\,d + bc)\,i$

4-dividing two complex number

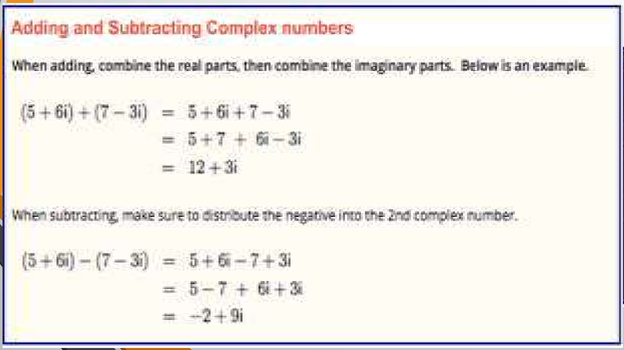
- To add or subtract two **complex numbers**, just add or subtract the corresponding real and **imaginary** parts.
- For instance, the sum of

  5 + 3i

  4 + 2i

  is 9 + 5i.

- For another, the sum of 3 + i and –1 + 2i is 2 + 3i.



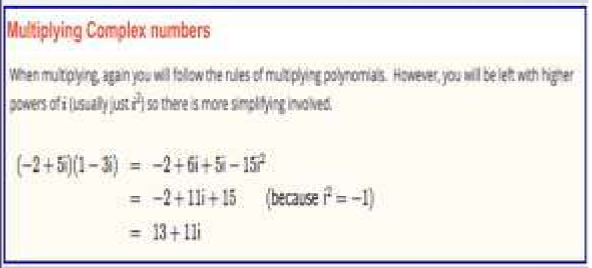
Adding and Subtracting Complex numbers

When adding, combine the real parts, then combine the imaginary parts. Below is an example.

$$\begin{aligned}(5+6i)+(7-3i) &= 5+6i+7-3i\\ &= 5+7+6i-3i\\ &= 12+3i\end{aligned}$$

When subtracting, make sure to distribute the negative into the 2nd complex number.

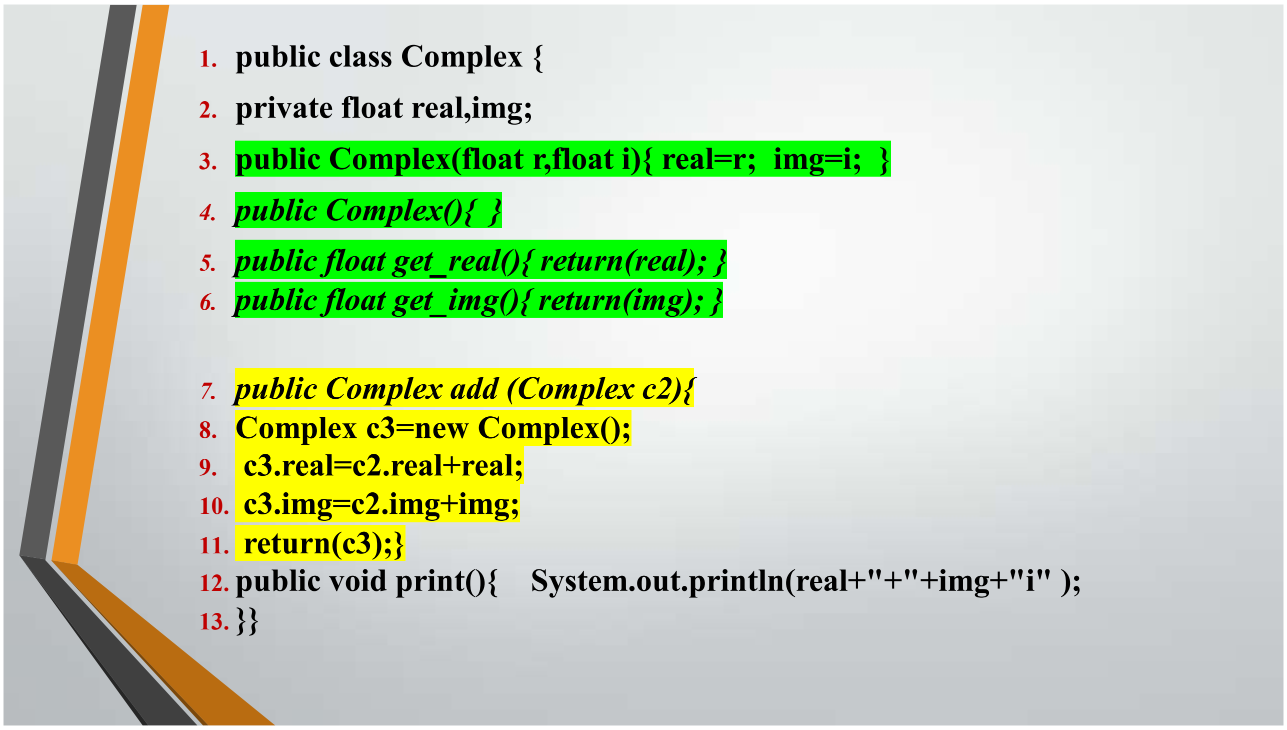
$$\begin{aligned}(5+6i)-(7-3i) &= 5+6i-7+3i\\ &= 5-7+6i+3i\\ &= -2+9i\end{aligned}$$



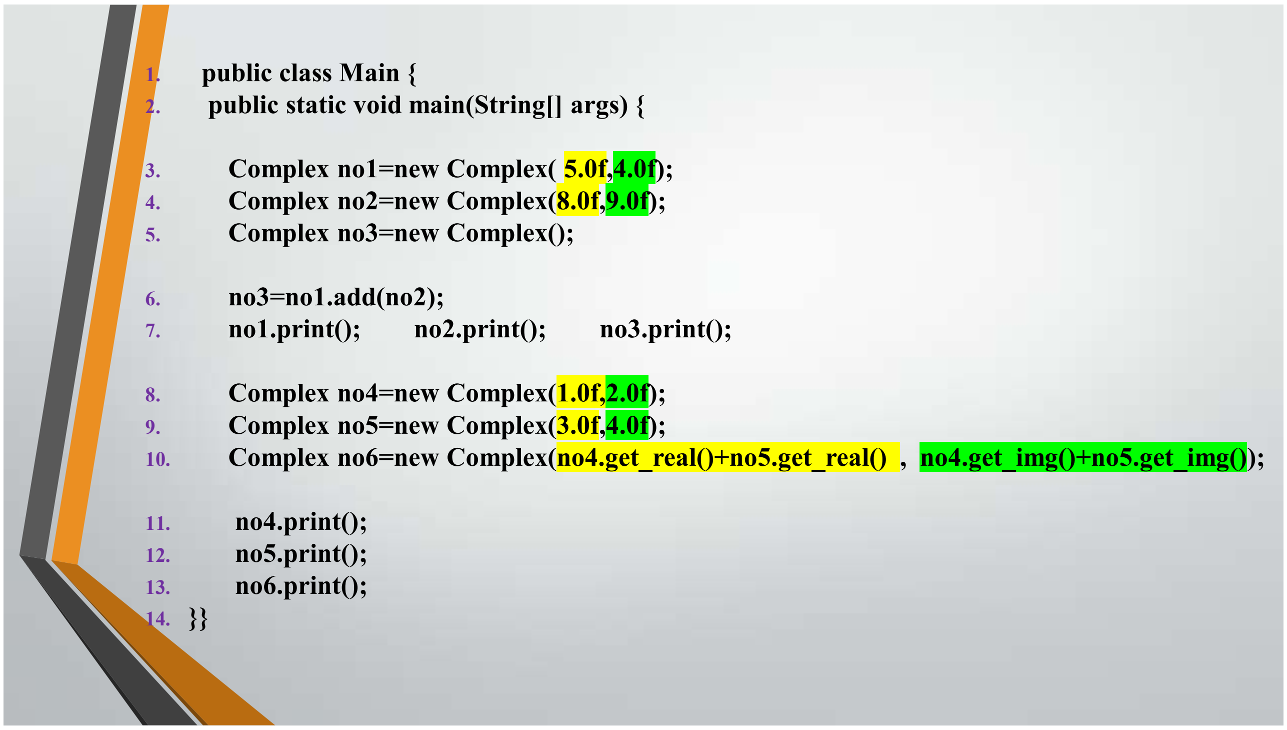
Multiplying Complex numbers

When multiplying, again you will follow the rules of multiplying polynomials. However, you will be left with higher powers of i (usually just i^2) so there is more simplifying involved.

$$\begin{aligned}(-2+5i)(1-3i) &= -2+6i+5i-15i^2\\ &= -2+11i+15 \quad (\text{because } i^2=-1)\\ &= 13+11i\end{aligned}$$



```
public class Complex {
private float real,img;
public Complex(float r,float i){ real=r;  img=i;  }
public Complex(){  }
public float get_real(){ return(real); }
public float get_img(){ return(img); }

public Complex add (Complex c2){
Complex c3=new Complex();
 c3.real=c2.real+real;
 c3.img=c2.img+img;
 return(c3);}
public void print(){    System.out.println(real+"+"+img+"i" );
}}
```



```
public class Main {
 public static void main(String[] args) {

    Complex no1=new Complex( 5.0f,4.0f);
    Complex no2=new Complex(8.0f,9.0f);
    Complex no3=new Complex();

    no3=no1.add(no2);
    no1.print();      no2.print();      no3.print();

    Complex no4=new Complex(1.0f,2.0f);
    Complex no5=new Complex(3.0f,4.0f);
    Complex no6=new Complex(no4.get_real()+no5.get_real() , no4.get_img()+no5.get_img());

     no4.print();
     no5.print();
     no6.print();
}}
```

[redacted]

1. **The results add operation**

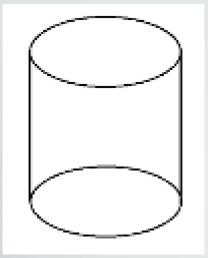
5.0+4.0i
8.0+9.0i
13.0+13.0i

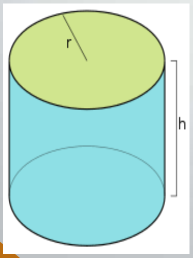
2. **The results without add operation**

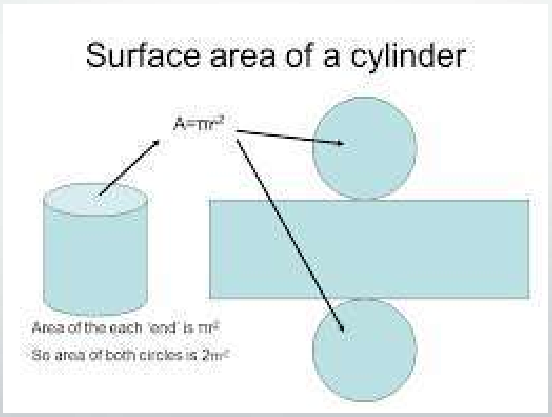
1.0+2.0i
3.0+4.0i
4.0+6.0i

5.0+4.0i
8.0+9.0i
13.0+13.0i
1.0+2.0i
3.0+4.0i
4.0+6.0i

Does circle is a part of cylinder?



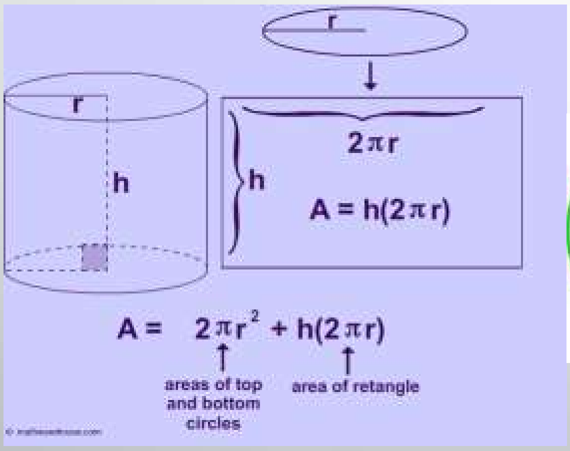


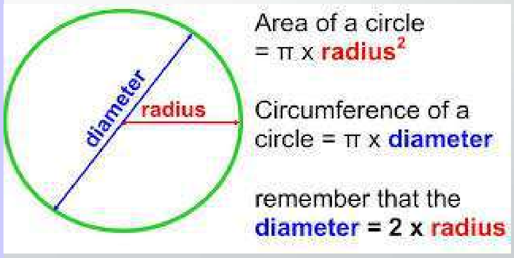

Surface area of a cylinder

Area of the each 'end' is πr^2
So area of both circles is $2\pi r^2$

[redacted]

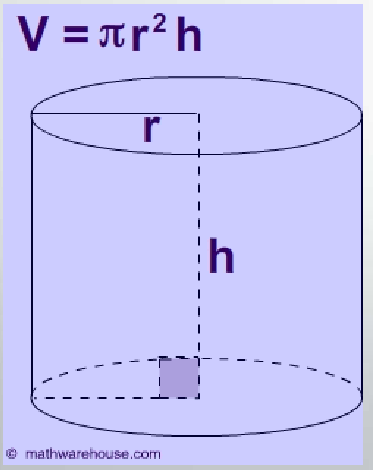
How to find out the surface area of a cylinder?

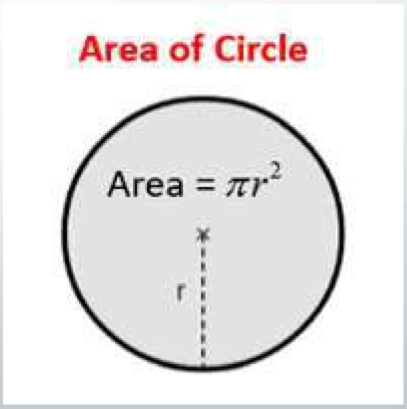



diameter
radius
Area of a circle = π x radius²
Circumference of a circle = π x diameter
remember that the diameter = 2 x radius

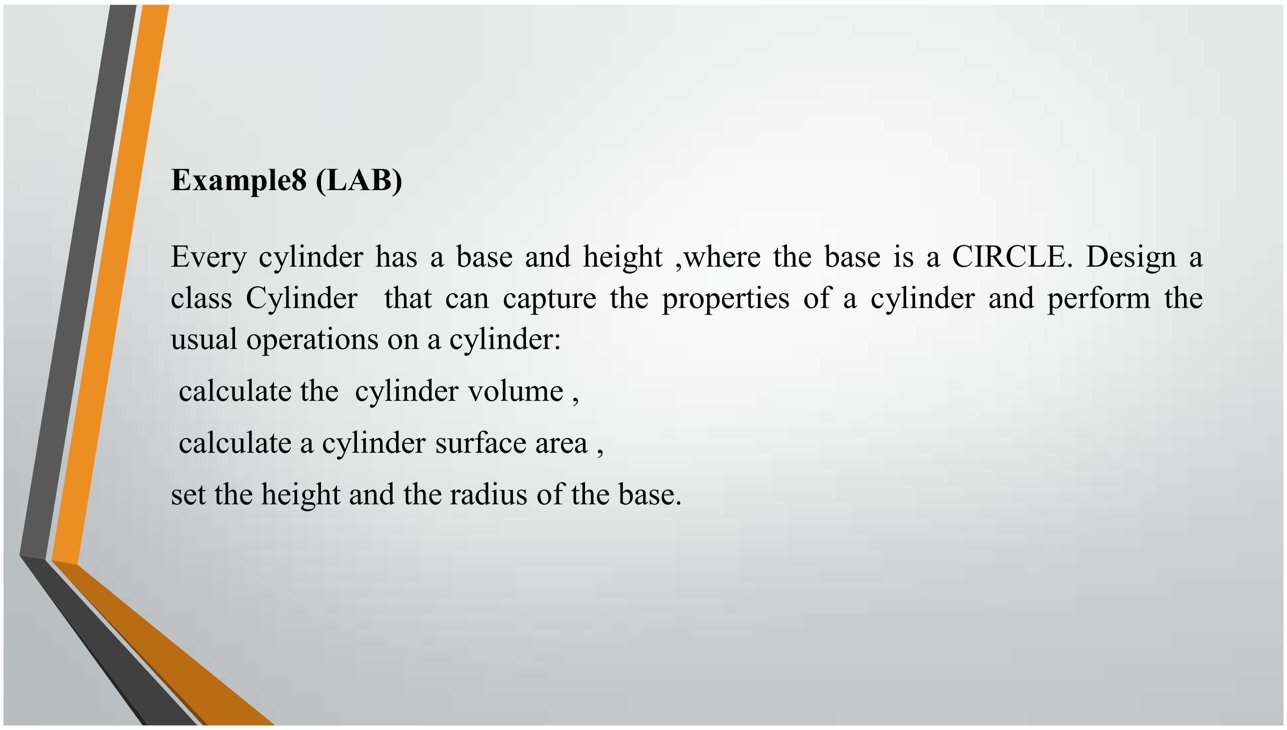
https://www.youtube.com/watch?v=9ez0Vw9Ddws

How to find out the volume of a cylinder?





[redacted]



Example8 (LAB)

Every cylinder has a base and height ,where the base is a CIRCLE. Design a class Cylinder that can capture the properties of a cylinder and perform the usual operations on a cylinder:

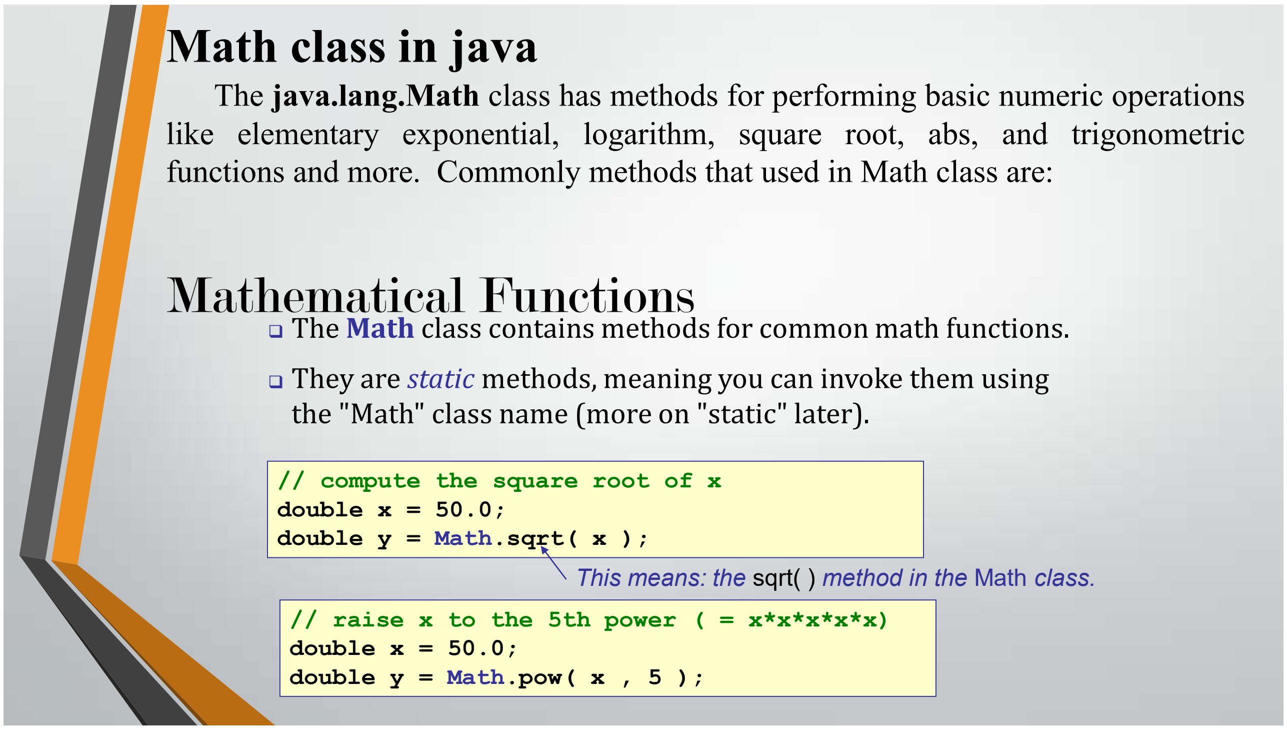
calculate the cylinder volume ,

calculate a cylinder surface area ,

set the height and the radius of the base.

- **Math class in java**
- **Examples**

[redacted]



Math class in java

The **java.lang.Math** class has methods for performing basic numeric operations like elementary exponential, logarithm, square root, abs, and trigonometric functions and more. Commonly methods that used in Math class are:

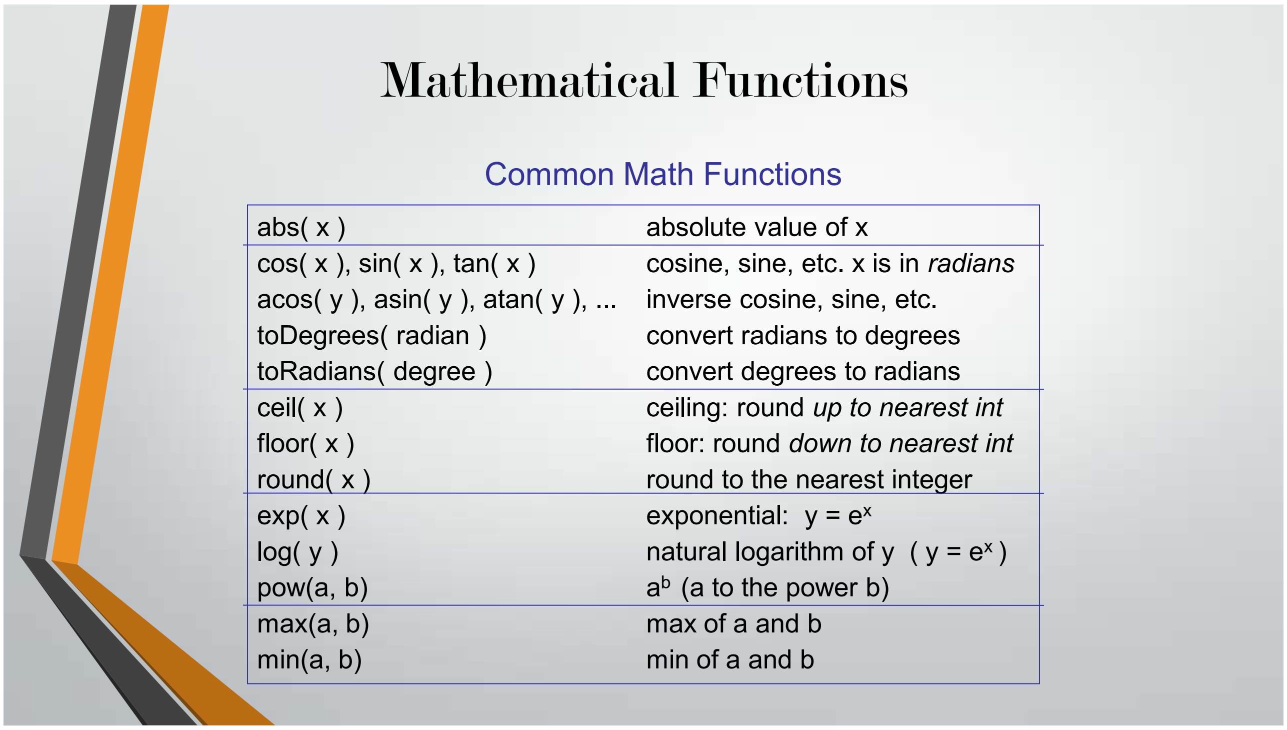
Mathematical Functions

- The **Math** class contains methods for common math functions.
- They are *static* methods, meaning you can invoke them using the "Math" class name (more on "static" later).

```
// compute the square root of x
double x = 50.0;
double y = Math.sqrt( x );
```

This means: the sqrt( ) *method in the* Math *class.*

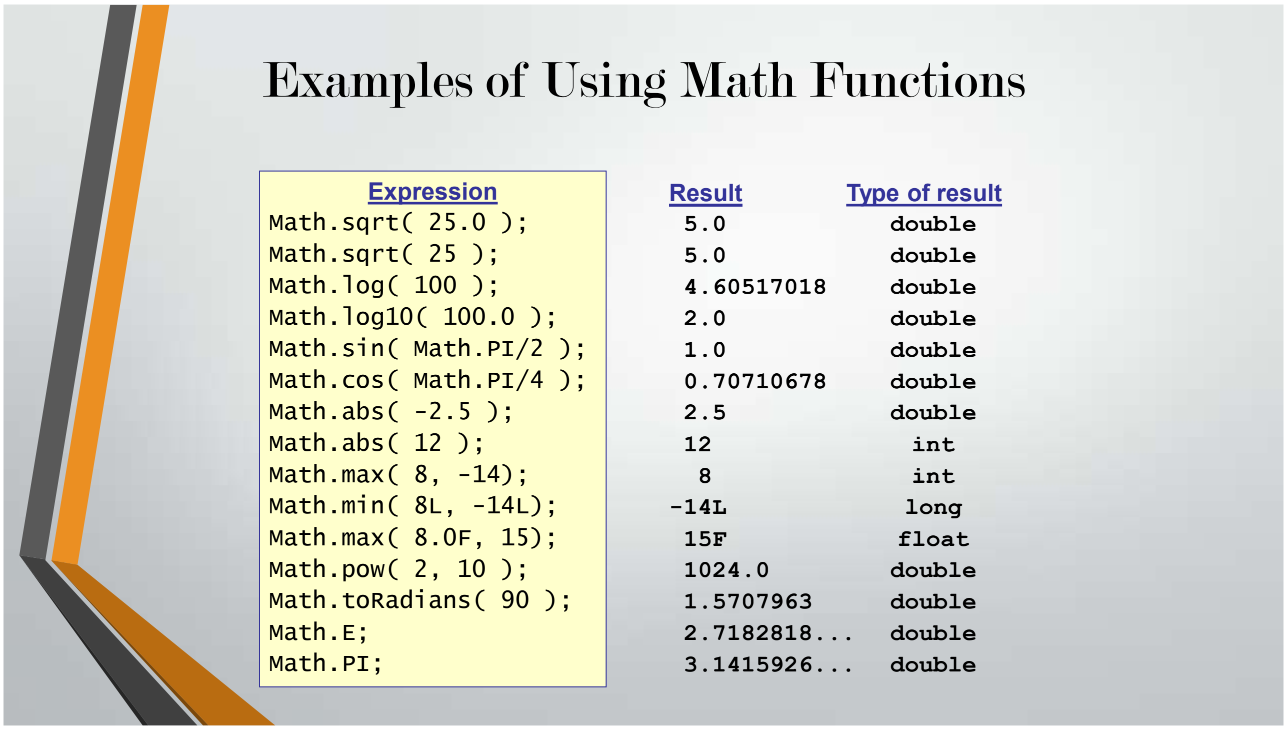
```
// raise x to the 5th power ( = x*x*x*x*x)
double x = 50.0;
double y = Math.pow( x , 5 );
```



Mathematical Functions

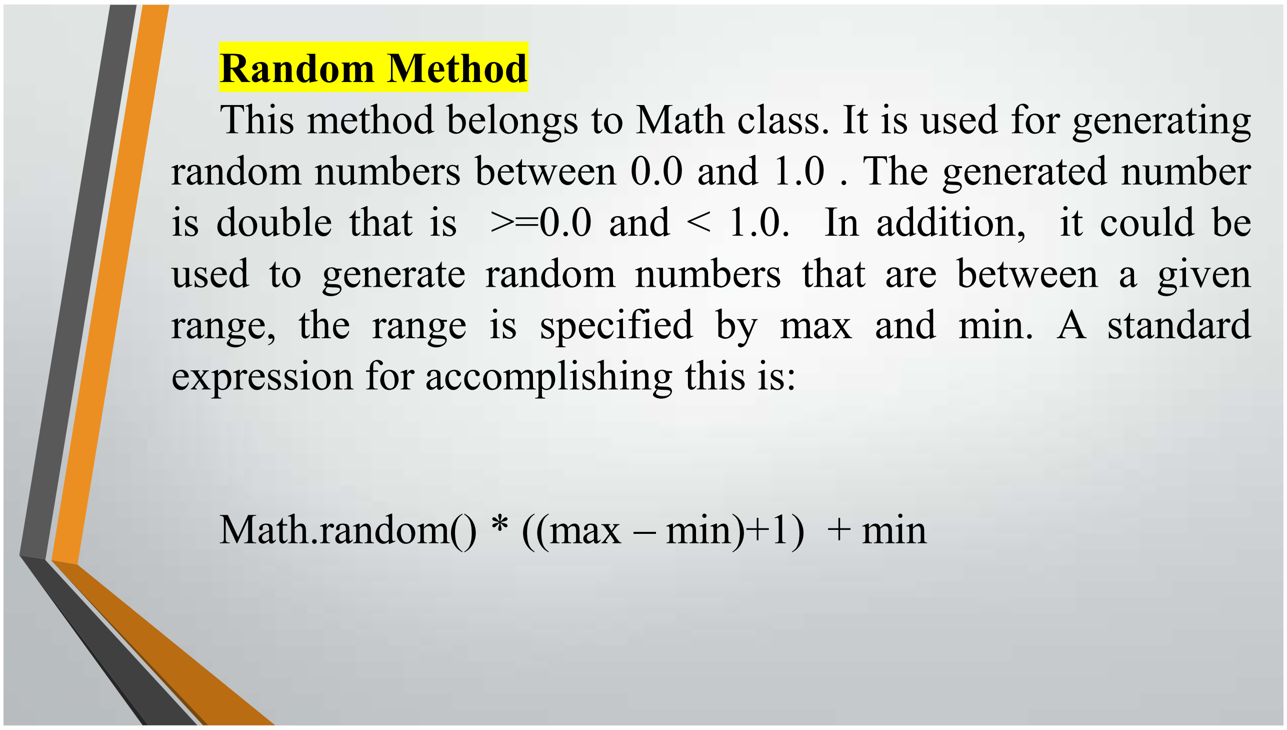
Common Math Functions

abs(x)	absolute value of x
cos(x), sin(x), tan(x)	cosine, sine, etc. x is in *radians*
acos(y), asin(y), atan(y), ...	inverse cosine, sine, etc.
toDegrees(radian)	convert radians to degrees
toRadians(degree)	convert degrees to radians
ceil(x)	ceiling: round *up to nearest int*
floor(x)	floor: round *down to nearest int*
round(x)	round to the nearest integer
exp(x)	exponential: $y = e^x$
log(y)	natural logarithm of y ($y = e^x$)
pow(a, b)	a^b (a to the power b)
max(a, b)	max of a and b
min(a, b)	min of a and b



Examples of Using Math Functions

Expression	Result	Type of result
`Math.sqrt( 25.0 );`	5.0	double
`Math.sqrt( 25 );`	5.0	double
`Math.log( 100 );`	4.60517018	double
`Math.log10( 100.0 );`	2.0	double
`Math.sin( Math.PI/2 );`	1.0	double
`Math.cos( Math.PI/4 );`	0.70710678	double
`Math.abs( -2.5 );`	2.5	double
`Math.abs( 12 );`	12	int
`Math.max( 8, -14);`	8	int
`Math.min( 8L, -14L);`	-14L	long
`Math.max( 8.0F, 15);`	15F	float
`Math.pow( 2, 10 );`	1024.0	double
`Math.toRadians( 90 );`	1.5707963	double
`Math.E;`	2.7182818...	double
`Math.PI;`	3.1415926...	double

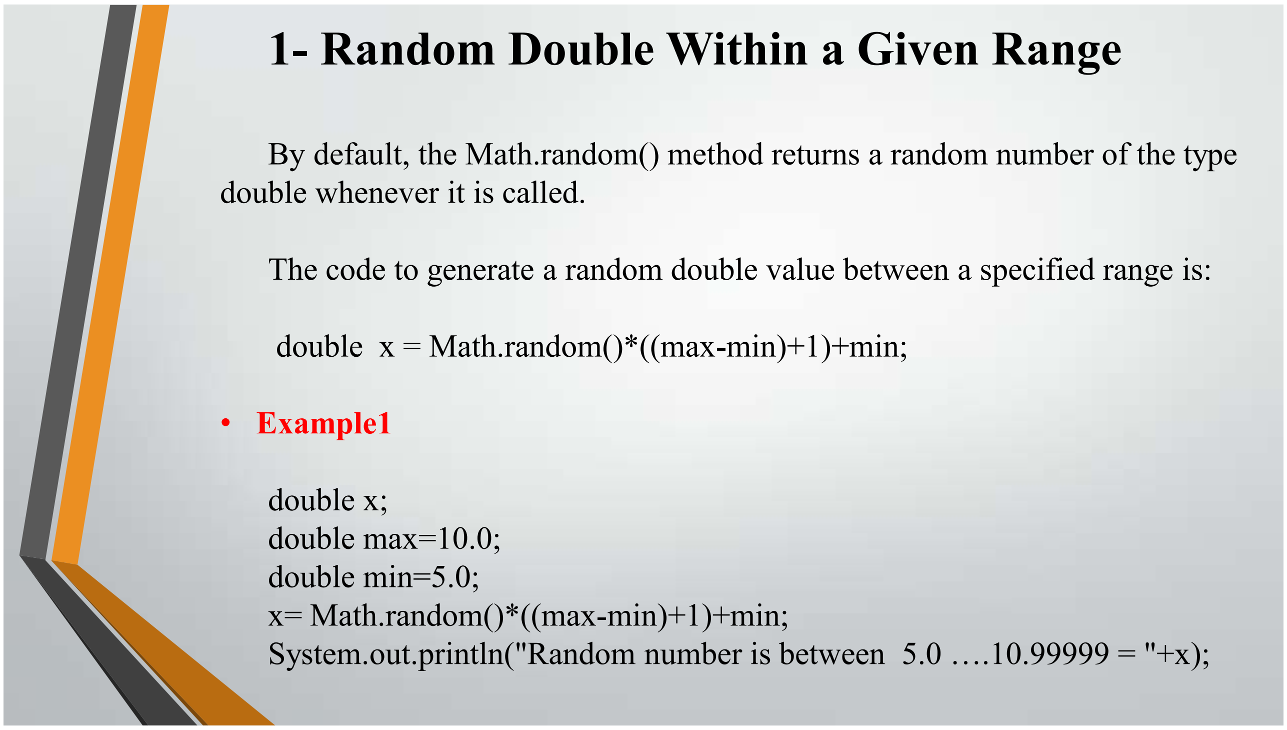


Random Method

This method belongs to Math class. It is used for generating random numbers between 0.0 and 1.0 . The generated number is double that is >=0.0 and < 1.0. In addition, it could be used to generate random numbers that are between a given range, the range is specified by max and min. A standard expression for accomplishing this is:

Math.random() * ((max – min)+1) + min

[redacted]



1- Random Double Within a Given Range

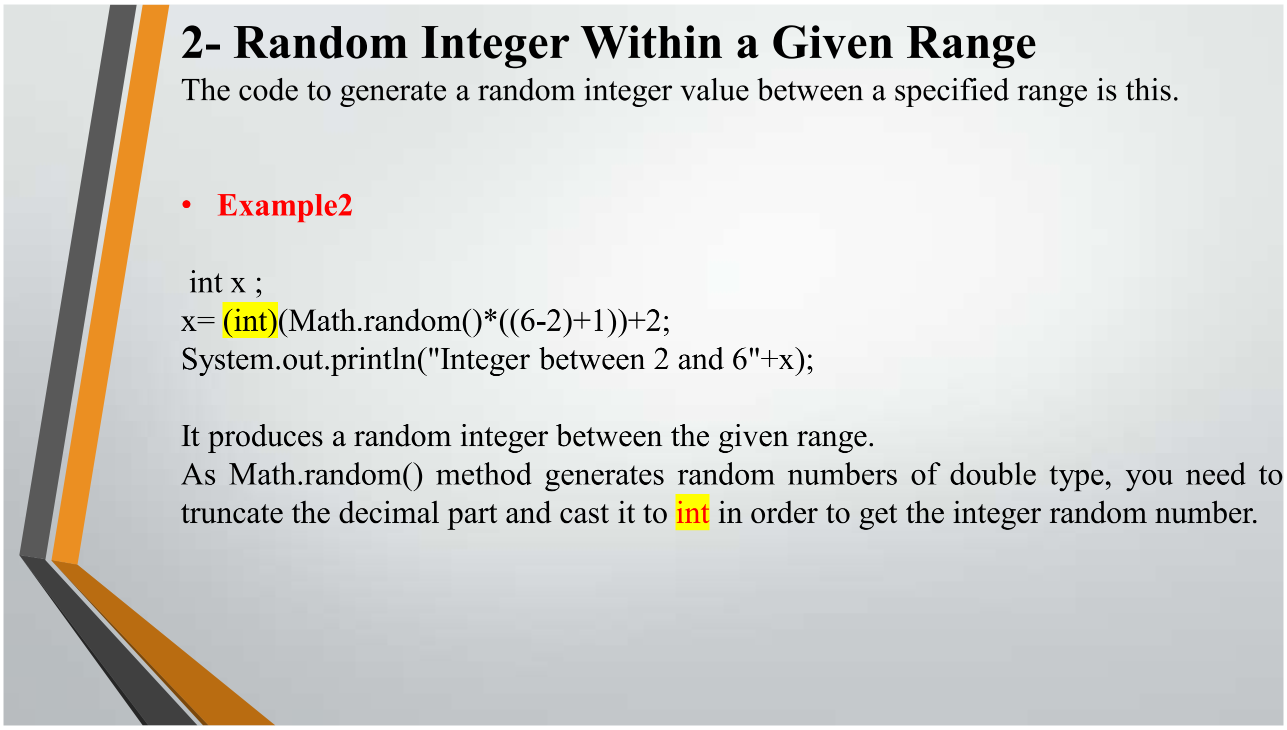
By default, the Math.random() method returns a random number of the type double whenever it is called.

The code to generate a random double value between a specified range is:

```
double  x = Math.random()*((max-min)+1)+min;
```

- **Example1**

```
double x;
double max=10.0;
double min=5.0;
x= Math.random()*((max-min)+1)+min;
System.out.println("Random number is between  5.0 ….10.99999 = "+x);
```



2- Random Integer Within a Given Range

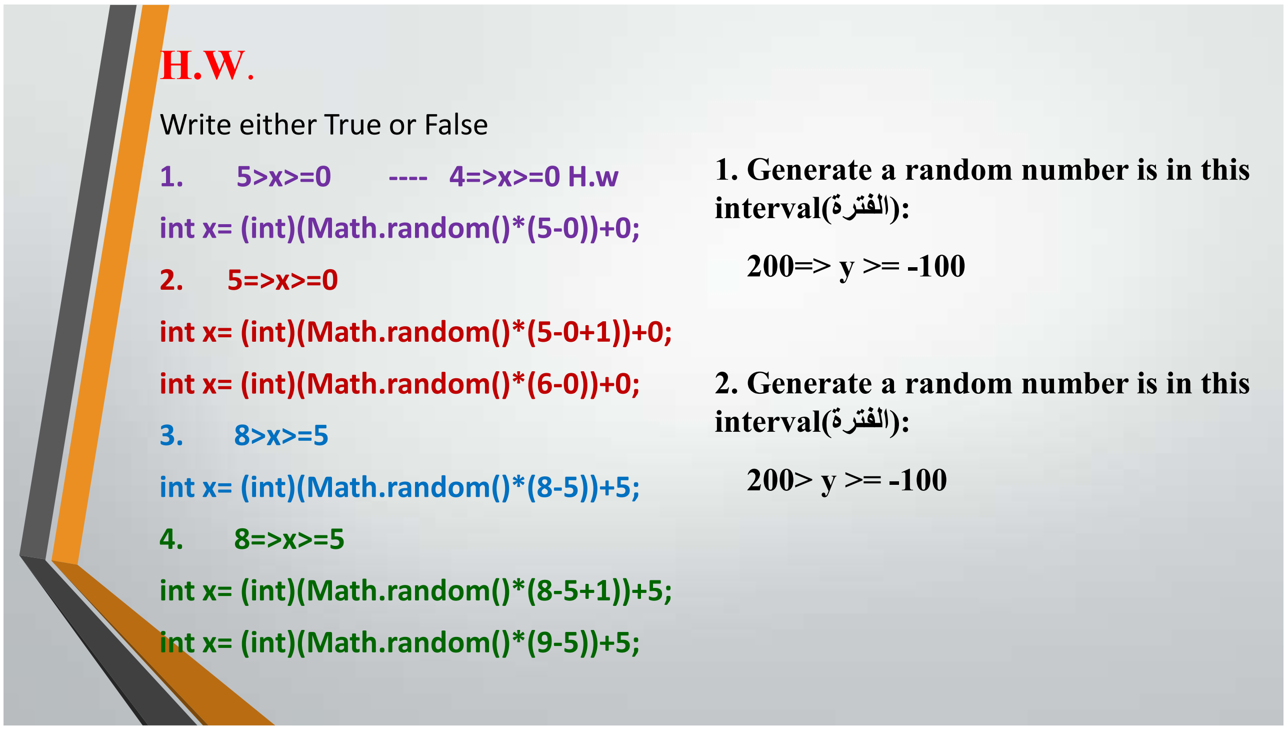
The code to generate a random integer value between a specified range is this.

- **Example2**

```
 int x ;
x= (int)(Math.random()*((6-2)+1))+2;
System.out.println("Integer between 2 and 6"+x);
```

It produces a random integer between the given range.
As Math.random() method generates random numbers of double type, you need to truncate the decimal part and cast it to int in order to get the integer random number.

[redacted]



H.W.

Write either True or False

1. 5>x>=0 ---- 4=>x>=0 H.w

```
int x= (int)(Math.random()*(5-0))+0;
```

2. 5=>x>=0

```
int x= (int)(Math.random()*(5-0+1))+0;
int x= (int)(Math.random()*(6-0))+0;
```

3. 8>x>=5

```
int x= (int)(Math.random()*(8-5))+5;
```

4. 8=>x>=5

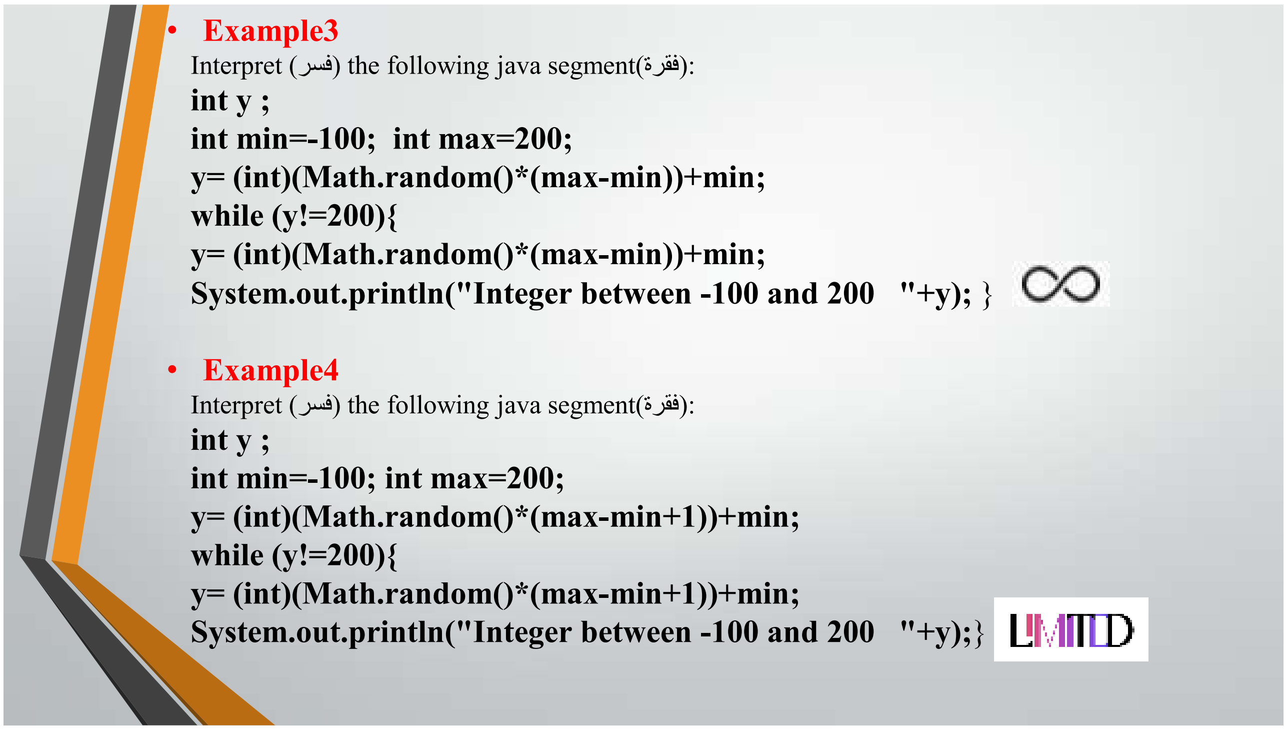
```
int x= (int)(Math.random()*(8-5+1))+5;
int x= (int)(Math.random()*(9-5))+5;
```

1. Generate a random number is in this interval(الفترة):

200=> y >= -100

2. Generate a random number is in this interval(الفترة):

200> y >= -100



- **Example3**

Interpret (فسر) the following java segment(فقرة):

```
int y ;
int min=-100;  int max=200;
y= (int)(Math.random()*(max-min))+min;
while (y!=200){
y= (int)(Math.random()*(max-min))+min;
System.out.println("Integer between -100 and 200   "+y); }
```

∞

- **Example4**

Interpret (فسر) the following java segment(فقرة):

```
int y ;
int min=-100; int max=200;
y= (int)(Math.random()*(max-min+1))+min;
while (y!=200){
y= (int)(Math.random()*(max-min+1))+min;
System.out.println("Integer between -100 and 200   "+y);}
```

LIMITD

• Example 5

1. Define a class which represents any point in the space. This class provides a method to find the distance between any point and the origin. Write a main method to find the smallest distance between each point in a set consists of 10 points and the origin.

```
public class Points{
private double x,y;
public void put_data(double a, double b){x=a;y=b;}
public double distance(){ return(Math.sqrt(x*x+y*y));}
public void print(){    System.out.println(x+"   ,   "+y);}}
```

```
public class Main {
public static void main(String[] args) {
 Points p[]=new Points [10];
 Points point=new Points();
 double d,mn;
 p[0]=new Points();
 p[0].put_data(13,12);
 mn=p[0].distance();point=p[0];
 for (int i=1;i<10;i++){
     p[i]=new Points();
     p[i].put_data(Math.random()*20,Math.random()*10);
     d=p[i].distance();
     System.out.println(d);
     if(d<mn) {mn=d;point=p[i];}          }
 System.out.println(mn);
 point.print();    } }
```

• Example 6

Define a class called Dice, this class has one integer variable members d which represents one dice.

It has two constructors the first one is used for setting the initial value for dice while the second one is used for calling the method roll.

The roll method is throwing the dice, so it needs to use the random method.

Use this class for representing a two players game each player has one dice.

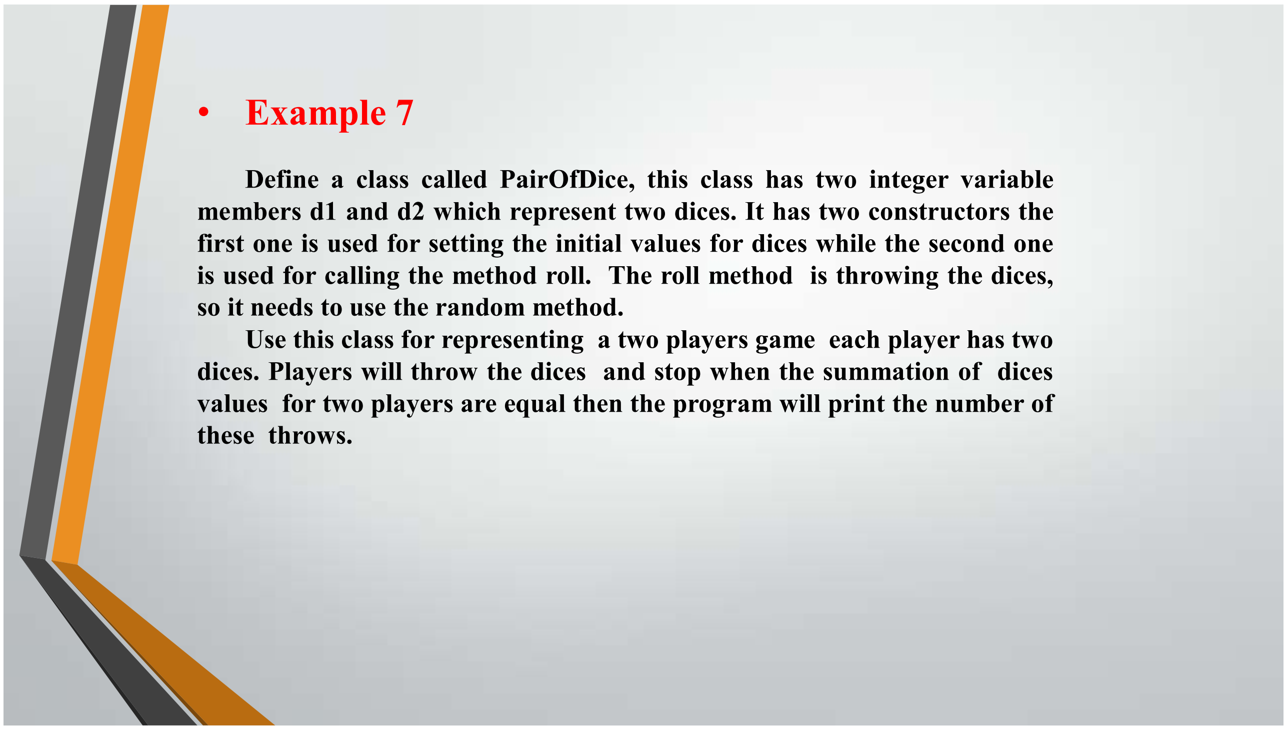
Players will throw the dices and print the greater value stop after 5 throws.

```
public class Dice {

public int d;
public Dice() {roll();   }
public Dice(int v) {
d = v;   }

public void roll() {
d = (int)(Math.random()*6) + 1;}
public int get_value(){return(d);}}
```

The output:

```
player1  6  player2  6
greater  6
player1  5  player2  6
greater  6
player1  5  player2  2
greater  5
player1  4  player2  6
greater  6
player1  5  player2  1
greater  5
```

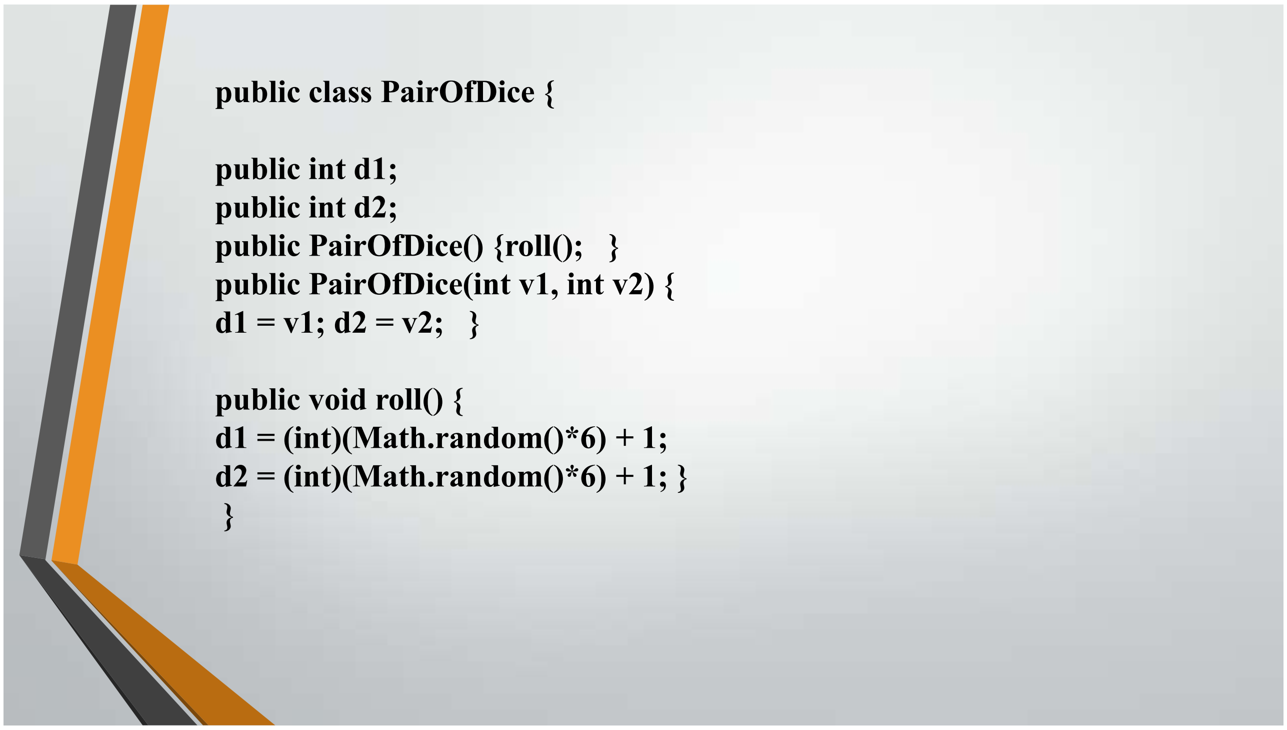
```
package javaapplication19;
public class JavaApplication19 {

    public static void main(String[] args) {
     Dice player1 = new Dice();       Dice player2 = new Dice ();
for (int i=1;i<=5;i++){
player1.roll();   player2.roll();
System.out.println("player1   "+player1.get_value() +"   player2   "+player2.get_value());

if(player1.get_value()>player2.get_value())
System.out.println("greater  "+player1.get_value());
else System.out.println("greater   "+player2.get_value()); }  } }
```



- **Example 7**

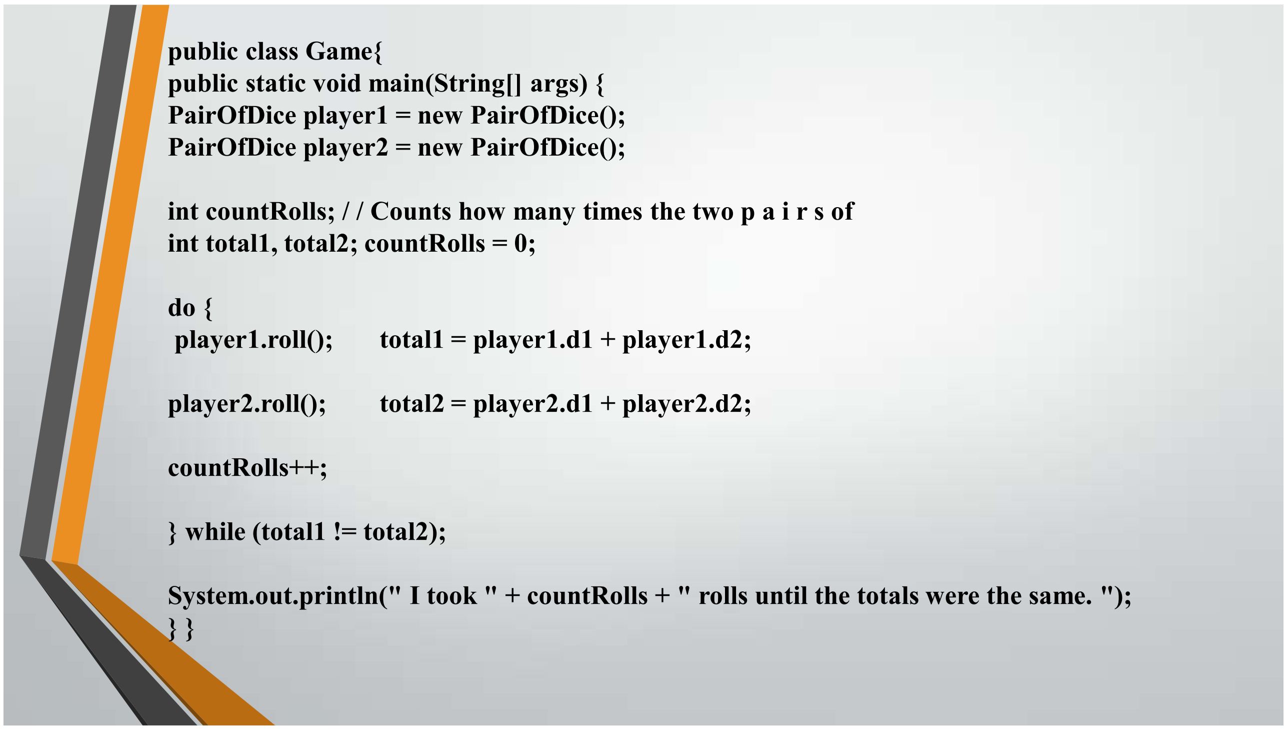
Define a class called PairOfDice, this class has two integer variable members d1 and d2 which represent two dices. It has two constructors the first one is used for setting the initial values for dices while the second one is used for calling the method roll. The roll method is throwing the dices, so it needs to use the random method.

Use this class for representing a two players game each player has two dices. Players will throw the dices and stop when the summation of dices values for two players are equal then the program will print the number of these throws.

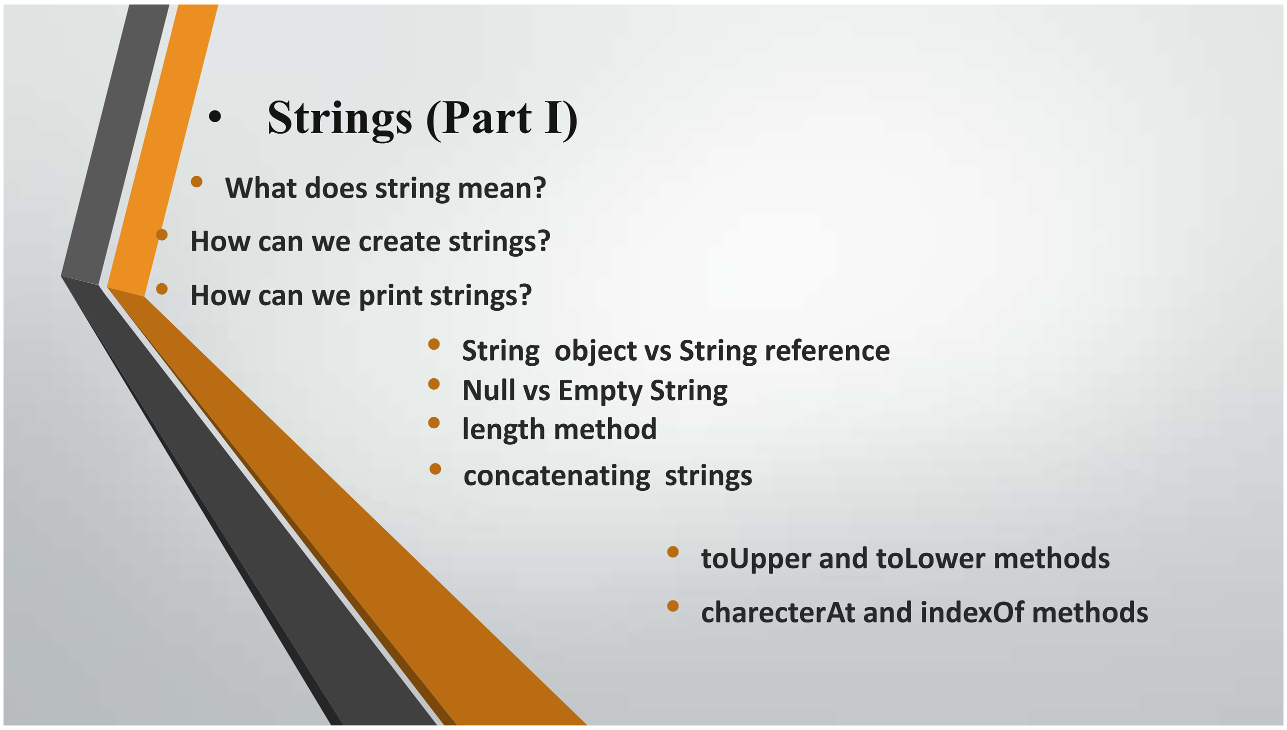


```
public class PairOfDice {

public int d1;
public int d2;
public PairOfDice() {roll();   }
public PairOfDice(int v1, int v2) {
d1 = v1; d2 = v2;   }

public void roll() {
d1 = (int)(Math.random()*6) + 1;
d2 = (int)(Math.random()*6) + 1; }
 }
```

[redacted]



```
public class Game{
public static void main(String[] args) {
PairOfDice player1 = new PairOfDice();
PairOfDice player2 = new PairOfDice();

int countRolls; / / Counts how many times the two p a i r s of
int total1, total2; countRolls = 0;

do {
 player1.roll();       total1 = player1.d1 + player1.d2;

player2.roll();       total2 = player2.d1 + player2.d2;

countRolls++;

} while (total1 != total2);

System.out.println(" I took " + countRolls + " rolls until the totals were the same. ");
} }
```



- **Strings (Part I)**
  - What does string mean?
  - How can we create strings?
  - How can we print strings?
    - String object vs String reference
    - Null vs Empty String
    - length method
    - concatenating strings
      - toUpper and toLower methods
      - charecterAt and indexOf methods

[redacted]

What does string mean?

- **Strings** which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects. The Java platform provides the String class to create and manipulate strings.
- Note that a Char is a single alphabet whereas String is zero or a sequence of characters. char is a primitive type whereas a String is a class.

char like ‘a’
String like "Iraq"

Write True or False
A=‘a’;
B=‘1’

C=‘12’
D=‘av’

S="a"
x="ab"
y="1"
z="123"
r="a123b"

How can we create strings?

How can we print strings?

Creating Strings

1. Literal: The most direct way to create a string is to write:

```
String greeting = "Hello world!";
String name="Raheeq";
System.out.println(greeting);
System.out.println(name);
```

In this case, "Hello world!" is a string literal—a series of characters in your code that is enclosed in double quotes. Whenever it encounters a string literal in your code, the compiler creates a String object with its value—in this case, Hello world!

[redacted]

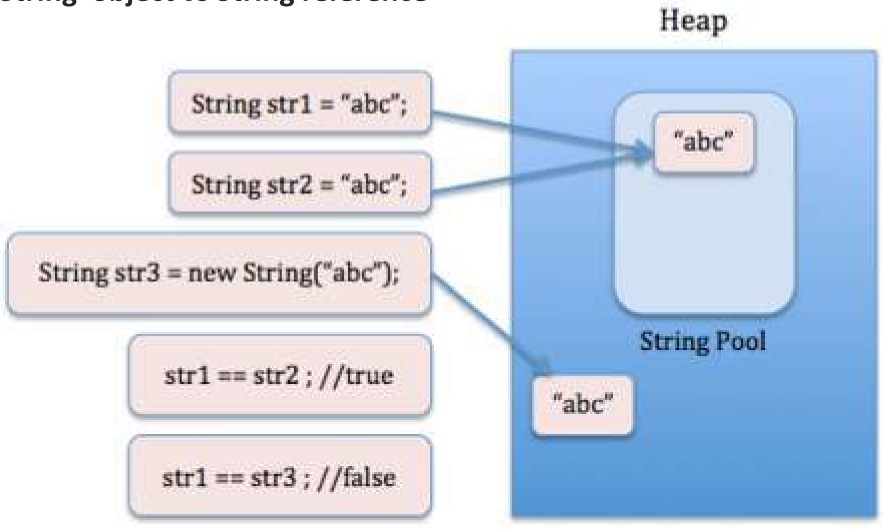
2. Constructor with Literal the string could be created using the <u>new keyword</u> and a constructor.

```
String s=new String("Welcome");   System.out.println(s);
String ss=new String("Raheeq"); System.out.println(ss);
```

3. Constructor with array of char: As with any other object, you can create String objects by using the <u>new keyword</u> and a constructor. The String class has thirteen constructors that allow you to provide the initial value of the string using different sources, such as an array of characters:

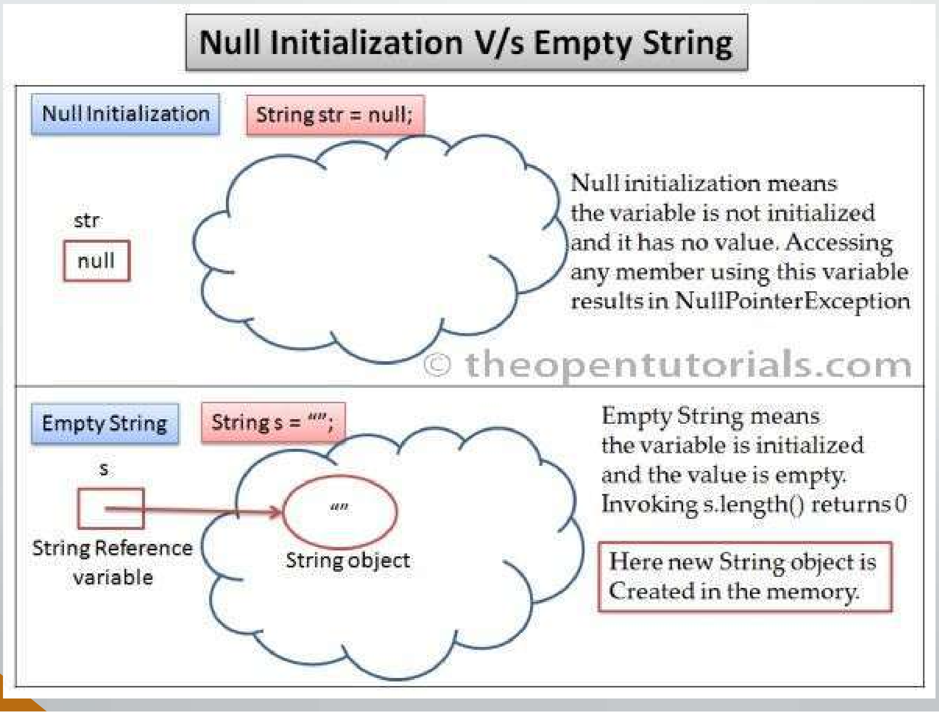
```
char [] a = { 'h', 'e', 'l', 'l', 'o', '.' };
String s = new String(a);   System.out.println(s);
```

- String object vs String reference



```
String str1 = "abc";
String str2 = "abc";
String str3 = new String("abc");
str1 == str2 ; //true
str1 == str3 ; //false
```

[redacted]

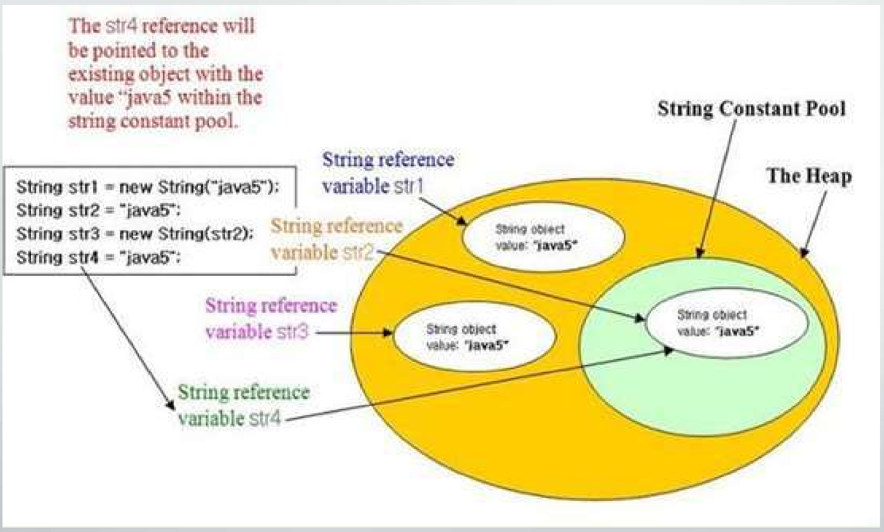


Null Initialization V/s Empty String

Null initialization means the variable is not initialized and it has no value. Accessing any member using this variable results in NullPointerException

Empty String means the variable is initialized and the value is empty. Invoking s.length() returns 0

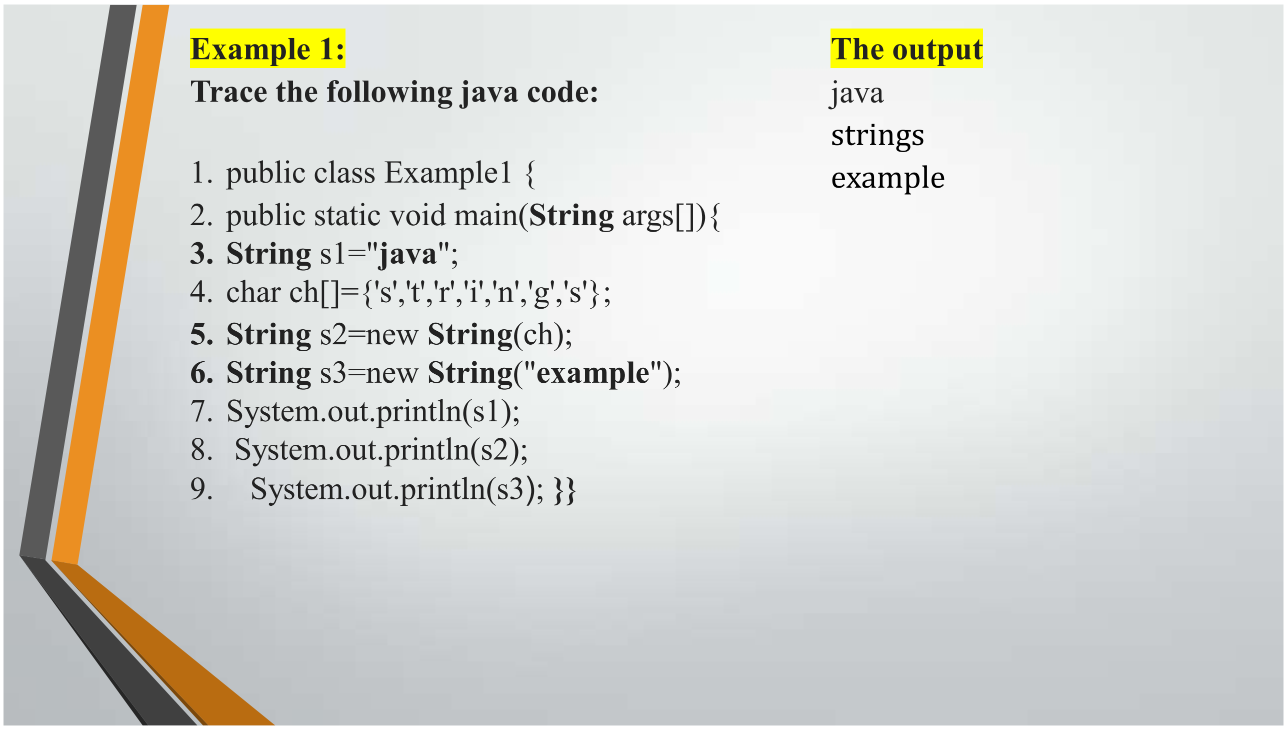
Here new String object is Created in the memory.



The str4 reference will be pointed to the existing object with the value "java5 within the string constant pool.

```
String str1 = new String("java5");
String str2 = "java5";
String str3 = new String(str2);
String str4 = "java5";
```

String Constant Pool
The Heap
String reference variable str1
String reference variable str2
String reference variable str3
String reference variable str4
String object value: "java5"
String object value: "java5"
String object value: "java5"



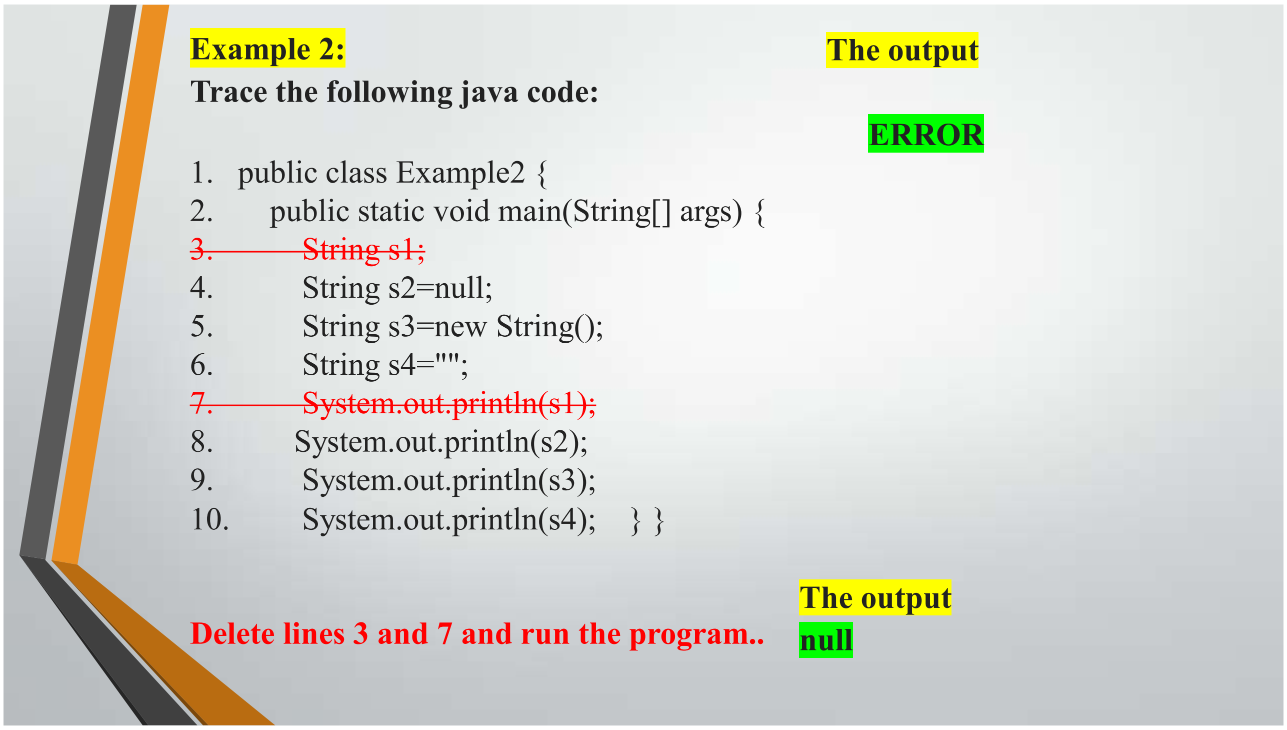
Example 1:

Trace the following java code:

```
public class Example1 {
public static void main(String args[]){
String s1="java";
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);
String s3=new String("example");
System.out.println(s1);
 System.out.println(s2);
   System.out.println(s3); }}
```

The output

```
java
strings
example
```



Example 2:

Trace the following java code:

```
1.   public class Example2 {
2.       public static void main(String[] args) {
~~3.           String s1;~~
4.           String s2=null;
5.           String s3=new String();
6.           String s4="";
~~7.           System.out.println(s1);~~
8.          System.out.println(s2);
9.           System.out.println(s3);
10.          System.out.println(s4);    } }
```

The output

ERROR

Delete lines 3 and 7 and run the program..

The output

null

- **length method**

•**Java String length():** The Java String length() method tells the length of the string. It returns count of total number of characters present in the String.

Example 3:

Trace the following java code:

```
public class Example3{
public static void main(String args[]{
String s1="hello";
String s2="whatsup";
System.out.println("String length is: "+s1.length());
System.out.println("String length is: "+s2.length()); }}
```

The output

```
String length is: 5
String length is: 7
```

- **Concat method**

Java String concat() or (+ operator) : The Java String concat() method combines a specific string at the end of another string and ultimately returns a combined string. It is like appending another string.

Example 4:

Trace the following java code:

```
public class Example4{
public static void main(String args[]){
String s1="hello";
s1=s1.concat("how are you");
System.out.println(s1); }}
```

The output

```
hellohow are you
```

[redacted]

Short Quiz

What are the values of s3,s4,s5 and s6

String s1=″ssss″;
String s2=″ddddd″;
String s3=s1+s2; s3 ssssddddd

String s4=s1.concat(s2); s4 ssssddddd

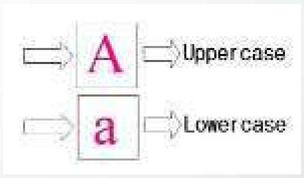
String s1=″sss″;
String s2=″bbbb″;
String s3=″dddd″;
String s5=s1+s2+s3; s5 sssbbbbdddd

String s6=s1.concat(s2.concat(s3)); s6 sssbbbbdddd

- **ToLowerCase and toUpperCase methods**

Java String toLowerCase(): The java string toLowerCase() method converts all the characters of the String to lower case.
Java String toUpperCase() : The Java String toUpperCase() method converts all the characters of the String to upper case.

Example5



1. public class Example5{
2. public static void main(String args[]){
3. String s1="HELLO HOW Are You?″;
4. String lower=s1.toLowerCase();
5. System.out.println(lower);
6. String s3="123AAbb456"; H.W.
7. String s2="AAbbCCdd123" ;
8. String upper=s2.toUpperCase();
9. System.out.println(upper);}}

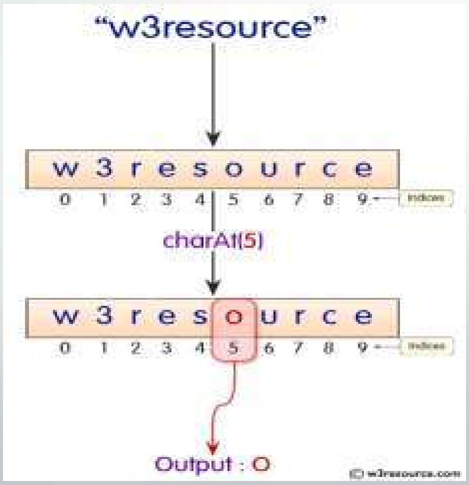
The output

hello how are you?

AABBCCDD123

[redacted]

The **charAt()** method: of the String class returns the char value at the specified index. An index ranges from 0 to length() - 1. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.



Example6: (Trace)

```
String s="IRAQ";
 char c=s.charAt(2);
System.out.println(c);
```

The output
A

```
for (int i=0;i<s.length();i++)
System.out.println(s.charAt(i));
```

The output
I
R
A
Q

Example7: (Trace)

```
String s="IRAQ" ;
char c=s.charAt(4);
 System.out.println(c);
```

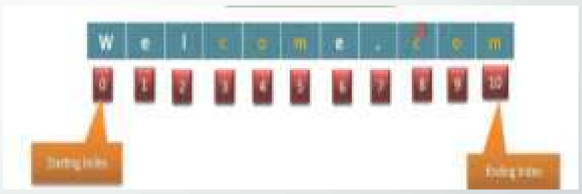
ERROR ...Why?

[redacted]

The method **indexOf()** is used for finding out the index of the specified character or substring in a particular String.

There are 4 variations of this method:

1- int indexOf(int ch): It returns the index of the first occurrence of character ch in a String.



2- int indexOf(int ch, int fromIndex): It returns the index of first occurrence if character ch, starting from the specified index "fromIndex".

Example8

Trace the following java code:

```
package javaapplication34;
 class Example8 {

    public static void main(String[] args) {
    String s1=new String("IRAQ IS MY COUNTRY I Love IRAQ");
       int i;
       i=s1.indexOf('I'); System.out.println(i);
       i=s1.indexOf('I',2); System.out.println(i); }}
```

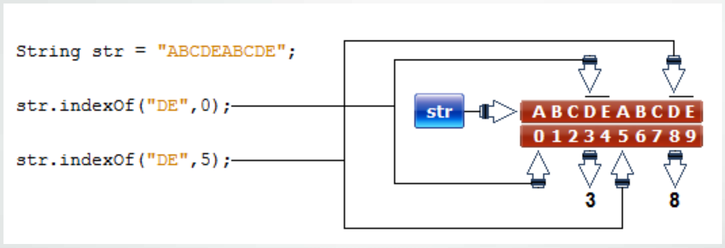
The output:

```
0
5
```

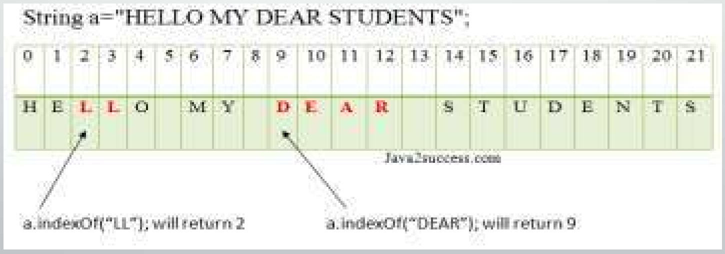
[redacted]

3- int indexOf(String str): Returns the index of string str in a particular String.

4- int indexOf(String str, int fromIndex): Returns the index of string str, starting from the specified index "fromIndex".



```
String str = "ABCDEABCDE";

str.indexOf("DE",0);

str.indexOf("DE",5);
```



String a="HELLO MY DEAR STUDENTS";

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
H	E	L	L	O		M	Y		D	E	A	R		S	T	U	D	E	N	T	S

a.indexOf("LL"); will return 2

a.indexOf("DEAR"); will return 9

Example9

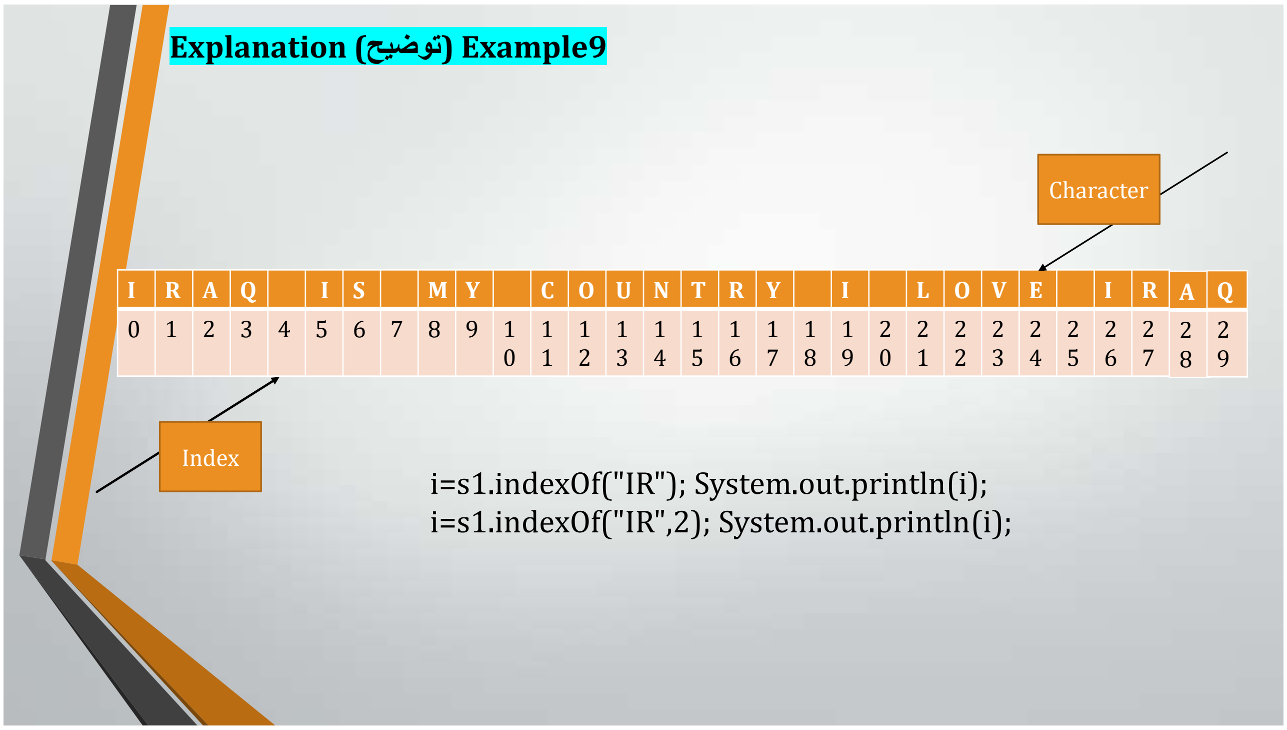
Trace the following java code:

```
package javaapplication34;
  class Example9 {
    public static void main(String[] args) {
     String s1=new String("IRAQ IS MY COUNTRY I LOVE IRAQ");
        int i;
        i=s1.indexOf("IR"); System.out.println(i);
        i=s1.indexOf("IR",2); System.out.println(i);}}
```

The output:

```
0
26
```

[redacted]

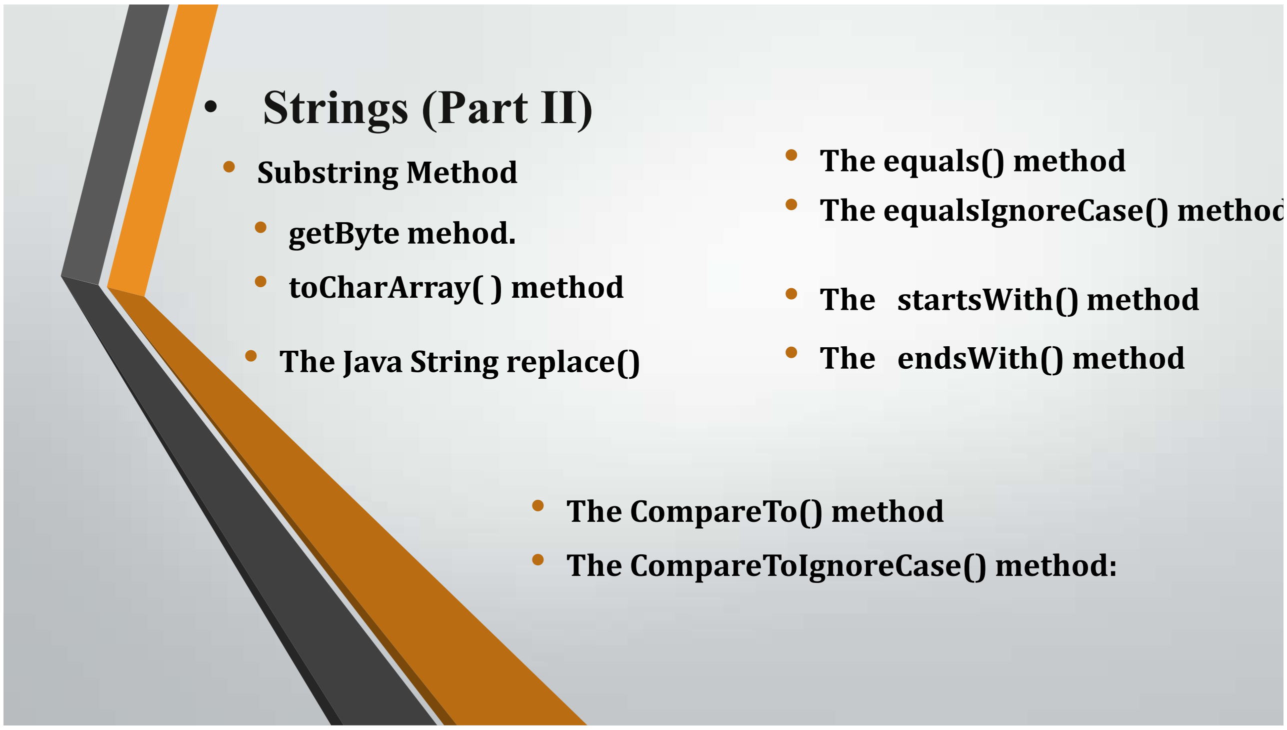


Explanation (توضيح) Example9

Character
Index

I	R	A	Q		I	S		M	Y		C	O	U	N	T	R	Y		I		L	O	V	E		I	R	A	Q
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

```
i=s1.indexOf("IR"); System.out.println(i);
i=s1.indexOf("IR",2); System.out.println(i);
```



- **Strings (Part II)**
  - **Substring Method**
    - **getByte mehod.**
    - **toCharArray( ) method**
  - **The Java String replace()**
  - **The equals() method**
  - **The equalsIgnoreCase() method**
  - **The startsWith() method**
  - **The endsWith() method**
  - **The CompareTo() method**
  - **The CompareToIgnoreCase() method:**

[redacted]

Substring Method

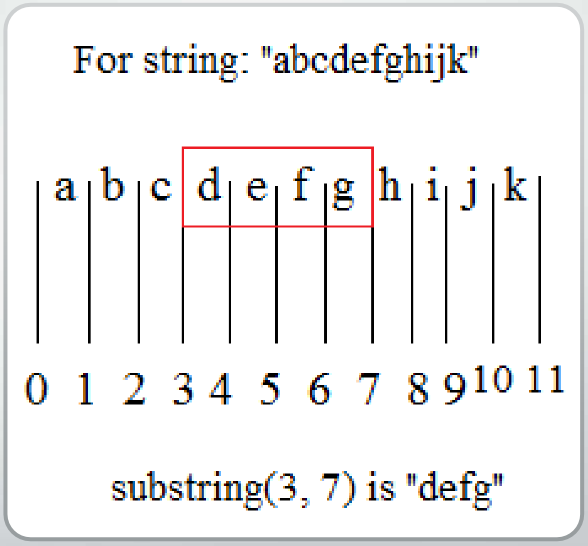
Substring is a subset of another string. Note: Index starts from 0.
You can get substring from the given string object by one of the two methods:

•public String substring(int startIndex): This method returns new String object containing the substring of the given string from specified startIndex (inclusive).

•public String substring(int startIndex, int endIndex): This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

In case of substring startIndex is inclusive and endIndex is exclusive.

- **Example 1**



• Example 2 (Trace)

Let's understand the startIndex and endIndex by the code given below:

```
String s="hello";
System.out.println(s.substring(0,2));
System.out.println(s.substring(2));
```

The output

```
he

llo
```

• getBytes method

Java. **getBytes**() method in **java** is used to convert a string into sequence of bytes and returns an array of bytes.

• Example 3 (Trace)

```
public class Example3 {
  public static void main(String[] args) {

  String s="ABCDEF";

 byte[] b= s.getBytes();

  for (int i=0;i<b.length;i++)
  System.out.println(b[i]);

  } }
```

The output :

```
65
66
67
68
69
70
```

If we replace the string s to "abcdef " in example3

The output will be:

```
97
98
99
100
101
102
```

H.W. what will be the output when the string s=”0123456”

• toCharArray() method

The **java** string **toCharArray**() method converts the given string into a sequence of characters. The returned array length is equal to the length of the string.

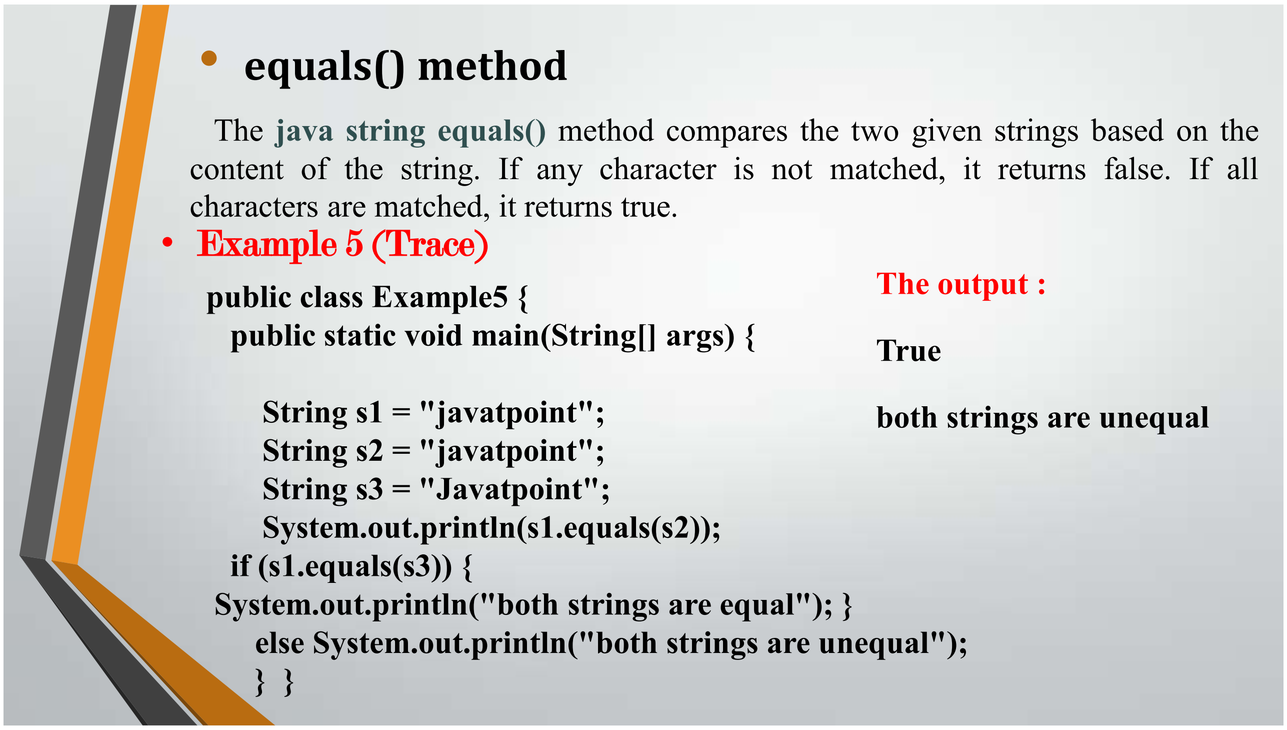
• **Example 4 (Trace)**

```
public class Example4 {
  public static void main(String[] args) {

  String s="abcdef";
    char[] c= s.toCharArray();

  for (int i=0;i<c.length;i++)
      System.out.println(c[i]);

} }
```

The output :

```
a
b
c
d
e
f
```

What will be the output if we replace the string s to “123456”?



equals() method

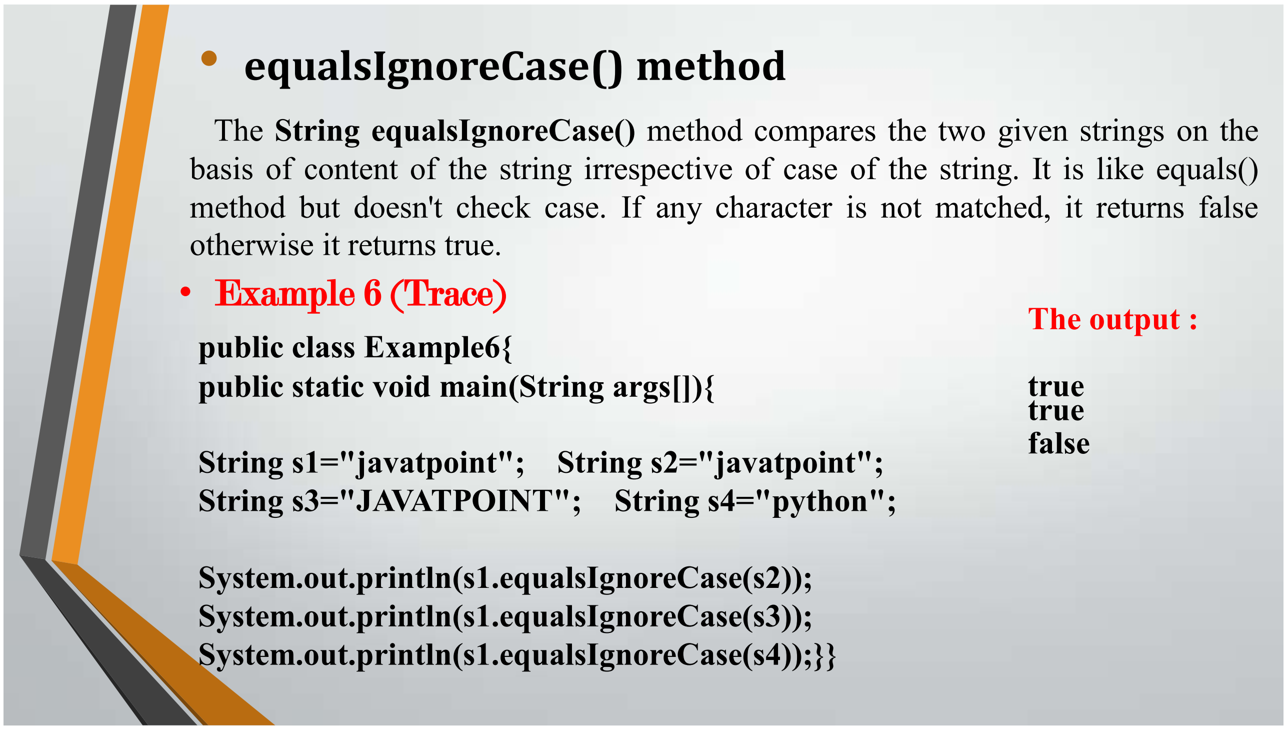
The **java string equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

- **Example 5 (Trace)**

```
public class Example5 {
  public static void main(String[] args) {

     String s1 = "javatpoint";
     String s2 = "javatpoint";
     String s3 = "Javatpoint";
     System.out.println(s1.equals(s2));
  if (s1.equals(s3)) {
 System.out.println("both strings are equal"); }
    else System.out.println("both strings are unequal");
    } }
```

The output :

```
True

both strings are unequal
```



equalsIgnoreCase() method

The **String equalsIgnoreCase()** method compares the two given strings on the basis of content of the string irrespective of case of the string. It is like equals() method but doesn't check case. If any character is not matched, it returns false otherwise it returns true.

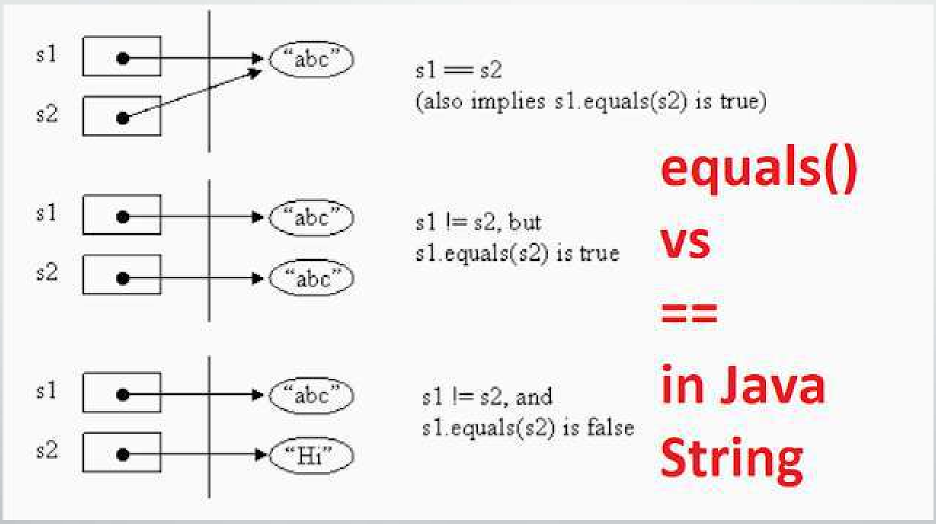
- **Example 6 (Trace)**

```
public class Example6{
public static void main(String args[]){

String s1="javatpoint";   String s2="javatpoint";
String s3="JAVATPOINT";   String s4="python";

System.out.println(s1.equalsIgnoreCase(s2));
System.out.println(s1.equalsIgnoreCase(s3));
System.out.println(s1.equalsIgnoreCase(s4));}}
```

The output :

```
true
true
false
```

[redacted]



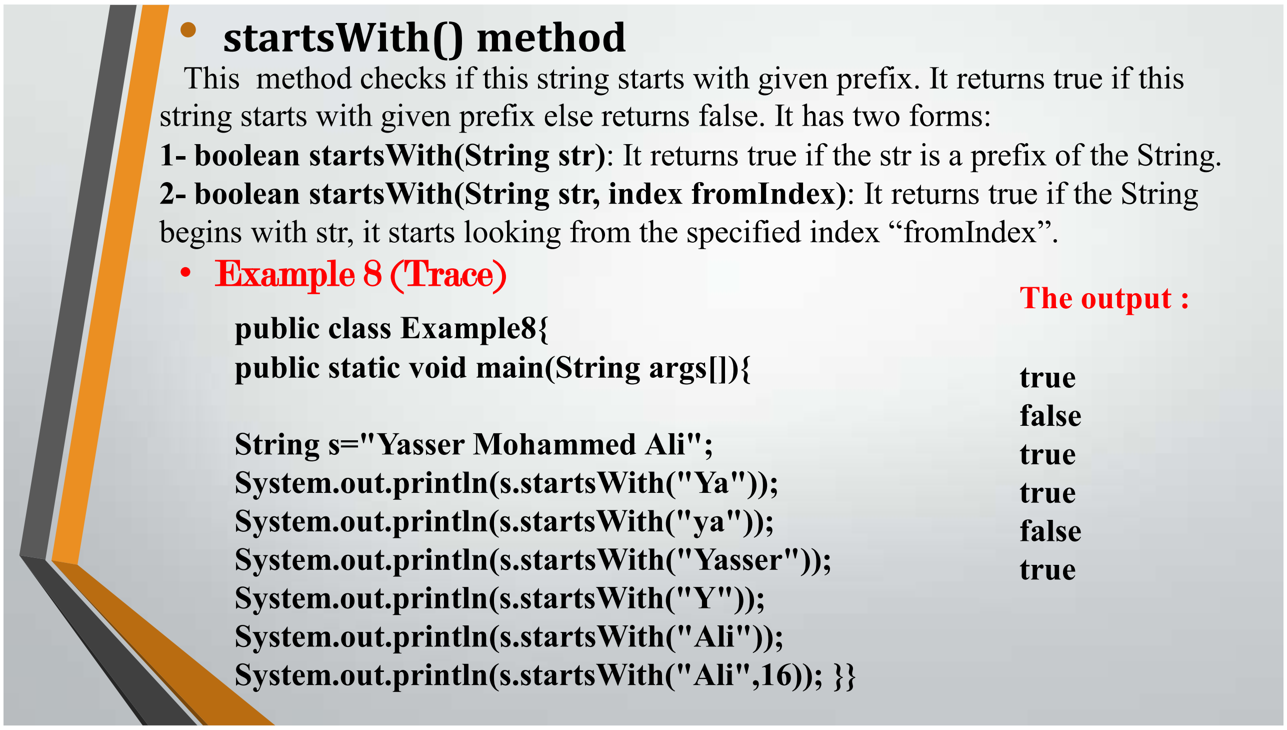
- **Example 7 (Trace) H.W.**

```
public class Example7 {

 public static void main(String[] args) {

 String s1="IRAQ";
 String s2=" IRAQ ";
 System.out.println(s1==s2);
 System.out.println(s1.equals(s2));

 String s3=new String(" IRAQ ");
 String s4=new String(" IRAQ ");
 System.out.println(s3==s4);
 System.out.println(s3.equals(s4));}
 }
```

Why the output will be as shown below?

```
true
true
false
true
```

[redacted]



startsWith() method

This method checks if this string starts with given prefix. It returns true if this string starts with given prefix else returns false. It has two forms:

1- boolean startsWith(String str): It returns true if the str is a prefix of the String.

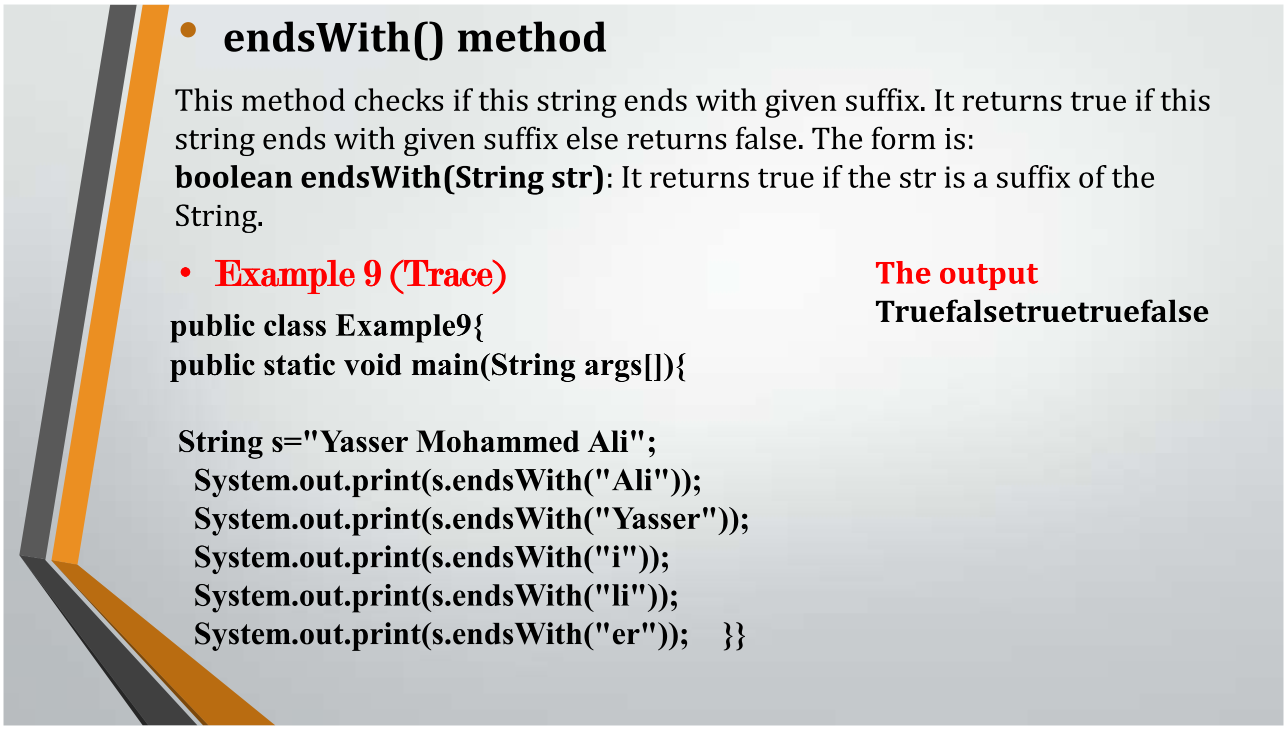
2- boolean startsWith(String str, index fromIndex): It returns true if the String begins with str, it starts looking from the specified index "fromIndex".

- Example 8 (Trace)

```
public class Example8{
public static void main(String args[]){

String s="Yasser Mohammed Ali";
System.out.println(s.startsWith("Ya"));
System.out.println(s.startsWith("ya"));
System.out.println(s.startsWith("Yasser"));
System.out.println(s.startsWith("Y"));
System.out.println(s.startsWith("Ali"));
System.out.println(s.startsWith("Ali",16)); }}
```

The output :

```
true
false
true
true
false
true
```



endsWith() method

This method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false. The form is:
boolean endsWith(String str): It returns true if the str is a suffix of the String.

- Example 9 (Trace)

```
public class Example9{
public static void main(String args[]){

 String s="Yasser Mohammed Ali";
   System.out.print(s.endsWith("Ali"));
   System.out.print(s.endsWith("Yasser"));
   System.out.print(s.endsWith("i"));
   System.out.print(s.endsWith("li"));
   System.out.print(s.endsWith("er"));    }}
```

The output

```
Truefalsetruetruefalse
```

[redacted]

replace() method

This method returns a string replacing all the old char or CharSequence to new char or CharSequence..

There are two type of replace methods in java string.

1. public String replace(char oldChar, char newChar).
2. public String replace(CharSequence target, CharSequence replacement)

- **Example 10 (Trace)**

```
public class ReplaceExample1{
public static void main(String args[]){
String s1="aaabbbccc";
String s2=s1.replace('a','e');  →  String s2=s1.replace("a","e"); √
System.out.println(s1+"  "+s2);
String s3=s1.replace("ab","zx");  →  String s3=s1.replace('ab','zx'); ✗

System.out.println(s1+"  "+s3); }}
```

The output :

```
aaabbbccc  eeebbbccc
aaabbbccc  aazxbbccc
```

compareTo() method

It is used for comparing two strings lexicographically. Each character of both strings is converted into a Unicode value. Assume that you have two strings a1 and a2, this method returns:

a1.compareTo(a2)

it returns positive number if a1 > a2 +

it returns negative number if a1 < a2 −

it returns 0 if a1 == a2, it returns 0 **0**

CompareToIgnoreCase() method: This method compares two strings lexicographically, ignoring case differences. This means that “aa” equals “AA”.

[redacted]

• Example 11 (Trace)

```
public class Example11{
public static void main(String args[]){

 String s1="AA";          String s2="AAA";
 String s3="BBB", s4="BB";
 String s5="aaa";
 System.out.println(s1.compareTo(s2));
System.out.println(s1.compareTo(s3));
        System.out.println(s4.compareTo(s2));
System.out.println(s1.compareTo(s2));
        System.out.println(s1.compareTo(s4));
System.out.println(s5.compareTo(s2)); }}
```

The output :

```
-1
-1
1
-1
-1
32
```

• Example 12 (Trace) H.W.

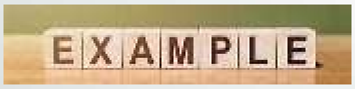
```
public class Example12 {
 public static void main(String args[]) {

     String str1 = "String method tutorial“,  str2 = "compareTo method example";
      String str3 = "String method tutorial";

      int var1=str1.compareTo( str2 );System.out.println("str1&str2 comparison: "+var1);
      int var2=str1.compareTo( str3 );System.out.println("str1&str3 comparison: "+var2);

      int var3 = str2.compareTo("compareTo method example");
      System.out.println("str2 & string argument comparison: "+var3);    } }
```

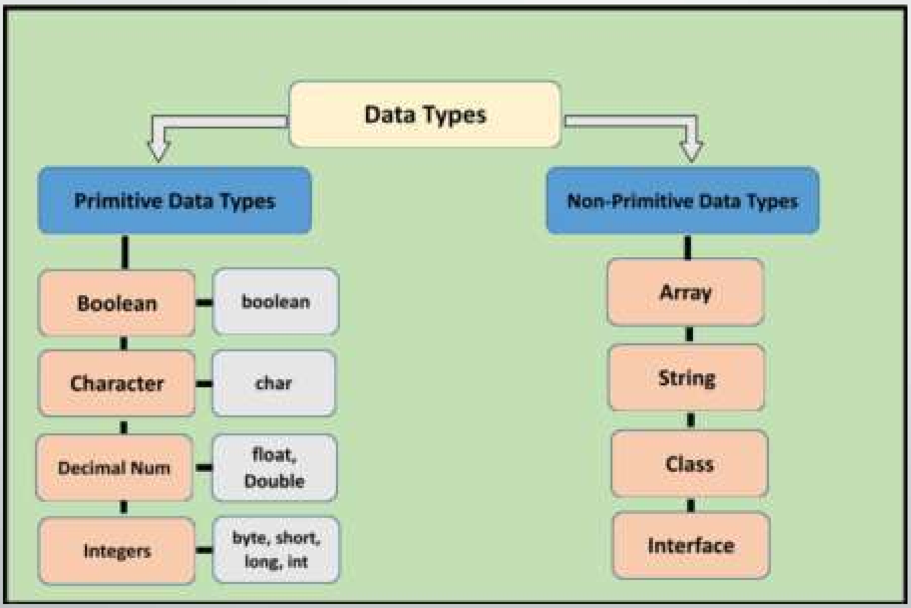
Why the output will be as shown below?

```
str1 & str2 comparison: -16
str1 & str3 comparison: 0
str2 & string argument comparison: 0
```

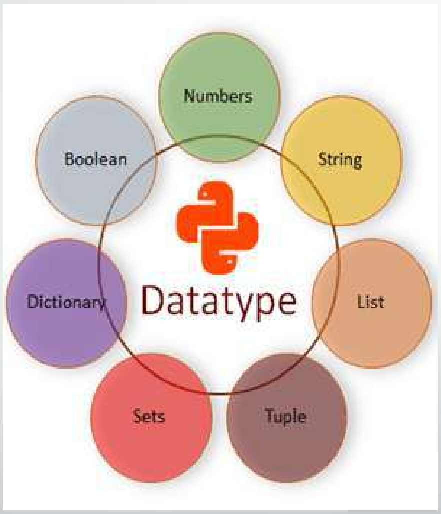
[redacted]

- Converting any data type to String and verse versa
- Reading from KB
- Examples





[redacted]



1- Converting from any data type to String

1. It is very simple way to convert any data type to a string using the + operator as shown at the following java code:

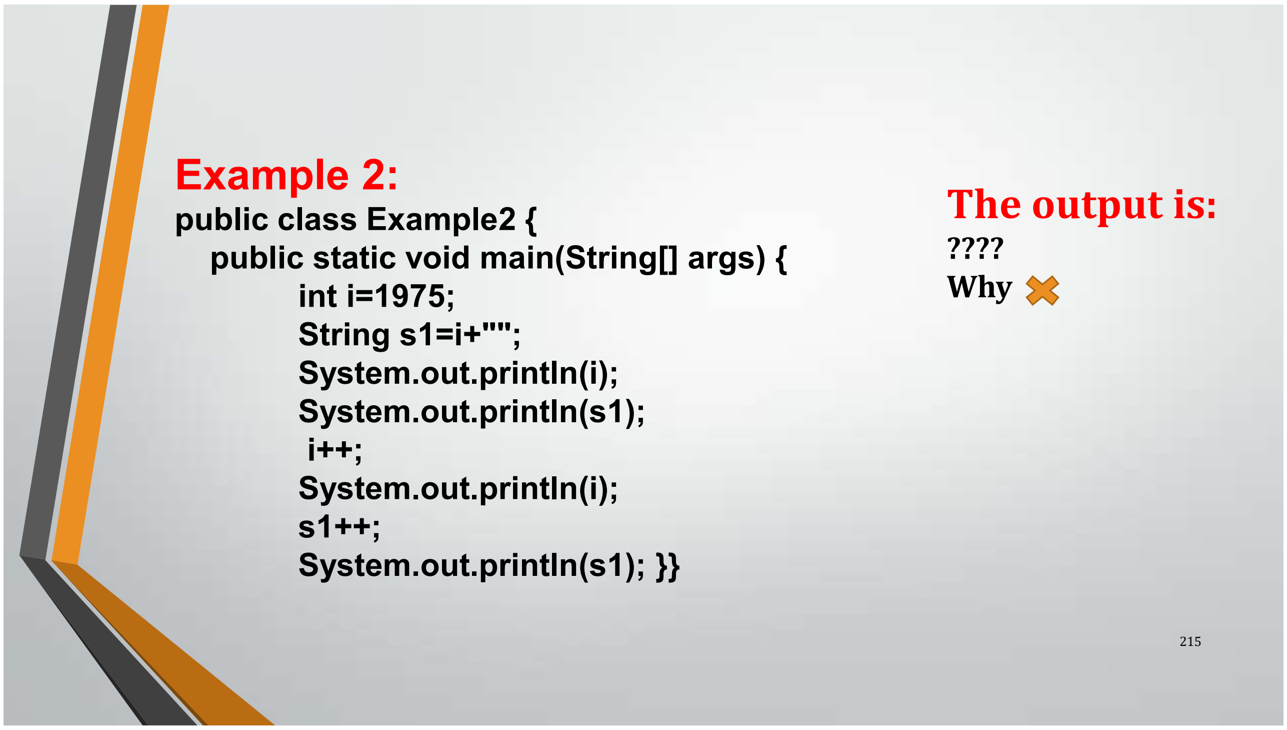
Exampl1:

```
public class Example1 {
   public static void main(String[] args) {
            int i=1975;
            String s1=i+"";
            System.out.println(i);
         System.out.println(s1);  }}
```

The output is:

```
1975
1975
```

[redacted]



Example 2:

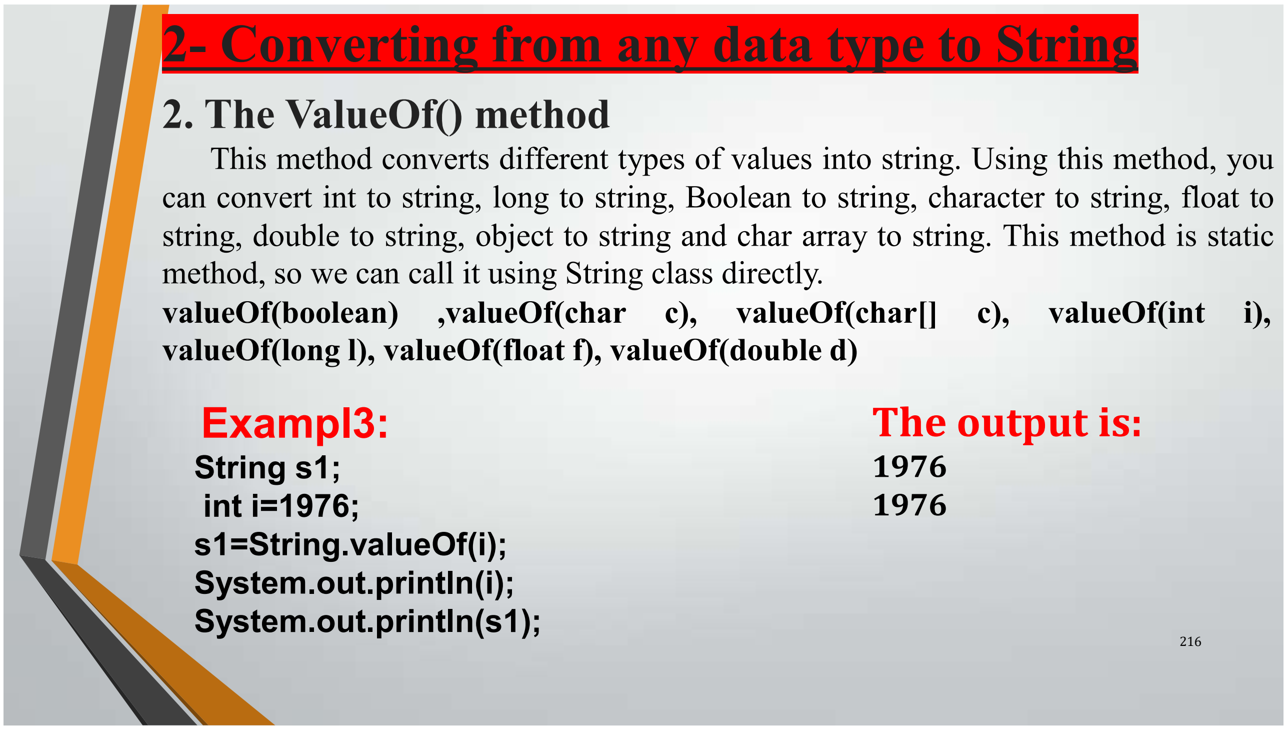
```
public class Example2 {
   public static void main(String[] args) {
            int i=1975;
            String s1=i+"";
            System.out.println(i);
            System.out.println(s1);
             i++;
            System.out.println(i);
            s1++;
            System.out.println(s1); }}
```

The output is:

????

Why

215



2- Converting from any data type to String

2. The ValueOf() method

This method converts different types of values into string. Using this method, you can convert int to string, long to string, Boolean to string, character to string, float to string, double to string, object to string and char array to string. This method is static method, so we can call it using String class directly.

valueOf(boolean) ,valueOf(char c), valueOf(char[] c), valueOf(int i), valueOf(long l), valueOf(float f), valueOf(double d)

Exampl3:

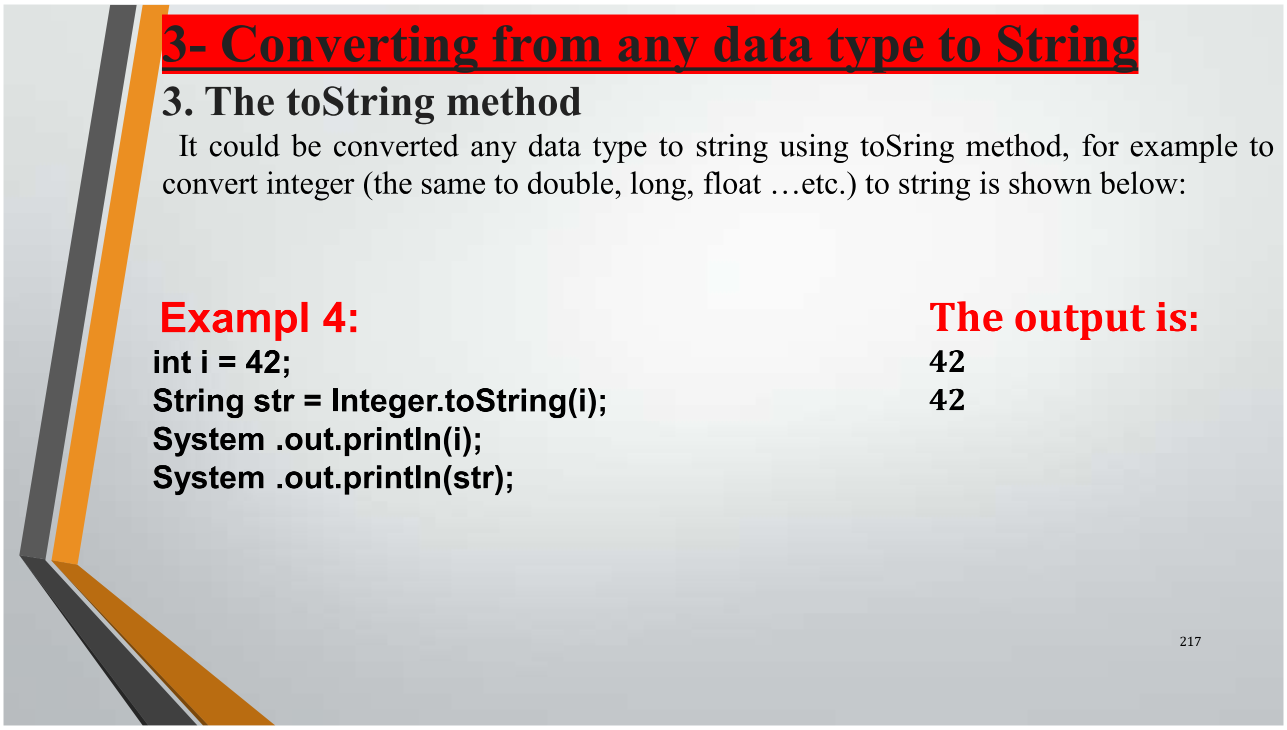
```
String s1;
 int i=1976;
s1=String.valueOf(i);
System.out.println(i);
System.out.println(s1);
```

The output is:

```
1976
1976
```

216

[redacted]



3- Converting from any data type to String

3. The toString method

It could be converted any data type to string using toSring method, for example to convert integer (the same to double, long, float …etc.) to string is shown below:

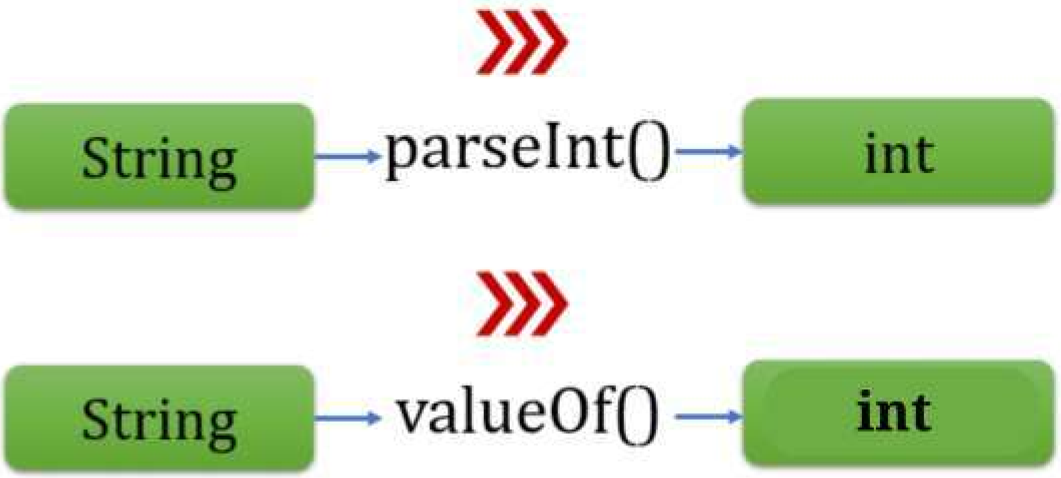
Exampl 4:

```
int i = 42;
String str = Integer.toString(i);
System .out.println(i);
System .out.println(str);
```

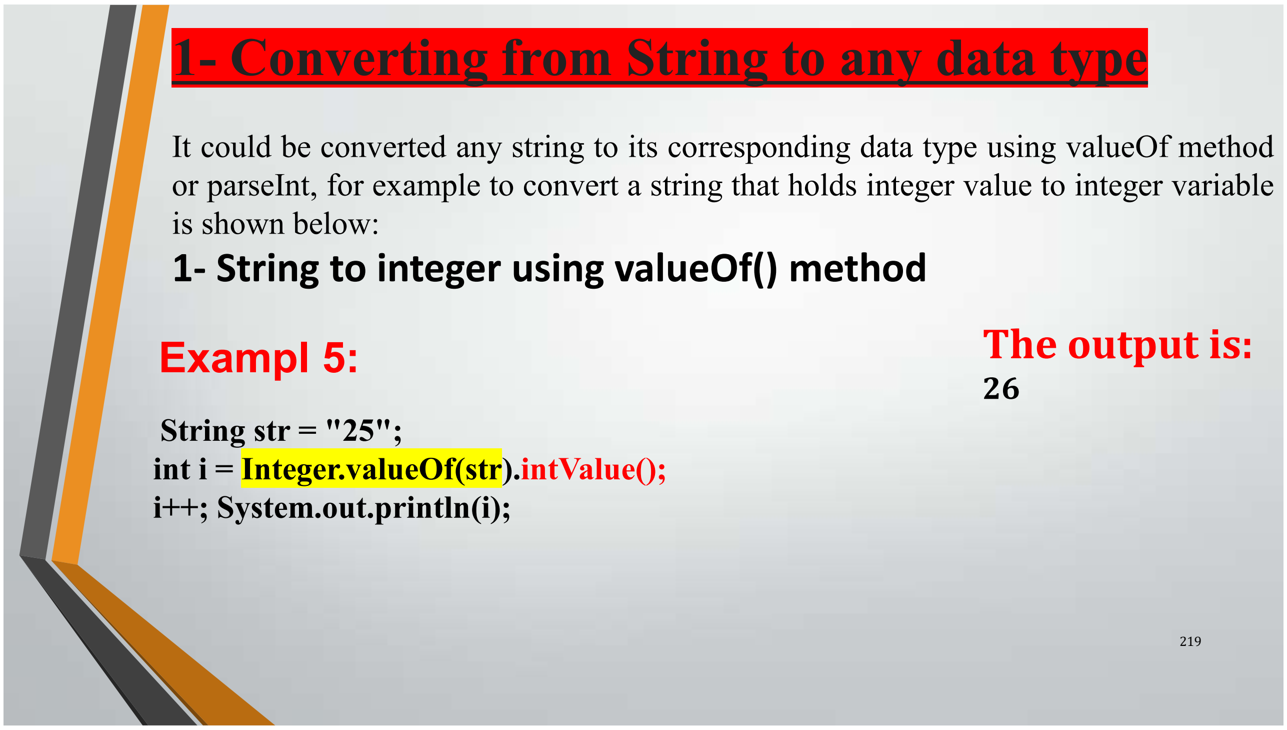
The output is:

```
42
42
```

217



[redacted]



1- Converting from String to any data type

It could be converted any string to its corresponding data type using valueOf method or parseInt, for example to convert a string that holds integer value to integer variable is shown below:

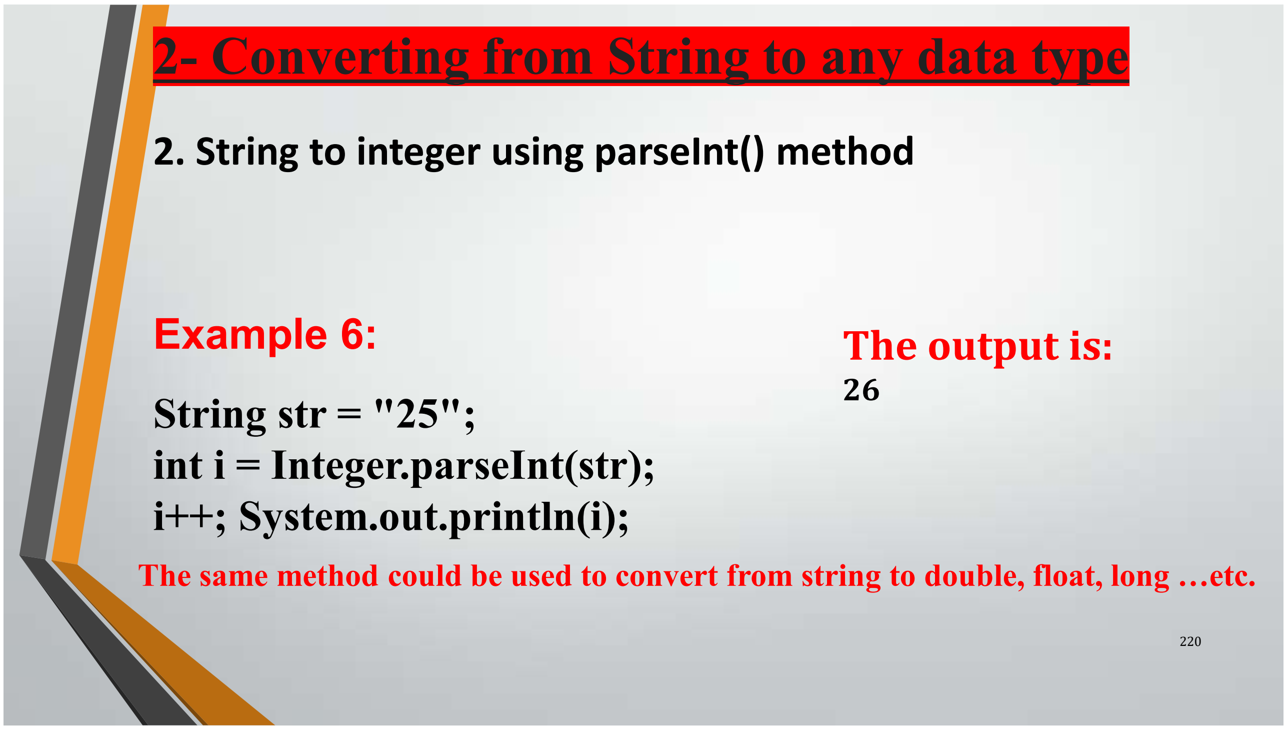
1- String to integer using valueOf() method

Exampl 5:

```
String str = "25";
int i = Integer.valueOf(str).intValue();
i++; System.out.println(i);
```

The output is:

```
26
```



2- Converting from String to any data type

2. String to integer using parseInt() method

Example 6:

```
String str = "25";
int i = Integer.parseInt(str);
i++; System.out.println(i);
```

The output is:

```
26
```

The same method could be used to convert from string to double, float, long …etc.

Example 7

```
String str = "25.8";
int i = Integer.parseInt(str);
i++;  System.out.println(i);
```

The output is:

??????? And Why?

Reading from Keyboard in java

Example 8

```
import java.util.Scanner;

public class Example 8  {
  public static void main (String[] args) {
    ...
    Scanner scanner = new Scanner(System.in);
    String inputString = scanner.nextLine();    or directly  (nextInt();
    ...
    System.out.println(inputString);
    ...
  }
}
```

Now the inputString has an input and you can convert the input string to any data type as explained previously

```
import java.util.Scanner;
public class RectangleArea {
 public static void main(String[] args) {
 int length;
int width;
int area;
Scanner console = new Scanner(System.in);
System.out.print("Enter length ");
length = console.nextInt();
System.out.print("Enter width ");
width = console.nextInt();
area = length * width;
System.out.println("The area of rectangle is " + area); } }
```

Problems

1. Find a number of digits before and after decimal point at any double number.

   e.g. 1234.56

   4 before point and 2 after point

2. Print a list of names that starts with an upper-case letter only.
3. Print a list of names that starts with ‘A’ or ‘a’ letter only.
4. Reversing (عكس) any string , example string s=”abcd”, the output is string “dcba”
5. Consider the following string:

   String hannah = "Did Hannah see bees? Hannah did.";

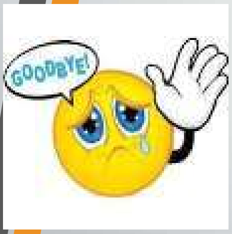
-What is the value displayed by the expression hannah.length()?
-What is the value returned by the method call hannah.charAt(12)?
-Write an expression that refers to the letter b in the string hannah.
-How long is the string returned by the following expression? What is the string?

"Was it a car or a cat I saw?".substring(9, 12)

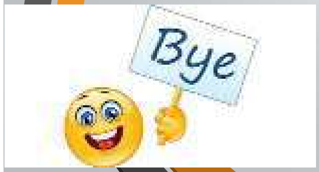
[redacted]

Goodbye

Thank you for listening



Any questions?



225

Inheritance in OOP – Java

- Examples
- Practical Part (trace)
- Writing programs
- Tracing programs

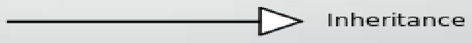
226

[redacted]

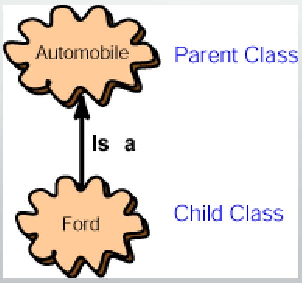
Inheritance def.

Inheritance is a mechanism in which one class acquires (يكتسب) the property of another class. For example, a child inherits the traits (سمات) of his/her parents.

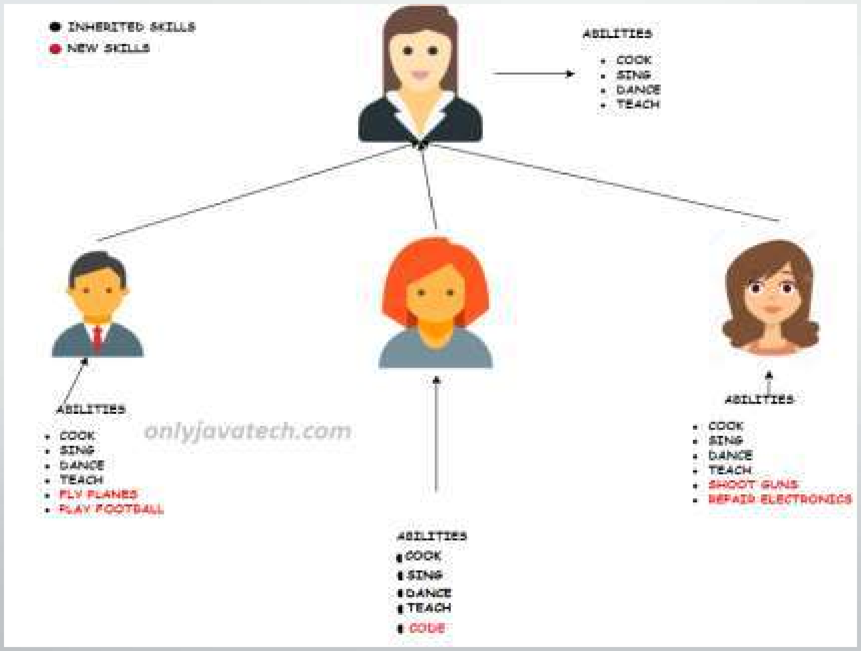
Inheritance is an important pillar (دعامة) and the most powerful mechanisms of OOP. The inheritance mechanism is allowing a class to inherit the features (fields and methods) of another class.

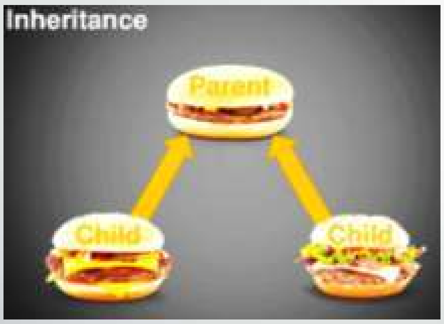


- **Inheritance** represents the **IS-A relationship,** also known as **parent-child relationship.**
- It allows the **reuse** of the members of a class (called the superclass or the mother class) in another class (called subclass, child class or the derived class) that inherits from it. Below three visual examples of inheritance from Real World.



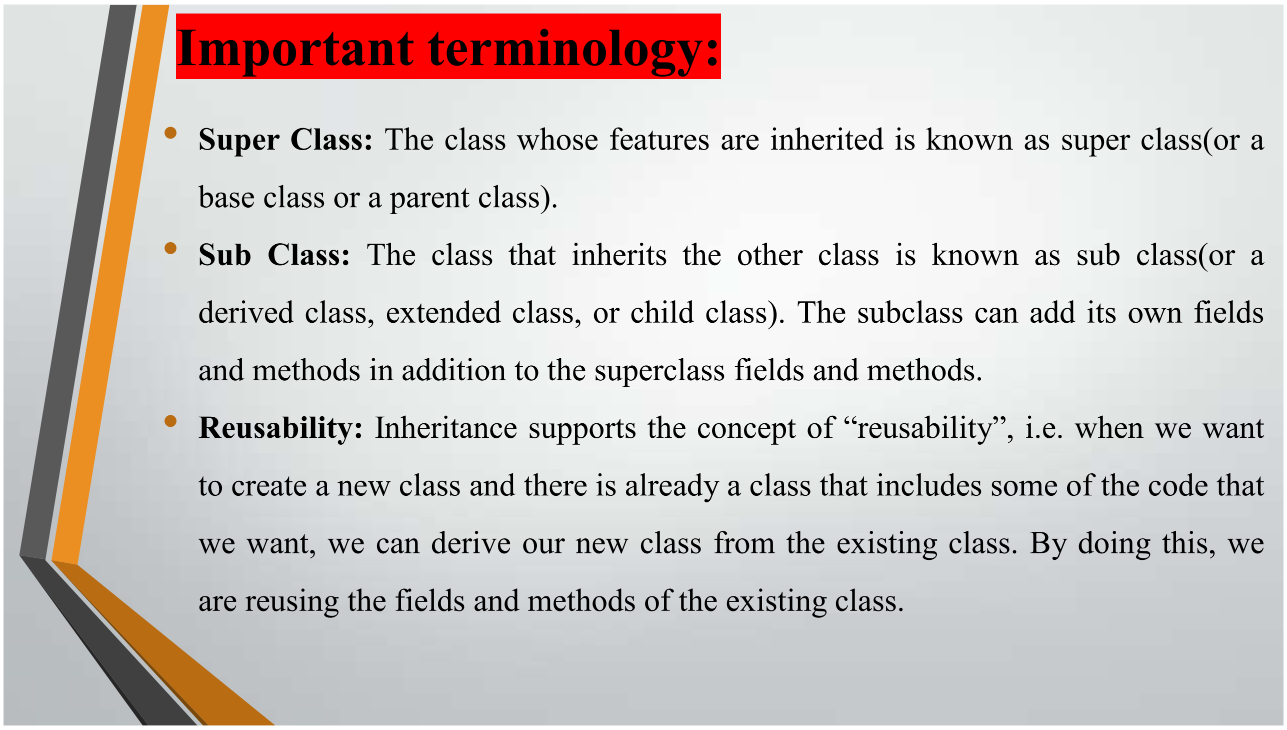
[redacted]


● INHERITED SKILLS
● NEW SKILLS
ABILITIES
• COOK
• SING
• DANCE
• TEACH
ABILITIES
• COOK
• SING
• DANCE
• TEACH
• FLY PLANES
• PLAY FOOTBALL
onlyjavatech.com
ABILITIES
• COOK
• SING
• DANCE
• TEACH
• CODE
ABILITIES
• COOK
• SING
• DANCE
• TEACH
• SHOOT GUNS
• REPAIR ELECTRONICS


Inheritance
Parent
Child
Child

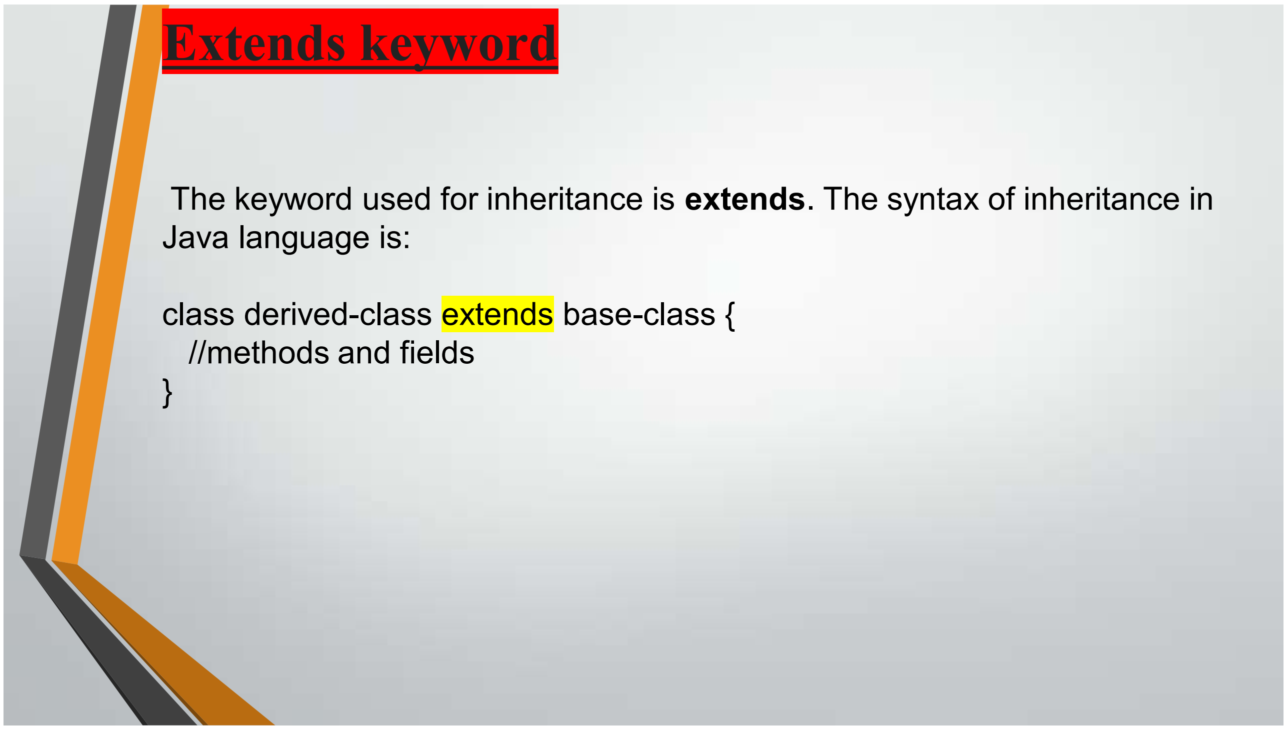
Inheritance
Phone
HTC phone
Microsoft phone
Samsung phone

[redacted]



Important terminology:

- **Super Class:** The class whose features are inherited is known as super class(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

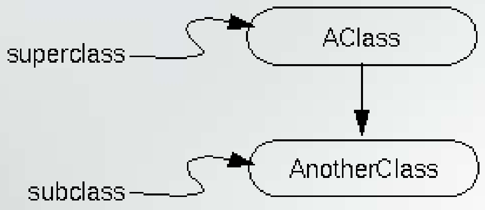


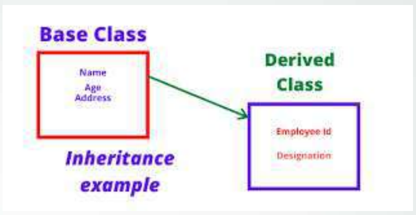
Extends keyword

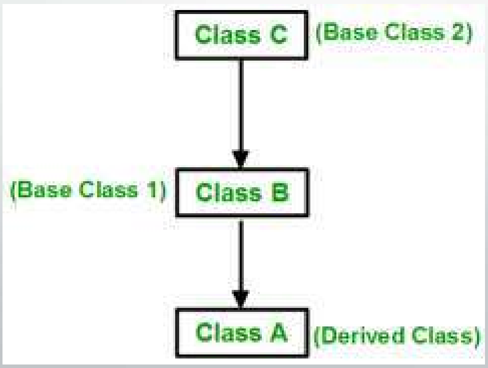
The keyword used for inheritance is **extends**. The syntax of inheritance in Java language is:

```
class derived-class extends base-class {
   //methods and fields
}
```

[redacted]

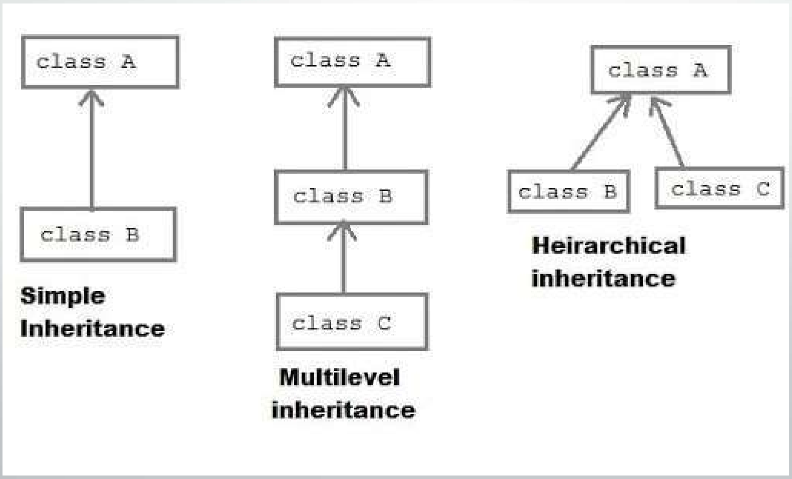




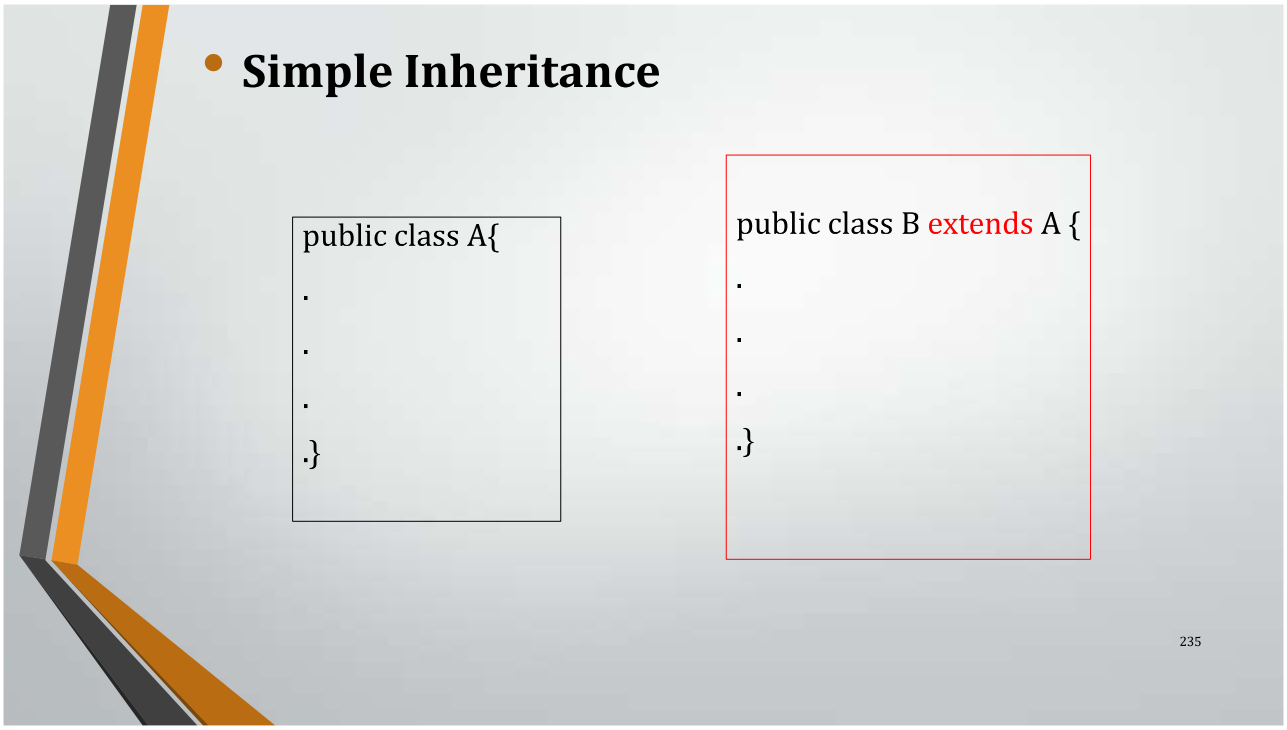


Example1

- How to write a Java structure for the following classes hierarchy



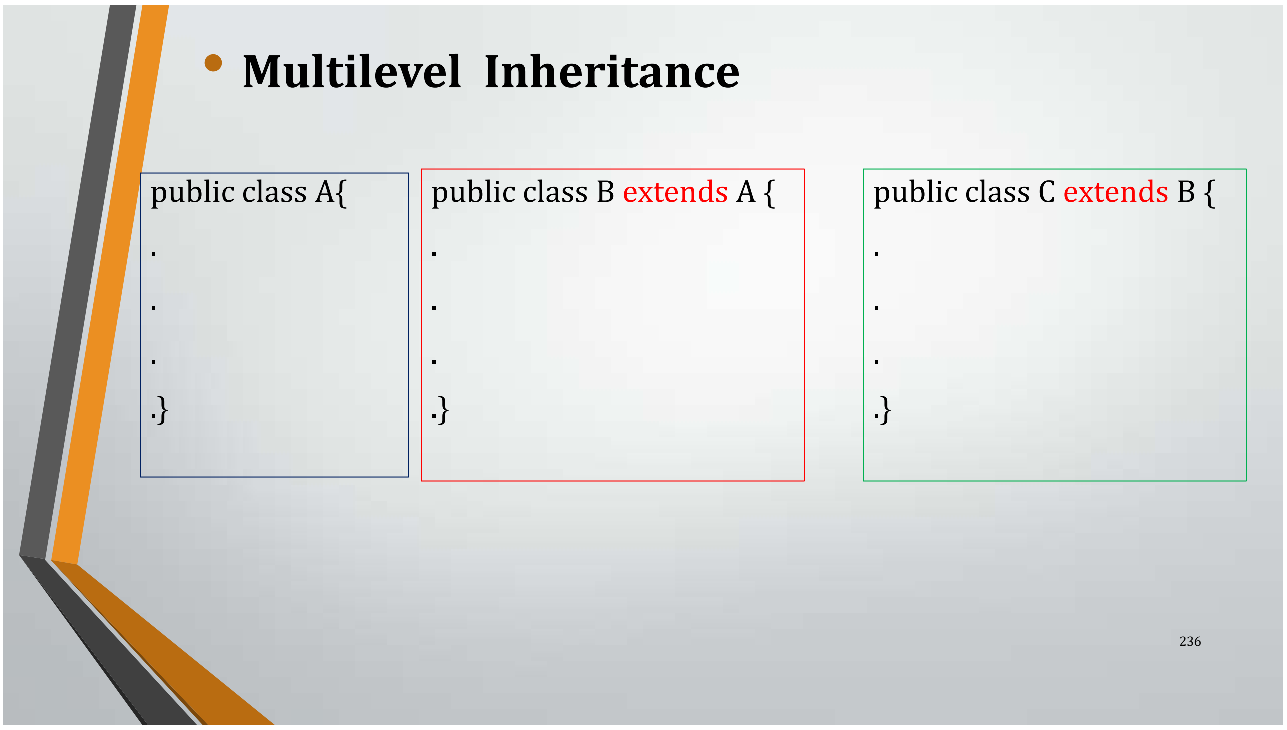
[redacted]



Simple Inheritance

```
public class A{
.
.
.
.}
```

```
public class B extends A {
.
.
.
.}
```



Multilevel Inheritance

```
public class A{
.
.
.
.}
```

```
public class B extends A {
.
.
.
.}
```

```
public class C extends B {
.
.
.
.}
```

• Hierarchal Inheritance

```
public class A{
.
.
.
.}
```

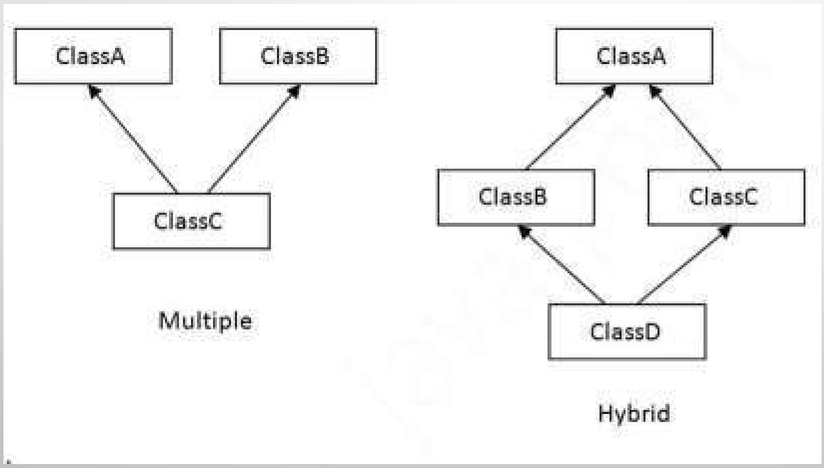
```
public class B extends A {
.
.
.
.}
```

```
public class C extends A {
.
.
.
.}
```

237

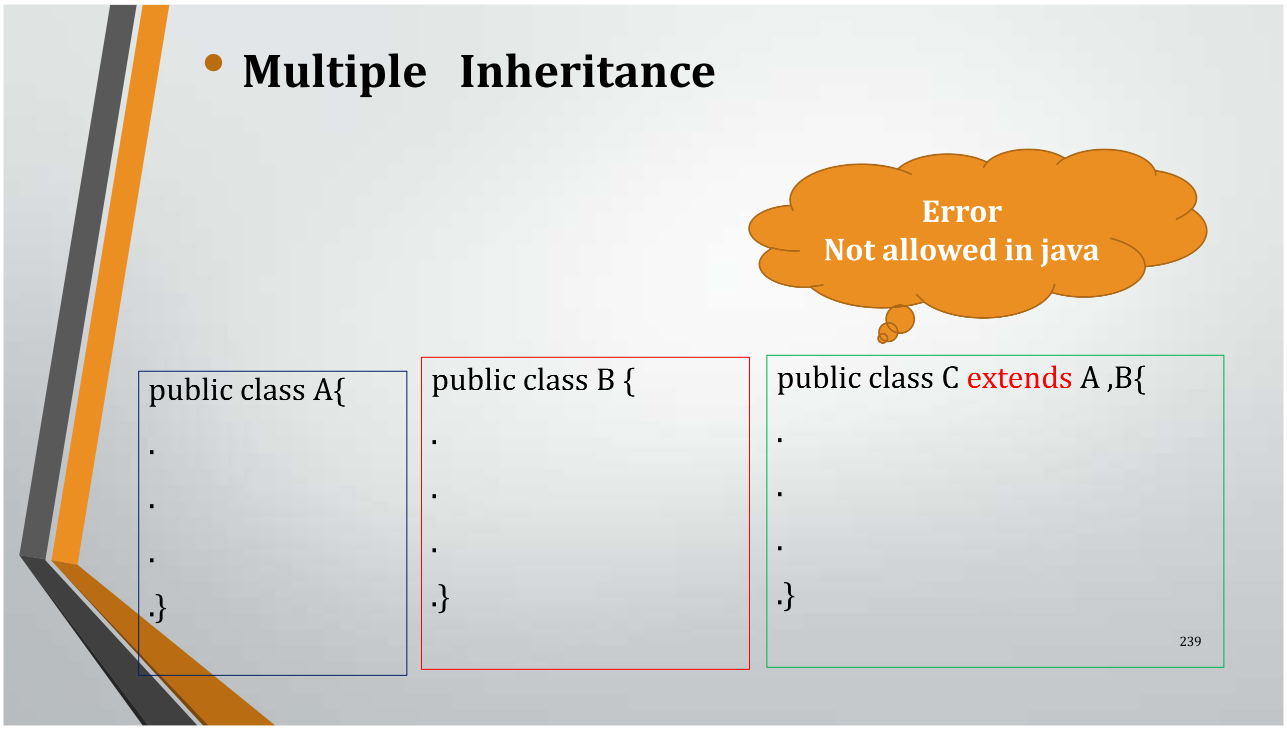
• Example2

• Define a java structure for the following two figures



238

[redacted]



Multiple Inheritance

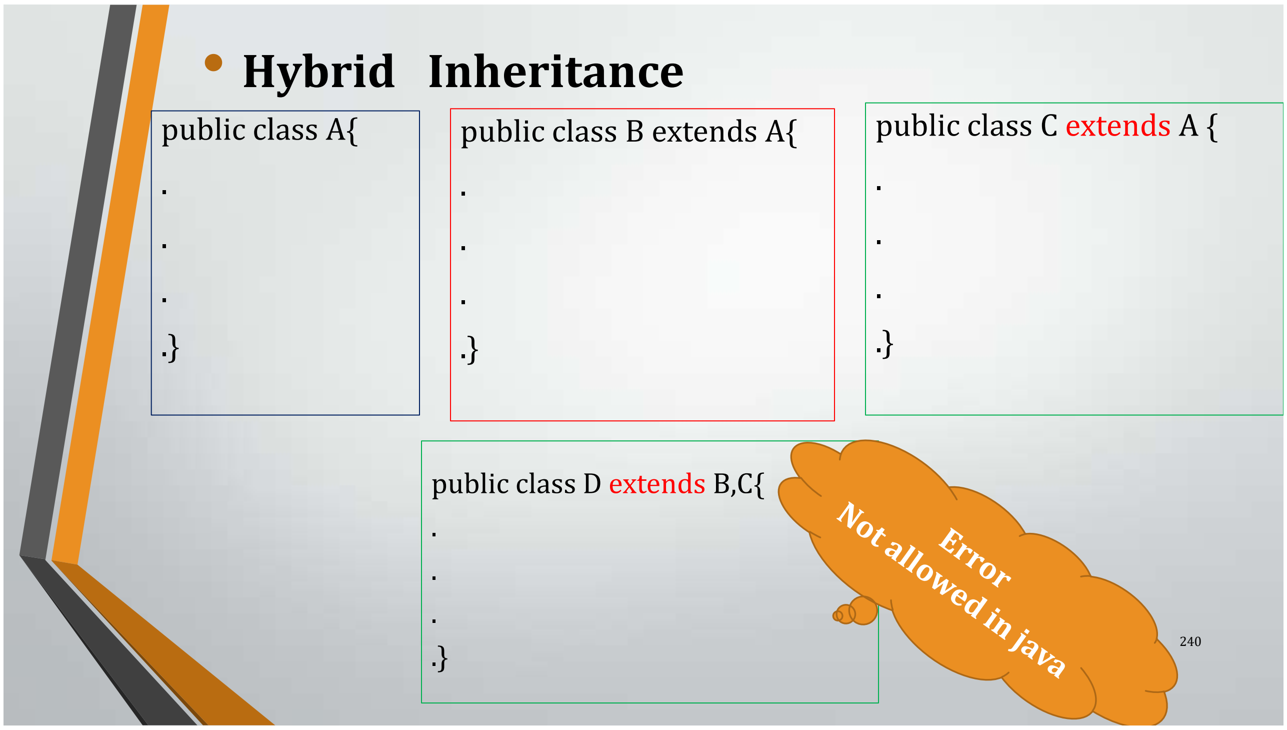
Error
Not allowed in java

```
public class A{
.
.
.
.}
```

```
public class B {
.
.
.
.}
```

```
public class C extends A ,B{
.
.
.
.}
```

239



Hybrid Inheritance

```
public class A{
.
.
.
.}
```

```
public class B extends A{
.
.
.
.}
```

```
public class C extends A {
.
.
.
.}
```

```
public class D extends B,C{
.
.
.
.}
```

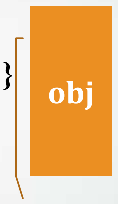
Error
Not allowed in java

240

[redacted]

Example3 Trace

```
public class Teacher {
 public   String designation = "Teacher";
  public String collegeName = "Beginners book";
  public void does(){  System.out.println("Teaching");  } }
public class PhysicsTeacher extends Teacher{
 public String mainSubject = "Physics";}
```



designation	collegeName	mainSubject	does()
Teacher	Beginners book	Physics	

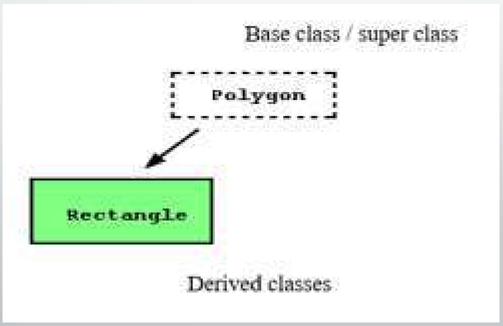
```
public class Main{
   public static void main(String args[]){
      PhysicsTeacher obj = new PhysicsTeacher();
      System.out.println(obj.collegeName);
      System.out.println(obj.designation);
      System.out.println(obj.mainSubject);
        obj.does(); }}
```

Output:

```
Beginners book
Teacher
Physics
Teaching
```

Example1 (Writing a program)

Define a base class called Polygon which has two integer attributes represent width and height of a Polygon. Set method is used for setting the width and the height. Derive one subclass Rectangle which inherits all members from Polygon and add a new method area that used for calculating rectangle area. Write a main class to create 1 rectangle object and use it to print its area.



[redacted]

```
public class Polygon {
  protected int width, height;
  public void set_values (int a, int b)
    { width=a; height=b;}  }

public class Rectangle extends Polygon{
  public   int area (){
    return (width * height); }  }

 public class Main {
     public static void main(String[] args) {
      Rectangle rect=new Rectangle();
      rect.set_values(4,5);
   System.out.println(rect.area());  }}
```

The output :
20

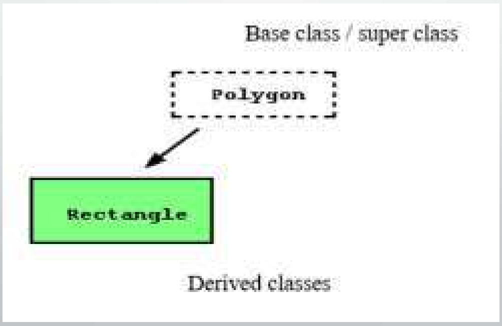
Screen

Object	width	height	area set
rect	4	5	

memory

Example2 (Writing a program):

Define a base class called Polygon which has two integer attributes represent width and height of a Polygon. Set method is used for setting the width and the height. Derive one subclass Rectangle which inherits all members from Polygon and add a new method area that used for calculating rectangle area. Write a main class to create 3 rectangles and print the areas of these rectangles.



[redacted]

```
public class Polygon {
  protected int width, height;
  public void set_values (int a, int b)
    { width=a; height=b;}  }
public class Rectangle extends Polygon{
  public   int area (){  return (width * height); }  }
 public class Main {
     public static void main(String[] args) {
      Rectangle rect1=new Rectangle(); Rectangle rect2=new Rectangle();
      Rectangle rect3=new Rectangle();
      rect1.set_values(4,5);      rect2.set_values(6,7);     rect3.set_values(8,9);
   System.out.println(rect1.area());  System.out.println(rect2.area());
   System.out.println(rect3.area());  }}
```

The output :

```
20
42
72
```

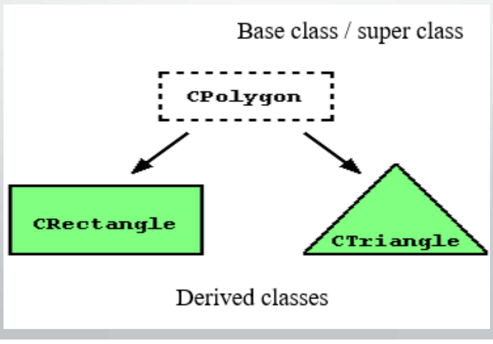
Screen

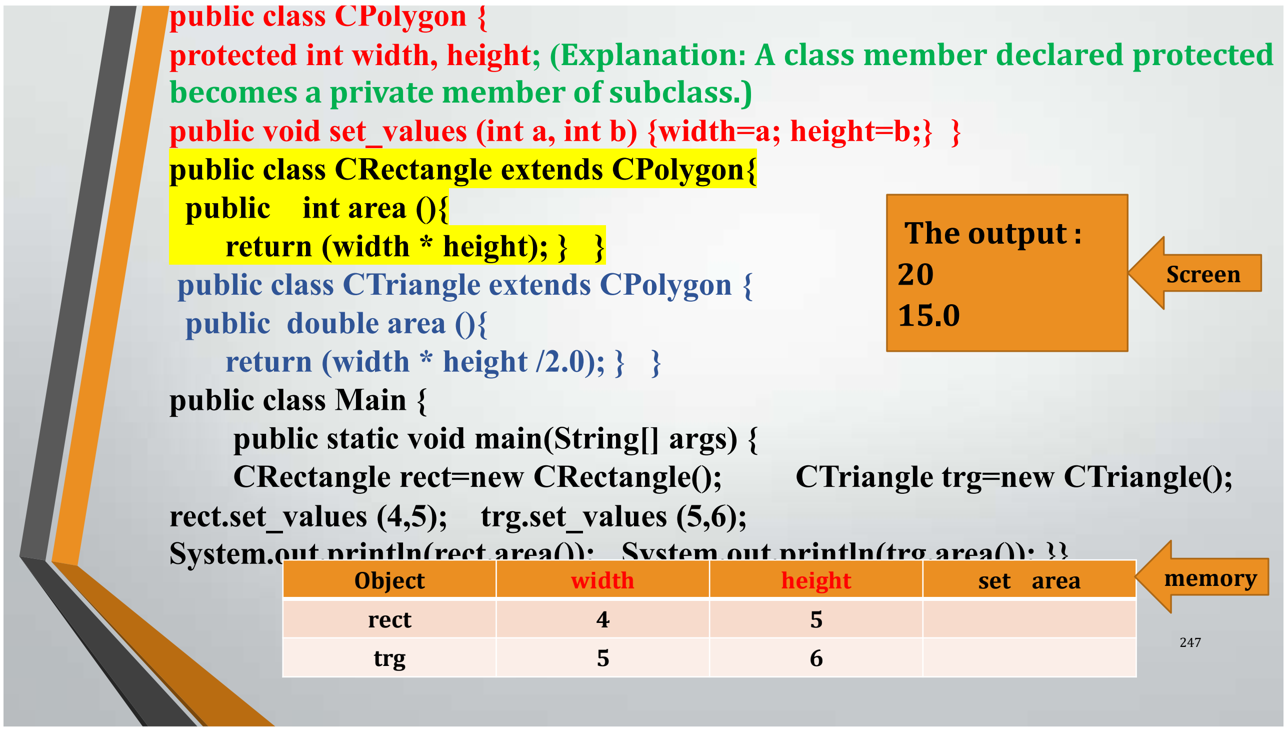
Object	width	height	area set
rect1	4	5	
rect2	6	7	
rect3	8	9	

memory

Example3 (Writing a program)

We are going to suppose that we want to declare a series of classes that describe polygons like our CRectangle, or CTriangle. They have certain common features, such as both can be described by means of only two sides: height and width. This could be represented in the world of classes with a class CPolygon from which we would derive the two referred ones, CRectangle and CTriangle. Use these three classes to define two objects: rectangle and triangle and print their areas.





```
public class CPolygon {
protected int width, height; (Explanation: A class member declared protected becomes a private member of subclass.)
public void set_values (int a, int b) {width=a; height=b;}  }
public class CRectangle extends CPolygon{
  public    int area (){
       return (width * height); }   }
 public class CTriangle extends CPolygon {
  public  double area (){
       return (width * height /2.0); }   }
public class Main {
        public static void main(String[] args) {
        CRectangle rect=new CRectangle();        CTriangle trg=new CTriangle();
rect.set_values (4,5);     trg.set_values (5,6);
System.out.println(rect.area());   System.out.println(trg.area()); }}
```

The output :

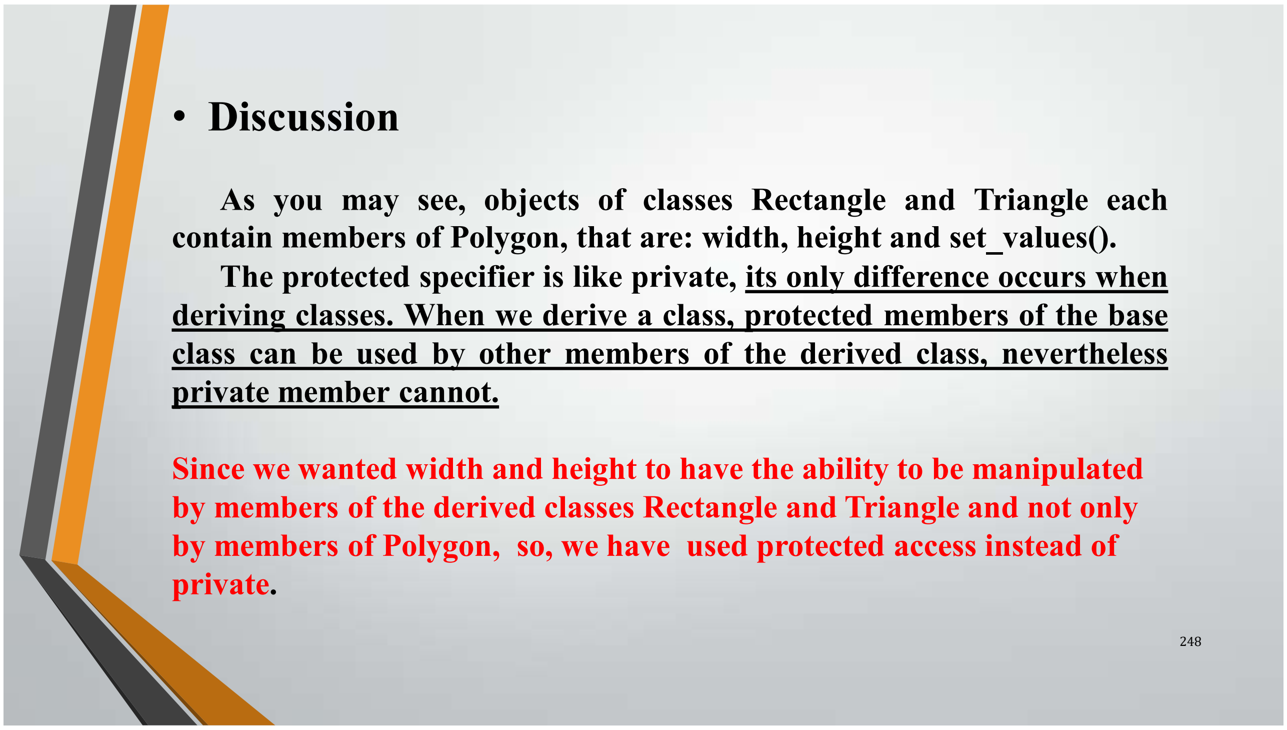
```
20
15.0
```

Screen

Object	width	height	set area
rect	4	5	
trg	5	6	

memory

247



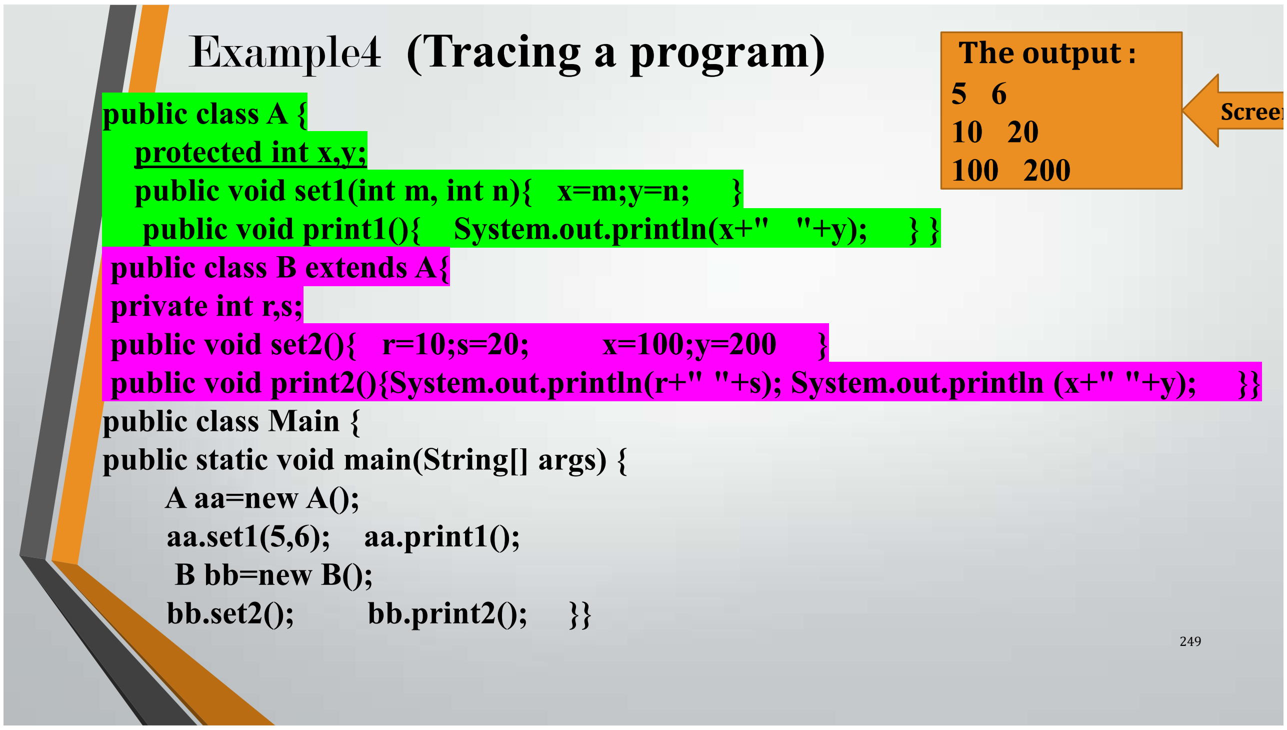
- **Discussion**

As you may see, objects of classes Rectangle and Triangle each contain members of Polygon, that are: width, height and set_values().

The protected specifier is like private, <u>its only difference occurs when deriving classes. When we derive a class, protected members of the base class can be used by other members of the derived class, nevertheless private member cannot.</u>

Since we wanted width and height to have the ability to be manipulated by members of the derived classes Rectangle and Triangle and not only by members of Polygon, so, we have used protected access instead of private.

248



Example4 (Tracing a program)

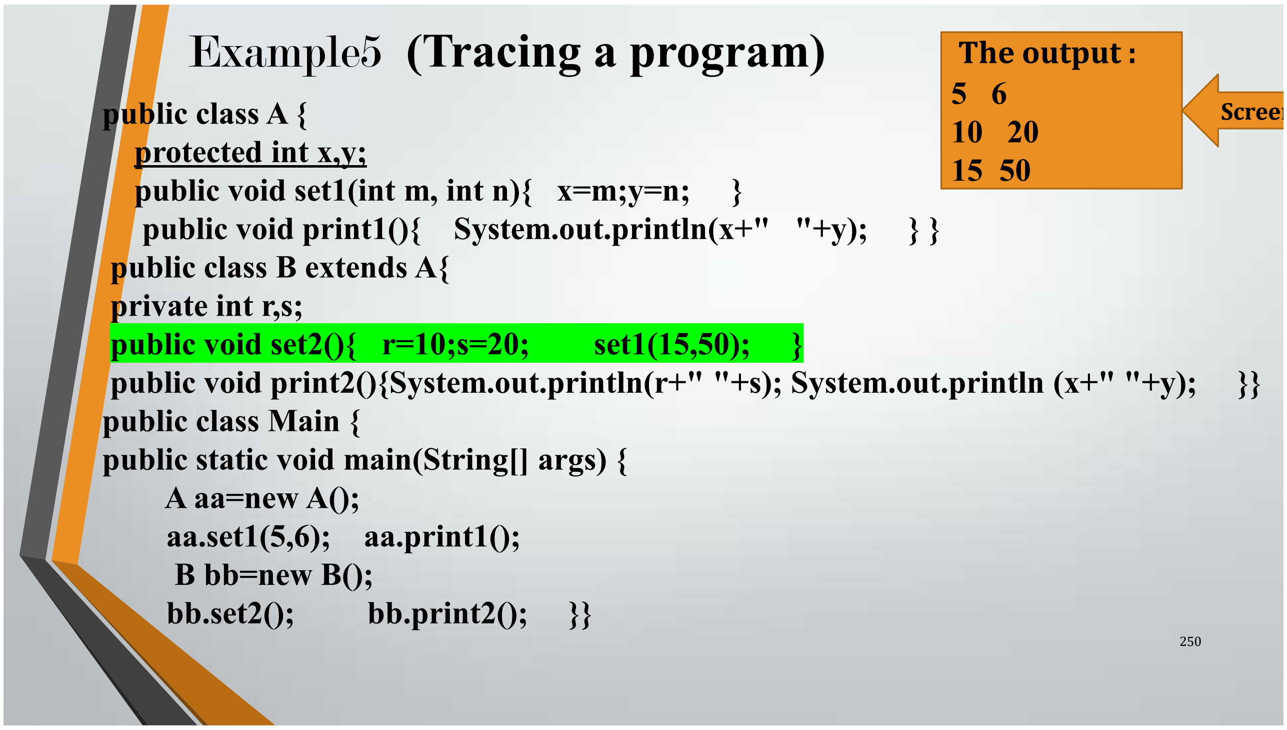
The output :

```
5   6
10   20
100   200
```

Scree

```
public class A {
   protected int x,y;
   public void set1(int m, int n){   x=m;y=n;     }
    public void print1(){    System.out.println(x+"    "+y);     } }
 public class B extends A{
 private int r,s;
 public void set2(){   r=10;s=20;          x=100;y=200     }
 public void print2(){System.out.println(r+" "+s); System.out.println (x+" "+y);     }}
public class Main {
public static void main(String[] args) {
        A aa=new A();
        aa.set1(5,6);     aa.print1();
         B bb=new B();
        bb.set2();         bb.print2();     }}
```

249



Example5 (Tracing a program)

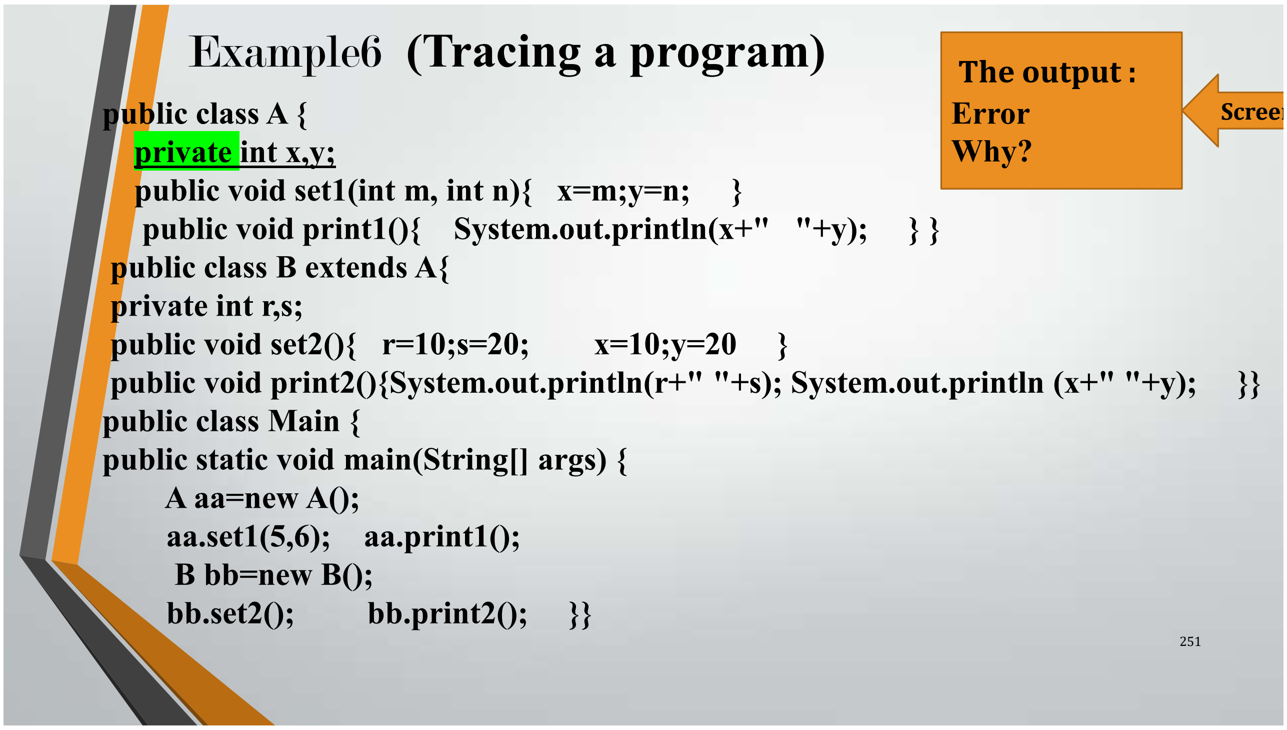
The output :

```
5   6
10   20
15  50
```

Scree

```
public class A {
   protected int x,y;
   public void set1(int m, int n){   x=m;y=n;     }
    public void print1(){    System.out.println(x+"    "+y);     } }
 public class B extends A{
 private int r,s;
 public void set2(){   r=10;s=20;        set1(15,50);     }
 public void print2(){System.out.println(r+" "+s); System.out.println (x+" "+y);     }}
public class Main {
public static void main(String[] args) {
        A aa=new A();
        aa.set1(5,6);     aa.print1();
         B bb=new B();
        bb.set2();         bb.print2();     }}
```

250



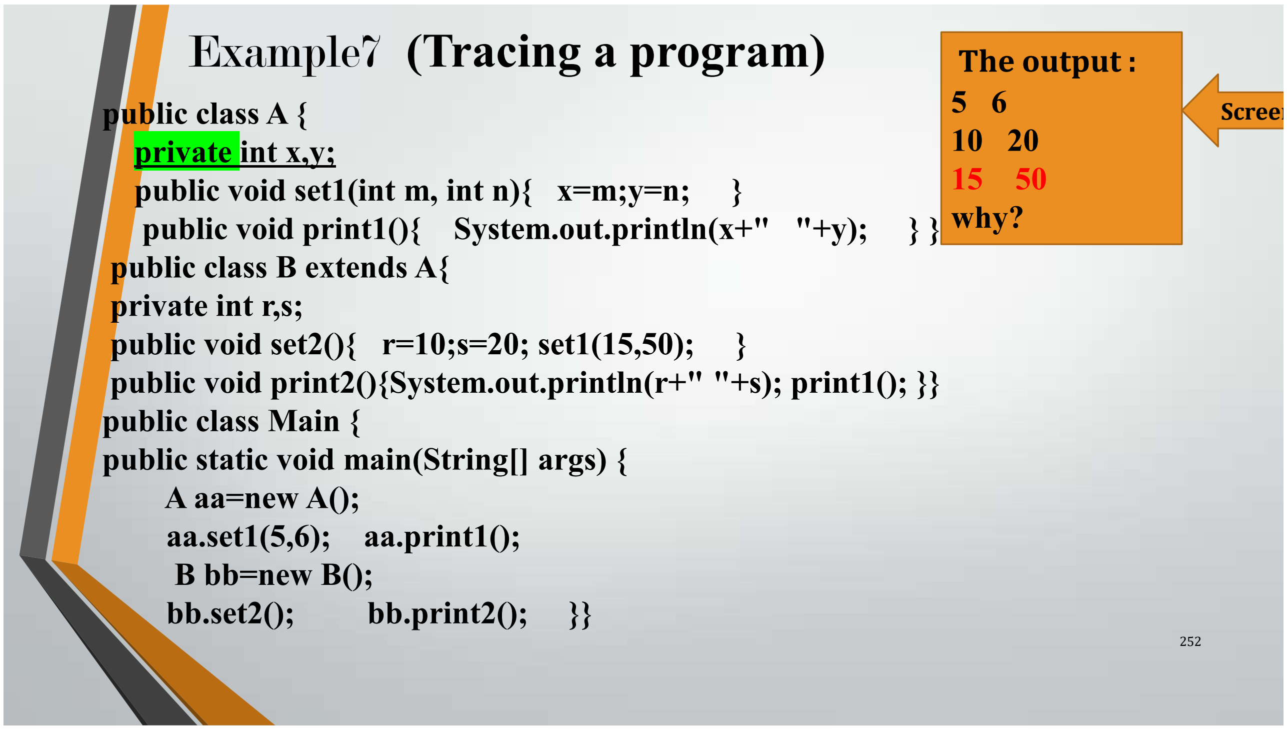
Example6 (Tracing a program)

```
public class A {
  private int x,y;
  public void set1(int m, int n){   x=m;y=n;     }
   public void print1(){    System.out.println(x+"    "+y);     } }
 public class B extends A{
 private int r,s;
 public void set2(){   r=10;s=20;        x=10;y=20     }
 public void print2(){System.out.println(r+" "+s); System.out.println (x+" "+y);     }}
public class Main {
public static void main(String[] args) {
        A aa=new A();
        aa.set1(5,6);     aa.print1();
         B bb=new B();
        bb.set2();        bb.print2();     }}
```

The output :
Error
Why?

Scree

251



Example7 (Tracing a program)

```
public class A {
  private int x,y;
  public void set1(int m, int n){   x=m;y=n;     }
   public void print1(){    System.out.println(x+"    "+y);     } }
 public class B extends A{
 private int r,s;
 public void set2(){   r=10;s=20; set1(15,50);     }
 public void print2(){System.out.println(r+" "+s); print1(); }}
public class Main {
public static void main(String[] args) {
        A aa=new A();
        aa.set1(5,6);     aa.print1();
         B bb=new B();
        bb.set2();        bb.print2();     }}
```

The output :

```
5   6
10   20
15    50
```

why?

Scree

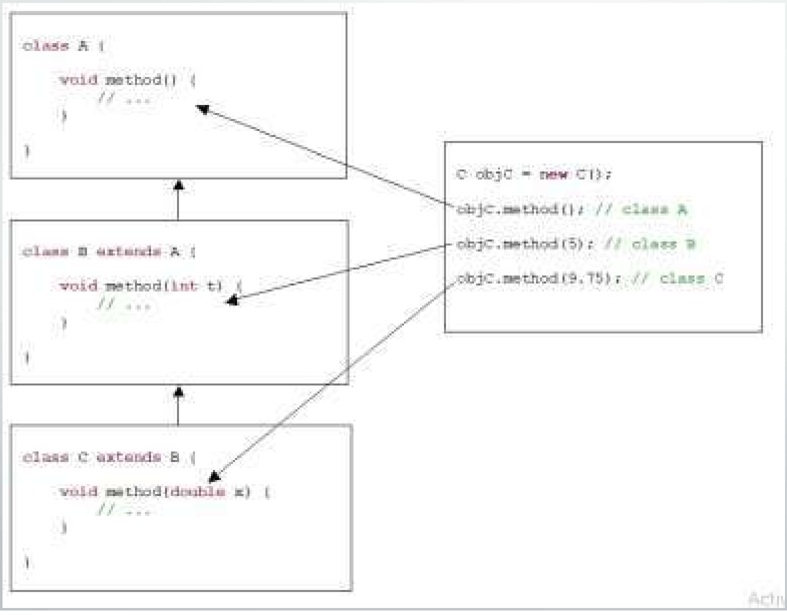
252

Inheritance in OOP – Java Cont. (overloading)

- Overloading methods in subclasses
- Examples (tracing and writing programs)

Overloading methods in subclasses

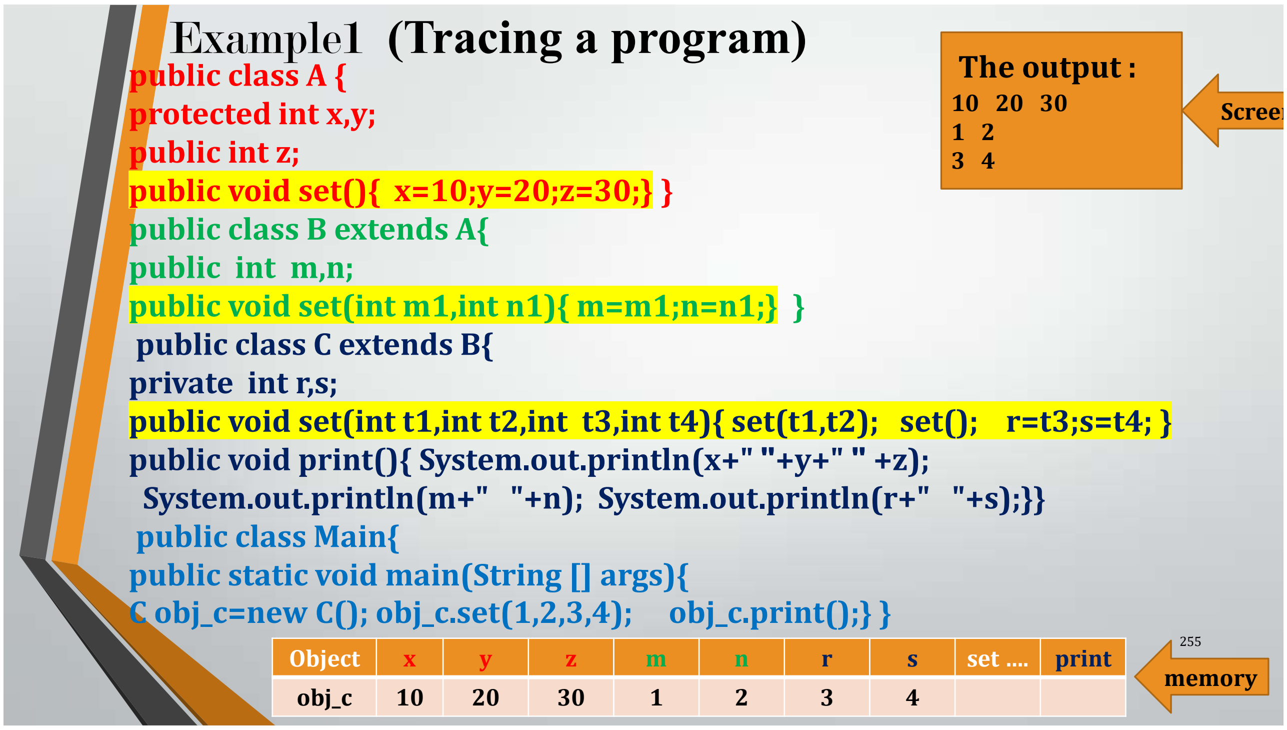
The methods of the super class could be overloaded in the sub class…



```
class A {
    void method() {
        // ...
    }
}

class B extends A {
    void method(int t) {
        // ...
    }
}

class C extends B {
    void method(double x) {
        // ...
    }
}
```

```
C objC = new C();
objC.method(); // class A
objC.method(5); // class B
objC.method(9.75); // class C
```

254



Example1 (Tracing a program)

```
public class A {
protected int x,y;
public int z;
public void set(){  x=10;y=20;z=30;} }
public class B extends A{
public  int  m,n;
public void set(int m1,int n1){ m=m1;n=n1;}  }
 public class C extends B{
private  int r,s;
public void set(int t1,int t2,int  t3,int t4){ set(t1,t2);   set();    r=t3;s=t4; }
public void print(){ System.out.println(x+" "+y+" " +z);
  System.out.println(m+"   "+n);  System.out.println(r+"   "+s);}}
 public class Main{
public static void main(String [] args){
C obj_c=new C(); obj_c.set(1,2,3,4);     obj_c.print();} }
```

The output :

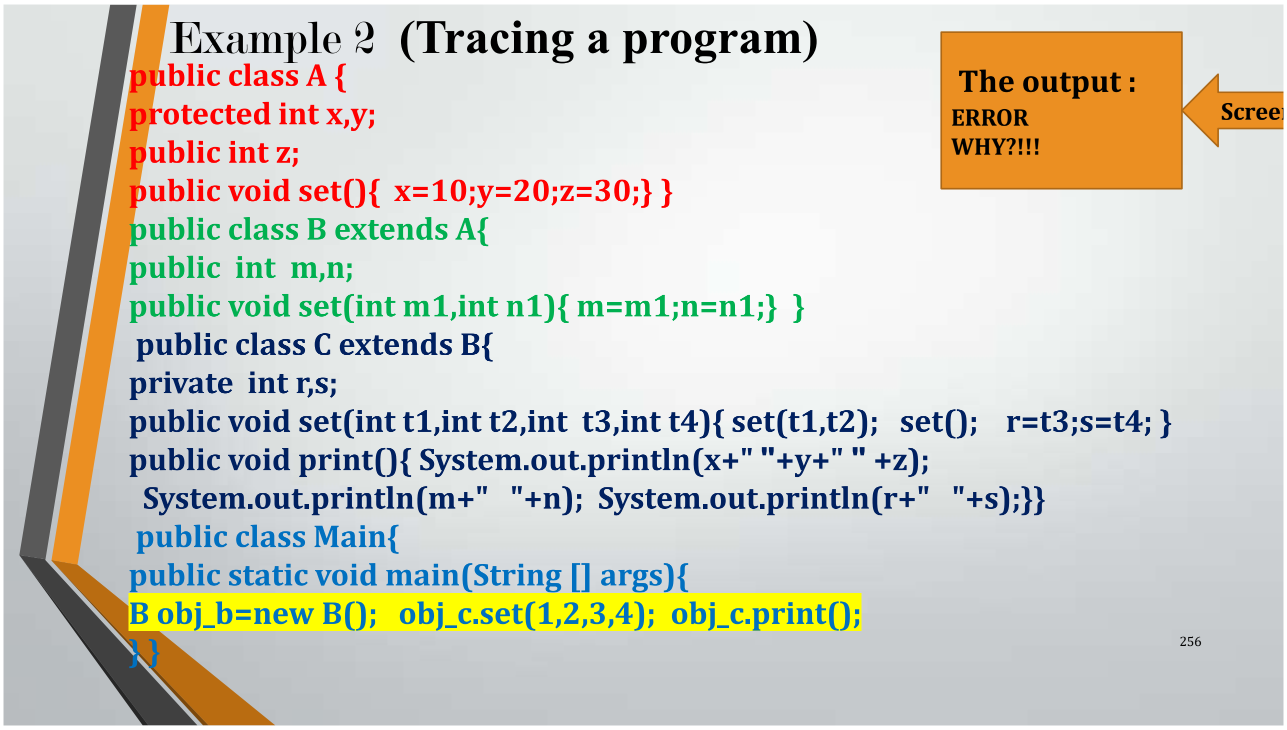
```
10  20  30
1  2
3  4
```

Scree

255

Object	x	y	z	m	n	r	s	set	print
obj_c	10	20	30	1	2	3	4		

memory



Example 2 (Tracing a program)

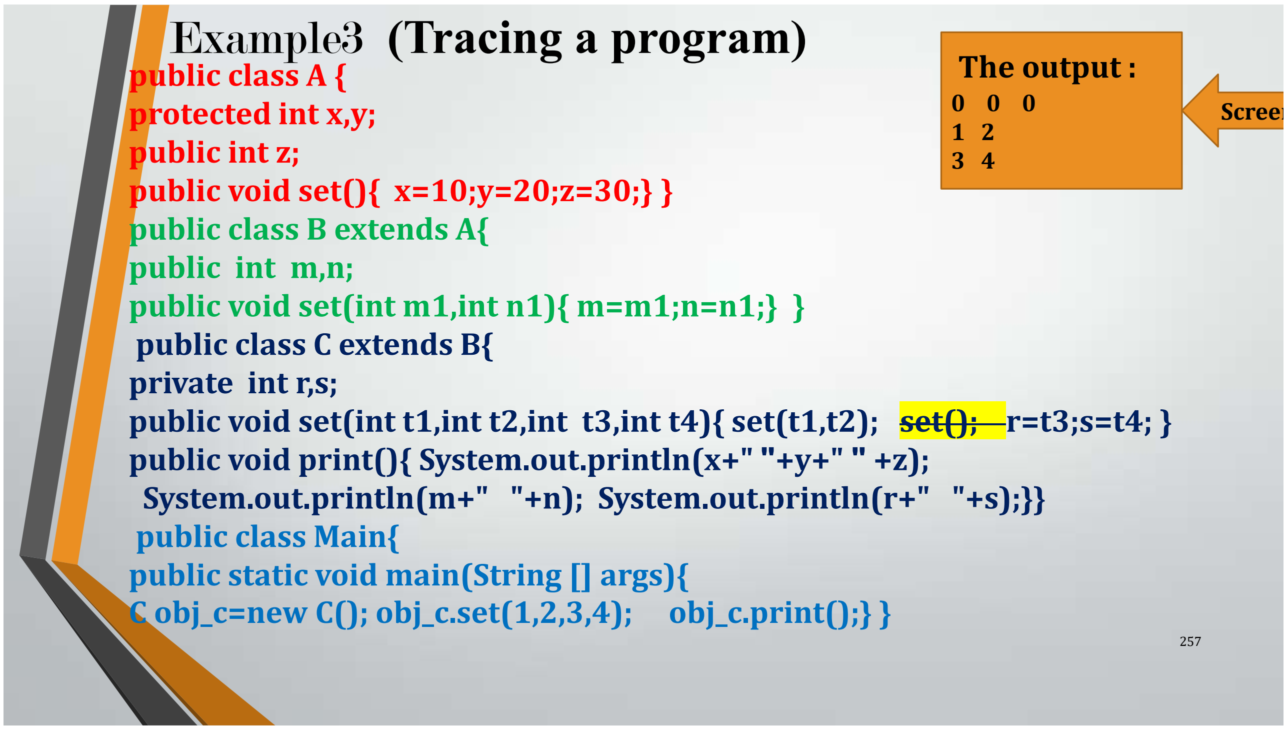
```
public class A {
protected int x,y;
public int z;
public void set(){  x=10;y=20;z=30;} }
public class B extends A{
public  int  m,n;
public void set(int m1,int n1){ m=m1;n=n1;}  }
 public class C extends B{
private  int r,s;
public void set(int t1,int t2,int  t3,int t4){ set(t1,t2);   set();    r=t3;s=t4; }
public void print(){ System.out.println(x+" "+y+" " +z);
  System.out.println(m+"   "+n);  System.out.println(r+"   "+s);}}
 public class Main{
public static void main(String [] args){
B obj_b=new B();   obj_c.set(1,2,3,4);  obj_c.print();
} }
```

The output :

ERROR
WHY?!!!

Scree

256



Example3 (Tracing a program)

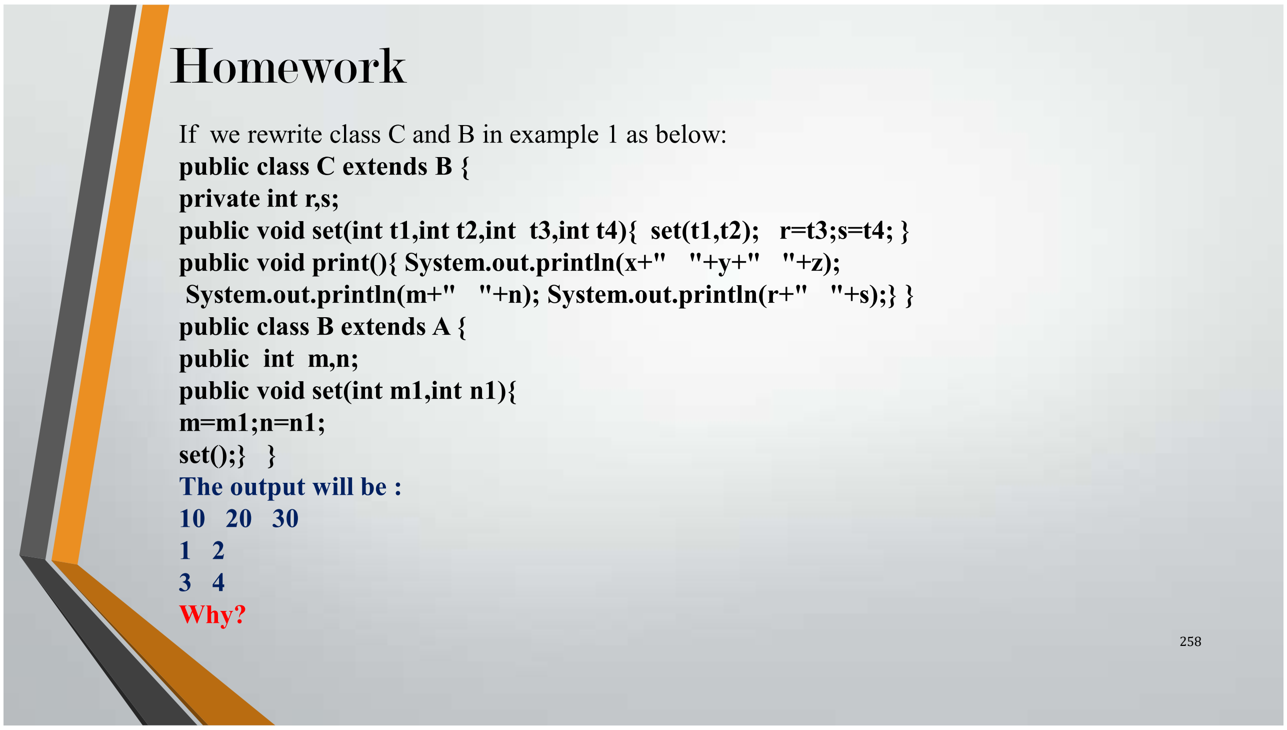
```
public class A {
protected int x,y;
public int z;
public void set(){ x=10;y=20;z=30;} }
public class B extends A{
public int m,n;
public void set(int m1,int n1){ m=m1;n=n1;} }
 public class C extends B{
private int r,s;
public void set(int t1,int t2,int t3,int t4){ set(t1,t2); ~~set();~~ r=t3;s=t4; }
public void print(){ System.out.println(x+" "+y+" " +z);
  System.out.println(m+"   "+n); System.out.println(r+"   "+s);}}
 public class Main{
public static void main(String [] args){
C obj_c=new C(); obj_c.set(1,2,3,4);    obj_c.print();} }
```

The output :

```
0   0   0
1  2
3  4
```

Scree

257



Homework

If we rewrite class C and B in example 1 as below:

```
public class C extends B {
private int r,s;
public void set(int t1,int t2,int t3,int t4){ set(t1,t2);  r=t3;s=t4; }
public void print(){ System.out.println(x+"   "+y+"   "+z);
 System.out.println(m+"   "+n); System.out.println(r+"   "+s);} }
public class B extends A {
public int m,n;
public void set(int m1,int n1){
m=m1;n=n1;
set();}  }
```

The output will be :

```
10   20   30
1   2
3   4
```

Why?

258

Example4 (Writing a program):

Define a base class called Person which has two string attributes represent name and gender of a Person. Set method is used for setting the name and gender for a Person. Derive one subclass called Student which inherits all members from Person and add two new methods: the first method is called set for setting three marks while the second method called average which is used for printing average. Write a main class to create two students Yazan and Nour and print their averages and details.

```
public class Person {
  protected String name;
  protected String gender;
public void set(String nm,String gn)
   name=new String(nm);
   gender=new String(gn);} }
```

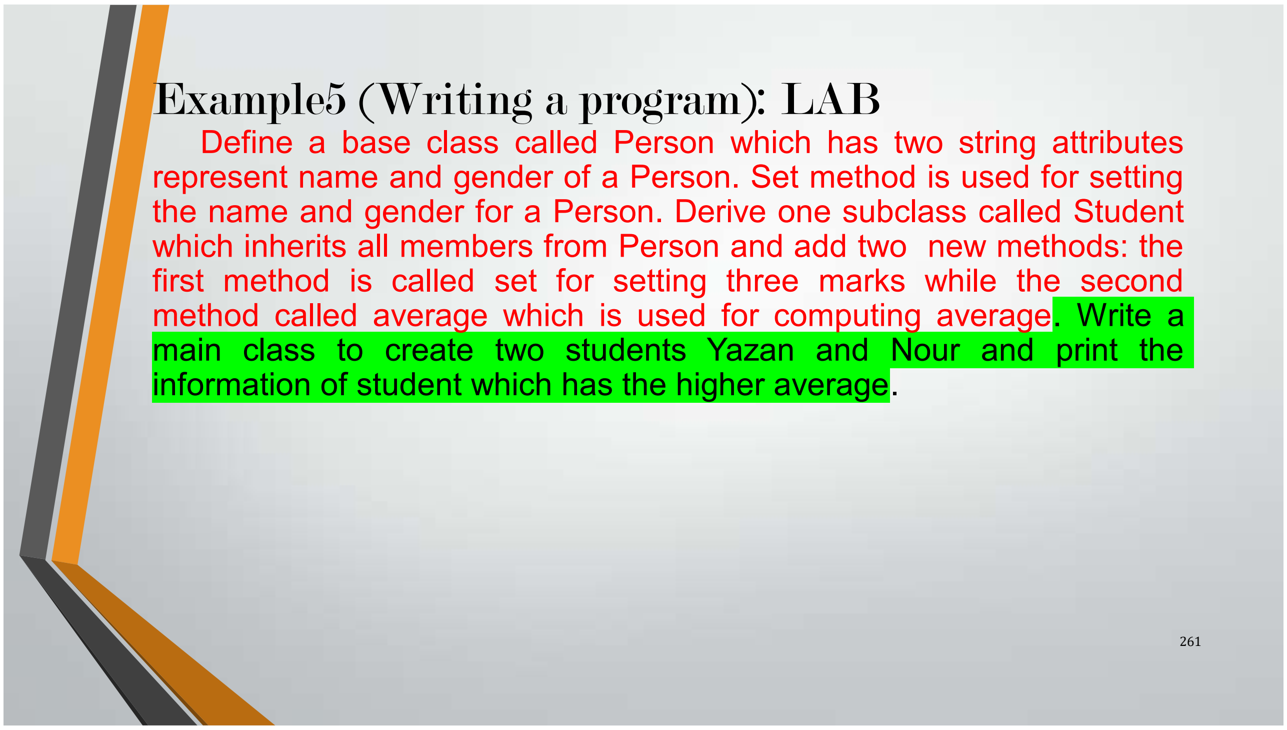
```
public class Student extends Person {
   private int m1,m2,m3;
   public void set(int a,int b,int c){
   m1=a;m2=b;m3=c;}
   public void average(){
double av=(m1+m2+m3)/3.0;
System.out.println(name+"   "+gender+"   "+av);  } }
```

```
public class Main {
   public static void main(String[] args) {
     Student st1=new Student();
     st1.set("Nour", "female");      st1.set(10,8,9);      st1.average();
          Student st2=new Student();
     st2.set("Yazan", "male");   st2.set(10,10,10);      st2.average(); } }
```

The output:

```
Nour   female   9.0
Yazan    male   10.0
```

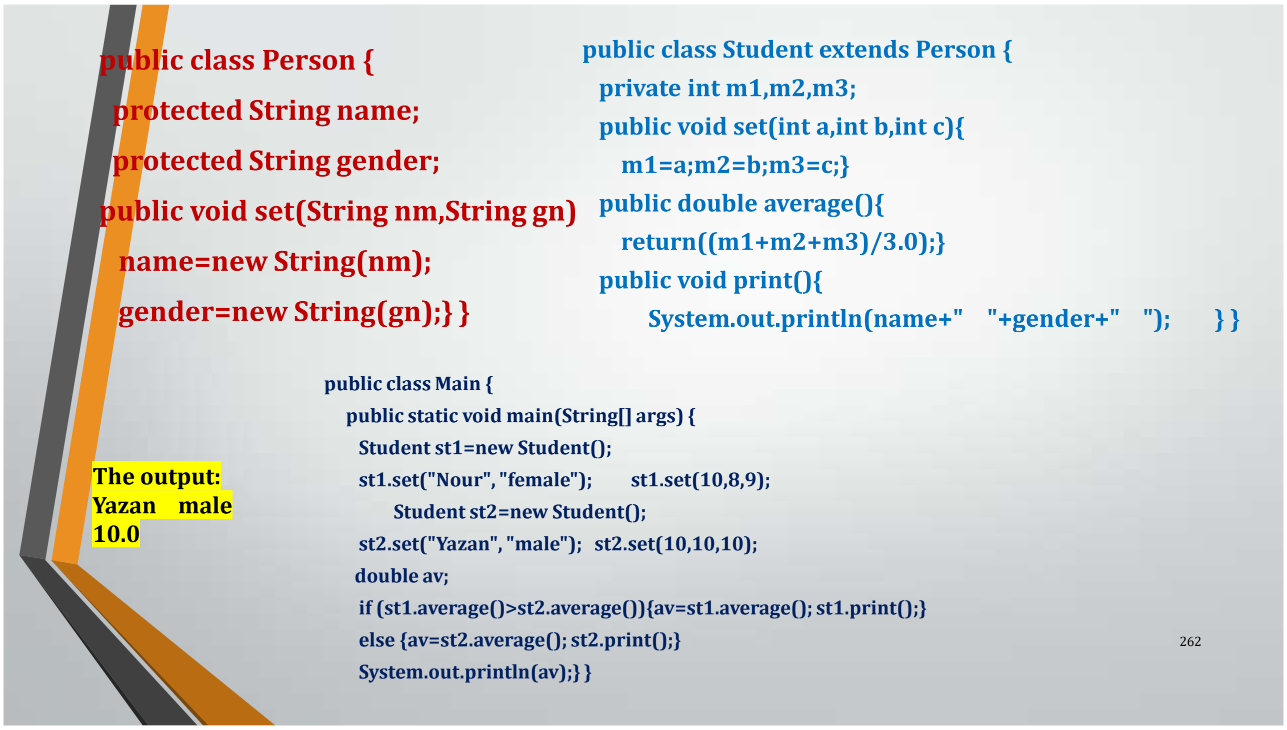
[redacted]



Example5 (Writing a program): LAB

Define a base class called Person which has two string attributes represent name and gender of a Person. Set method is used for setting the name and gender for a Person. Derive one subclass called Student which inherits all members from Person and add two new methods: the first method is called set for setting three marks while the second method called average which is used for computing average. Write a main class to create two students Yazan and Nour and print the information of student which has the higher average.

261



```
public class Person {
 protected String name;
 protected String gender;
public void set(String nm,String gn)
  name=new String(nm);
  gender=new String(gn);} }
```

```
public class Student extends Person {
  private int m1,m2,m3;
  public void set(int a,int b,int c){
     m1=a;m2=b;m3=c;}
  public double average(){
     return((m1+m2+m3)/3.0);}
  public void print(){
        System.out.println(name+"    "+gender+"    ");      } }
```

```
public class Main {
   public static void main(String[] args) {
     Student st1=new Student();
     st1.set("Nour", "female");       st1.set(10,8,9);
          Student st2=new Student();
     st2.set("Yazan", "male");   st2.set(10,10,10);
    double av;
     if (st1.average()>st2.average()){av=st1.average(); st1.print();}
     else {av=st2.average(); st2.print();}
     System.out.println(av);} }
```

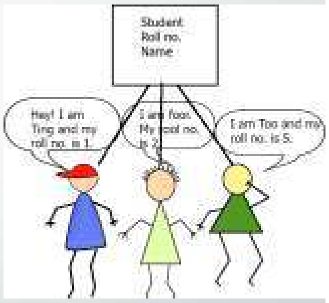
The output:
```
Yazan    male
10.0
```

262

[redacted]

Constructors in base and sub-classes

- **Constructor definition in subclass**
- **Calling a base class constructor in subclass**
- **Examples (tracing and writing programs)**


Student
Roll no.
Name
Hey! I am Ting and my roll no. is 1.
I am Foo. My roll no. is 2.
I am Too and my roll no. is 5.

Java Programming

Understanding the Java Class

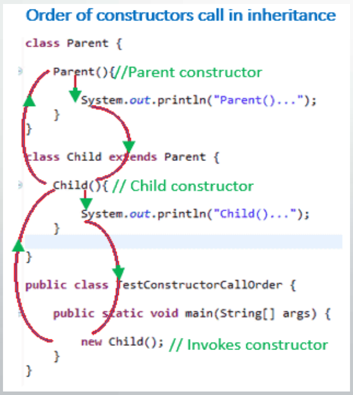
Superclass Constructors

Constructors in Subclasses

- **<u>Constructors are not inherited</u>**. That is, if you extend an existing class to make a subclass, the constructors in the superclass do not become a part of the subclass.
- If you want constructors in the subclass, you have to define new ones.
- If you don't define any constructors in the subclass, then the computer will make up a default constructor, with no parameters, for you. This could be a problem, if there is a constructor in the superclass that does a lot of necessary work. It looks like you might have to repeat all that work in the subclass!

[redacted]

- constructor of sub-class is invoked when we create the object of subclass, it by default invokes the constructor of super class **(which has no parameters)**. Hence, in inheritance the objects are **constructed top-down.**
- The superclass constructor can be called explicitly using the **super keyword**, but it should be first statement in a constructor.
- The super keyword refers to the superclass, immediately above of the calling class in the hierarchy. The use of multiple super keywords to access an ancestor class other than the direct parent **is not permitted**.
- The super keyword should be used when we need to call a constructor **with parameters**.


Order of constructors call in inheritance

```
class Parent {
    Parent(){//Parent constructor
        System.out.println("Parent()...");
    }
}
class Child extends Parent {
    Child(){ // Child constructor
        System.out.println("Child()...");
    }
}
public class TestConstructorCallOrder {
    public static void main(String[] args) {
        new Child(); // Invokes constructor
    }
}
```

Example1 (Tracing a program)

```
public class One {
   public One(){
      System.out.println("One");}
   public One(int a){
      System.out.println("One with parameter");}   }
```

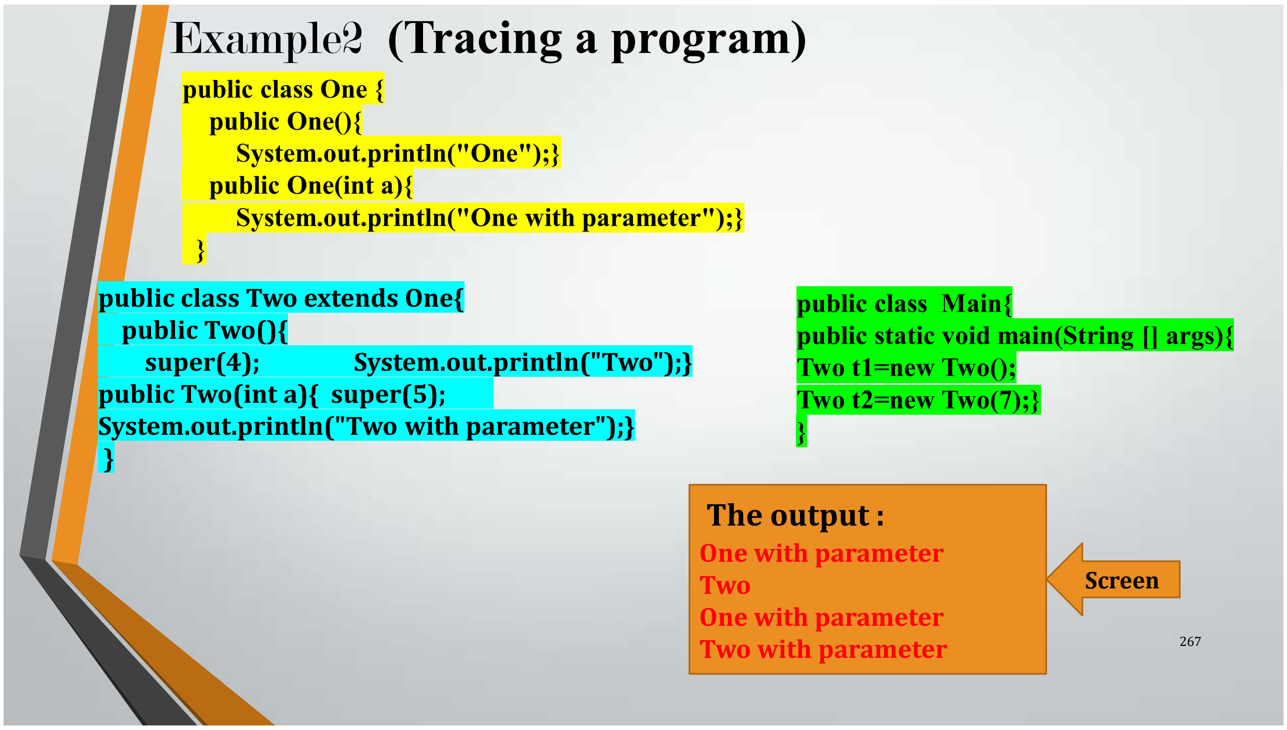
```
public class Two extends One{
   public Two(){
      super(); // optional
      System.out.println("Two");}
public Two(int a){
     super(); //optional
   System.out.println("Two  with  parameter");
} }
```

```
public class  Main{
public static void main(String [] args){
Two t1=new Two();
Two t2=new Two(7);}}
```

The output :

```
One
Two
One
Two with parameter
```

Screen



Example2 (Tracing a program)

```
public class One {
   public One(){
      System.out.println("One");}
   public One(int a){
      System.out.println("One with parameter");}
 }
```

```
public class Two extends One{
   public Two(){
      super(4);            System.out.println("Two");}
public Two(int a){  super(5);
System.out.println("Two with parameter");}
 }
```

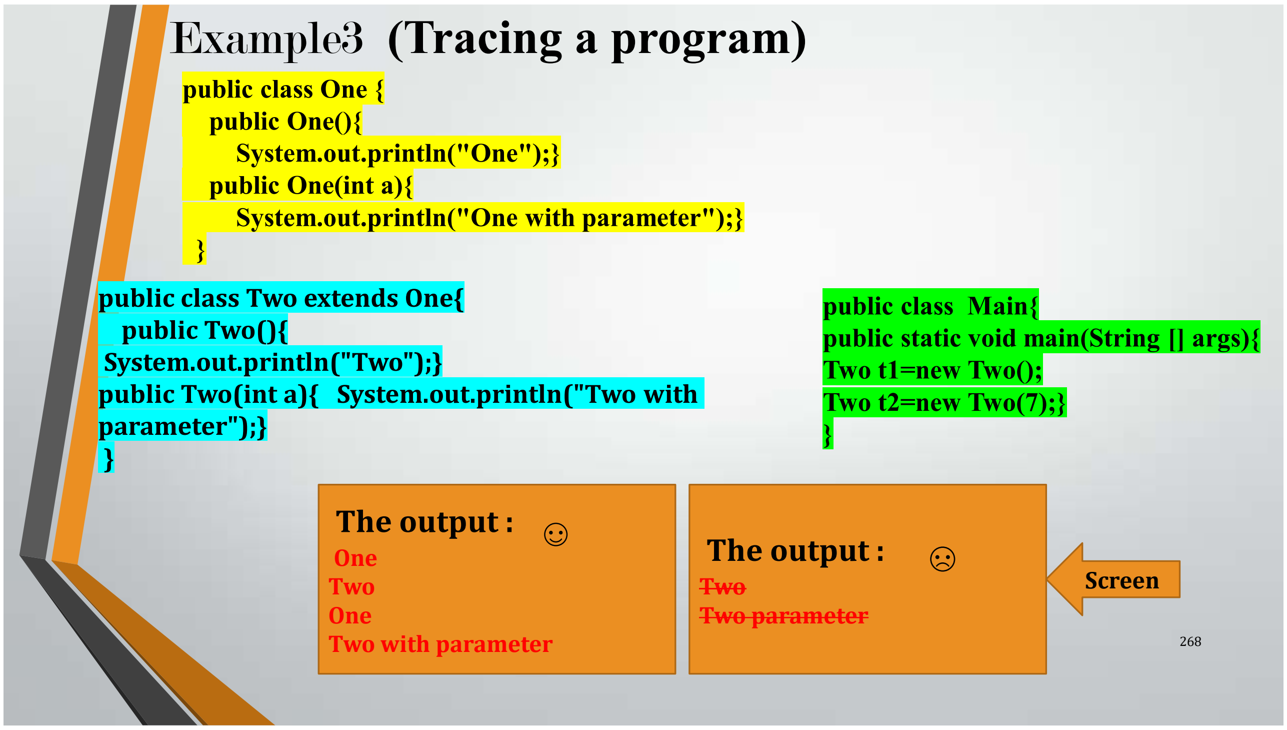
```
public class  Main{
public static void main(String [] args){
Two t1=new Two();
Two t2=new Two(7);}
}
```

The output :

```
One with parameter
Two
One with parameter
Two with parameter
```

Screen

267



Example3 (Tracing a program)

```
public class One {
   public One(){
      System.out.println("One");}
   public One(int a){
      System.out.println("One with parameter");}
 }
```

```
public class Two extends One{
   public Two(){
 System.out.println("Two");}
public Two(int a){  System.out.println("Two with
parameter");}
 }
```

```
public class  Main{
public static void main(String [] args){
Two t1=new Two();
Two t2=new Two(7);}
}
```

The output : ☺

```
 One
Two
One
Two with parameter
```

The output : ☹

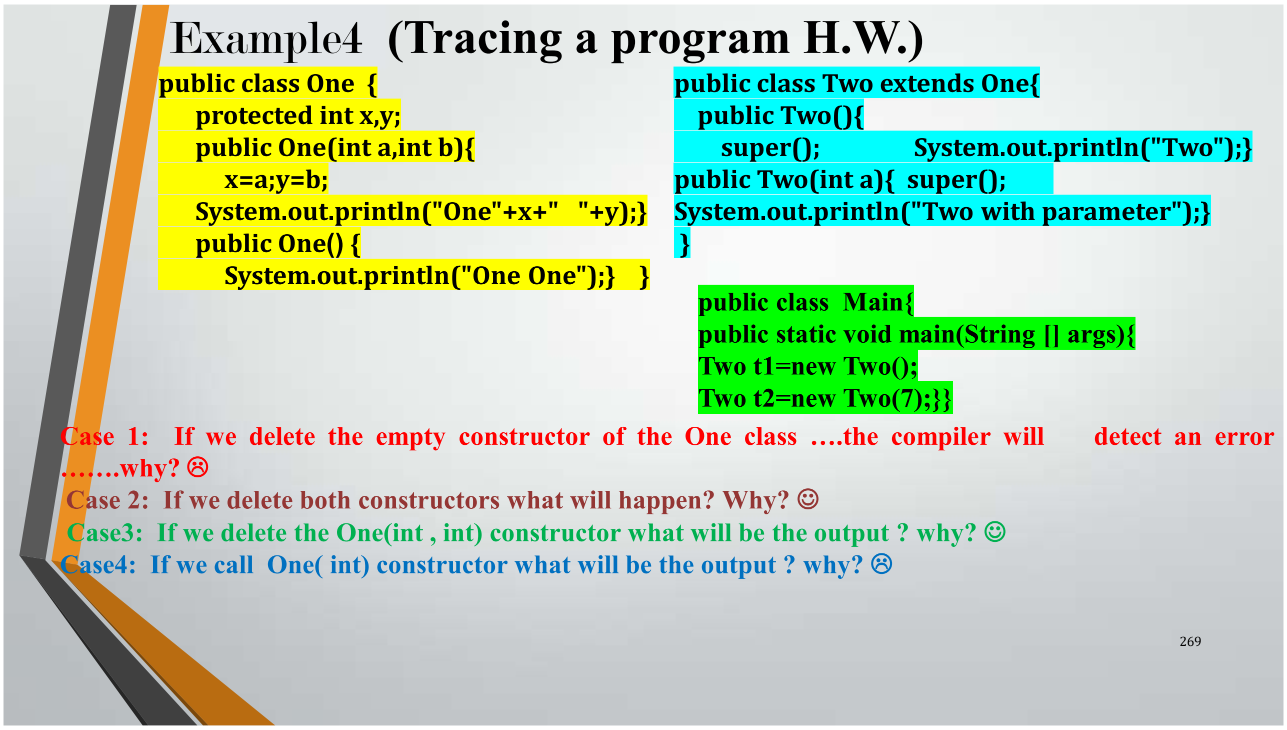
~~Two~~

~~Two parameter~~

Screen

268

[redacted]



Example4 (Tracing a program H.W.)

```
public class One  {
    protected int x,y;
    public One(int a,int b){
        x=a;y=b;
    System.out.println("One"+x+"   "+y);}
    public One() {
        System.out.println("One One");}   }
```

```
public class Two extends One{
   public Two(){
      super();            System.out.println("Two");}
public Two(int a){  super();
System.out.println("Two with parameter");}
}
```

```
public class  Main{
public static void main(String [] args){
Two t1=new Two();
Two t2=new Two(7);}}
```

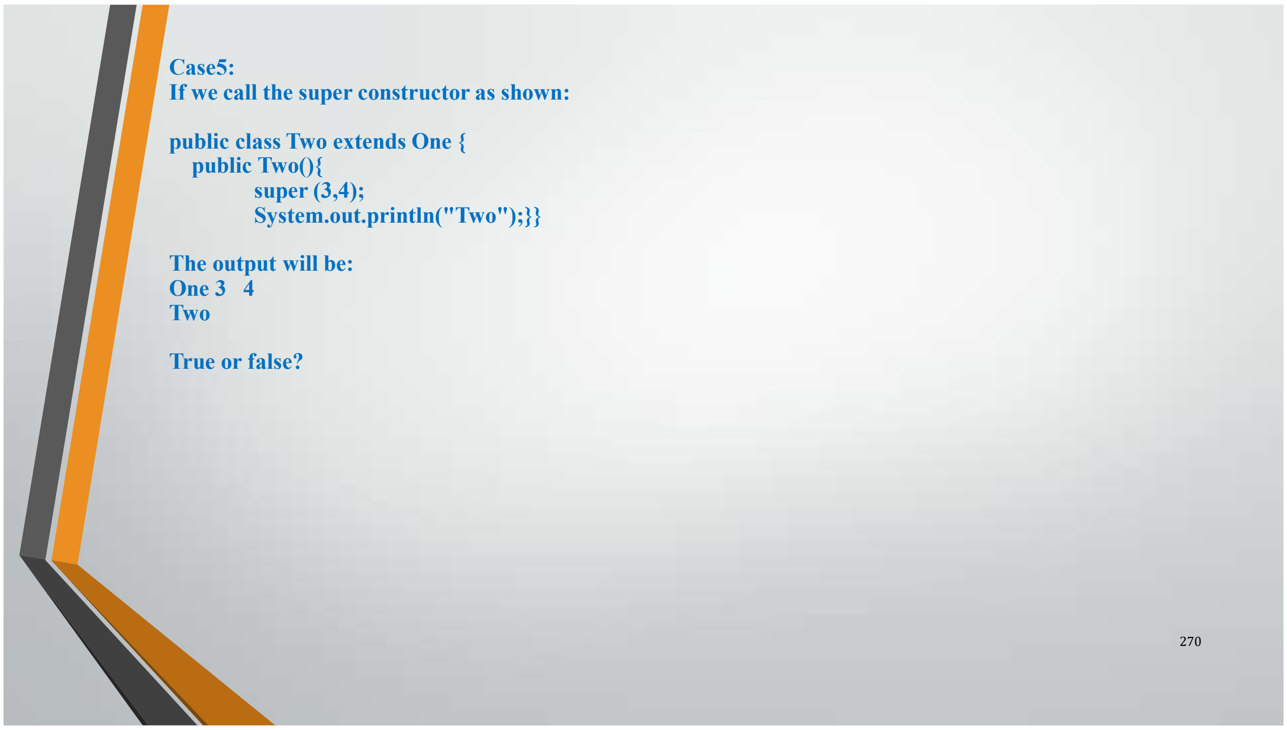
Case 1: If we delete the empty constructor of the One class ….the compiler will detect an error …….why? ☹

Case 2: If we delete both constructors what will happen? Why? ☺

Case3: If we delete the One(int , int) constructor what will be the output ? why? ☺

Case4: If we call One(int) constructor what will be the output ? why? ☹

269



Case5:
If we call the super constructor as shown:

```
public class Two extends One {
   public Two(){
            super (3,4);
            System.out.println("Two");}}
```

The output will be:

```
One 3   4
Two
```

True or false?

270

[redacted]

Example 4 (Writing a program)

Emergency contacts

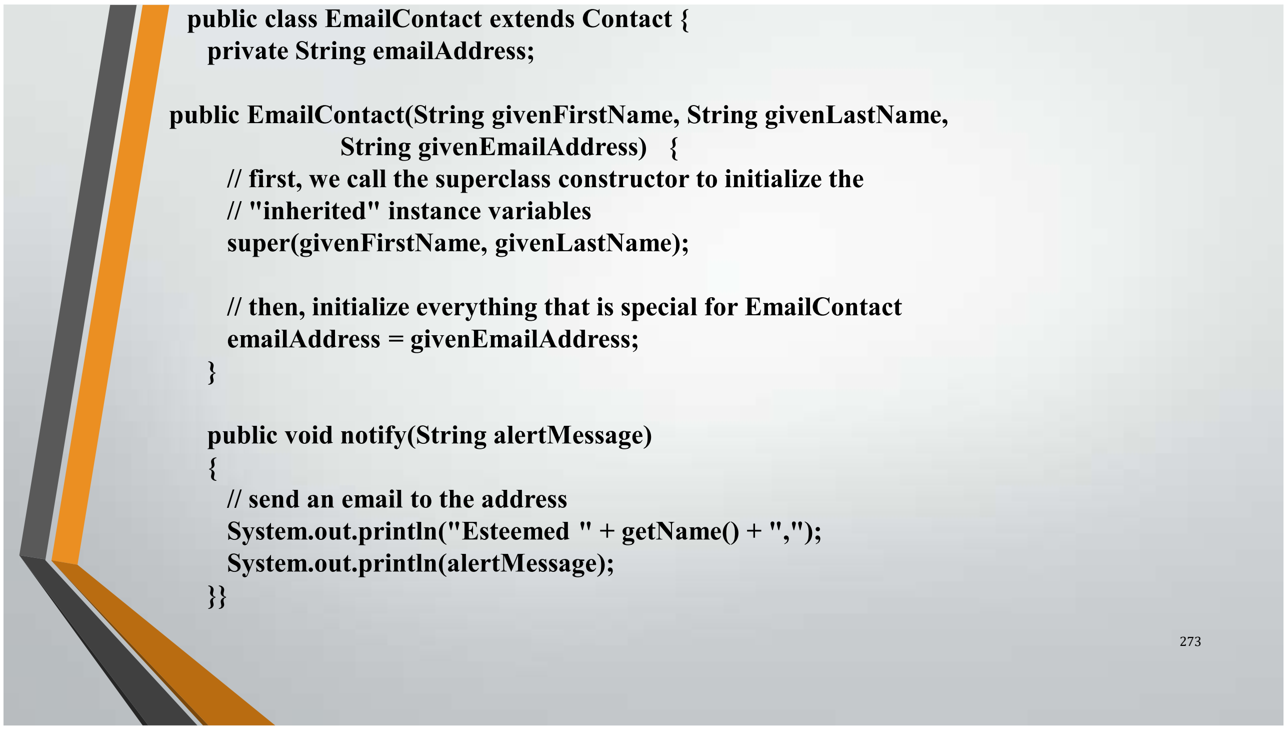
Crisis alert systems are all the rage these days. When an emergency manifests itself, all folks who have registered with the emergency contact database are notified via email, phone, text message , etc. We can use the concepts of inheritance to reduce the system's complexity and allow for future ways of contacting individuals.

First, we model the general Contact. All contacts should have a name, but the particular way in which they are contacted depends upon their preferred method of communication. We will leave it for the subclasses of Contact to decide how to implement the notify method.

Defining a subclass using the extends keyword

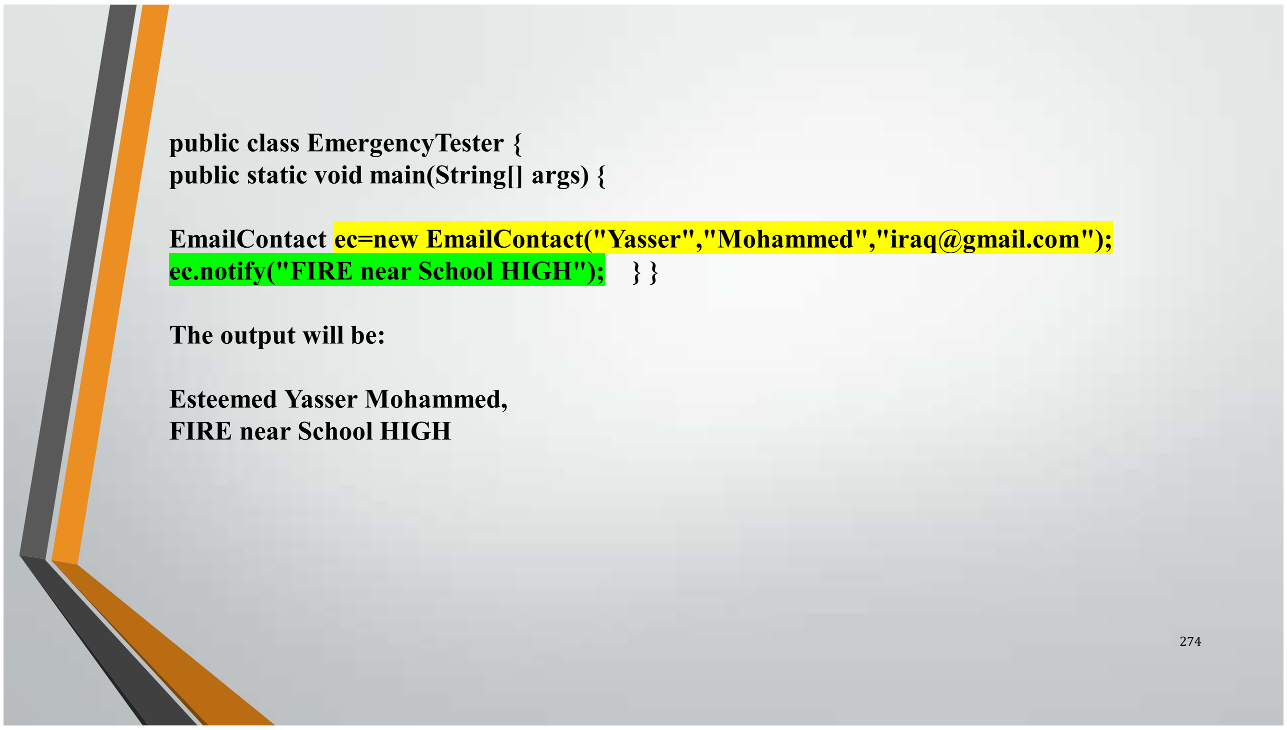
Now, let's make an EmailContact which is a subclass of Contact that is specialized for email notification. In order to define a subclass of any other class,

```
public class Contact {
  private String firstName;
  private String lastName;

  public Contact(String givenFirstName,    String givenLastName) {
    firstName = givenFirstName;
    lastName = givenLastName;   }

  public String getName() {
    return (firstName + "  " + lastName);   }}
```



```
 public class EmailContact extends Contact {
   private String emailAddress;

public EmailContact(String givenFirstName, String givenLastName,
                    String givenEmailAddress)   {
     // first, we call the superclass constructor to initialize the
     // "inherited" instance variables
     super(givenFirstName, givenLastName);

     // then, initialize everything that is special for EmailContact
     emailAddress = givenEmailAddress;
   }

   public void notify(String alertMessage)
   {
     // send an email to the address
     System.out.println("Esteemed " + getName() + ",");
     System.out.println(alertMessage);
   }}
```

273



```
public class EmergencyTester {
public static void main(String[] args) {

EmailContact ec=new EmailContact("Yasser","Mohammed","iraq@gmail.com");
ec.notify("FIRE near School HIGH");    } }
```

The output will be:

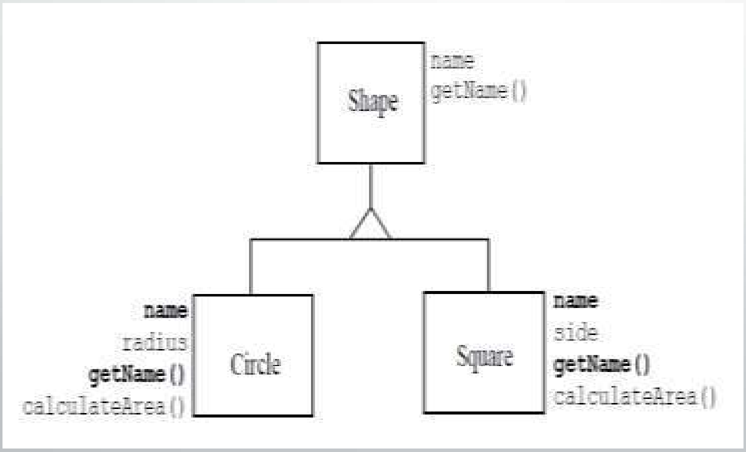
```
Esteemed Yasser Mohammed,
FIRE near School HIGH
```

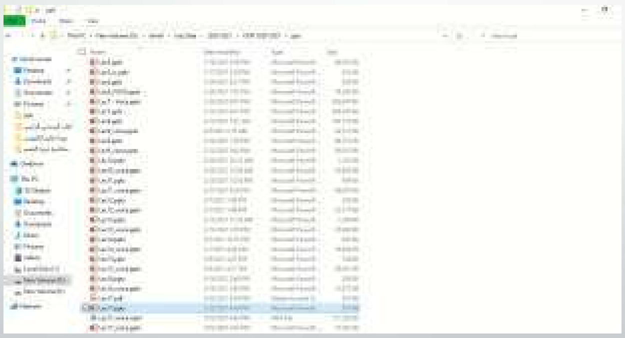
274

[redacted]

LAB :

The following figure is a class hierarchy of shapes. Shape is a generalized class of Circle and Square. All shapes have a name and a measurement by which the area of the shape is calculated. The attribute name and method getName() are defined as properties of Shape. Circle and Square, being subclasses of Shape, inherit these properties (highlighted in bold in the following figure).use constructors to set the values of attributes.



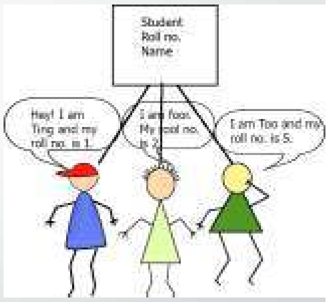


276

[redacted]

"this" keyword ←

- **Using this key word in variables**
  - **Using this with constructor**
  - **Shadowed and hided Variables**
  - **The Special Variable super**
- **Examples (tracing and writing programs)**

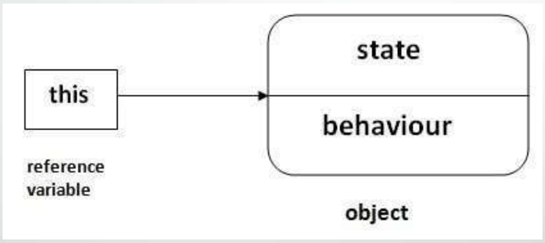

Student
Roll no.
Name
Hey! I am Ting and my roll no. is 1.
I am Foo. My roll no. is 2.
I am Too and my roll no. is 5.

278

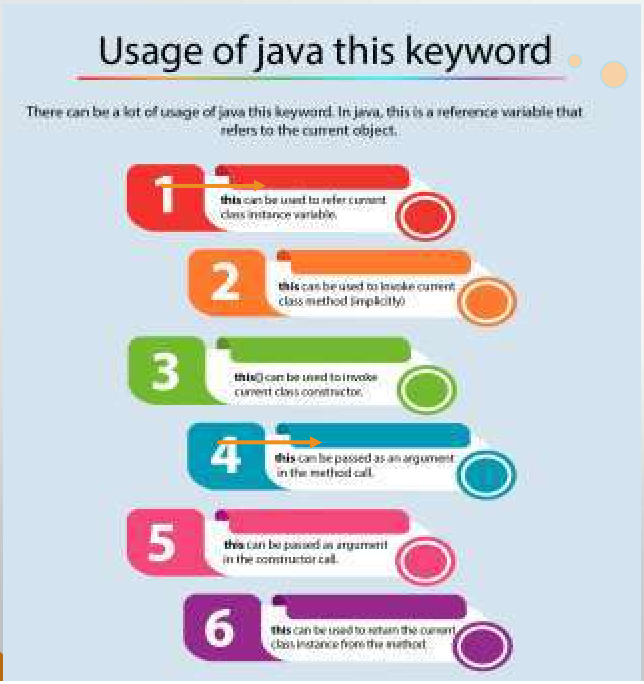
[redacted]

• 'this' keyword ←

There can be a lot of usage of **java this keyword**.
In java, this is a **reference variable** that refers to the current object.



279



Usage of java this keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

1 **this** can be used to refer current class instance variable.

2 **this** can be used to invoke current class method (implicitly)

3 **this()** can be used to invoke current class constructor.

4 **this** can be passed as an argument in the method call.

5 **this** can be passed as argument in the constructor call.

6 **this** can be used to return the current class instance from the method.

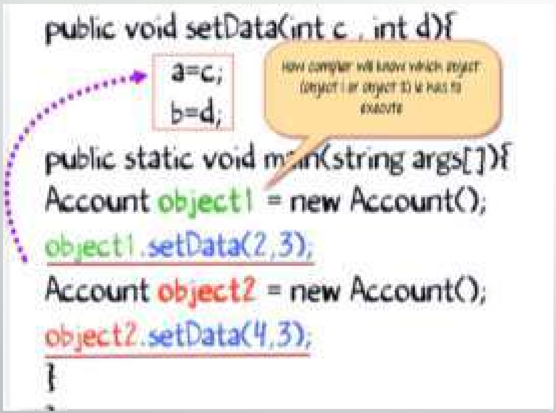
Preparing an example for each point in the figure.

280

[redacted]

• Using **this** with a Field

- The most common reason for using this keyword is because a field is shadowed by a method or constructor parameter.



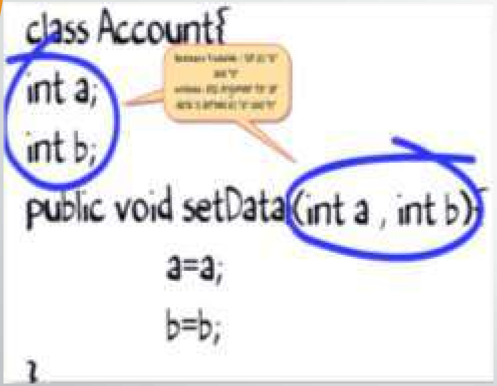
```
public void setData(int c , int d){
        a=c;
        b=d;

public static void main(string args[]){
Account object1 = new Account();
object1.setData(2,3);
Account object2 = new Account();
object2.setData(4,3);
}
```

How compiler will know which object (object1 or object2) he has to execute

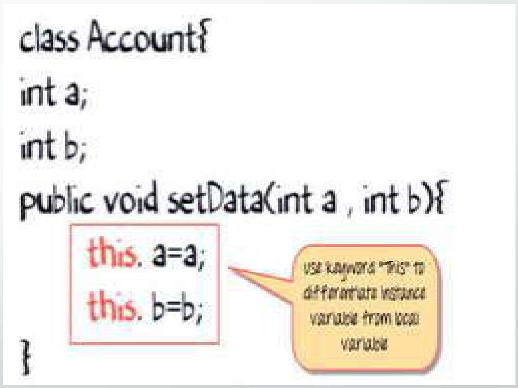
281

• Using **this** with a Field

- The most common reason for using this keyword is because a field is shadowed by a method or constructor parameter.



```
class Account{
int a;
int b;
public void setData(int a , int b){
        a=a;
        b=b;
}
```

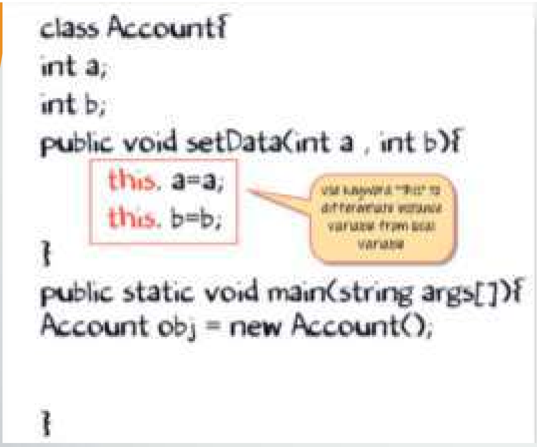


```
class Account{
int a;
int b;
public void setData(int a , int b){
      this. a=a;
      this. b=b;
}
```

use keyword "this" to differentiate instance variable from local variable

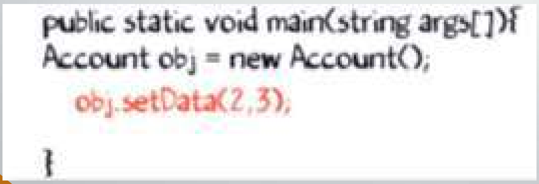
282

[redacted]



```
class Account{
int a;
int b;
public void setData(int a , int b){
        this. a=a;
        this. b=b;
}
public static void main(string args[]){
Account obj = new Account();


}
```

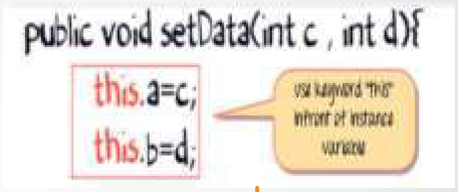
use keyword "this" to differentiate instance variable from local variable



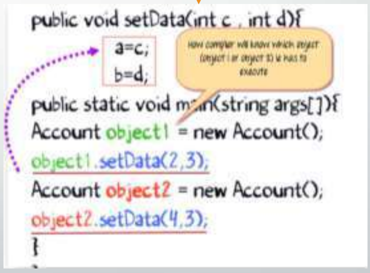
```
public static void main(string args[]){
Account obj = new Account();
    obj.setData(2,3);

}
```

- **QUIZ**

Choose the suitable face for each case



```
public void setData(int c , int d){
      this.a=c;
      this.b=d;
```



```
public void setData(int c , int d){
        a=c;
        b=d;
public static void main(string args[]){
Account object1 = new Account();
object1.setData(2,3);
Account object2 = new Account();
object2.setData(4,3);
}
```

How compiler will know which object (object 1 or object 2) it has to execute

283

☺ ● ☺ ☺

- **Example1 (Trace)**

```
public class Point {
public int x = 0;
public int y = 0;
public Point(int a, int b) {     x = a;        y = b;     }}
```

☺

- **Example2 (Trace)**

but it could have been written like this:

```
public class Point {
 public int x = 0;
 public int y = 0;
 public Point(int x, int y) {
  this.x = x;         this.y = y;     }}
```

☺

284

[redacted]

- **Example3 (Trace)**

```
public class Account{
private int a; int b;
public void set(int a ,int b){
a = a;    b = b; }
```

- **Example4 (Trace)**

but it could have been written like this:

```
public void show(){
System.out.println("Value of A ="+a);
System.out.println("Value of B ="+b); } }

public class Main{
public static void main(String args[]){
Account obj = new Account();
      obj.set(2,3);    obj.show(); } }
```

The output is:

```
Value of A =0
Value of B =0
```

- **Using this with Constructor**

- **From within a constructor, you can also use “this” keyword to call another constructor in the same class. Doing so is called an explicit(واضح) constructor invocation (استدعاء).**

[redacted]

• Example5 (Trace)

```
public class Rectangle {
   private int x, y;
   private int width, height;

   public Rectangle() { this(0, 0, 1, 1); }

   public Rectangle(int width, int height) {
   this(0, 0, width, height); }

   public Rectangle(int x, int y, int width, int height){
   this.x=x;
   this.y=y;
   this.width = width;
   this.height = height;    }

   public void print(){
   System.out.print("   x="+x);
   System.out.print("   y="+y);
   System.out.print("   width="+width);
   System.out.print("   height="+height);
}}
```

```
public class Main {
public static void main(String[] args) {

Rectangle r1=new Rectangle();
   r1.print();

Rectangle r2=new Rectangle(10,20);
   r2.print();
Rectangle r3=new Rectangle(5,55,6,66);
   r3.print();    } }
```

The output:

```
x=0    y=0    width=1    height=1
x=0   y=0   width=10   height=20
x=5   y=55   width=6   height=66
```

• Shadowed and hided Variables

- One variable *shadow* another if they have the same name and are accessible in the same place.

• Example6 (Trace)

```
public class One {
   private int x = 1000;
   public void set(){
      int x;
      x=100;
   System.out.println(x);
   System.out.println(((One)this).x);

((One)this).x=2000;
 System.out.println(((One)this).x);} }

public class JavaMain {
public static void main(String[] args) {
      One obj=new One();
       obj.set();    }}
```

The output is:

```
 100
1000
2000
```

H.W

```
System.out.println(x);
  System.out.println(this.x);
```

[redacted]

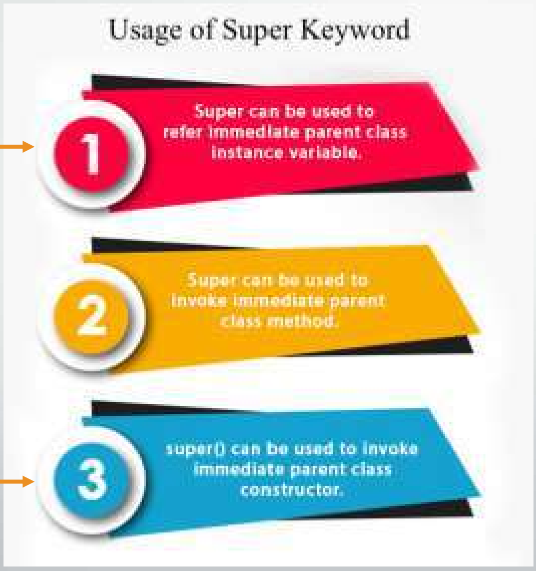
- What happens if an inherited variable has the same name as a variable of the subclass? The variable of the subclass is said to hide (sometimes called shadow) the inherited variable with the same name. The inherited variable is visible in the subclass, yet it cannot be accessed by the same name.

- We can say that a field is said to *hide* all fields with the same name in super classes.

- But what if you need to use the inherited variable in the subclass; how can it be accessed? The answer is to use the reserved word super.
- For example, if class B is a subclass of class A, and both contain a variable named x as follows.

289

- **The Special Variable super**


Usage of Super Keyword
1
Super can be used to refer immediate parent class instance variable.
2
Super can be used to invoke immediate parent class method.
3
super() can be used to invoke immediate parent class constructor.

290

[redacted]

• Example7 (Trace)

```
public class A  {
protected int x; .................. }
public class B extends A {
// hide (shadow) the inherited variable common from class A
protected int x;
................................ }
```

- Then in class B, the variable common may be referred to by either **x** or **this.x**. However, the inherited variable common is referred to by **super.x** or by **((A)this).x**.

- Notice that the keyword 'this' may be cast to refer to the appropriate class, in this case class A. This technique is useful if you want to refer to a variable in a class beyond (وراء) the immediate superclass higher up the class hierarchy.

- Although you may refer to shadowed variables by casting an object to the appropriate type, this technique cannot be used to refer to overridden methods as explained in next lecture.

[redacted]

• Example8 (Trace)

```
public class Two extends One {
 private int x ;
   public void print(){    set();          x=2000;
       System.out.println(this.x);
System.out.println(x);
       System.out.println(((One)this).x);
System.out.println(super.x);

       super.x=10;           this.x=20;

   System.out.println(this.x);       System.out.println(x);
   System.out.println(((One)this).x);
System.out.println(super.x);

   ((One)this).x=30;         x=40;
   System.out.println(((One)this).x);
System.out.println(super.x);
   System.out.println(this.x);      System.out.println(x); }  }
```

```
public class One {
  protected  int x ;
         public void set(){   x=1000;
   System.out.println(x);
System.out.println(this.x);} }
```

```
public class Main {
        public static void main(String[] args){
        Two obj=new Two();
        obj.print();     } }
```

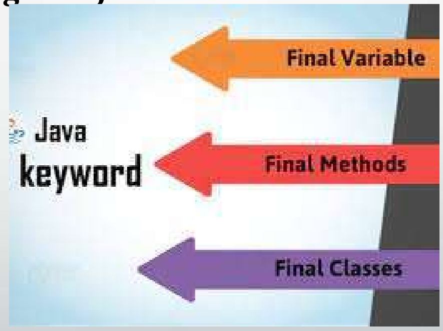
The output will be:

```
1000
1000
2000
2000
1000
1000
20
20
10
10
30
30
40
40
```

[redacted]

final keyword in Java

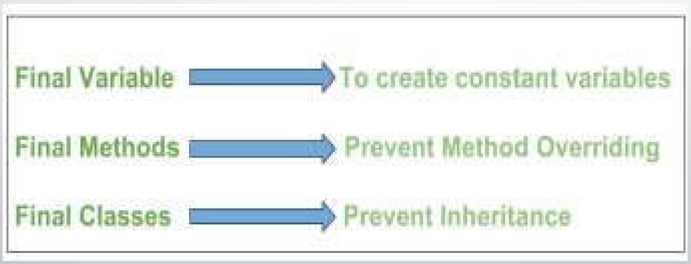
- Final variable and blank variable
- Final method
- Final class
- Examples (tracing programs)



295

• Final Keyword in Java

Java, ***final* keyword** is applied in various contexts. The *final* keyword is a modifier means the *final* class can't be extended, a variable can't be modified, and a method can't be override it means that an entity cannot later be changed.
It used for the following purposes:


Final Variable → To create constant variables
Final Methods → Prevent Method Overriding
Final Classes → Prevent Inheritance

296

Final variable

- **The field declared as final behaves like constant. Means once it is declared, it can't be changed. Before compiling, only once it can be set; after that you can't change its value. Attempt to change in its value lead to exception or compile time error. If the final variable dose not initialize the compiler will throw compile-time error. It could be declared as shown below:**

Example1:

public final double radius = 126.45;
public final double PI = 3.145;

The fields which are declared as static, final and public are known as *named constants*.

Example2:

public class Maths{
public final double x = 2.998E8; }

297

Blank variable

- A **final** variable does not need not be initialized at the point of declaration: this is called a **"blank final"** variable.
- A blank final instance variable of a class must be assigned at the end of every constructor of the class in which it is declared; similarly, a blank **final variable must be assigned in a initializer of the class in which it is declared**: otherwise, a compile-time error occurs in both cases.

Example3:

```
public class Sphere {
  public  final double PI = 3.141592653589793;

//blank final
  public final double radius;
  public final double xpos;
  public final double ypos;
  public final double zpos;
   public Sphere(double x, double y, double z, double r) {
       radius = r;       xpos = x;       ypos = y;       zpos = z;    }
  [...]  }
```

298

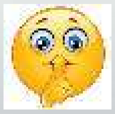
[redacted]

• Note

- **Any attempt to reassign radius, xpos, ypos, or zpos will meet with a compile error.**
- **In fact, even if the constructor doesn't set a final variable, attempting to set it outside the constructor will result in a compilation error.**

• Example4

```
public class Test {
public static void main(String args[]) {
    final int i = 10;
   i = 30;  // Error because i is final.
    i=i+1; // Error because i is final.    } }
```



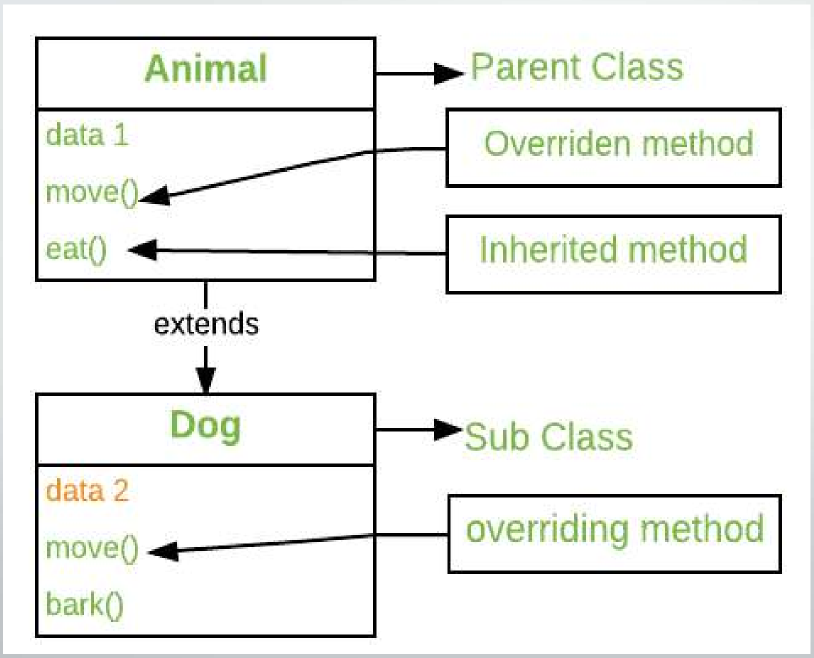
299

• Final method

- **You can declare some or all of a class's methods final. A final method can't be overridden by subclasses but it is still could be overloaded. This is used to prevent unexpected behavior from a subclass altering a method that may be crucial to the function or consistency of the class.**
- A final method within a class could be declared in java as shown:

public class MyClass {
public final void myFinalMethod() {...}}

300



Example 4 (Trace):

```
public class FinalExample {
public final void display( ){
System.out.println("Hello welcome to Tutorialspoint");    }}

public  class Sample extends FinalExample{
   public void display(){        System.out.println("hi");     }    }


public class Main{   public static void main(String args[]){
  Sample s=new Sample();      s.display();    }}
```

Output

```
FinalMethodExample.java:12: error: display() in FinalMethodExample.Sample cannot
override display() in FinalMethodExample
public void display(){
           ^
overridden method is final
1 error
```

302

[redacted]

• Final method arguments

- You can also declare method's argument as final. The final argument can't be modified by the method directly.

• Final class

- A **final** **class** cannot be extended. This is done for reasons of security (المنية والحماية) and efficiency (الكفاءة و الفعالية). Accordingly, many of the Java standard library classes are final, for example java.lang.System and java.lang.String. **All methods in a final class are implicitly final.**
- The final class is declared in java as shown:

public final class A {}

If we say:

public class B extends A ✖

- **This means that A cannot be further extended or subclassed. This feature has a big implication. It allows control over a class, so that no one can subclass the class and possibly introduce anomalous behavior (سلوك غير طبيعي). For example, java.lang.String is a final class. This means, for example, that I can't subclass String and provide my own length() method that does something very different from returning the string length.**

[redacted]

1- H.W.

1. Constructor can't be final …why?
2. Could you define a final class without final methods?
3. Could you use a set method to initialize the final attributes within a class.
4. Does the final method process the final variables?
5. Explain the blank final.
6. Discuss that " The value of a final variable is not necessarily known at compile time"

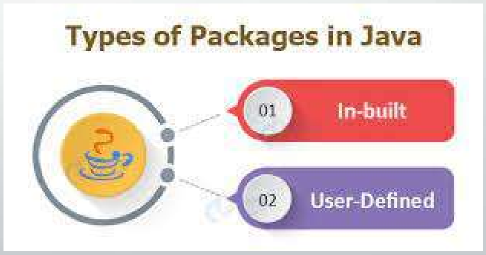
2- H.W.

- <u>Give a programming example with its execution that describes g the following :</u>

- **Methods called from constructors should generally be declared final. If a constructor calls a non-final method, a subclass may redefine that method with surprising or undesirable results.**

Packages

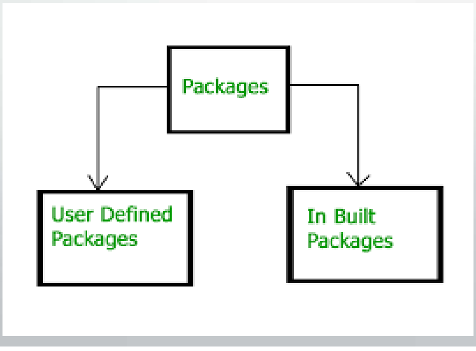
- **Package def. and types**
- **Creating a package**
- **Importing a package**
- **Access to classes within the same package**
- **Examples**

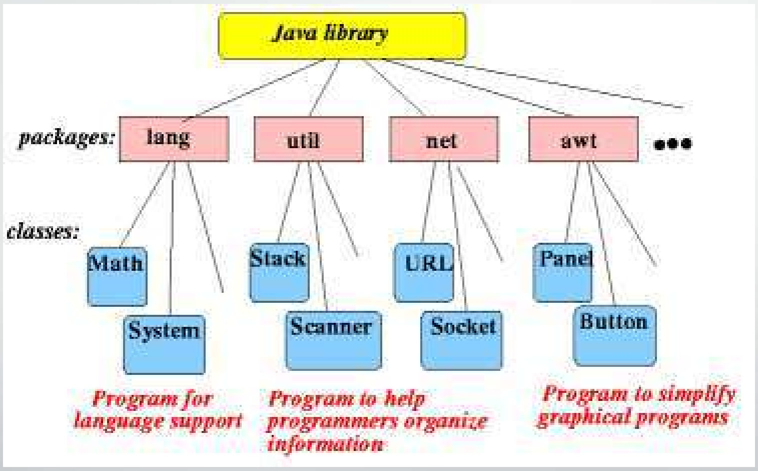


[redacted]

• Def.

- A java package is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two forms, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.




Java library
packages:
lang
util
net
awt
•••
classes:
Math
System
Stack
Scanner
URL
Socket
Panel
Button
Program for language support
Program to help programmers organize information
Program to simplify graphical programs

[redacted]

• Advantage of Java Package

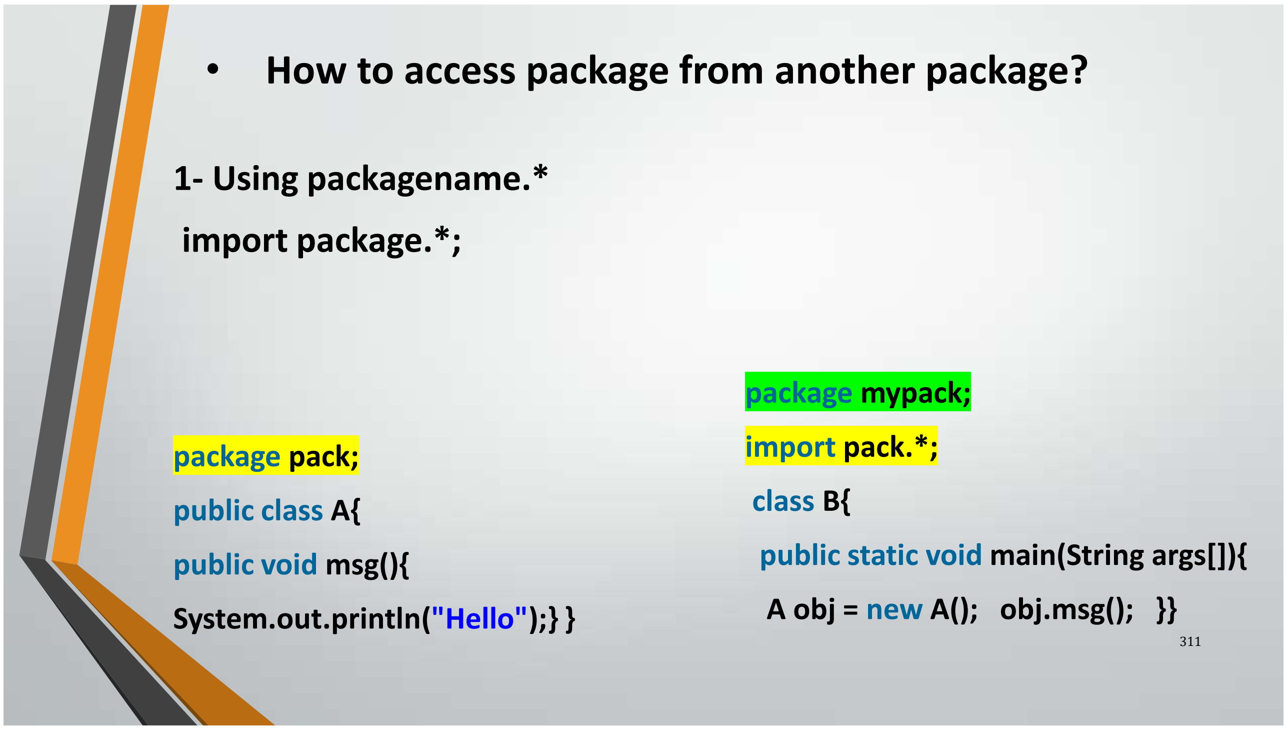
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.

• Creating Package

- The **package keyword** is used to create a package in java.

```
package mypack;
public class Simple{
 public static void main(String args[]){
   System.out.println("Welcome to package");
  }
}
```

[redacted]



- How to access package from another package?

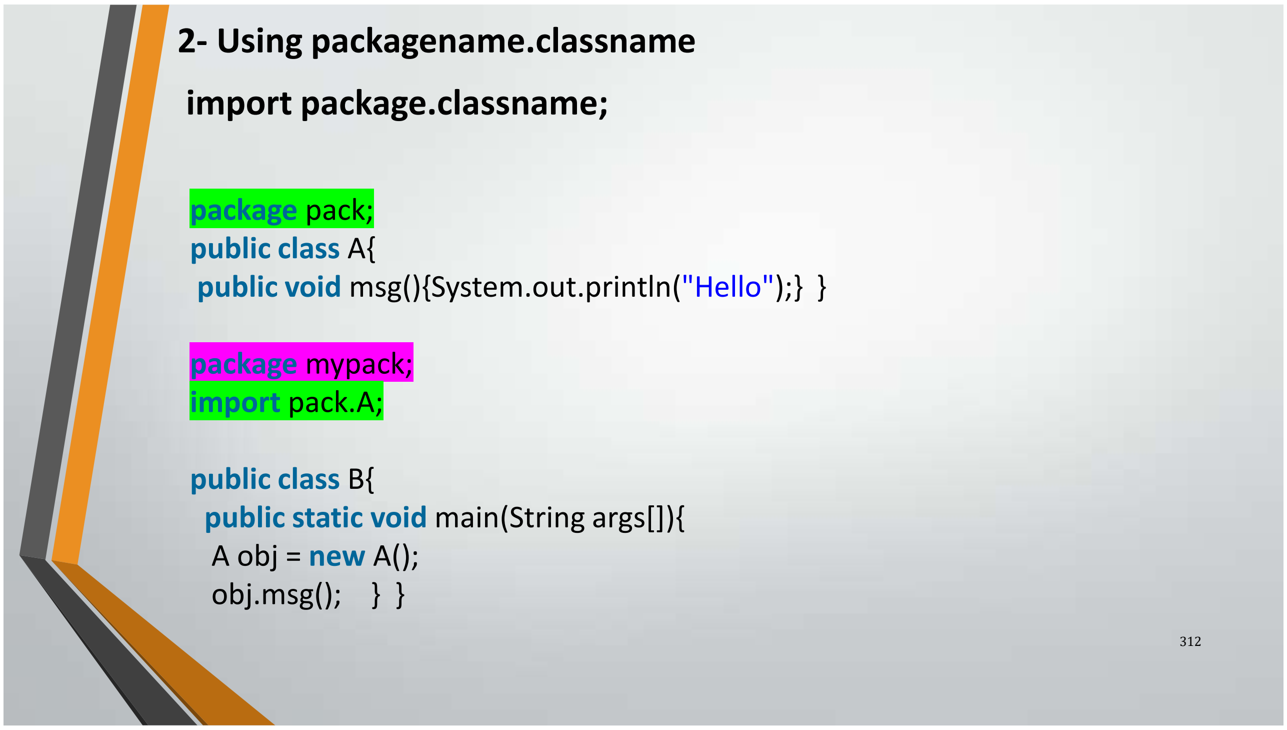
1- Using packagename.*

import package.*;

```
package pack;
public class A{
public void msg(){
System.out.println("Hello");} }
```

```
package mypack;
import pack.*;
 class B{
  public static void main(String args[]){
   A obj = new A();   obj.msg();   }}
```

311

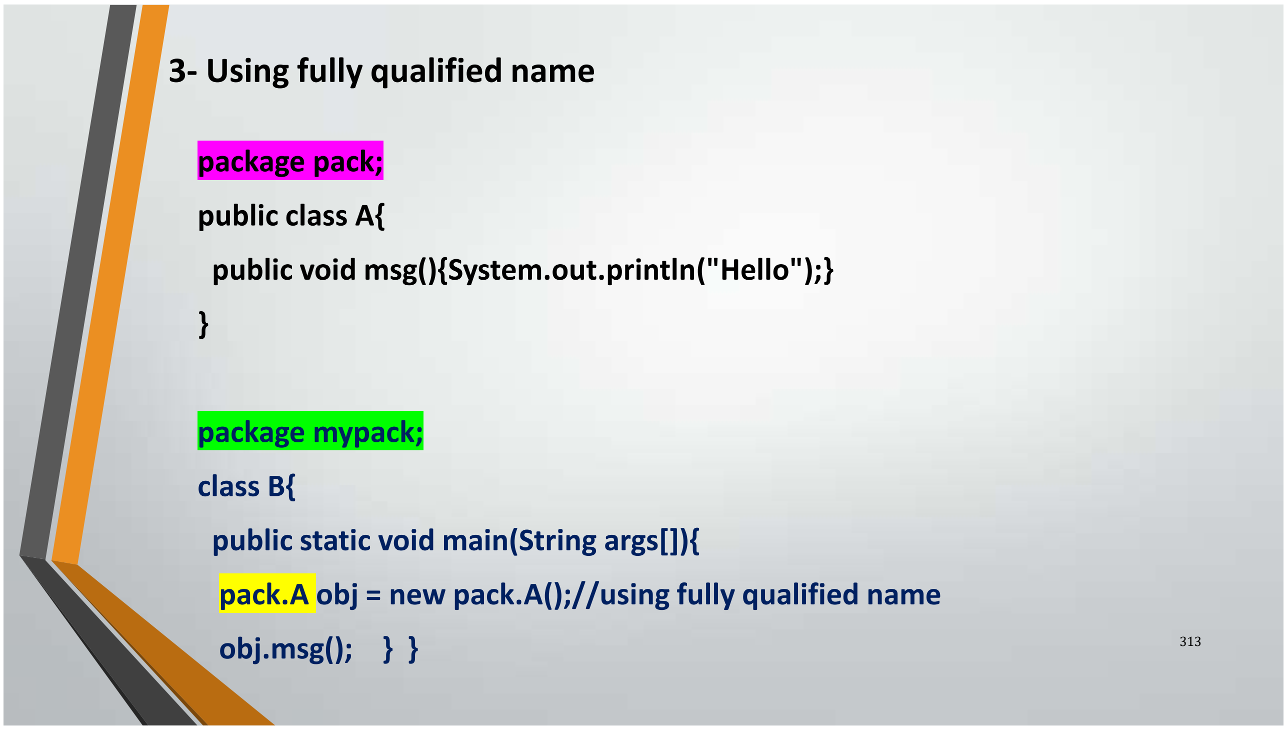


2- Using packagename.classname

import package.classname;

```
package pack;
public class A{
 public void msg(){System.out.println("Hello");}  }

package mypack;
import pack.A;

public class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();    }  }
```

312



3- Using fully qualified name

```
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}

package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();    } }
```

313

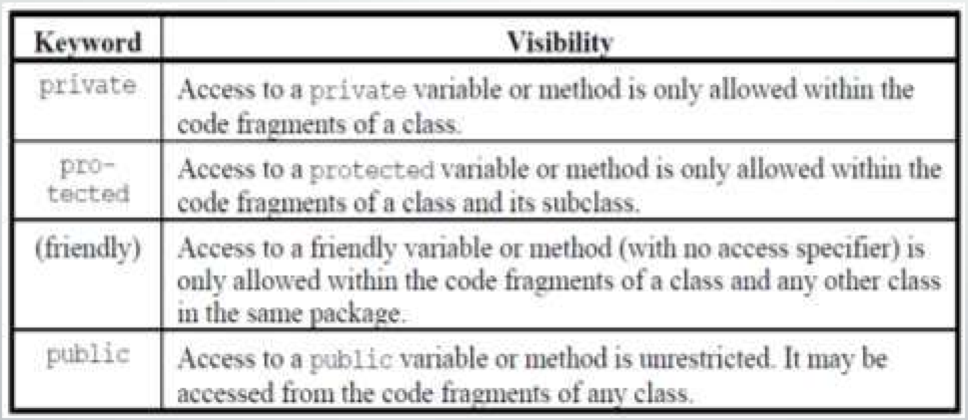
• Sub -package

```
package com.javatpoint.core;
class Simple{
 public static void main(String args[]){
  System.out.println("Hello subpackage");
 } }
```

314

[redacted]

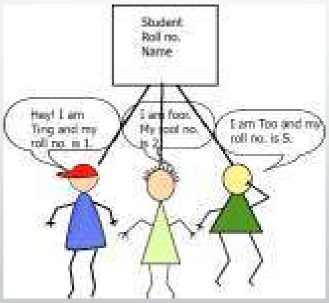
- **Access Modifiers**



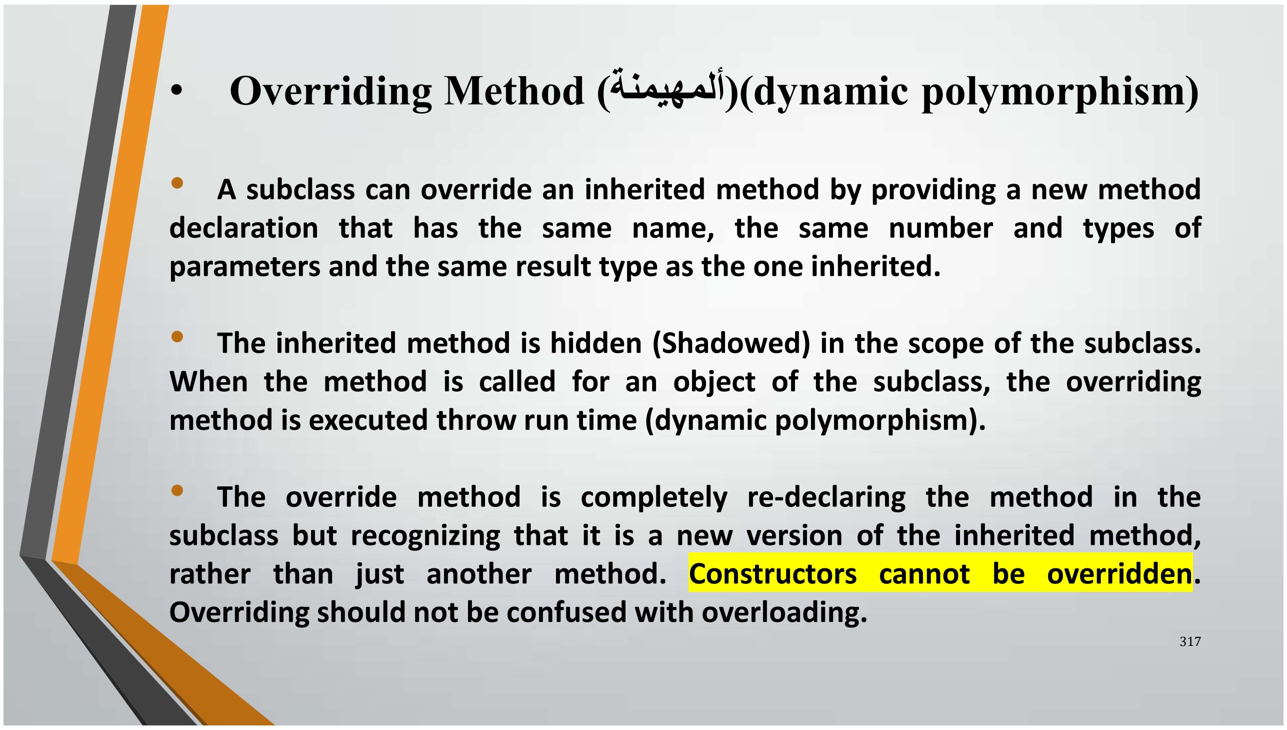
Keyword	Visibility
private	Access to a private variable or method is only allowed within the code fragments of a class.
pro-tected	Access to a protected variable or method is only allowed within the code fragments of a class and its subclass.
(friendly)	Access to a friendly variable or method (with no access specifier) is only allowed within the code fragments of a class and any other class in the same package.
public	Access to a public variable or method is unrestricted. It may be accessed from the code fragments of any class.

Override method

- **Definition**
- **Rules**
- **The Special Variable super and this for method calling**
- **Examples (graphical , tracing)**


Student
Roll no.
Name
Hey! I am Ting and my roll no. is 1.
I am Foo. My roll no. is 2.
I am Too and my roll no. is 5.

[redacted]



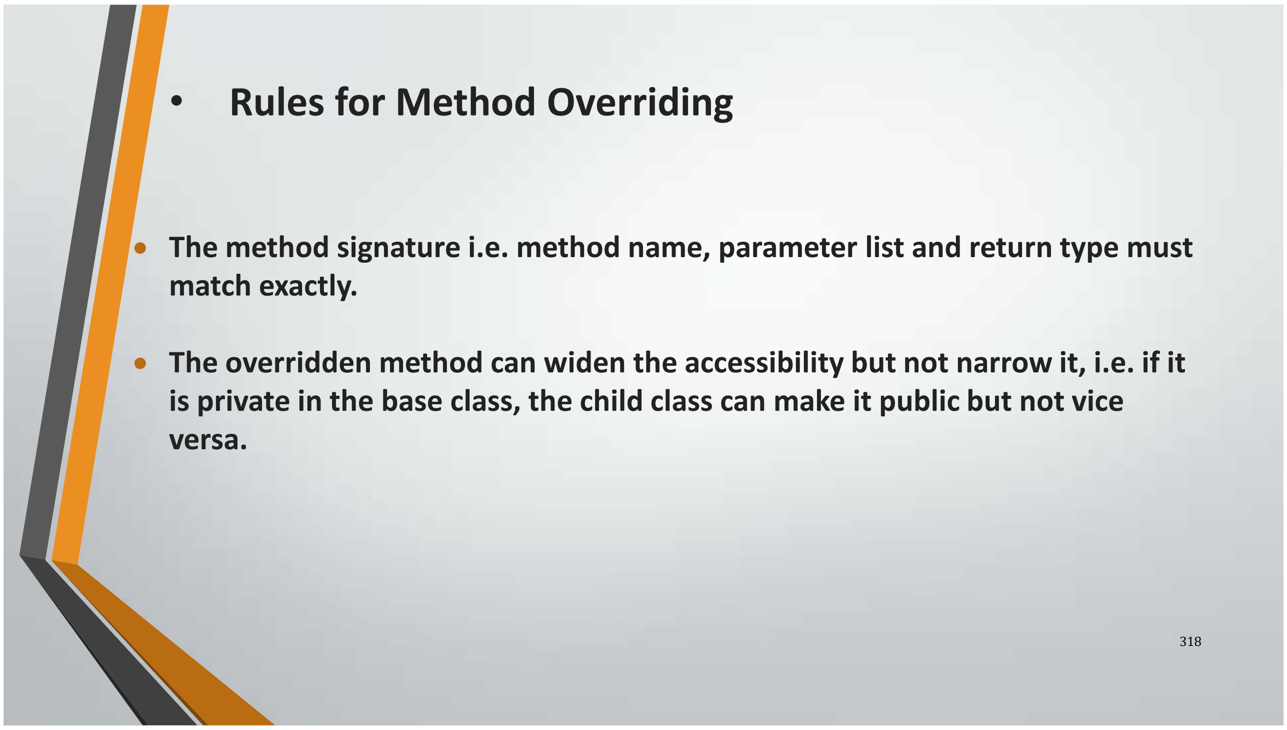
• Overriding Method (ألمهيمنة)(dynamic polymorphism)

- A subclass can override an inherited method by providing a new method declaration that has the same name, the same number and types of parameters and the same result type as the one inherited.

- The inherited method is hidden (Shadowed) in the scope of the subclass. When the method is called for an object of the subclass, the overriding method is executed throw run time (dynamic polymorphism).

- The override method is completely re-declaring the method in the subclass but recognizing that it is a new version of the inherited method, rather than just another method. Constructors cannot be overridden. Overriding should not be confused with overloading.

317



• Rules for Method Overriding

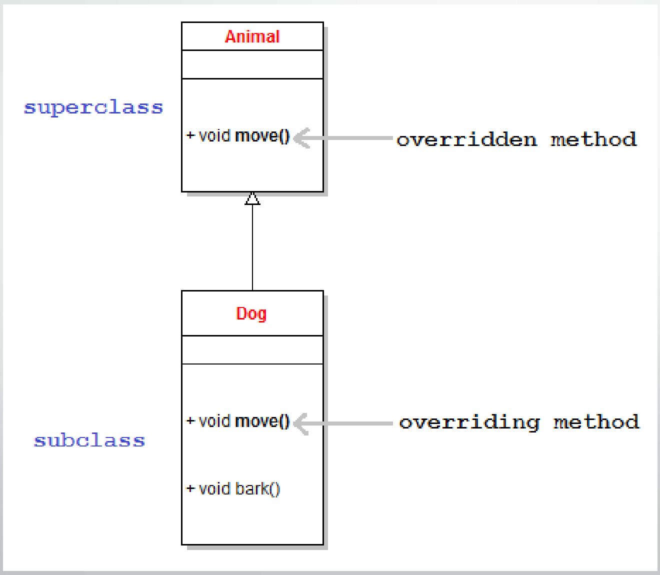
- The method signature i.e. method name, parameter list and return type must match exactly.

- The overridden method can widen the accessibility but not narrow it, i.e. if it is private in the base class, the child class can make it public but not vice versa.

318

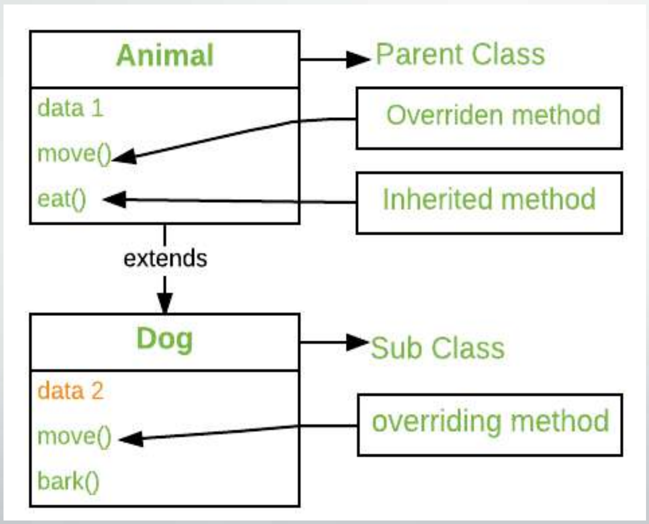
[redacted]

- Graphical example 1



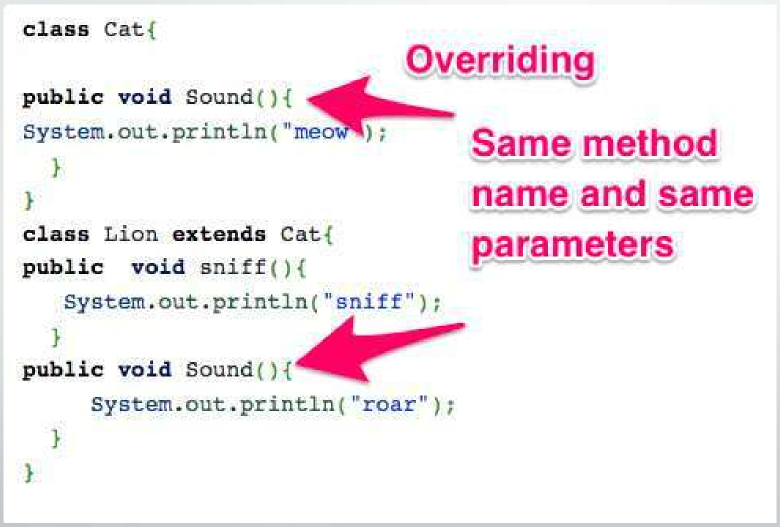
319

- Graphical example 2



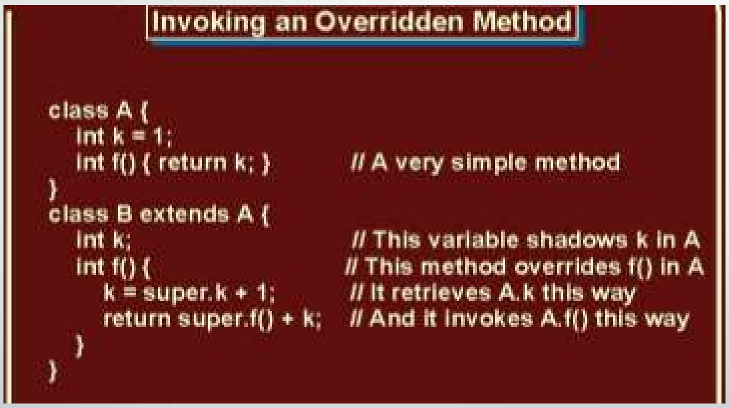
320

Example 3



```
class Cat{

public void Sound(){
System.out.println("meow");
   }
}
class Lion extends Cat{
public  void sniff(){
   System.out.println("sniff");
   }
public void Sound(){
     System.out.println("roar");
   }
}
```

- ## Example 4 (Explanation

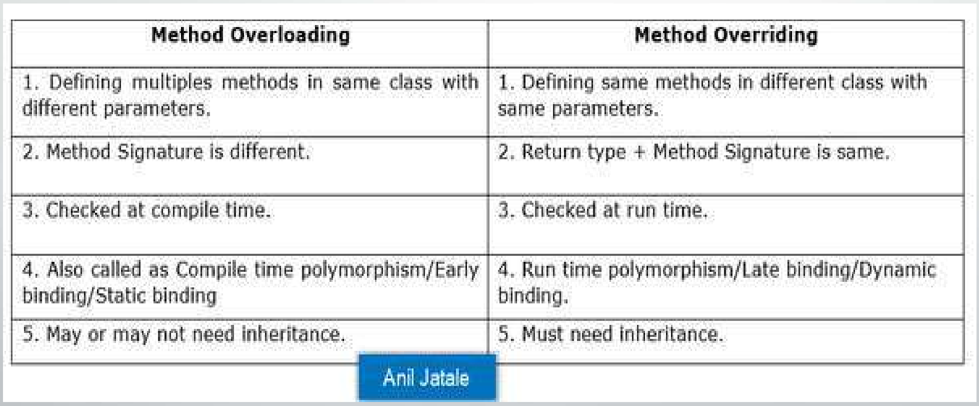


Invoking an Overridden Method

```
class A {
   int k = 1;
   int f() { return k; }          // A very simple method
}
class B extends A {
   int k;                         // This variable shadows k in A
   int f() {                      // This method overrides f() in A
      k = super.k + 1;            // It retrieves A.k this way
      return super.f() + k;       // And it invokes A.f() this way
   }
}
```

[redacted]

• The differences between method overloading and overriding



Method Overloading	Method Overriding
1. Defining multiples methods in same class with different parameters.	1. Defining same methods in different class with same parameters.
2. Method Signature is different.	2. Return type + Method Signature is same.
3. Checked at compile time.	3. Checked at run time.
4. Also called as Compile time polymorphism/Early binding/Static binding	4. Run time polymorphism/Late binding/Dynamic binding.
5. May or may not need inheritance.	5. Must need inheritance.

• Example 5 (Tracing)

```
public class One {
protected int a,b;
public void set(){    a=50;b=500;}
public void print(){    System.out.println(a+"   "+b);}  }

public class Two extends One{
   private int x,y;
   public void set(){        super.set();         x=100; y=200;   }
   public void print(){        super.print();  System.out.println(x+"   "+y);    }   }

public class Main {
public static void main(String[] args) {
Two t=new Two();   t.set();      t.print();}  }
```

The output:

```
50   500        ☺
100   200
```

When the super method call for both print and set methods are cancelled the output will be:

```
  100   200   ☺
```

[redacted]

• Note1

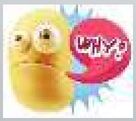
- *Private methods cannot be overridden, so a matching method declared in a subclass is considered separate. Set and print methods in One class are not overridden).*

• Example 6 (Tracing)

```
public class One {
protected  int a,b;
private void set(int a,int b){     this.a=a;this.b=b;}
private void print(){      System.out.println(a+"    "+b);} }

public class Two extends One{
    private int x,y;
   public  void set(int x,int y){      this.x=x;this.y=y;            }
    public void print(){       set(100,200); System.out.println(x+"    "+y);    }}

public class Main {
         public static void main(String[] args) {
    Two t=new Two();      t.print();} }
```

The output:

```
100   200
```

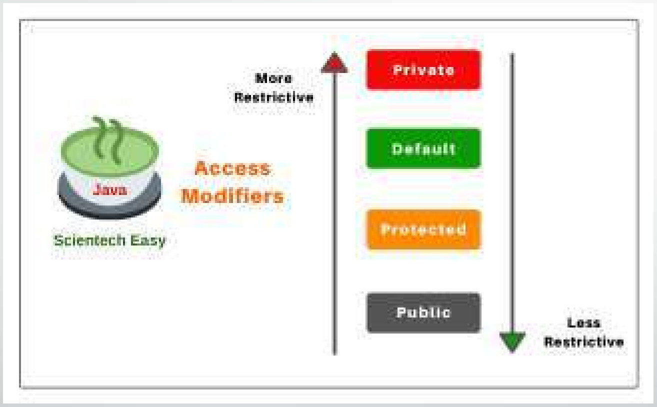


- Note2

- If we override the set method and set the modifier to "protected" in sub class while it is public in super class, then the compiler will detect an error. Access to the overridden method using <u>public, protected or the default</u> if no modifier, must be either the same as that of the super class method or made more accessible (change it from protected to public).

- An overriding method cannot be made less accessible (e.g change from public to protected). Static method cannot be overridden. Instance methods cannot be overridden by static method. The final instance method cannot be overridden also a final static method cannot be re-declared in a subclass(<u>this point will be explained later</u>).

327



[redacted]

• Example 7 (Tracing)

```
public class One {
protected  int a,b;

protected void set(int a,int b){     this.a=a;this.b=b;}

protected void print(){     System.out.println(a+"   "+b);} }

public class Two extends One{
   private int a,b;
  public  void set(int a,int b){ this.a=a;this.b=b;  ((One)this).a=10;     //or super.a=10;
 super.b=20;}

   public void print(){
      this.set(100,200);          //or  set(100,200);
      super.print();          System.out.println(a+"   "+b);     }}

   public class Main {
   public static void main(String[] args) {
   Two t=new Two();      t.print();}}
```

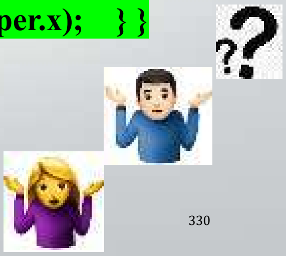
The output:

```
10 20
100 200
```

329

• Example 8 (Tracing) Test1

```
public class One {
protected int x,y;
public One(int a,int b){ x=a;y=b;       System.out.println("One"+x+"   "+y);}
 public One(){ System.out.println("One One");}   }

public class Two extends One{
   private int x;
   public Two(){         super(3,4);           x=100;
             System.out.println("Two"+x+"   "+y+"    "+super.x);
             System.out.println("Two"+this.x+"   "+y+"    "+super.x);    } }
public class Main {
      public static void main(String[] args) {
   Two t=new Two();      t.print();} }
```



What is the output?

330

[redacted]

- ## Example 9 (Tracing) Test2

```
public class One {
protected int a,b;
public void set(){     a=50;b=500;}
public void set(int c,int d){a=c;b=d;}
public void print(){     System.out.println(a+"    "+b);}  }

public class Two extends One{
   private int x,y;
   public void set(){        super.set();          x=100; y=200;    }
  public void set(int m1,int m2,int m3,int m4){a=m1;b=m2;x=m3;y=m4;}
   public void print(){         super.print();  System.out.println(x+"    "+y);     }    }

public class Main {
public static void main(String[] args) {
Two t=new Two();   t.set(10,20,30,40);      t.print();}  }
```

The output:

```
10   20
30   40
```

Correct or Not? Yes it is

[redacted]

static keyword in Java

- **Static variable**
- **Static method**
- **Static block and class**
- **Examples (tracing programs)**

- **Static variable**

- **Static variable** in **Java** is **variable** which belongs to the class and initialized only once at the start of the execution.
- It is a **variable** which belongs to the class and not to object(instance).
- **Static variables** are initialized only once, at the start of the execution.
- It could be accessed by className.variableName (if it is public for example),so it is named class variable.

[redacted]

Key Features of Static Variables:

1.Declared with the static keyword inside a class but outside any method, constructor, or block.

2.Shared among all instances of the class (i.e., a single copy exists for the entire class).

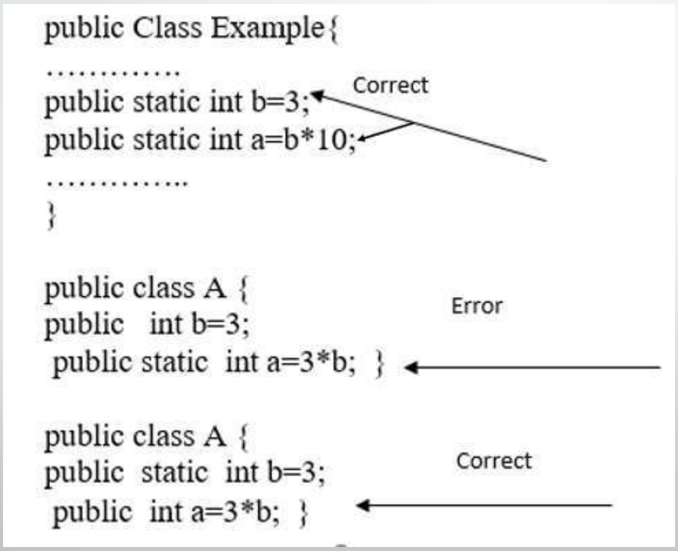
3.Allocated memory once at the time of class loading.

4.Can be accessed using the class name (e.g., ClassName.variableName).

5.Default value: If uninitialized, it takes the default value based on its data type (e.g., 0 for integers, null for objects).

6.Useful for constants and properties shared across all instances.

- **Example1 (Explanation)**



```
public Class Example{
.............
public static int b=3;
public static int a=b*10;
..............
}

public class A {
public   int b=3;
 public static  int a=3*b;  }

public class A {
public  static  int b=3;
 public  int a=3*b;  }
```

Correct
Error
Correct

336

• Static method

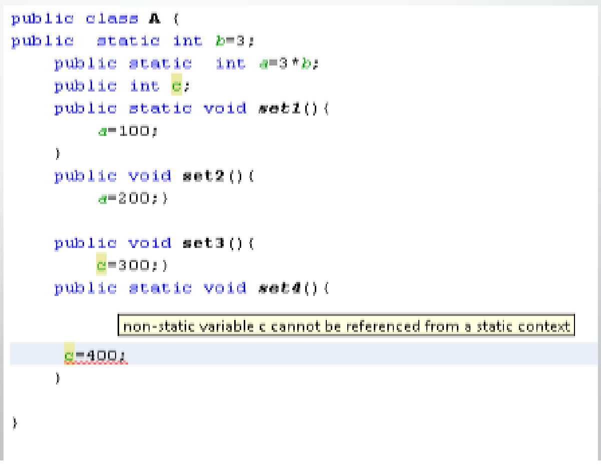
- **Static method** in **Java** is a **method** which belongs to the class and not to the object.
- A **static method** can access only **static** data and cannot access non-**static** data (instance variables).
- A **static method** can call only other **static methods** and can not call a non-**static method** from it.
- It could be invoked using **classname.methodname**(), it named class method, like random method in Math class and possible to invoke it using object too.

337

Example 2 (Explanation

```
public class C {
private int n=0;
public static int s=0;
public int getn() {  return (n);  //OK }
public int gets() { return s;  //OK }
public  static  int  get2n()  {  return  n;  //ERROR!!  }  }
WHY?!!!
```

Example 3 (Explanation)



```
public class A {
public  static int b=3;
    public static  int a=3*b;
    public int c;
    public static void set1(){
        a=100;
    }
    public void set2(){
        a=200;}

    public void set3(){
        c=300;}
    public static void set4(){

           non-static variable c cannot be referenced from a static context
      c=400;
    }

}
```

Example 4 (Trace):

```
public class Demo{
public static void main(String args[]){
Student s1 = new Student();
System.out.print("object s1   ");   s1.showData();
s1.increment();
System.out.print("object s1   ");s1.showData();

 Student s2 = new Student();
 System.out.print("object s2   ");   s2.showData();
 s2.increment();
System.out.print("object s2   "); s2.showData();
s2.increment();
System.out.print("object s2   ");s2.showData();
Student.b++;
System.out.print("object s2   "); s2.showData();
 System.out.print("object s1   "); s1.showData();
 }}
```

The output:

```
object s1   Value of a = 1   Value of b = 1
object s1   Value of a = 2   Value of b = 2
object s2   Value of a = 1   Value of b = 3
object s2   Value of a = 2   Value of b = 4
object s2   Value of a = 3   Value of b = 5
object s2   Value of a = 3   Value of b = 6
object s1   Value of a = 2   Value of b = 6
```

```
public class Student {
 private int a;
public static int b;
  Student(){ b++;a++; }

   public void showData(){
      System.out.print("Value of a = "+a+"    ");
      System.out.print("Value of b = "+b);
      System.out.println();   }

public void increment(){ a++; b++;} }
```

Object in memory	a	b
s1	~~1~~ 2	~~1 2 3 4 5~~ 6
s2	~~1 2~~ 3	

339

Static block

- The **static block** is a block of statement inside a Java class that will be executed when a class is first loaded into the JVM.
- A **static block helps to initialize the static data members**, just like constructors help to initialize instance members.

```
public class Test{
static { //Code goes here }
}
```

340

Example 5 (Trace: How to access static block)

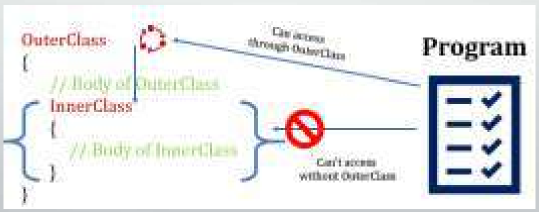
```
public class Demo {
 public static int a;
 public static int b;
 static {
   a = 10;   b = 20; }
}
public class Main{
 public static void main(String args[]) {
Demo dm=new Demo();
  System.out.println("Value of a = " +dm. a);
  System.out.println("Value of b = " +dm. b);

System.out.println("Value of a = " +Demo. a);
  System.out.println("Value of b = " +Demo. b);    }}
```

Output:

```
Value of a = 10
Value of b = 20
```

Static class

- **Can a class be static in Java? The answer is Yes, some classes can be made static in Java.**
- **Java supports Static Instance Variables, Static Methods, Static Block and Static Classes.**
- **Java allows a class to be defined within another class. These are called Nested Classes. The class in which the nested class is defined is known as the Outer Class. Unlike top level classes, Inner classes can be Static. Non-static nested classes are also known as Inner classes.**
- **The nested classes will be explained later.**



• Static method with inheritance

- The static method cannot be overriding but redefining (redeclaring).
- The static method can be redefining (redeclaring).
- An instance method cannot override a static method, and a static method cannot hide an instance method.

• Example 6 (Trace)

```
public class Scott {
public static void abc() { System.out.println("aaa"); } }

public class Group extends Scott {
public static void abc() {
~~super.abc();~~
System.out.println("zzz");}}

public class Main{
public static void main(String[] args){
Group.abc();  }  }
```

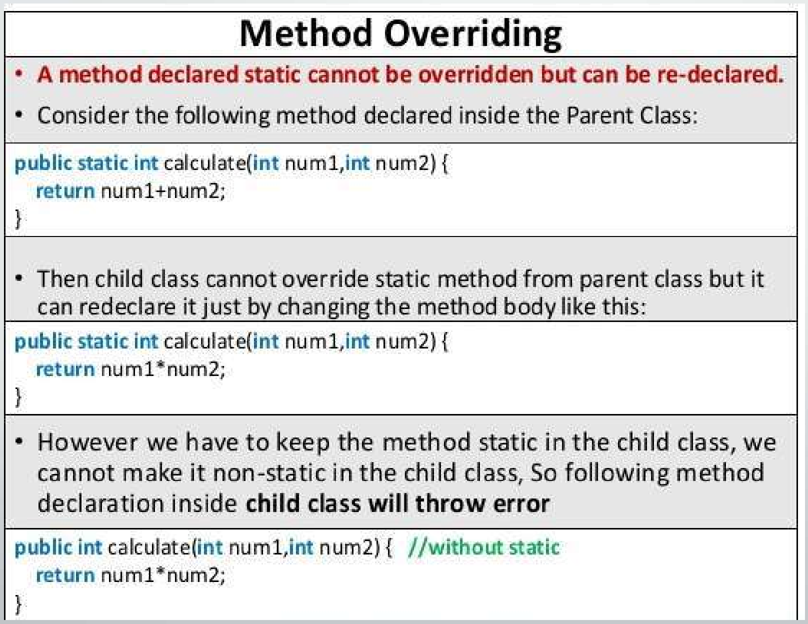
The output:

```
zzz
```

- QUIZ

Q1:What are the differences between method overriding and redefining (redeclaring) in java?

[redacted]



Method Overriding

- A method declared static cannot be overridden but can be re-declared.
- Consider the following method declared inside the Parent Class:

```
public static int calculate(int num1,int num2) {
    return num1+num2;
}
```

- Then child class cannot override static method from parent class but it can redeclare it just by changing the method body like this:

```
public static int calculate(int num1,int num2) {
    return num1*num2;
}
```

- However we have to keep the method static in the child class, we cannot make it non-static in the child class, So following method declaration inside **child class will throw error**

```
public int calculate(int num1,int num2) {   //without static
    return num1*num2;
}
```

Trace the following java code

- **Example 7 (Trace)**

```
public  class One {
   protected int x,y;
   public static void f1(){  System.out.println("static f1 from class One");}
   public void f3(){  System.out.println("f3 from class One");     }    }

public class Two extends One {
   private int a;
   public static void f1(){        // super.f1();
      System.out.println("redefining  method f1 from class Two");    }
      public void f3(){System.out.println("method overriding from class Two");
     super.f3();   } }

public class Main {
public static void main(String[] args) {
Two t=new Two();
 t.f1();    t.f3()
One.f1();   Two.f1();     } }
```

Does the output correct or not and why?

```
redefining  method f1 from class Two
method overriding from class Two
f3 from class One
static f1 from class One
redefining  method f1 from class Two
```

346

[redacted]

• Using static with array of objects • Example 8(Trace)

```
public class A {
 public static int x;
public A(){     x=x+1; }
public void print (){     System.out.println("printing x using object "+"x="+x);} }

public class Main {
  public static void main(String[] args) {

            //A.x=100;

     A a1=new A();          a1.print();
     A a2=new A();         a2.print();
     System.out.println("printing x using class "+A.x);

     A a_array[]=new A[5];
     for(int i=0;i<5;i++){
     a_array[i]=new A();  }

System.out.println("printing x using class "+A.x);
a_array[0].print();
a_array[3].print(); } }
```

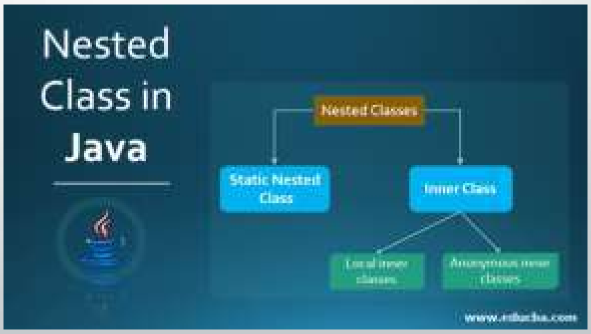
The output: (Correct or not?)

```
printing x using object x=1
printing x using object x=2
printing x using class 2
printing x using class 7
printing x using object x=7
printing x using object x=7
```

347

Nested classes

- Outer class
- Inner class
- Static   class
- Examples (tracing programs)



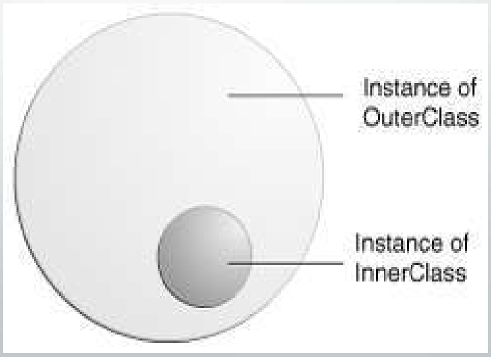
[redacted]

• Nested class

The Java programming language allows you to define a class within another class which is called nested class.
The following java segment illustrates the nested class declaration in java:

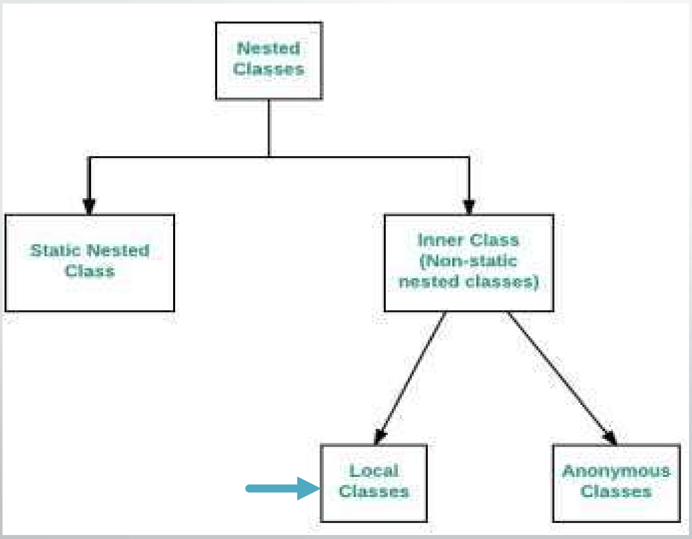
Example 1

```
public class OuterClass {    ...
public class inner {       ...   }
                  }
```



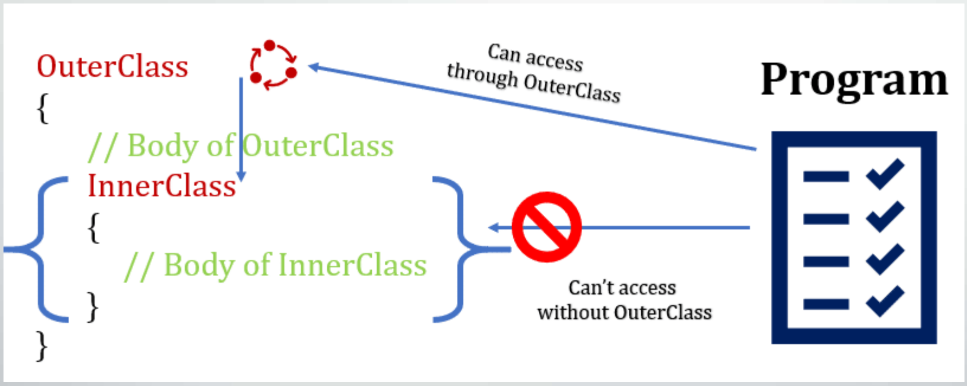
• Very important note (Terminology):

Nested classes are divided into two categories



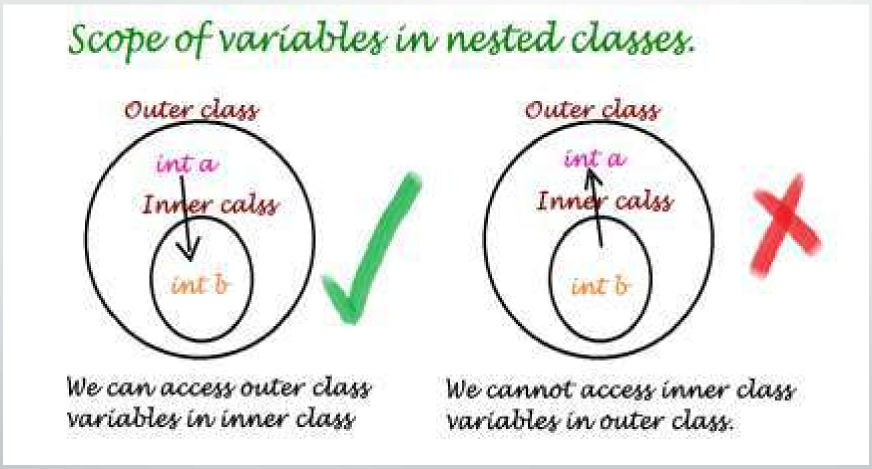
• Inner class

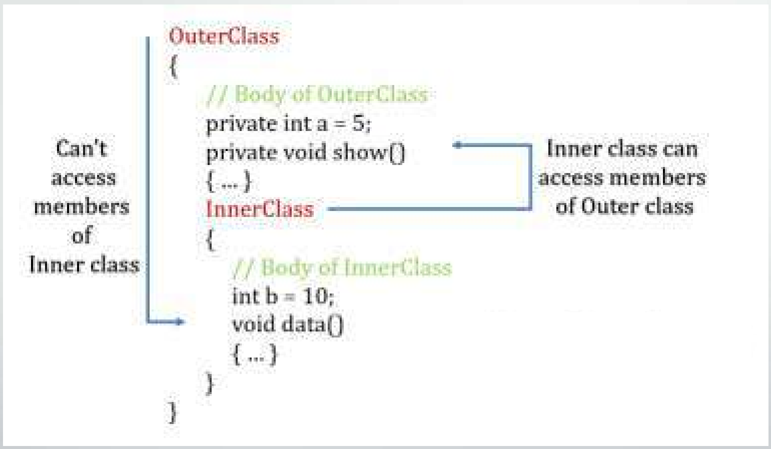
- The inner class is known as a *member class.*
- *Is* another class component in the same way that constants, variables, and methods are also class components.
- As with instance methods and variables, an inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields.
- Also, because an inner class is associated with an instance, it cannot define any static members itself.
- Objects that are instances of an inner class exist *within* an instance of the outer class.



[redacted]

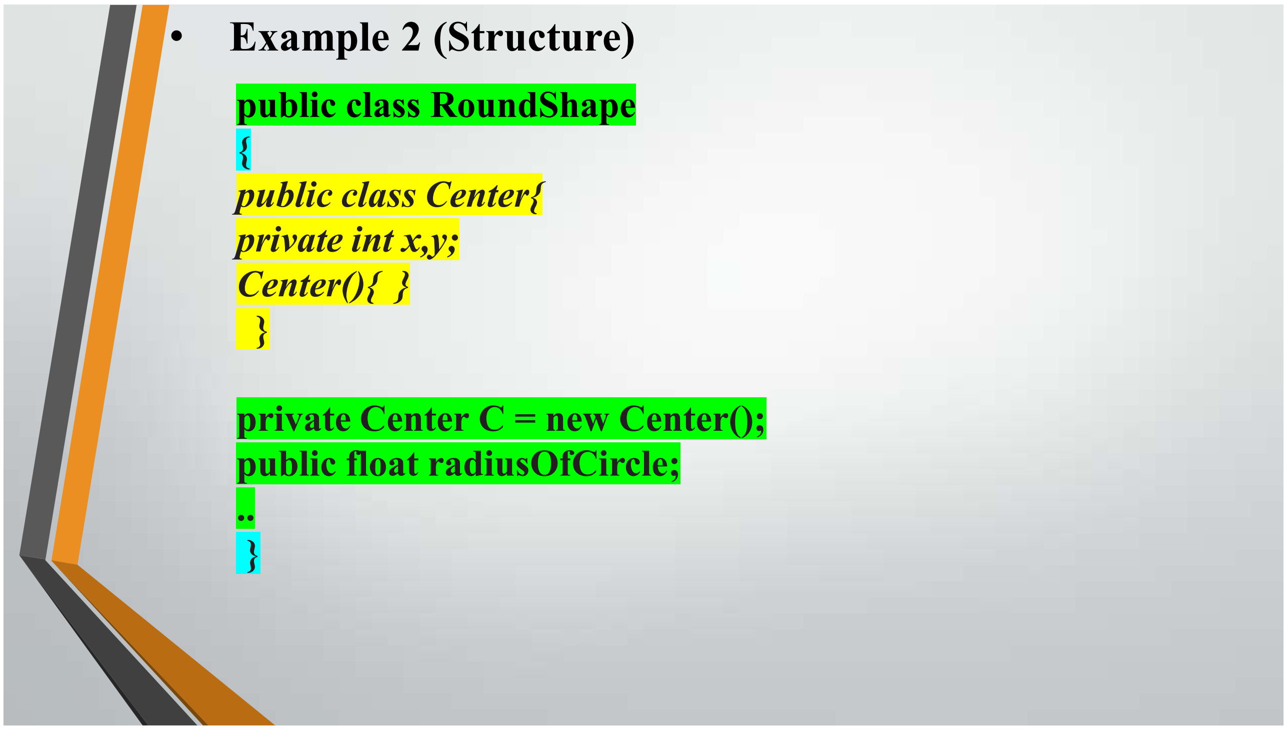
- Access modifiers


Scope of variables in nested classes.
Outer class
int a
Inner calss
int b
Outer class
int a
Inner calss
int b
We can access outer class variables in inner class
We cannot access inner class variables in outer class.


Can't access members of Inner class
Inner class can access members of Outer class

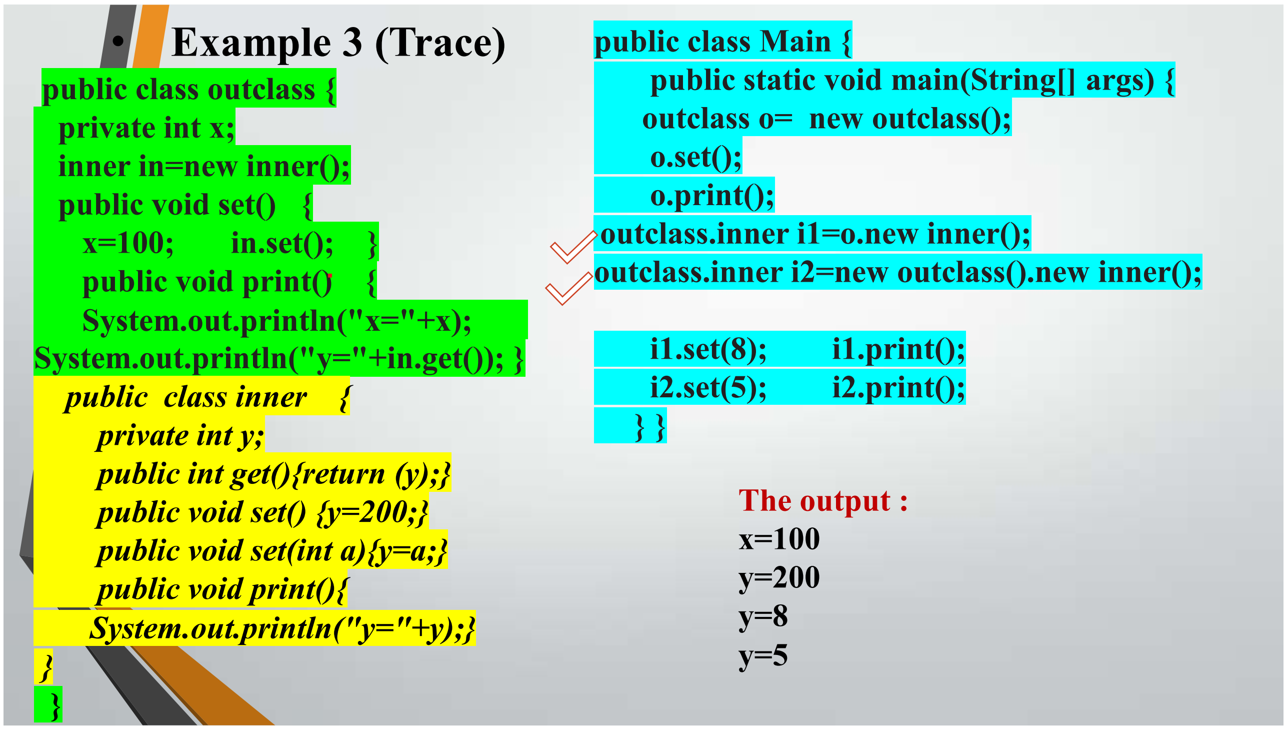
```
OuterClass
{
    // Body of OuterClass
    private int a = 5;
    private void show()
    { ... }
    InnerClass
    {
      // Body of InnerClass
      int b = 10;
      void data()
      { ... }
    }
}
```

[redacted]



Example 2 (Structure)

```
public class RoundShape
{
public class Center{
private int x,y;
Center(){ }
  }

private Center C = new Center();
public float radiusOfCircle;
..
}
```



Example 3 (Trace)

```
public class outclass {
  private int x;
  inner in=new inner();
  public void set()   {
     x=100;      in.set();   }
     public void print()    {
     System.out.println("x="+x);
System.out.println("y="+in.get()); }
   public  class inner    {
       private int y;
       public int get(){return (y);}
       public void set() {y=200;}
       public void set(int a){y=a;}
       public void print(){
      System.out.println("y="+y);}
}
}
```

```
public class Main {
       public static void main(String[] args) {
      outclass o=  new outclass();
       o.set();
       o.print();
✓ outclass.inner i1=o.new inner();
✓ outclass.inner i2=new outclass().new inner();

       i1.set(8);        i1.print();
       i2.set(5);        i2.print();
     } }
```

The output :

```
x=100
y=200
y=8
y=5
```

[redacted]

```
public class JavaApplication24 {

   public static void main(String[] args) {
      Outer o=  new Outer();
      o.set();         o.print();

      Outer.inner i1=o.new inner();
i1.set(8);       i1.print();   } }
```

The output
```
x=100
y=200
y=8
x=1000
```

Scope of variables in nested classes
Outer class
Outer class
We can access outer class variables in inner class
We cannot access inner class variables in outer class

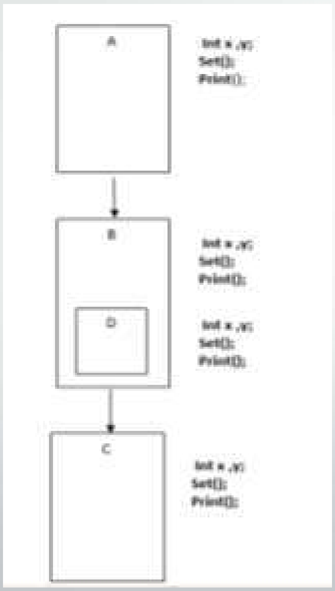
```
public class Outer {
   private int x;
  inner in=new inner();
  public void set()   {
 x=100;
✕ y=2000;
 in.set();    }
public void print(){
   System.out.println("x="+x);
System.out.println("y="+in.get()); }
   public  class inner    {
      private int y;
      public int get(){return (y);}
      public void set() {y=200;}
      public void set(int a){y=a;}
      public void print(){
     System.out.println("y="+y);x=1000;
 ✓    System.out.println("x="+x);  }   }   }
```

- Example 4 (Trace)

```
public  class Test {
public int num = 25;
public int getNum()
{  return num; }

   public class MyInner {
   public int num = 38;
    public int getNum() { return num; }
   } }
public class Main{
public static void main(String[] args) {
Test  abt = new Test();
//OuterClass.InnerClass  innerObject = outerObject.new InnerClass();
Test.MyInner minn =abt.new MyInner();
System.out.println(abt.num);
System.out.println(minn.num);} }
```

The output:
```
25
38
```

- **H.W. LAB In class main find the summation of all x's and y's**



• Static nested class

- A **static nested class** is a regular class defined inside of a package level class or inside of another static nested class.
- They are actually defined inside the body of the parent class, not only in the same file.
- As with any high level facility offered by a programming language it can be of real help in structuring clear programs or it can be just the opposite of this when abused.

- **Static nested class** facts:
  - is defined as a static member of the parent class
  - accepts all accessibility modifiers
  - it is NOT linked to an outer instance (it can live independently)
  - has direct access to static members of the parent class regardless of the access modifiers declared in the parent class
  - has direct access to all members of an instance of the parent class regardless of the access modifiers declared in the parent class
- Here is a brief example of how nested classes are declared and how they access members of the parent classes.

- Example 5

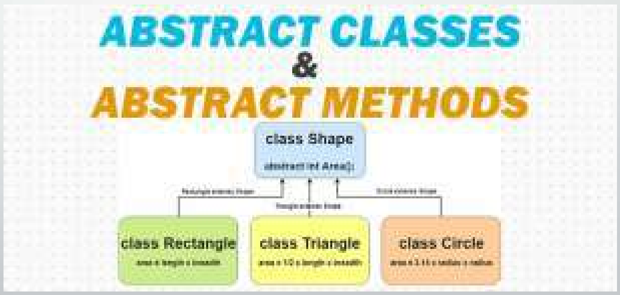
```
public class Top{
 private static int staticCounter = 0;
private int nestedCounter = 0;

 public static class Nested1 {
 private static int staticCounter = 0;
 private int nestedCounter = 0;
 public static class Nested2 {
 public Nested2(Top t, Top.Nested1 tn1) {
  Top.staticCounter++;
 t.nestedCounter++;
  Top.Nested1.staticCounter++;
 tn1.nestedCounter++;  }  }
```

```
public Nested1(Top t) {
 Top.staticCounter++;
 t.nestedCounter++; }

 public String toString() {
 return
 getClass().getName() + ".nestedCounter: " + nestedCounter +
 System.getProperty("line.separator") +
 getClass().getName() + ".staticCounter: " + staticCounter;
 } }
  public String toString() {
 return
 getClass().getName() + ".nestedCounter: " + nestedCounter +
 System.getProperty("line.separator") +
 getClass().getName() + ".staticCounter: " + staticCounter;
 }
  public static void main(String[] args) {
 Top t = new Top();
 Top.Nested1 nested1 = new Top.Nested1(t);
 Top.Nested1.Nested2 nested2 = new Top.Nested1.Nested2(t, nested1);
  System.out.println(t);
 System.out.println(nested1);
 }
```

[redacted]

Abstract class and Dynamic binding

- Dynamic polymorphism
- Dynamic binding
- Abstract class and abstract method
- Examples (tracing and writing programs)

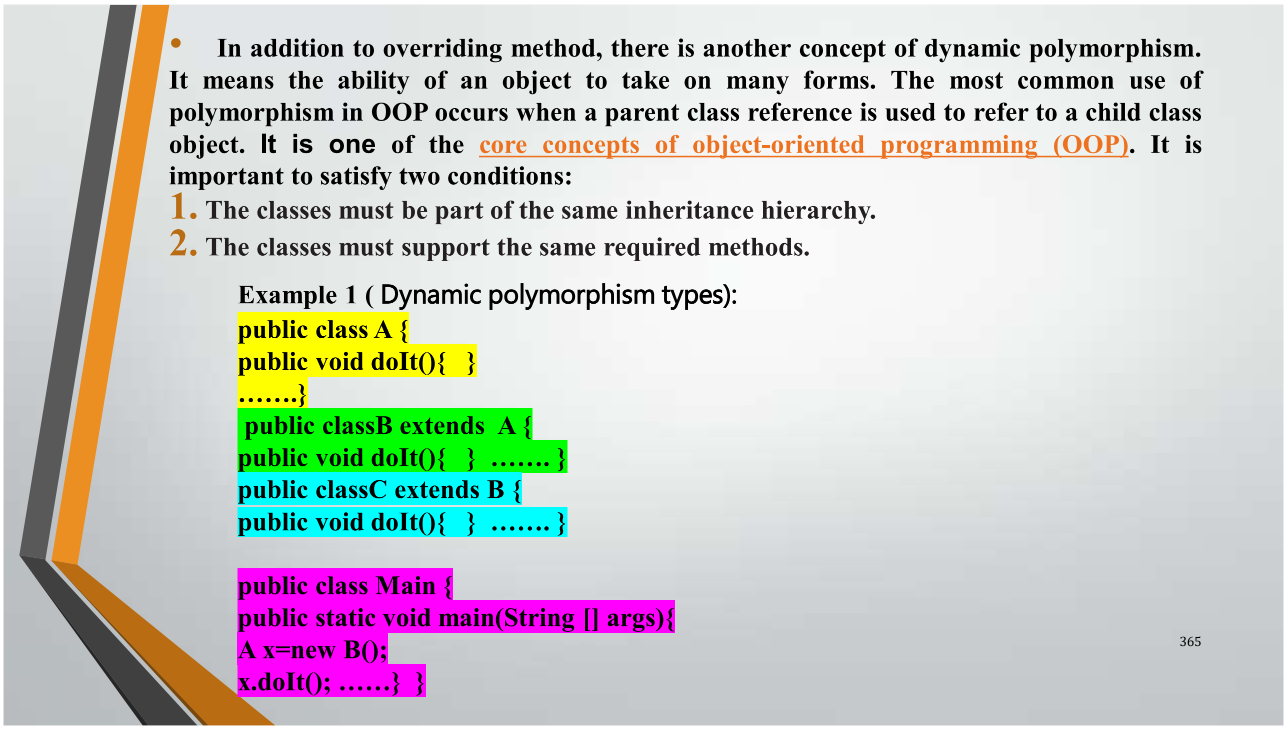


- Polymorphism in OOP as explained previously

In the previous lectures we discussed Polymorphism in Java. In this lecture we will see types of polymorphism. There are two types of polymorphism in java:
1- Static Polymorphism also known as compile time polymorphism.
2- Dynamic Polymorphism also known as runtime polymorphism.

Method Overloading in Java – This is an example of ***compile time(or static polymorphism)*** and it has been discussed previously.
Method Overriding in Java – This is an example of ***run time (or dynamic polymorphism).***

[redacted]

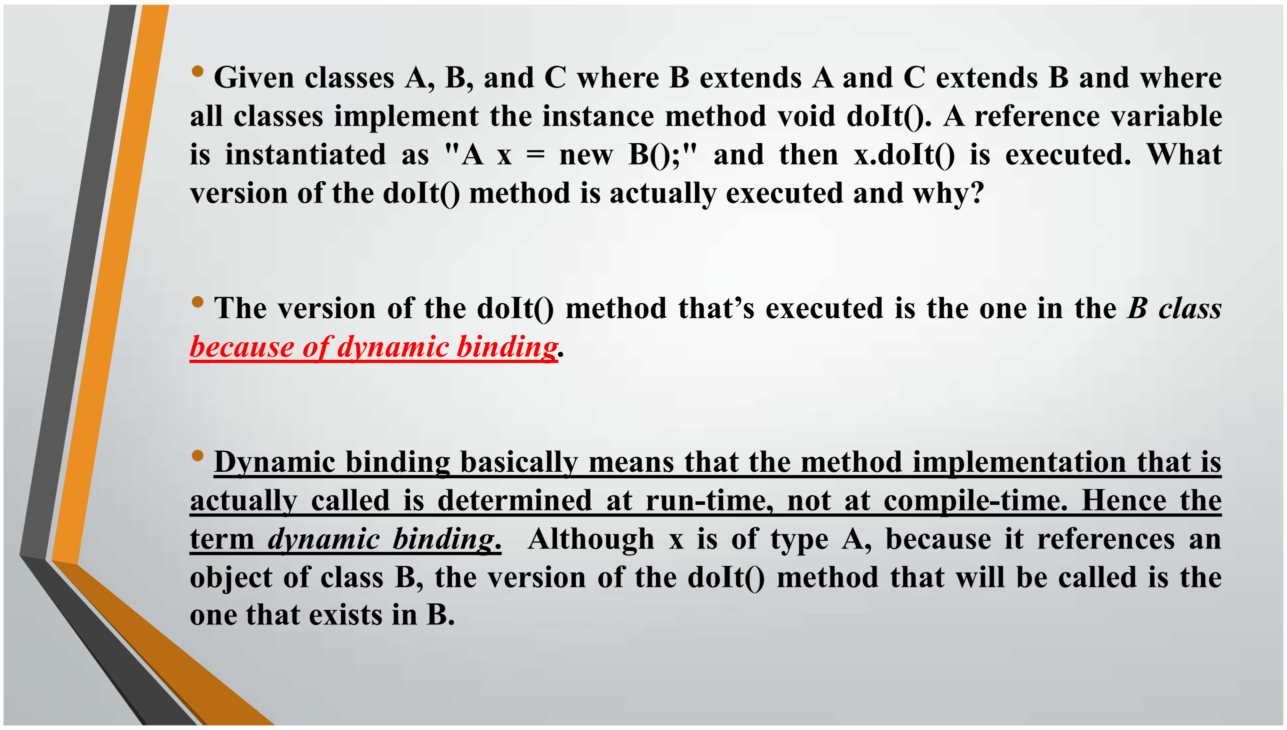


- In addition to overriding method, there is another concept of dynamic polymorphism. It means the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. It is one of the core concepts of object-oriented programming (OOP). It is important to satisfy two conditions:

1. The classes must be part of the same inheritance hierarchy.
2. The classes must support the same required methods.

Example 1 (Dynamic polymorphism types):

```
public class A {
public void doIt(){   }
…….}
 public classB extends  A {
public void doIt(){   }  ……. }
public classC extends B {
public void doIt(){   }  ……. }

public class Main {
public static void main(String [] args){
A x=new B();
x.doIt(); ……}  }
```



- Given classes A, B, and C where B extends A and C extends B and where all classes implement the instance method void doIt(). A reference variable is instantiated as "A x = new B();" and then x.doIt() is executed. What version of the doIt() method is actually executed and why?

- The version of the doIt() method that's executed is the one in the *B class* *because of dynamic binding*.

- Dynamic binding basically means that the method implementation that is actually called is determined at run-time, not at compile-time. Hence the term *dynamic binding*. Although x is of type A, because it references an object of class B, the version of the doIt() method that will be called is the one that exists in B.

[redacted]

Example 2 (Dynamic Polymorphism and Dynamic binding):

```
public class ABC{
  public void myMethod(){
     System.out.println("Overridden Method");    } }

public class XYZ extends ABC{
  public void myMethod(){
     System.out.println("Overriding Method");   }}
```

```
public class Main{
public static void main(String args[]){

ABC obj = new ABC();
obj.myMethod();
// This would call the myMethod() of parent class
ABC

XYZ obj = new XYZ();
obj.myMethod();
// This would call the myMethod() of child class
XYZ

ABC obj = new XYZ();
obj.myMethod();
// This would call the myMethod() of child class
XYZ   } }
```

Example 3 (Trace):

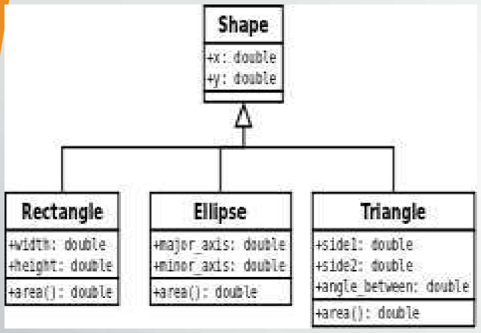
```
public class Animal{
  public void move(){ System.out.println("Animals can move"); } }

class Dog extends Animal{
  public void move(){    super.move(); // invokes the super class method
    System.out.println("Dogs can walk and run"); }}

public class TestDog{
  public static void main(String args[]){

Animal b = new Dog();
    b.move();//Runs the method in Dog class   }}
```

The output:

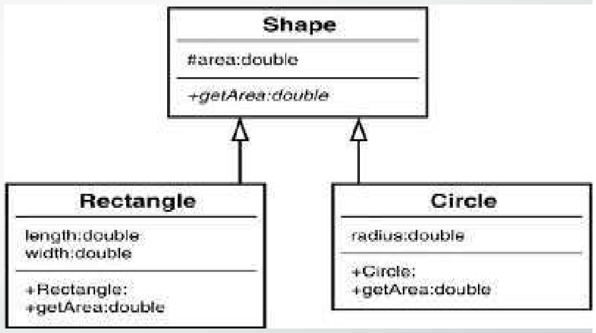
```
Animals can move
Dogs can walk and run
```

[redacted]

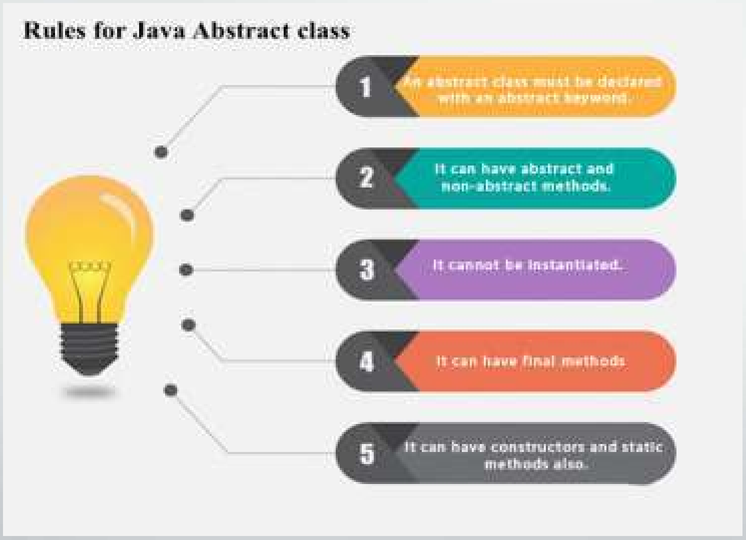
Case I



Case II



- **Abstract Class**



Rules for Java Abstract class

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

- **If a class is abstract and cannot be instantiated, the class does not have much use unless it has subclasses.**

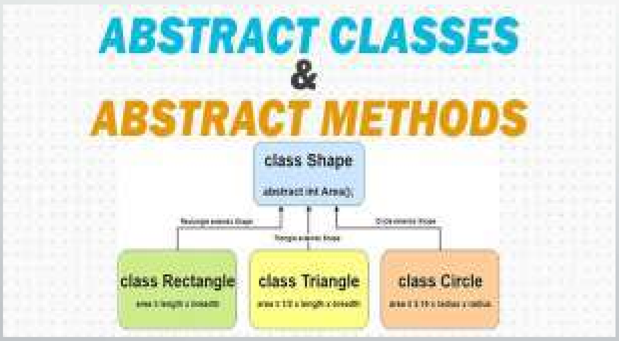
370

[redacted]

Abstract Method

•An abstract method is a method that is declared without an implementation (without {}), and followed by a semicolon, like this:

```
abstract int area();
abstract int sum(int x,int y);
abstract int mul(int ,int);
```



- Example 5 (writing   a program)

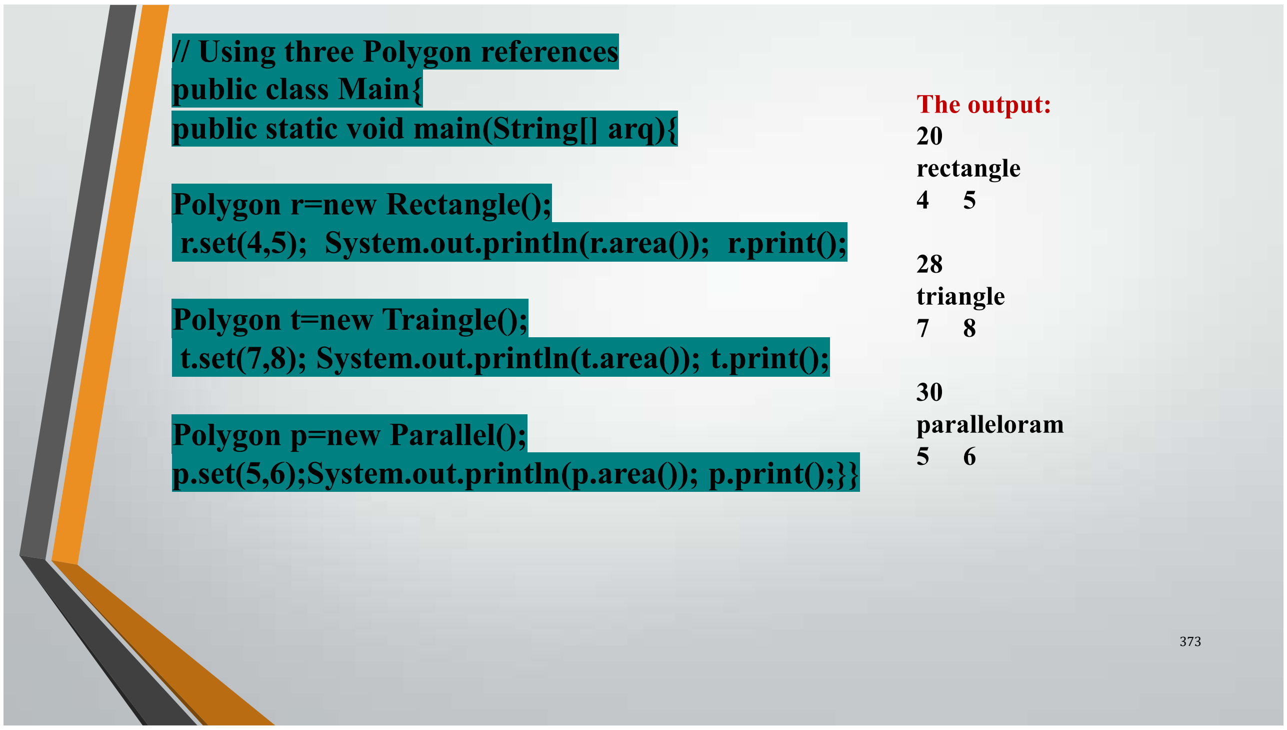
Define a base class called Polygon which has two integer attributes represent the dimensions of a Polygon. Set method is used for setting the dimensions. Derive three subclasses Rectangle, Triangle and Parallelogram. Write a main class to print the areas of the three Polygons using:

1: three polygon references   2:one polygon reference 3:using array of three objects.

```
public abstract class Polygon {
protected int d1,d2;
public  void set(int a ,int b){ d1=a;d2=b;}
public abstract int area();
public abstract void print(); }

public class Rectangle extends Polygon{
   public int area(){return (d1*d2);}
   public void print(){ System.out.println("rectangle"); System.out.println(d1+"          "+d2);}}

public class Triangle extends Polygon {
public int area(){return (d1*d2/2);}
  public void print(){ System.out.println("triangle");  System.out.println(d1+"             "+d2);}}

public class Parallel extends Polygon {
public int area(){return (d1*d2);}
public void print(){System.out.println("parallelogram");System.out.println(d1+"      "+d2);   }}
```

372

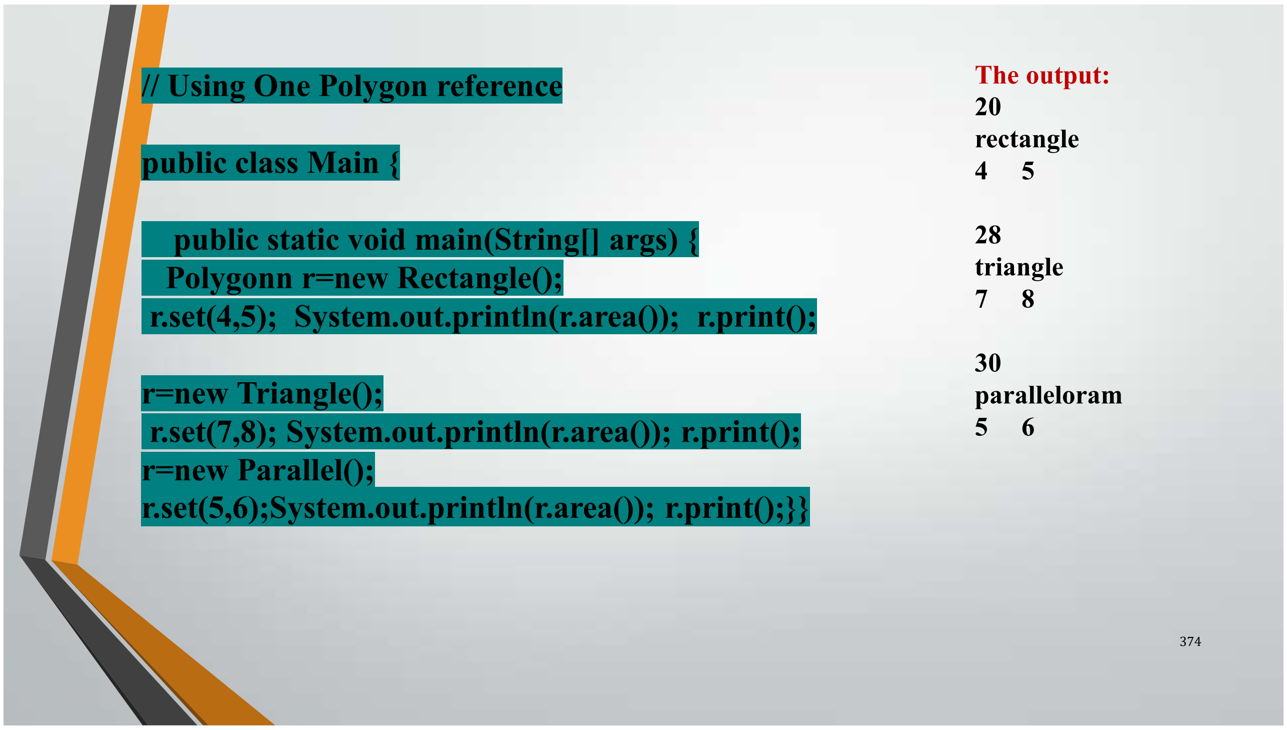
[redacted]



```
// Using three Polygon references
public class Main{
public static void main(String[] arq){

Polygon r=new Rectangle();
 r.set(4,5);  System.out.println(r.area());  r.print();

Polygon t=new Traingle();
 t.set(7,8); System.out.println(t.area()); t.print();

Polygon p=new Parallel();
p.set(5,6);System.out.println(p.area()); p.print();}}
```

The output:

```
20
rectangle
4     5

28
triangle
7     8

30
paralleloram
5     6
```



```
// Using One Polygon reference

public class Main {

    public static void main(String[] args) {
   Polygonn r=new Rectangle();
 r.set(4,5);  System.out.println(r.area());  r.print();

r=new Triangle();
 r.set(7,8); System.out.println(r.area()); r.print();
r=new Parallel();
r.set(5,6);System.out.println(r.area()); r.print();}}
```

The output:

```
20
rectangle
4     5

28
triangle
7     8

30
paralleloram
5     6
```

[redacted]

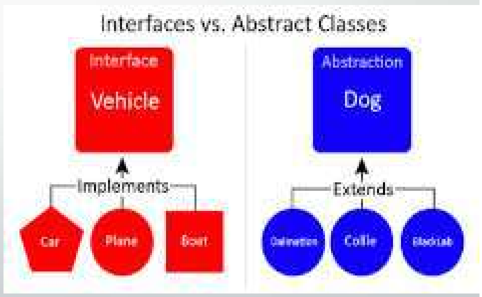
```
// Using array of  Polygons

public class Main{
public static void main(String[] args) {
 Polygon [] a=new Polygon[3];
  for (int i=0;i<3;i++)
     switch (i) {
         case 0: a[i]=new Rectangle();a[i].set(5,4);break;
         case 1: a[i]=new Triangle();a[i].set(7,8);break;
         case 2: a[i]=new Parallel();a[i].set(5,6);break;
     }
   for(int i=0;i<3;i++){
      System.out.println(a[i].area());
      a[i].print();}
  }
}
```

The output:

```
20
rectangle
4     5

28
triangle
7     8

30
paralleloram
5     6
```

Interface

- **Multiple inheritance**
- **Interface declaration**
- **Default method and tag interface**
- **Examples (tracing and writing programs)**



[redacted]

Interface Def.

- An interface is programming structure, is not a class but it is a blueprint of a class.
- its definition is similar to a class definition except that it uses the **interface** keyword.
- All methods in an interface are either *abstract methods or default method (java 8)*, that is, they are declared without the implementation part. They are to be implemented in the subclasses that use them.
- It can also include a static constant declaration.
- Writing an interface is like writing a class, but they are two different concepts:

A class describes the attributes and behaviors of an object while, an interface contains behaviors that a class implements. **Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.**

377

Interface and class (similarity)

- An interface is **similar** to a class in the following ways:
  - An interface can contain any number of methods.
  - An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
  - The bytecode of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name

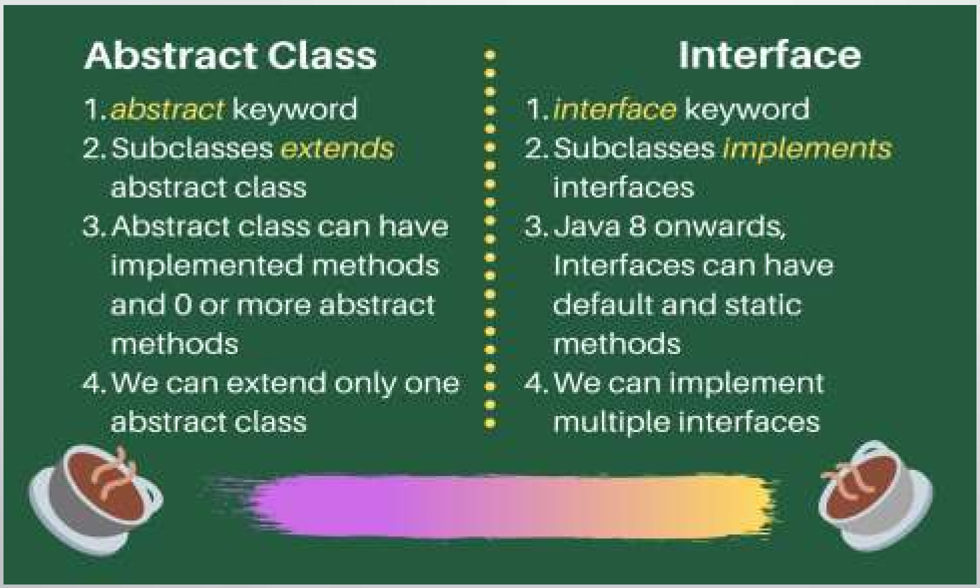
378

[redacted]

• Interface and class (differences)

- You cannot instantiate an interface.
- One public specifier is used.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.(except default java8)
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

379

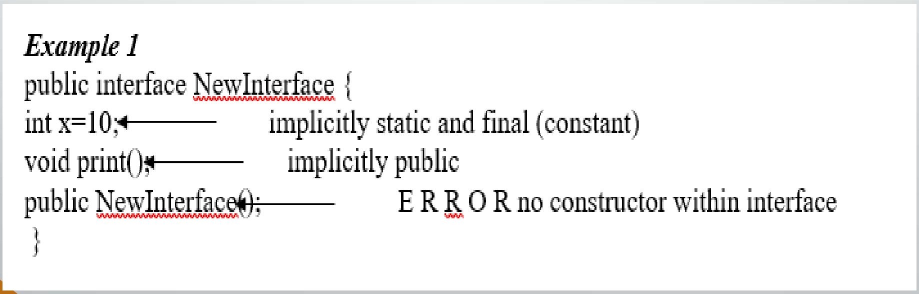


Abstract Class	Interface
1. *abstract* keyword	1. *interface* keyword
2. Subclasses *extends* abstract class	2. Subclasses *implements* interfaces
3. Abstract class can have implemented methods and 0 or more abstract methods	3. Java 8 onwards, Interfaces can have default and static methods
4. We can extend only one abstract class	4. We can implement multiple interfaces

[redacted]

• Declaring Interfaces

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface. Let us look at an example that depicts encapsulation:
Interfaces have the following properties:
•An interface is implicitly (ضمنيا) abstract. You do not need to use the **abstract** keyword when declaring an interface.
•Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
•Methods in an interface are implicitly public.
•No static methods within Interface.



Example 1

```
public interface NewInterface {
int x=10;←———        implicitly static and final (constant)
void print();←———        implicitly public
public NewInterface();———        E R R O R no constructor within interface
 }
```

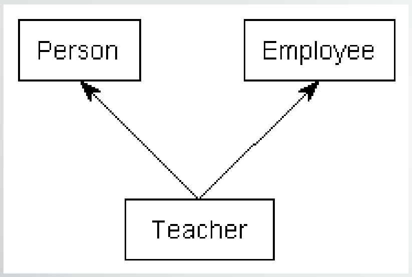
• Implementing interface

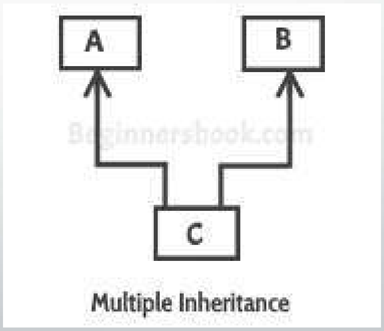
- A class uses the implements keyword to implement an interface.
- A class can implement more than one interface at a time.
- A class can extend only one class but implement many interfaces.
- An interface itself can extend another interface

- The implements keyword appears in the class declaration following the extends portion of the declaration.
- If a class does not define all the behaviors of the interface, the class must declare itself as abstract.

[redacted]

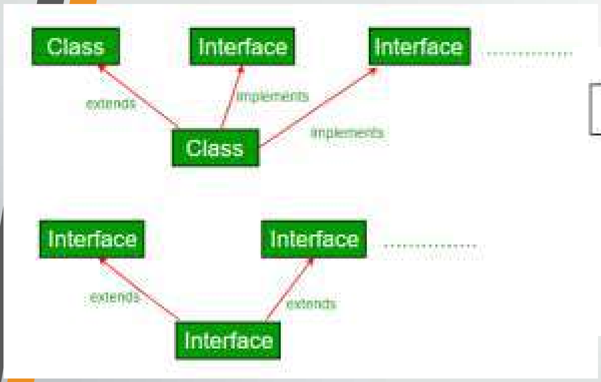
- **Multiple Inheritance**

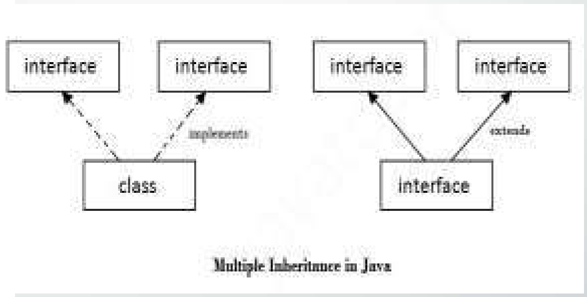




Multiple Inheritance

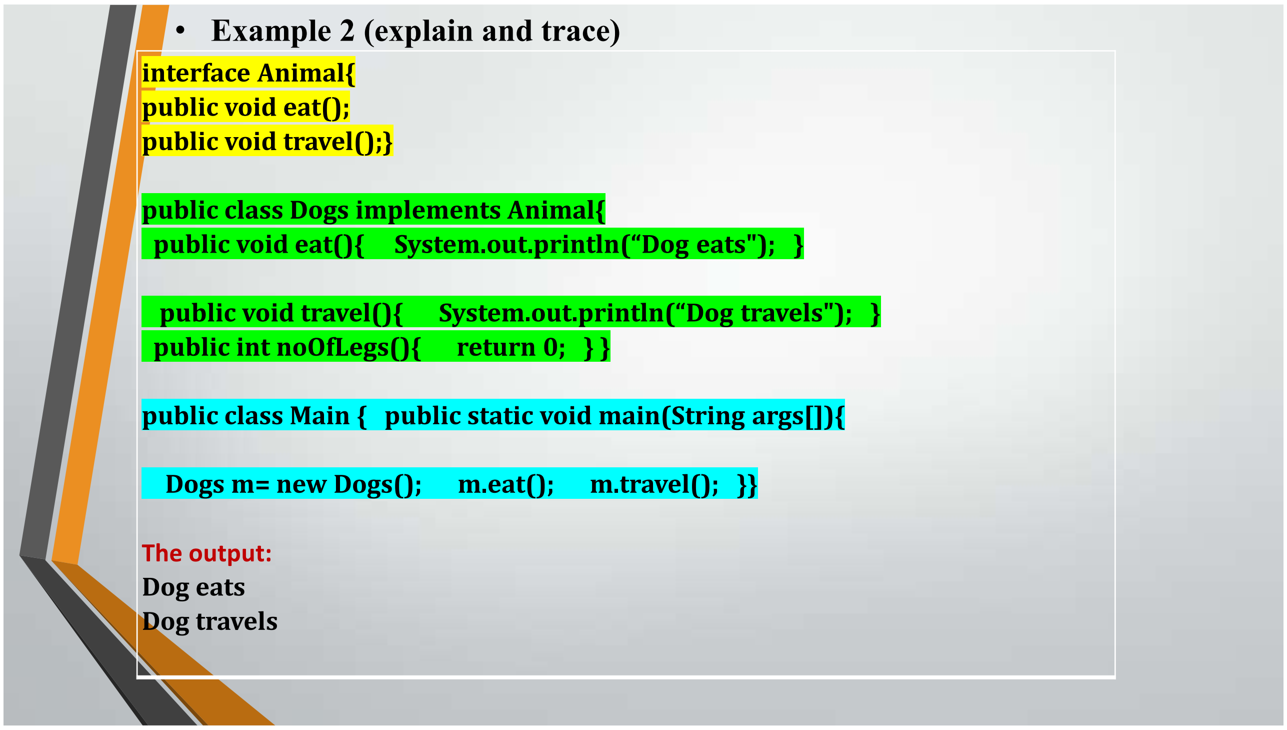
383





Multiple Inheritance in Java

[redacted]

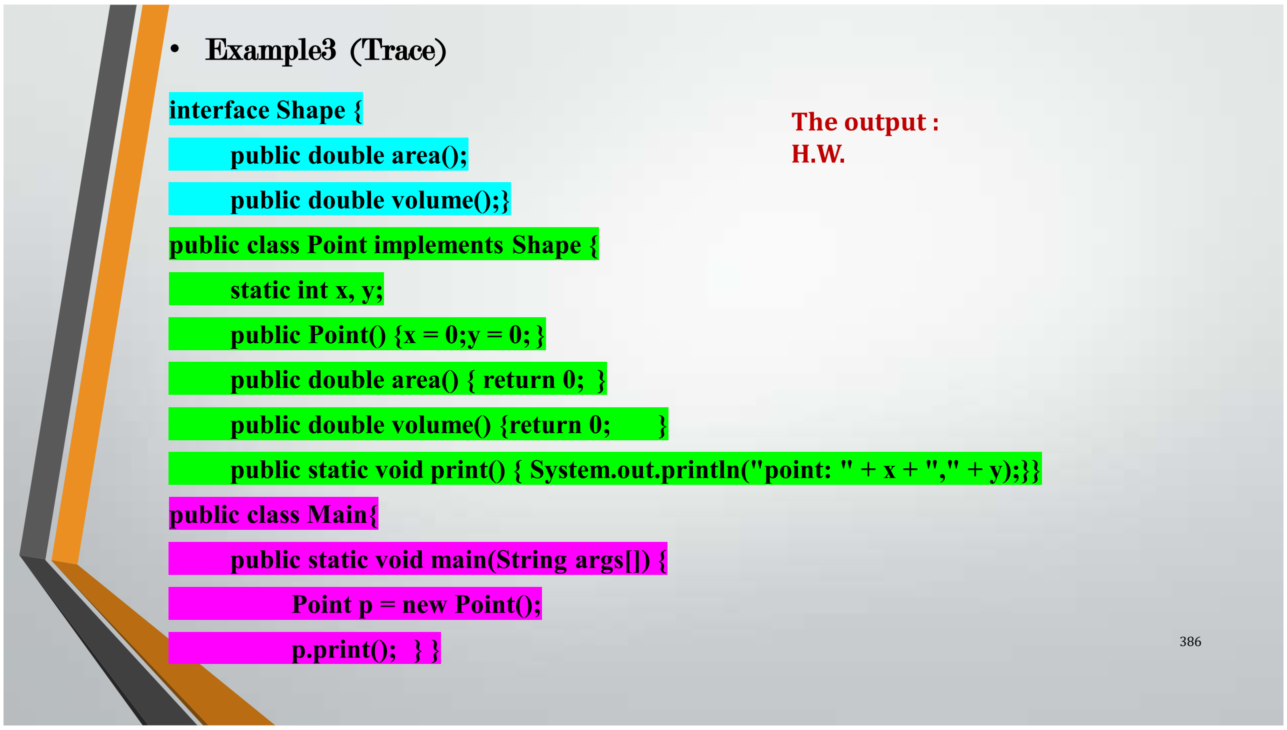


- Example 2 (explain and trace)

```
interface Animal{
public void eat();
public void travel();}

public class Dogs implements Animal{
 public void eat(){    System.out.println(“Dog eats");   }

  public void travel(){     System.out.println(“Dog travels");  }
 public int noOfLegs(){     return 0;   } }

public class Main {   public static void main(String args[]){

   Dogs m= new Dogs();      m.eat();      m.travel();   }}
```

The output:

```
Dog eats
Dog travels
```



- Example3 (Trace)

```
interface Shape {
        public double area();
        public double volume();}
public class Point implements Shape {
        static int x, y;
        public Point() {x = 0;y = 0; }
        public double area() { return 0;  }
        public double volume() {return 0;        }
        public static void print() { System.out.println("point: " + x + "," + y);}}
public class Main{
        public static void main(String args[]) {
                Point p = new Point();
                p.print();   } }
```

The output :
H.W.

386

[redacted]

• Default Methods In Java 8

- Before Java 8, interfaces could have only abstract methods.
- The implementation of these methods has to be provided in a separate class. So, if a new method is to be added in an interface, then its implementation code has to be provided in the class implementing the same interface.
- **To overcome this issue, Java 8 has introduced the concept of default methods which allow the interfaces to have methods with implementation without affecting the classes that implement the interface.**

387

• Example 4 (Trace)

```
interface TestInterface  {
default void show(){System.out.println("Default Method Executed");
public void square(int a);
   } }


class TestClass implements TestInterface {
   public void square(int a)    {    System.out.println(a*a);    } }


  public class Main{
   public static void main(String args[])   {
      TestClass d = new TestClass();
      d.square(4);
        d.show();    }  }
```

The output :

```
16
 Default Method Executed
```

388

[redacted]

• Multiple Inheritances Using Interface

- Example 5 (explain and trace)

```
interface vehicleone{
int  speed=90;
public void distance(); }

interface vehicletwo{
int distance=100;
public void speed(); }
```

```
class Vehicle  implements vehicleone,vehicletwo{
public void distance(){
            int  distance=speed*100;
            System.out.println("distance travelled is "+distance); }

public void speed(){
int speed=distance/100; }}

public class Main{
            public static void main(String args[]){
Vehicle obj=new Vehucle();
            System.out.println("Vehicle");
            obj.distance();
            obj.speed();        }}
```

The output :

```
Vehicle
distance travelled is 9000
```

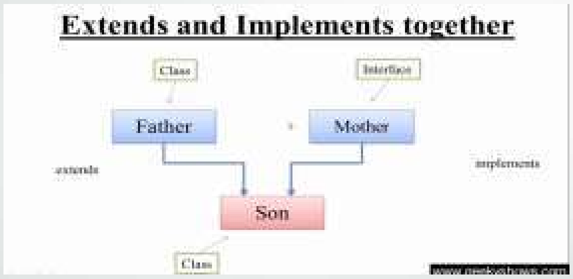
389

• Extending Interface

```
public interface Hockey extends Sports
{
}
```

• Extending Multiple Interfaces

```
public interface Hockey extends Sports, Event
{
}
```



• Tagging Interfaces

```
package java.util;
public interface EventListener {}
```

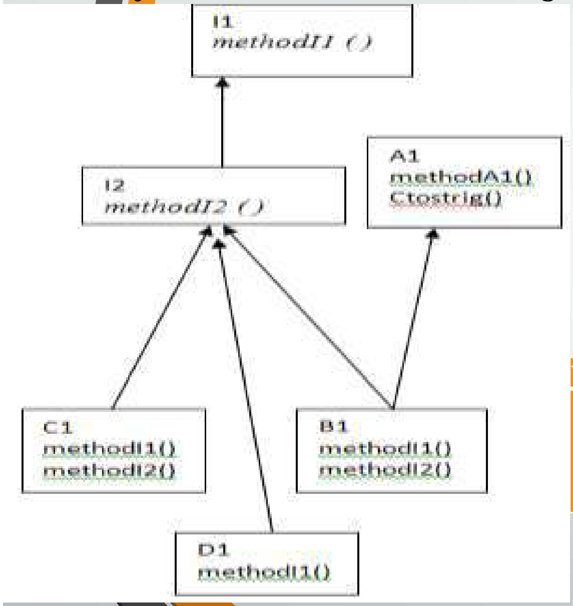
An interface with no methods in it is referred to as a **tagging** interface.

390

[redacted]

H.W.

Define a java structure for the following Hierarchy diagram



- Quiz

What is the default modifier in Interface?	
Answer	public + abstract for methods public + abstract for interface declaration public + static + final for the interface declaration variable

Lec25 Files

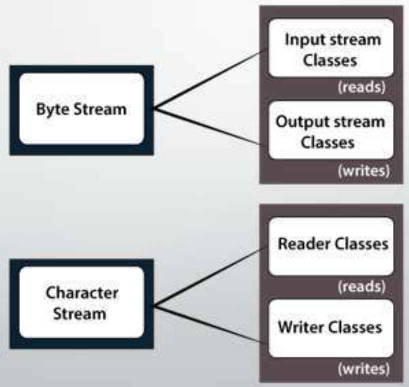
- File class
- Creating file
- Reading and writing from/to file
- Deleting file
    - Examples

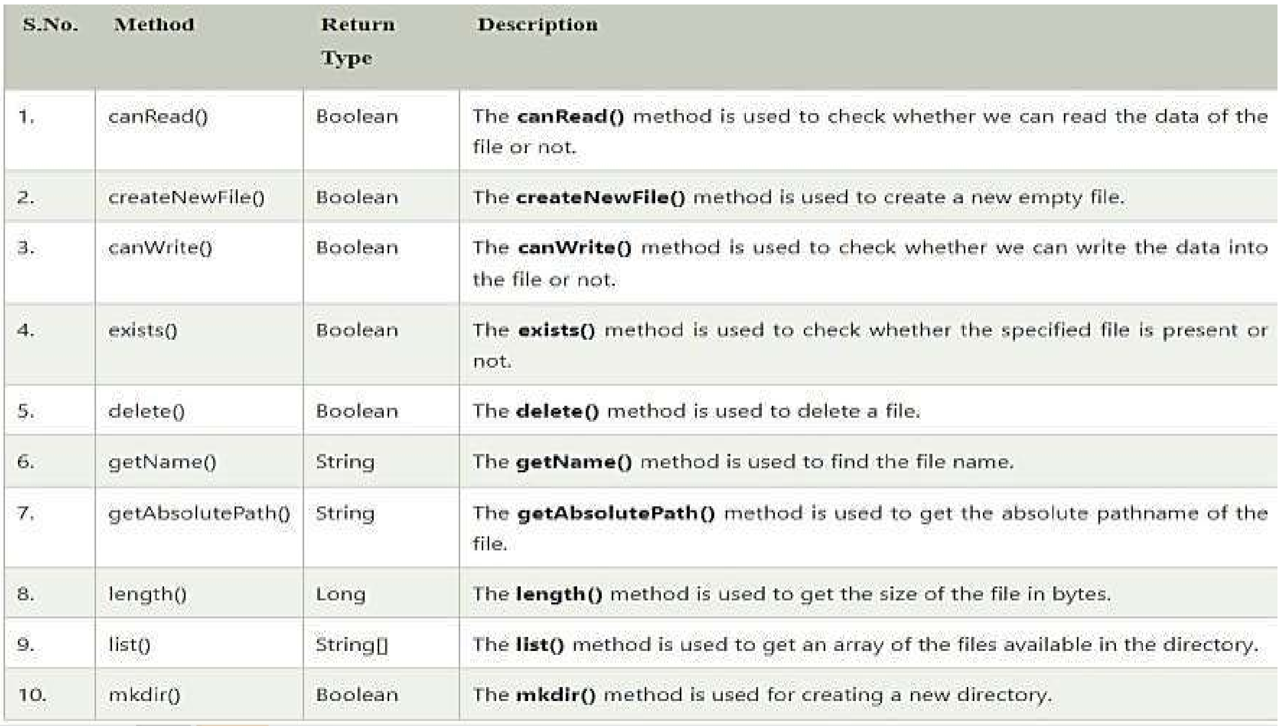
[redacted]

- In Java, a File is an abstract data type. A named location used to store related information is known as a File. There are several File Operations like creating a new File, getting information about File, writing into a File, reading from a File and deleting a File.

Stream

A series of data is referred to as **a stream**. In Java, **Stream** is classified into two types, i.e., **Byte Stream** and **Character Stream**.

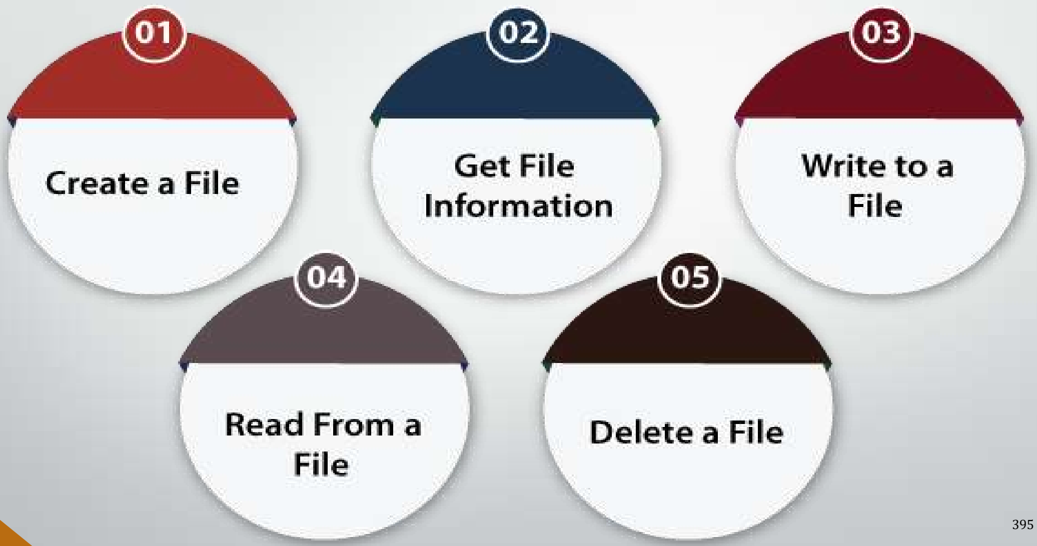




S.No.	Method	Return Type	Description
1.	canRead()	Boolean	The **canRead()** method is used to check whether we can read the data of the file or not.
2.	createNewFile()	Boolean	The **createNewFile()** method is used to create a new empty file.
3.	canWrite()	Boolean	The **canWrite()** method is used to check whether we can write the data into the file or not.
4.	exists()	Boolean	The **exists()** method is used to check whether the specified file is present or not.
5.	delete()	Boolean	The **delete()** method is used to delete a file.
6.	getName()	String	The **getName()** method is used to find the file name.
7.	getAbsolutePath()	String	The **getAbsolutePath()** method is used to get the absolute pathname of the file.
8.	length()	Long	The **length()** method is used to get the size of the file in bytes.
9.	list()	String[]	The **list()** method is used to get an array of the files available in the directory.
10.	mkdir()	Boolean	The **mkdir()** method is used for creating a new directory.

[redacted]

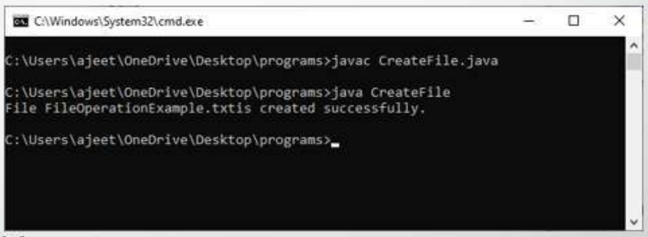
File Operations in Java



395

1- Create a File

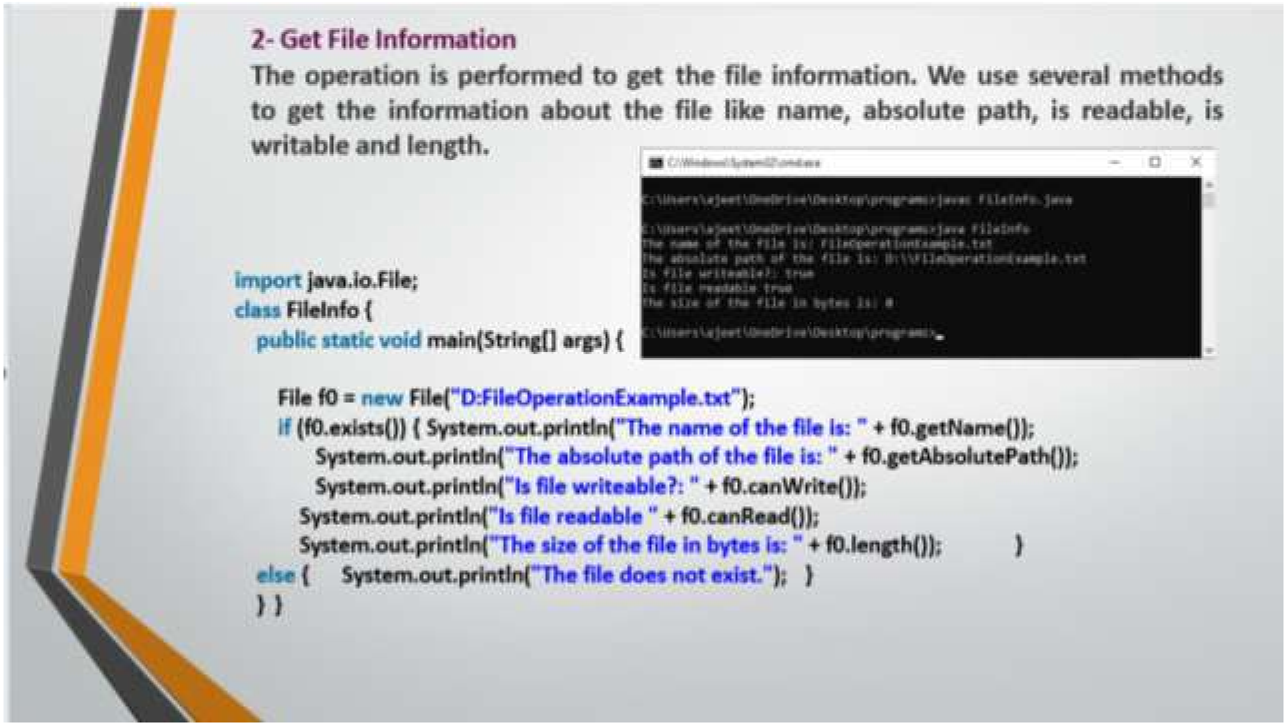
Create a File operation is performed to create a new file. We use the createNewFile() method of file. The createNewFile() method returns true when it successfully creates a new file and returns false when the file already exists.
Let's take an example of creating a file to understand how we can use the createNewFile() method to perform this operation.


C:\Windows\System32\cmd.exe

```
C:\Users\ajeet\OneDrive\Desktop\programs>javac CreateFile.java

C:\Users\ajeet\OneDrive\Desktop\programs>java CreateFile
File FileOperationExample.txtis created successfully.

C:\Users\ajeet\OneDrive\Desktop\programs>_
```

```
import java.io.File;
import java.io.IOException;
 class CreateFile {
          public static void main(String args[]) {
          try {  File f0 = new File("D:FileOperationExample.txt");
          if (f0.createNewFile()) { System.out.println("File " + f0.getName() + " is created successfully."); }
                else {    System.out.println("File is already exist in the directory.");        } }
             catch (IOException exception) {  System.out.println("An unexpected error is occurred.");
           exception.printStackTrace();  }} }
```

[redacted]

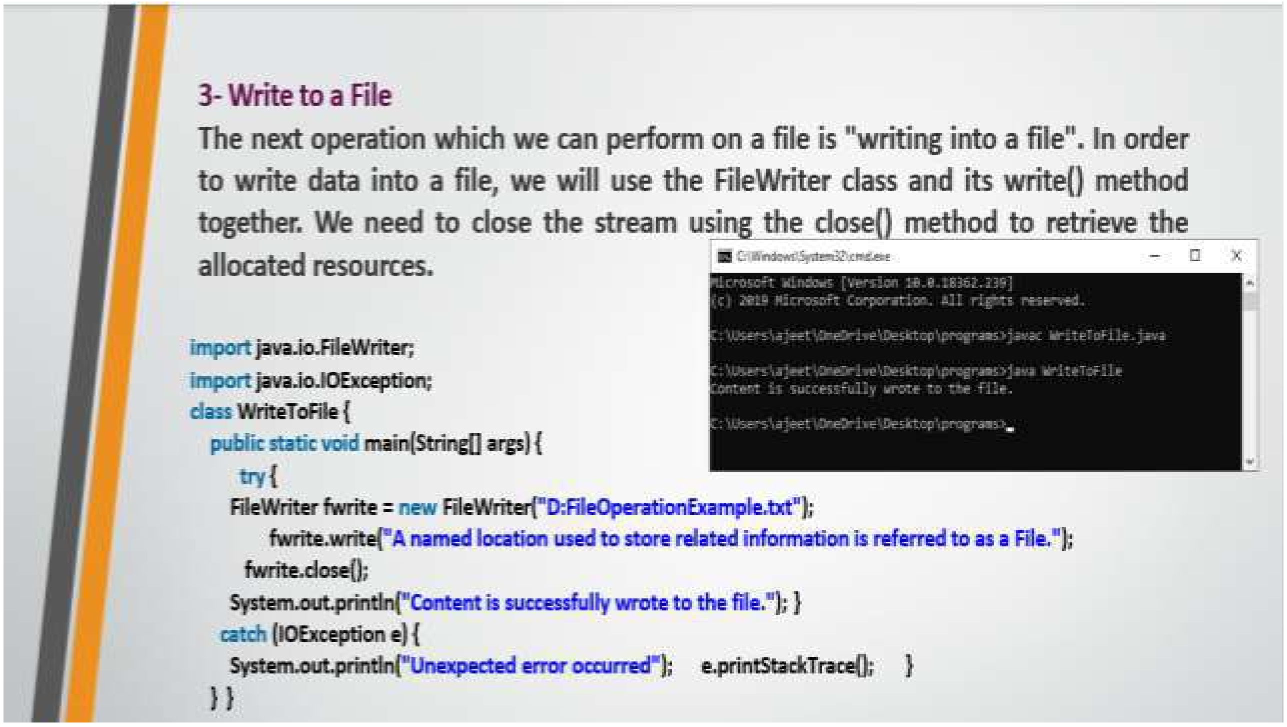


2- Get File Information

The operation is performed to get the file information. We use several methods to get the information about the file like name, absolute path, is readable, is writable and length.

```
C:\Users\ajeet\OneDrive\Desktop\programs>javac FileInfo.java

C:\Users\ajeet\OneDrive\Desktop\programs>java FileInfo
The name of the file is: FileOperationExample.txt
The absolute path of the file is: D:\\FileOperationExample.txt
Is file writeable?: true
Is file readable true
The size of the file in bytes is: 0

C:\Users\ajeet\OneDrive\Desktop\programs>_
```

```
import java.io.File;
class FileInfo {
  public static void main(String[] args) {

    File f0 = new File("D:FileOperationExample.txt");
    if (f0.exists()) { System.out.println("The name of the file is: " + f0.getName());
        System.out.println("The absolute path of the file is: " + f0.getAbsolutePath());
        System.out.println("Is file writeable?: " + f0.canWrite());
      System.out.println("Is file readable " + f0.canRead());
      System.out.println("The size of the file in bytes is: " + f0.length());          }
  else {    System.out.println("The file does not exist.");  }
  } }
```



3- Write to a File

The next operation which we can perform on a file is "writing into a file". In order to write data into a file, we will use the FileWriter class and its write() method together. We need to close the stream using the close() method to retrieve the allocated resources.

```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet\OneDrive\Desktop\programs>javac WriteToFile.java

C:\Users\ajeet\OneDrive\Desktop\programs>java WriteToFile
Content is successfully wrote to the file.

C:\Users\ajeet\OneDrive\Desktop\programs>_
```

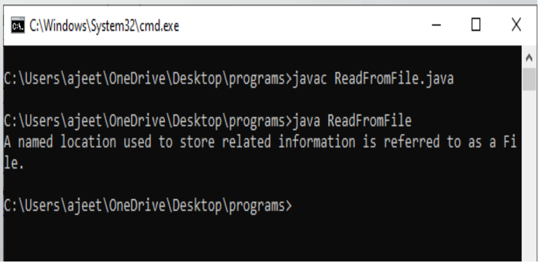
```
import java.io.FileWriter;
import java.io.IOException;
class WriteToFile {
  public static void main(String[] args) {
     try {
    FileWriter fwrite = new FileWriter("D:FileOperationExample.txt");
        fwrite.write("A named location used to store related information is referred to as a File.");
      fwrite.close();
    System.out.println("Content is successfully wrote to the file."); }
   catch (IOException e) {
    System.out.println("Unexpected error occurred");    e.printStackTrace();    }
  } }
```

[redacted]

4- Read from a File

The next operation which we can perform on a file is "read from a file". In order to write data into a file, we will use the Scanner class. Here, we need to close the stream using the close() method. We will create an instance of the:
Scanner class and use the hasNextLine() method nextLine() method to get data from the file.

```
import java.io.File;
 import java.io.FileNotFoundException;
import java.util.Scanner;
 class ReadFromFile {
 public static void main(String[] args) {
  try { File f1 = new File("D:FileOperationExample.txt"); Scanner dataReader = new Scanner(f1);
 while (dataReader.hasNextLine()) {
      String fileData = dataReader.nextLine(); System.out.println(fileData); }
   dataReader.close(); }
  catch (FileNotFoundException exception) {
   System.out.println("Unexcpected error occurred!");
   exception.printStackTrace(); }
   } }
```

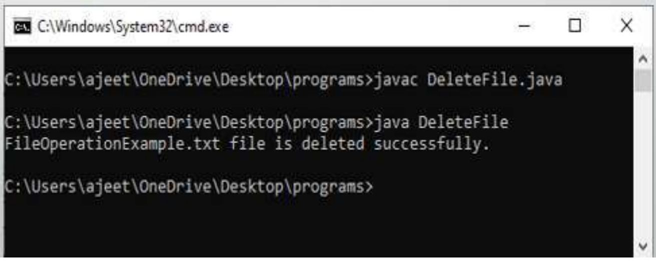


```
C:\Users\ajeet\OneDrive\Desktop\programs>javac ReadFromFile.java

C:\Users\ajeet\OneDrive\Desktop\programs>java ReadFromFile
A named location used to store related information is referred to as a Fi
le.

C:\Users\ajeet\OneDrive\Desktop\programs>
```

5-Delete a File

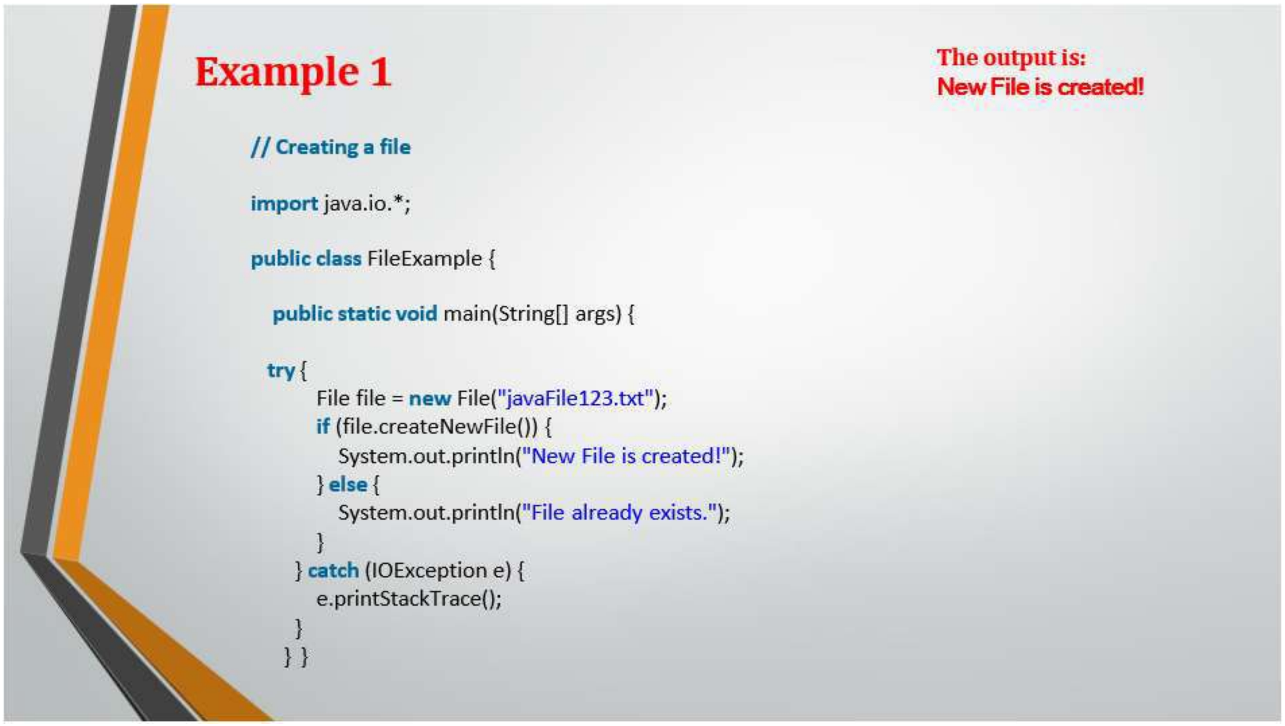
The next operation which we can perform on a file is "deleting a file". In order t delete a file, we will use the delete() method of the file. We don't need to close th stream using the close() method because for deleting a file, we neither use th FileWriter class nor the Scanner class.

```
import java.io.File;
class DeleteFile {
  public static void main(String[] args) {
    File f0 = new File("D:FileOperationExample.txt");
    if (f0.delete()) {
      System.out.println(f0.getName()+ " file is deleted successfully."); }
    else { System.out.println("Unexpected error found in deletion of the file."); }
  } }
```



```
C:\Users\ajeet\OneDrive\Desktop\programs>javac DeleteFile.java

C:\Users\ajeet\OneDrive\Desktop\programs>java DeleteFile
FileOperationExample.txt file is deleted successfully.

C:\Users\ajeet\OneDrive\Desktop\programs>
```

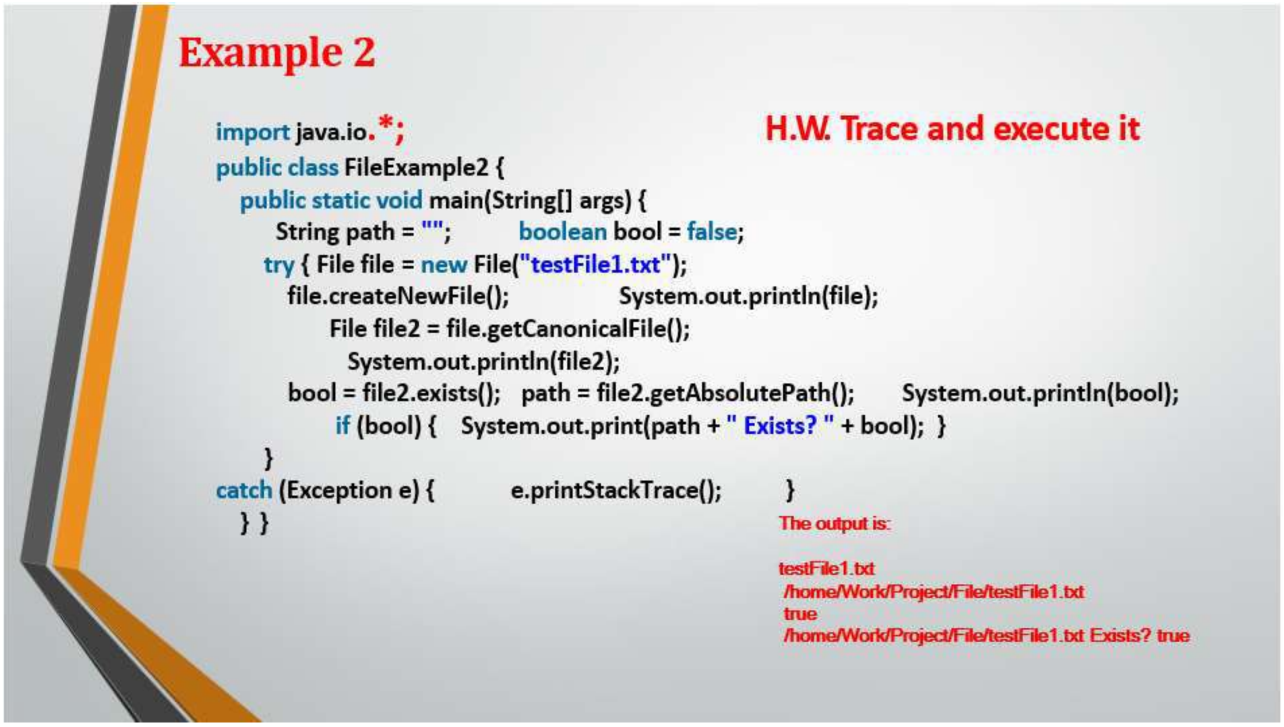
[redacted]



Example 1

The output is:
New File is created!

```
// Creating a file

import java.io.*;

public class FileExample {

  public static void main(String[] args) {

 try {
       File file = new File("javaFile123.txt");
       if (file.createNewFile()) {
         System.out.println("New File is created!");
       } else {
         System.out.println("File already exists.");
       }
     } catch (IOException e) {
       e.printStackTrace();
     }
   } }
```



Example 2

H.W. Trace and execute it

```
import java.io.*;
public class FileExample2 {
   public static void main(String[] args) {
       String path = "";        boolean bool = false;
     try { File file = new File("testFile1.txt");
         file.createNewFile();           System.out.println(file);
             File file2 = file.getCanonicalFile();
               System.out.println(file2);
         bool = file2.exists();   path = file2.getAbsolutePath();      System.out.println(bool);
              if (bool) {   System.out.print(path + " Exists? " + bool);  }
     }
catch (Exception e) {          e.printStackTrace();          }
   } }
```

The output is:

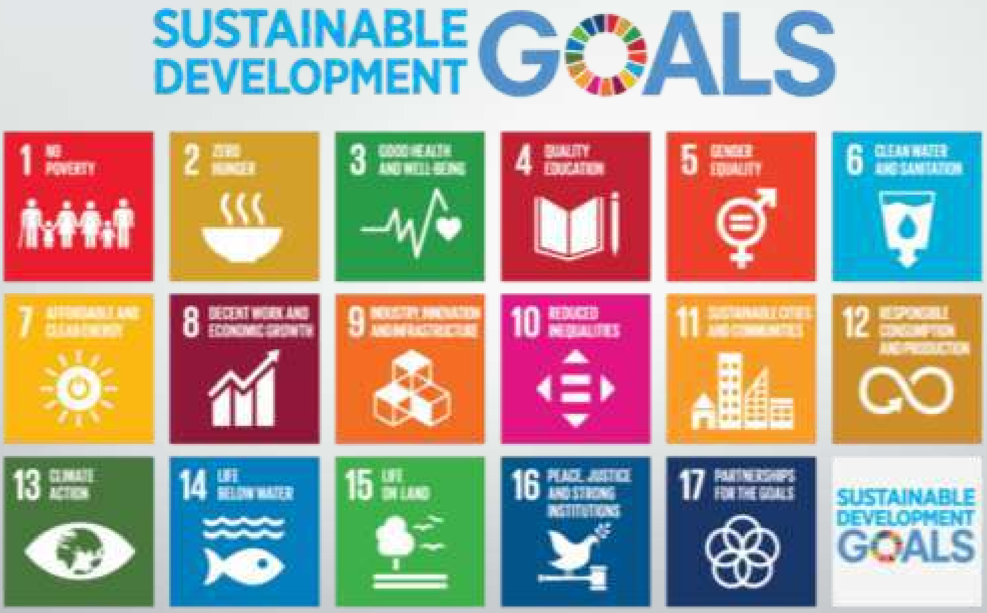
```
testFile1.txt
 /home/Work/Project/File/testFile1.txt
 true
 /home/Work/Project/File/testFile1.txt Exists? true
```

[redacted]

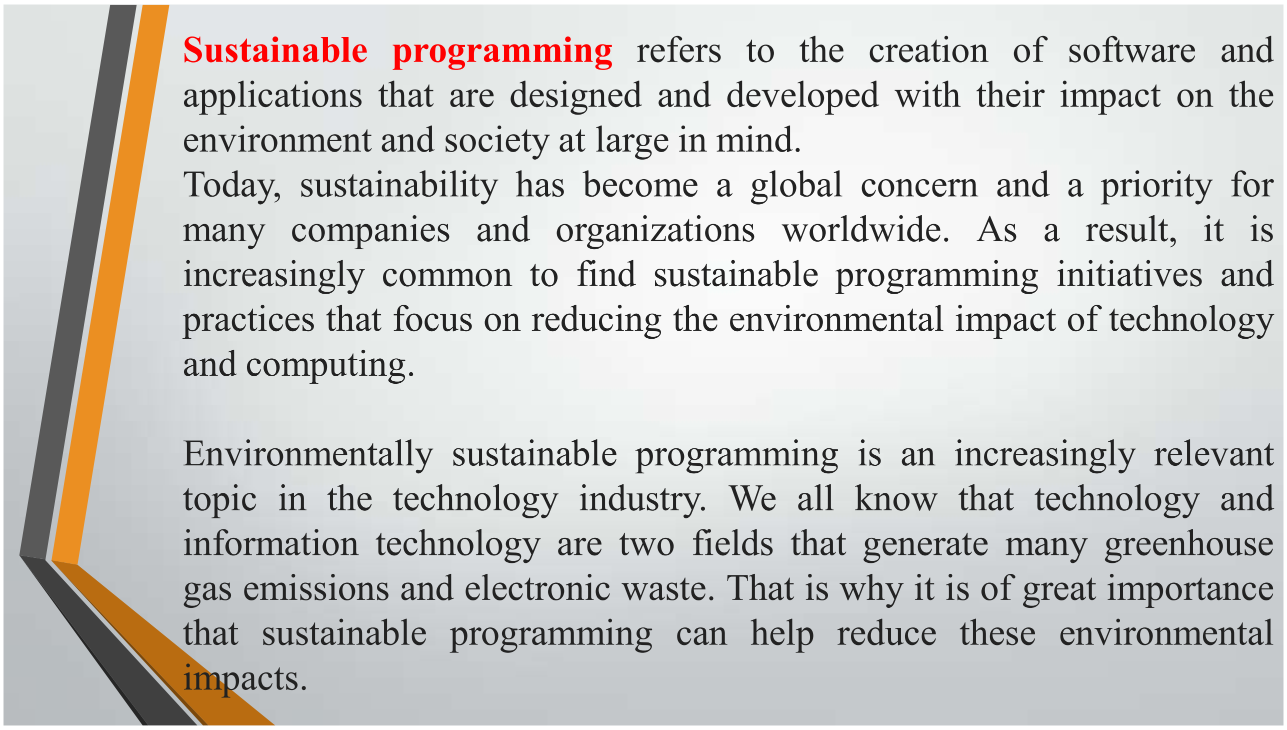
Sustainable development in Programming Part1

- Sustainable development def.
- Sustainable development goals
- Energy Efficiency
- Examples

403

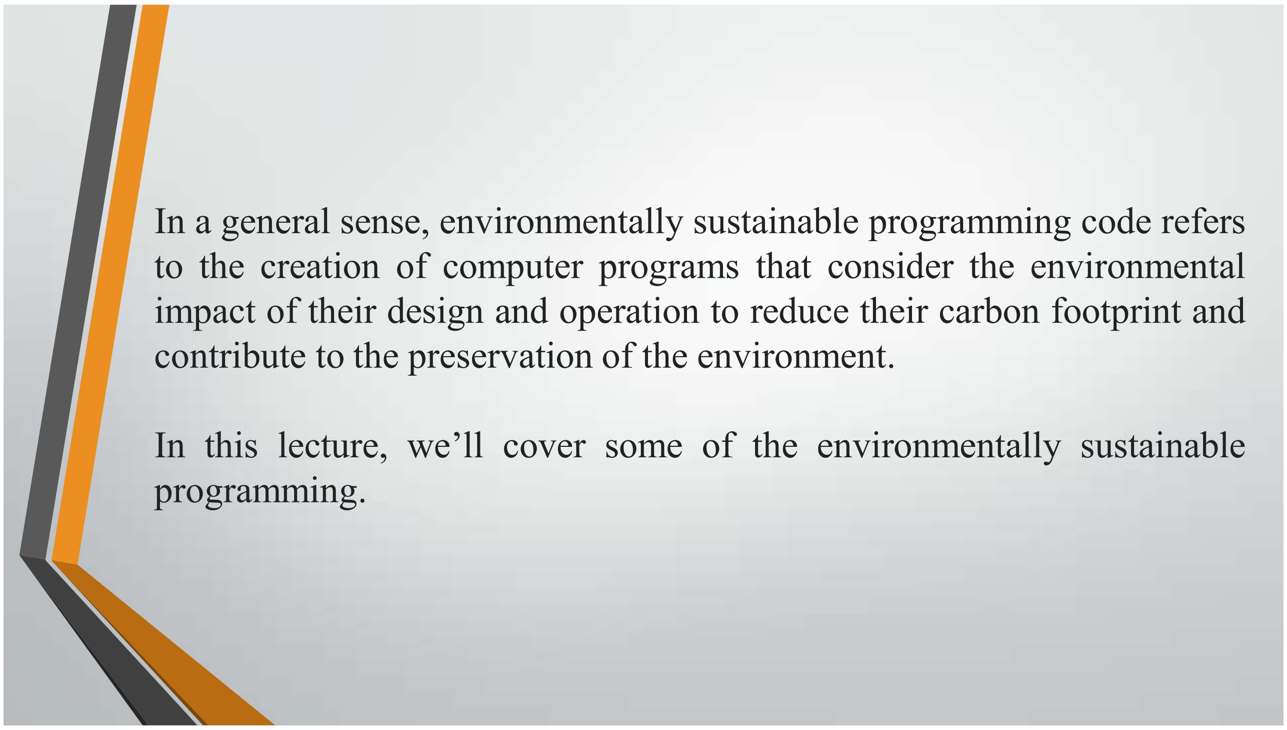

SUSTAINABLE DEVELOPMENT GOALS
1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

[redacted]



Sustainable programming refers to the creation of software and applications that are designed and developed with their impact on the environment and society at large in mind.
Today, sustainability has become a global concern and a priority for many companies and organizations worldwide. As a result, it is increasingly common to find sustainable programming initiatives and practices that focus on reducing the environmental impact of technology and computing.

Environmentally sustainable programming is an increasingly relevant topic in the technology industry. We all know that technology and information technology are two fields that generate many greenhouse gas emissions and electronic waste. That is why it is of great importance that sustainable programming can help reduce these environmental impacts.



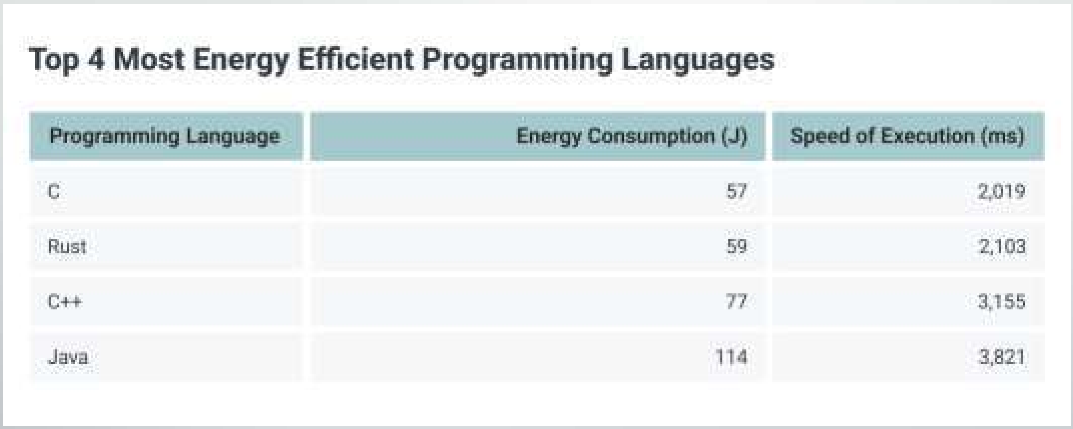
In a general sense, environmentally sustainable programming code refers to the creation of computer programs that consider the environmental impact of their design and operation to reduce their carbon footprint and contribute to the preservation of the environment.

In this lecture, we'll cover some of the environmentally sustainable programming.

[redacted]

Energy Efficiency

Sustainable programming code is written with energy efficiency in mind. This means that the software is designed to use as few resources and energy as possible. For example, the binary search algorithm is more efficient than the linear search algorithm since it uses fewer operations to find a result. Similarly, the use of compression algorithms can help reduce file sizes and decrease the amount of power required to transfer and store data.



Top 4 Most Energy Efficient Programming Languages

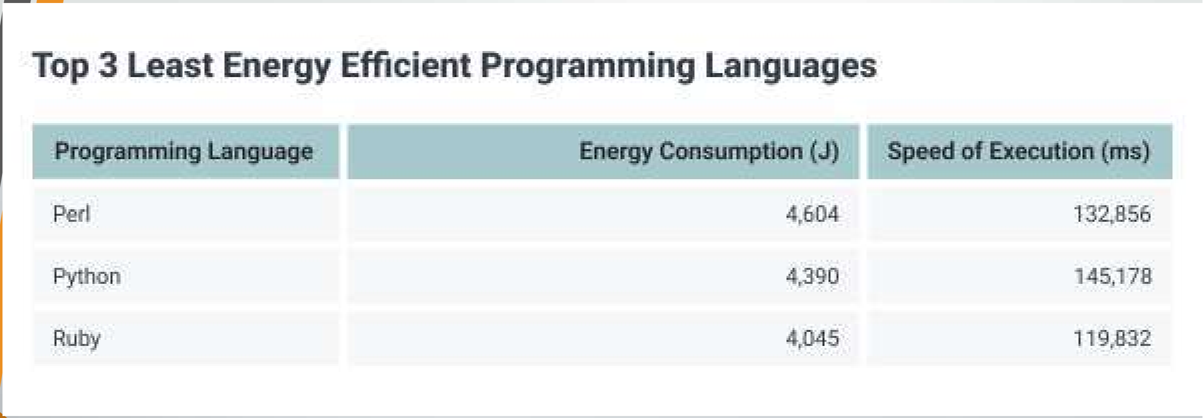
Programming Language	Energy Consumption (J)	Speed of Execution (ms)
C	57	2,019
Rust	59	2,103
C++	77	3,155
Java	114	3,821

[redacted]

C/C++: Known for its compilation speed and runtime efficiency, C/C++ consumes minimal memory and processor resources, making it one of the greenest programming options available.

JavaScript: Essential for web development and ubiquitous across development platforms, JavaScript can lead to inefficient code that requires higher resource use if not properly optimized.
Rust: With a focus on memory safety without the use of a garbage collector, Rust offers optimized performance that reduces power usage, making it an increasingly popular choice for energy-efficient applications.

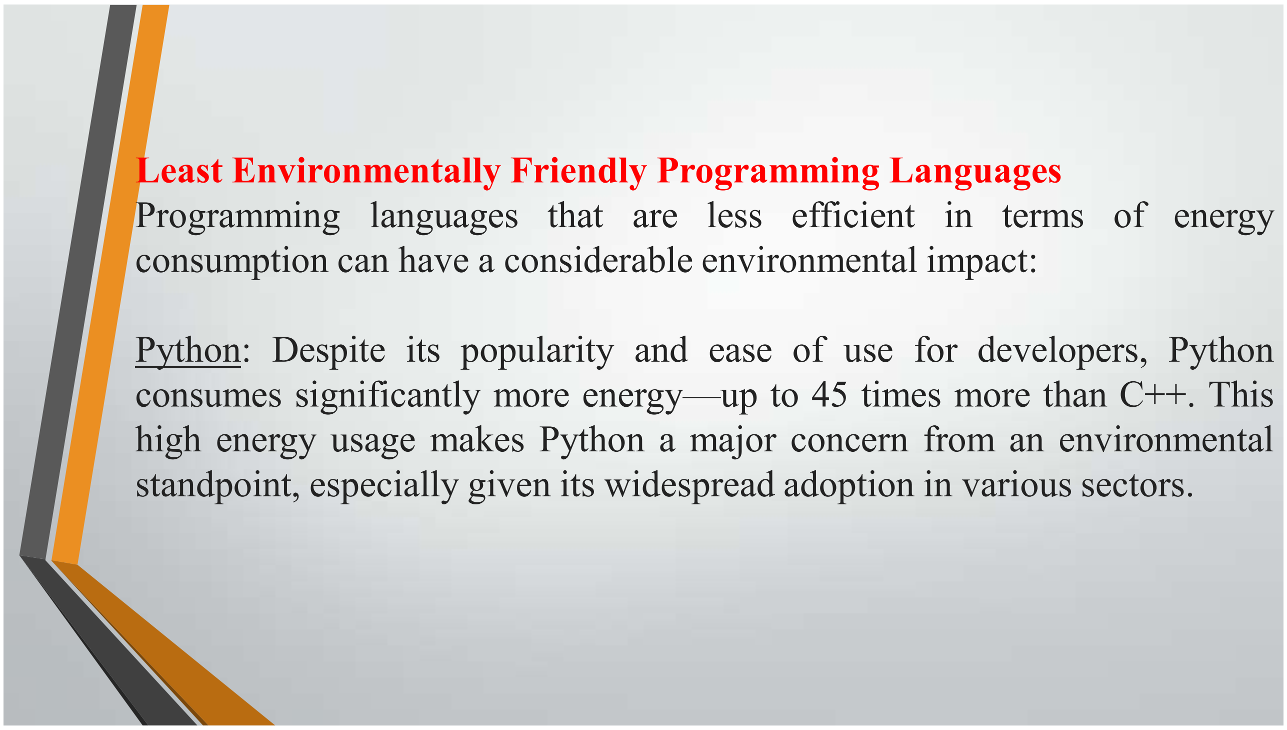
Java: Despite its extensive use and well-optimized Java Virtual Machine (JVM), Java still requires significant energy, particularly in large-scale systems, although recent improvements have helped mitigate its environmental impact.



Top 3 Least Energy Efficient Programming Languages

Programming Language	Energy Consumption (J)	Speed of Execution (ms)
Perl	4,604	132,856
Python	4,390	145,178
Ruby	4,045	119,832

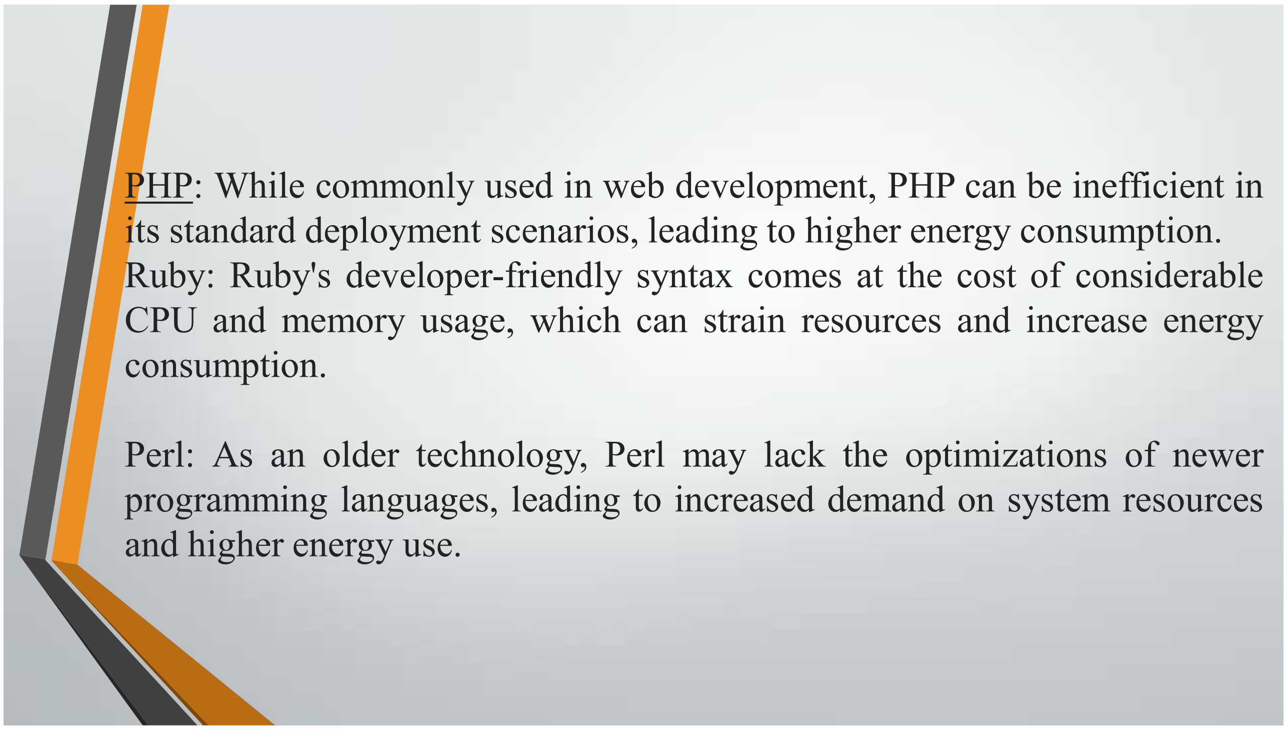
[redacted]



Least Environmentally Friendly Programming Languages

Programming languages that are less efficient in terms of energy consumption can have a considerable environmental impact:

Python: Despite its popularity and ease of use for developers, Python consumes significantly more energy—up to 45 times more than C++. This high energy usage makes Python a major concern from an environmental standpoint, especially given its widespread adoption in various sectors.



PHP: While commonly used in web development, PHP can be inefficient in its standard deployment scenarios, leading to higher energy consumption.
Ruby: Ruby's developer-friendly syntax comes at the cost of considerable CPU and memory usage, which can strain resources and increase energy consumption.

Perl: As an older technology, Perl may lack the optimizations of newer programming languages, leading to increased demand on system resources and higher energy use.

[redacted]

How Businesses Can Make More Environmentally Friendly Programming Choices?

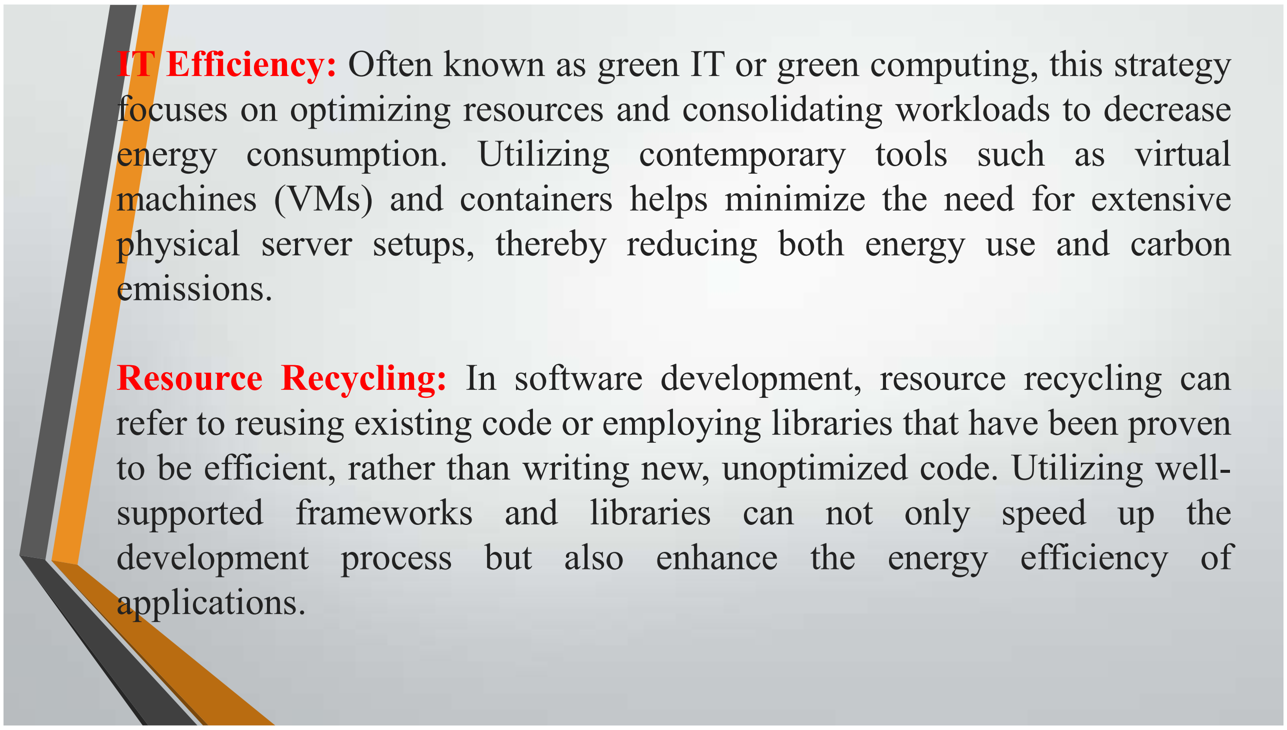
- Green coding aligns with and enhances the existing IT sustainability protocols and practices within an organization. Here are sustainability efforts that can be made to create a more efficient and greener practice for your business:

- Choosing the Right Programming Languages: Some languages inherently require less computational power, which translates into lower energy consumption. Languages like Rust, C, or C++ are generally more efficient than interpreted languages like Python or Perl. When performance and efficiency are critical, selecting a compiled language can lead to significant energy savings.

Core Energy Efficiency: Applications utilizing multi-core processors can be developed to boost energy efficiency. For instance, programmers can write code that commands processors to shut down and reboot in microseconds, offering a more efficient alternative to standard energy-saving settings.

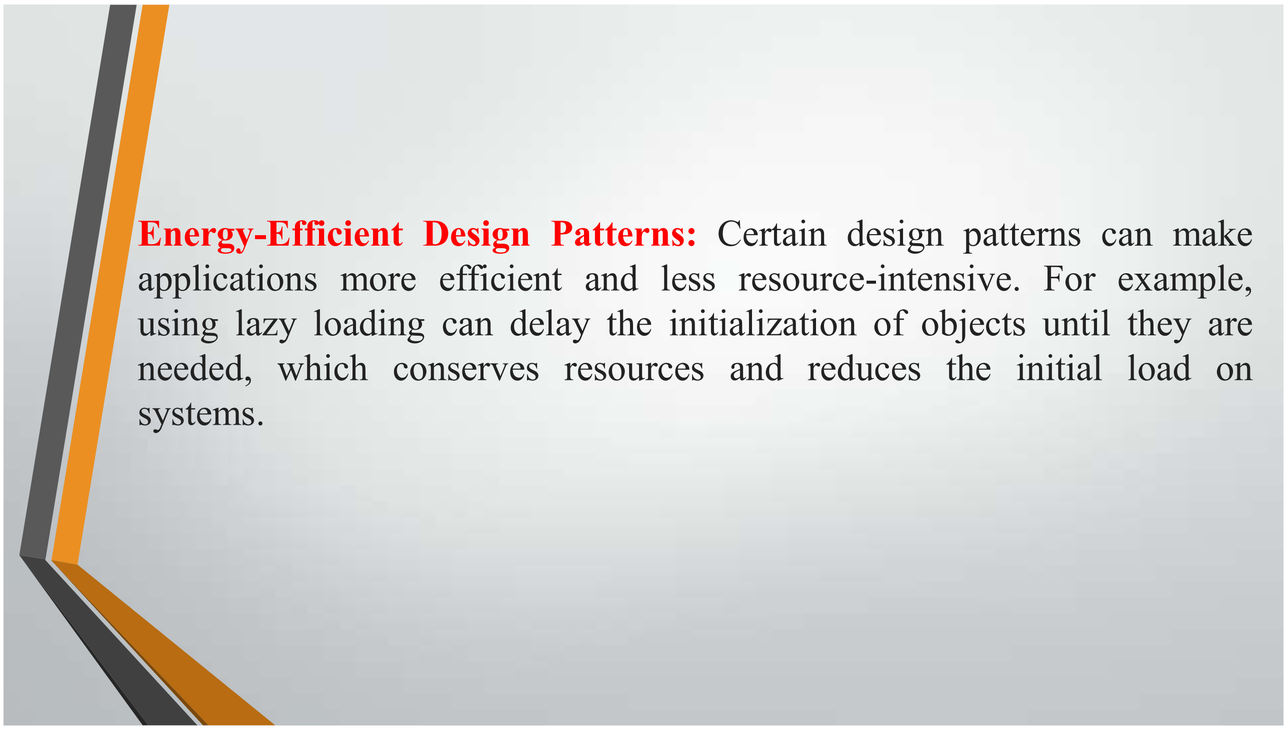
Microservices: This modern technique involves decomposing complex software into smaller, manageable components known as microservices. These services operate only when required, rather than maintaining a large, continuously running system. This leads to more resource-efficient application performance.

[redacted]



IT Efficiency: Often known as green IT or green computing, this strategy focuses on optimizing resources and consolidating workloads to decrease energy consumption. Utilizing contemporary tools such as virtual machines (VMs) and containers helps minimize the need for extensive physical server setups, thereby reducing both energy use and carbon emissions.

Resource Recycling: In software development, resource recycling can refer to reusing existing code or employing libraries that have been proven to be efficient, rather than writing new, unoptimized code. Utilizing well-supported frameworks and libraries can not only speed up the development process but also enhance the energy efficiency of applications.



Energy-Efficient Design Patterns: Certain design patterns can make applications more efficient and less resource-intensive. For example, using lazy loading can delay the initialization of objects until they are needed, which conserves resources and reduces the initial load on systems.

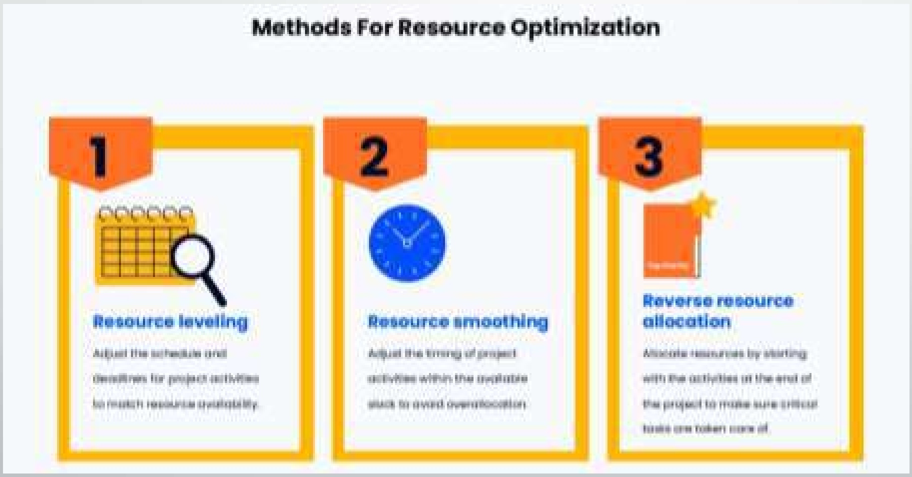
[redacted]

Sustainable development in Programming Part2

- Resource Optimization
- Hardware Recycling
- Sustainable Programming
- Examples

Resource Optimization

Sustainable programming code uses resources and materials more efficiently. This includes using hardware and software that is tailored to the needs of the system and reusing code rather than creating new programs from scratch.

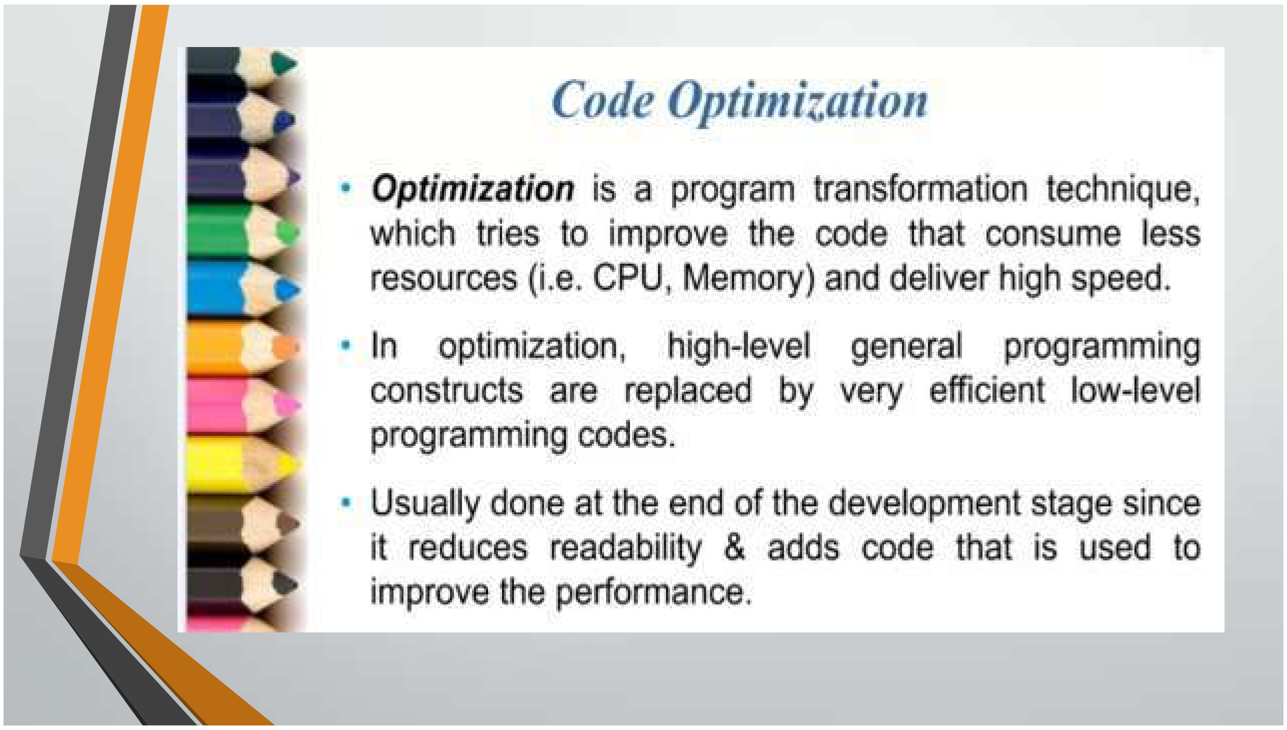


Resource leveling
Adjust the schedule and deadlines for project activities to match resource availability.

Resource smoothing
Adjust the timing of project activities within the available slack to avoid overallocation.

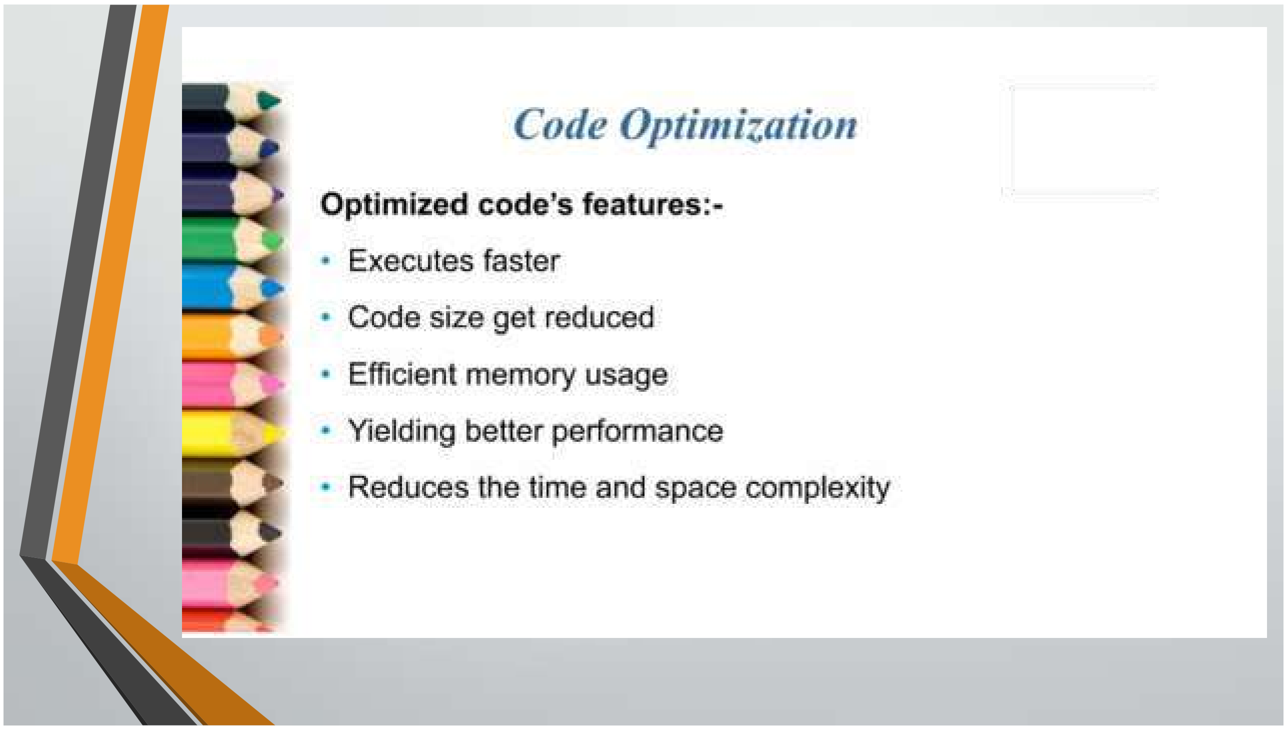
Reverse resource allocation
Allocate resources by starting with the activities at the end of the project to make sure critical tasks are taken care of.

[redacted]



Code Optimization

- ***Optimization*** is a program transformation technique, which tries to improve the code that consume less resources (i.e. CPU, Memory) and deliver high speed.
- In optimization, high-level general programming constructs are replaced by very efficient low-level programming codes.
- Usually done at the end of the development stage since it reduces readability & adds code that is used to improve the performance.

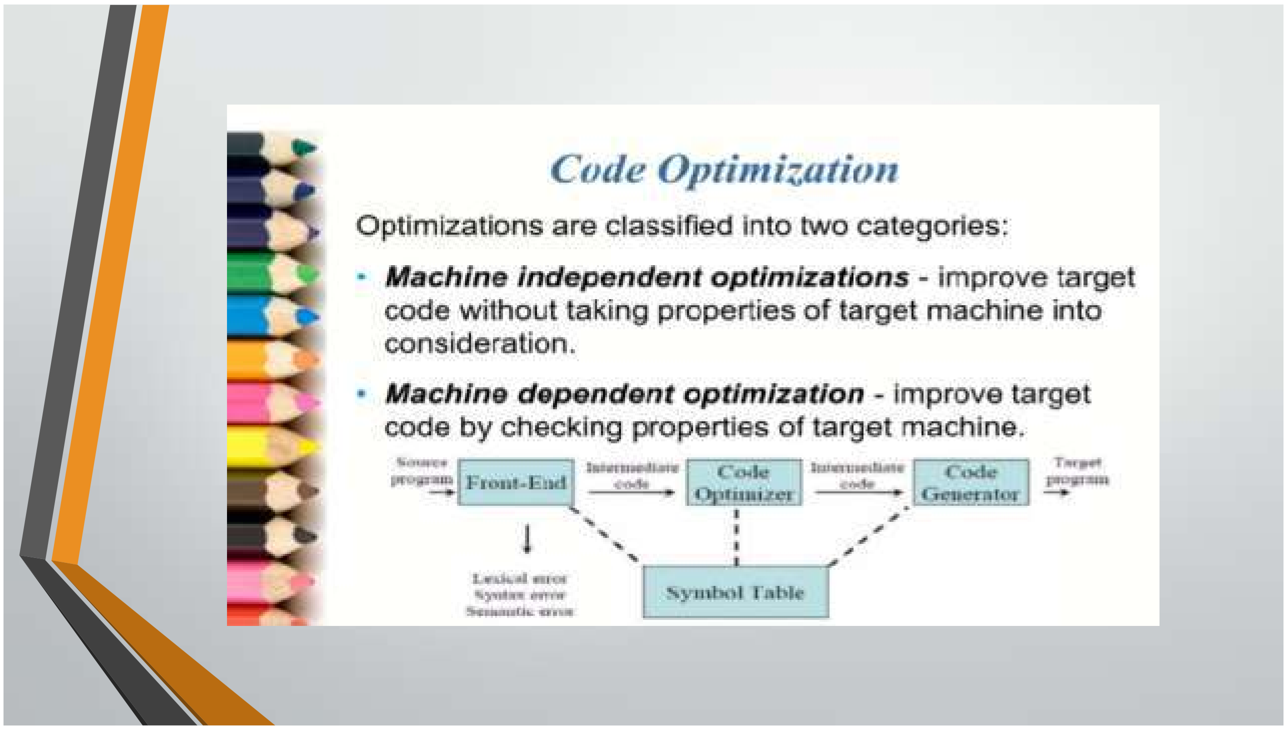


Code Optimization

Optimized code's features:-

- Executes faster
- Code size get reduced
- Efficient memory usage
- Yielding better performance
- Reduces the time and space complexity

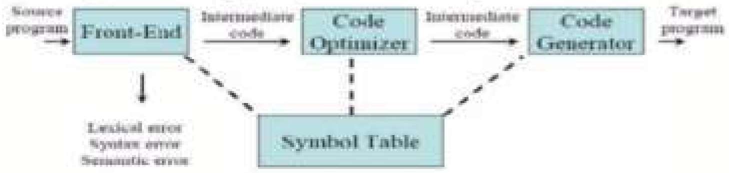
[redacted]

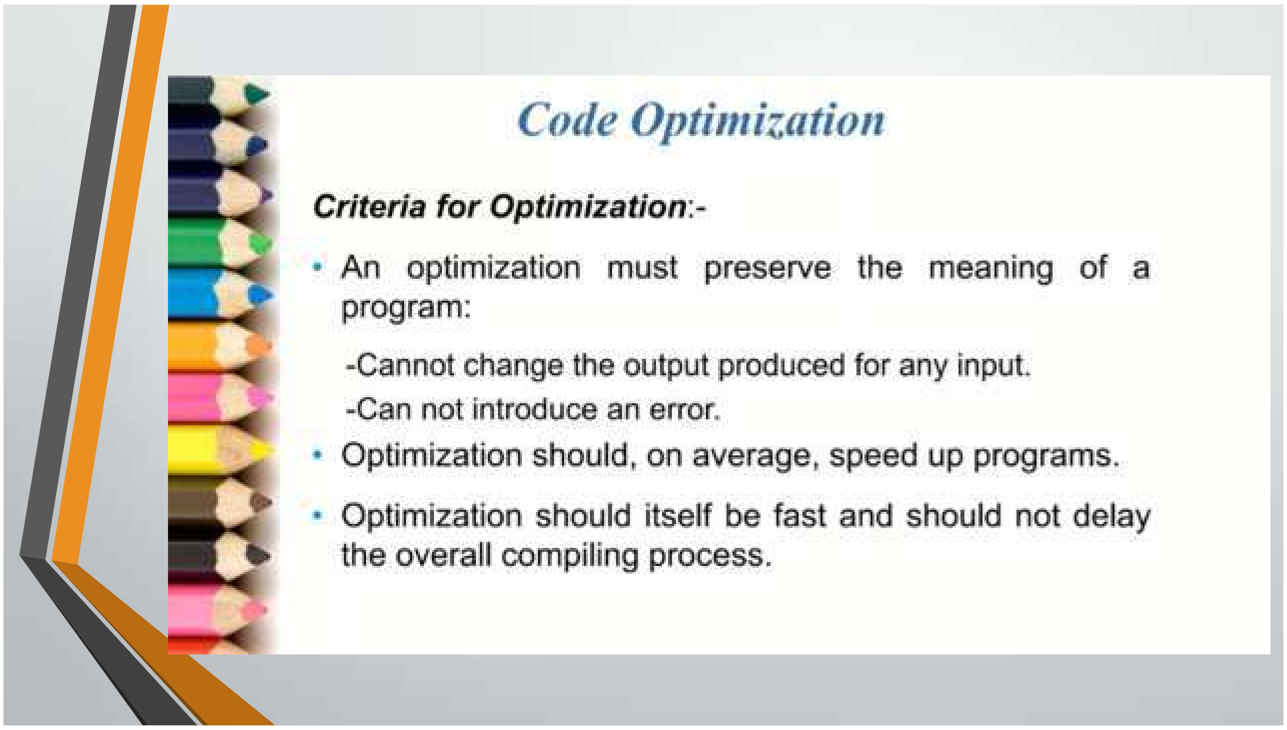


Code Optimization

Optimizations are classified into two categories:

- ***Machine independent optimizations*** - improve target code without taking properties of target machine into consideration.
- ***Machine dependent optimization*** - improve target code by checking properties of target machine.





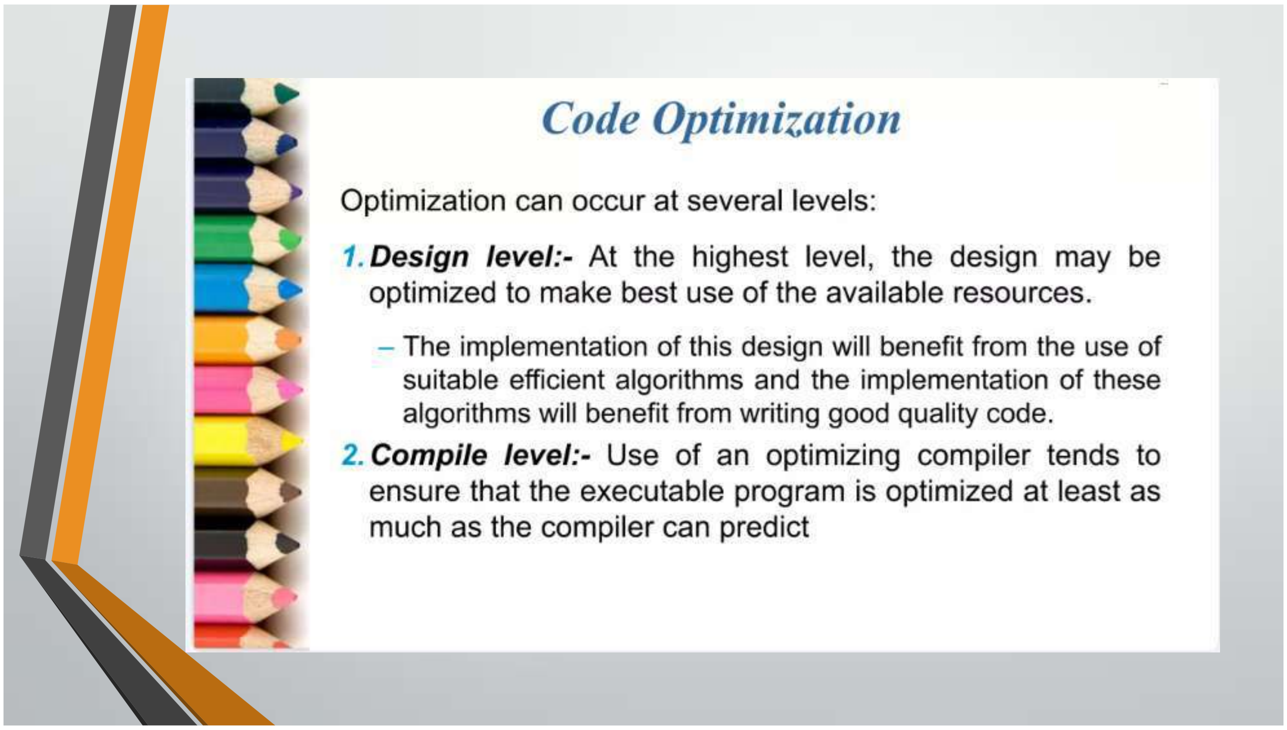
Code Optimization

Criteria for Optimization:-

- An optimization must preserve the meaning of a program:

  -Cannot change the output produced for any input.

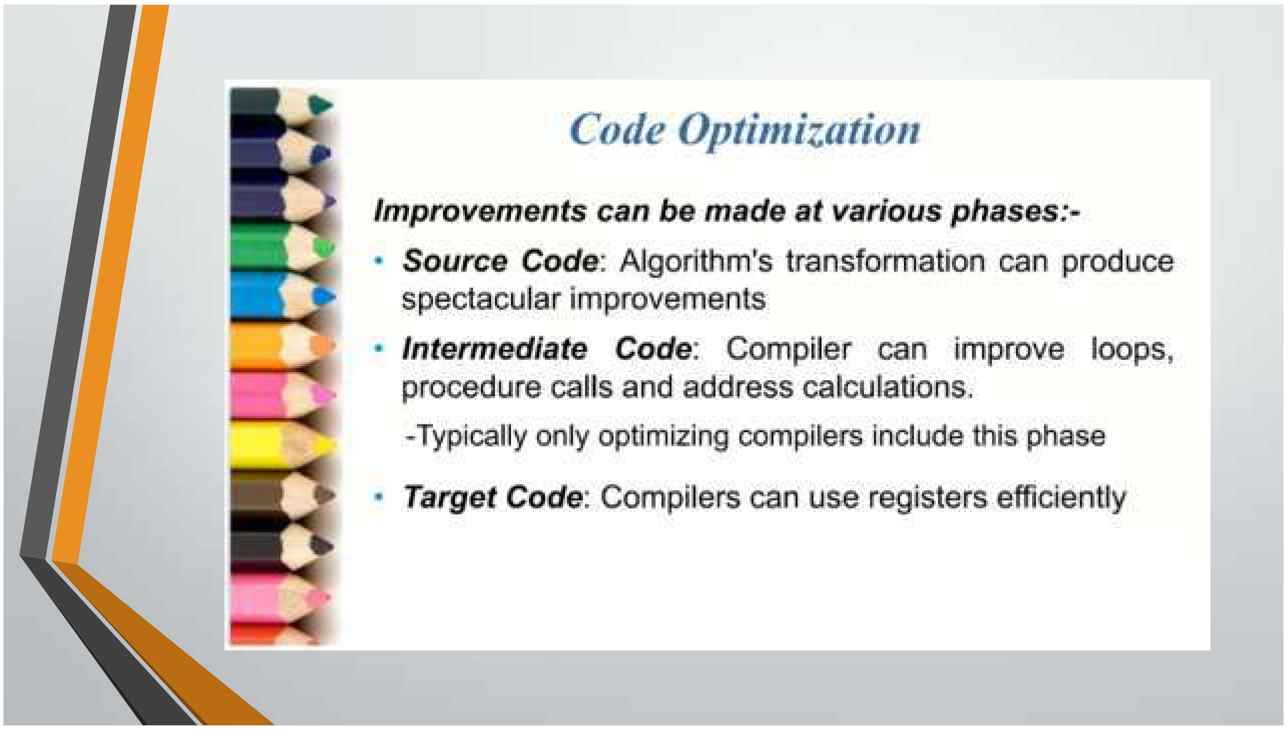
  -Can not introduce an error.
- Optimization should, on average, speed up programs.
- Optimization should itself be fast and should not delay the overall compiling process.



Code Optimization

Optimization can occur at several levels:

1. ***Design level:-*** At the highest level, the design may be optimized to make best use of the available resources.
   - The implementation of this design will benefit from the use of suitable efficient algorithms and the implementation of these algorithms will benefit from writing good quality code.
2. ***Compile level:-*** Use of an optimizing compiler tends to ensure that the executable program is optimized at least as much as the compiler can predict



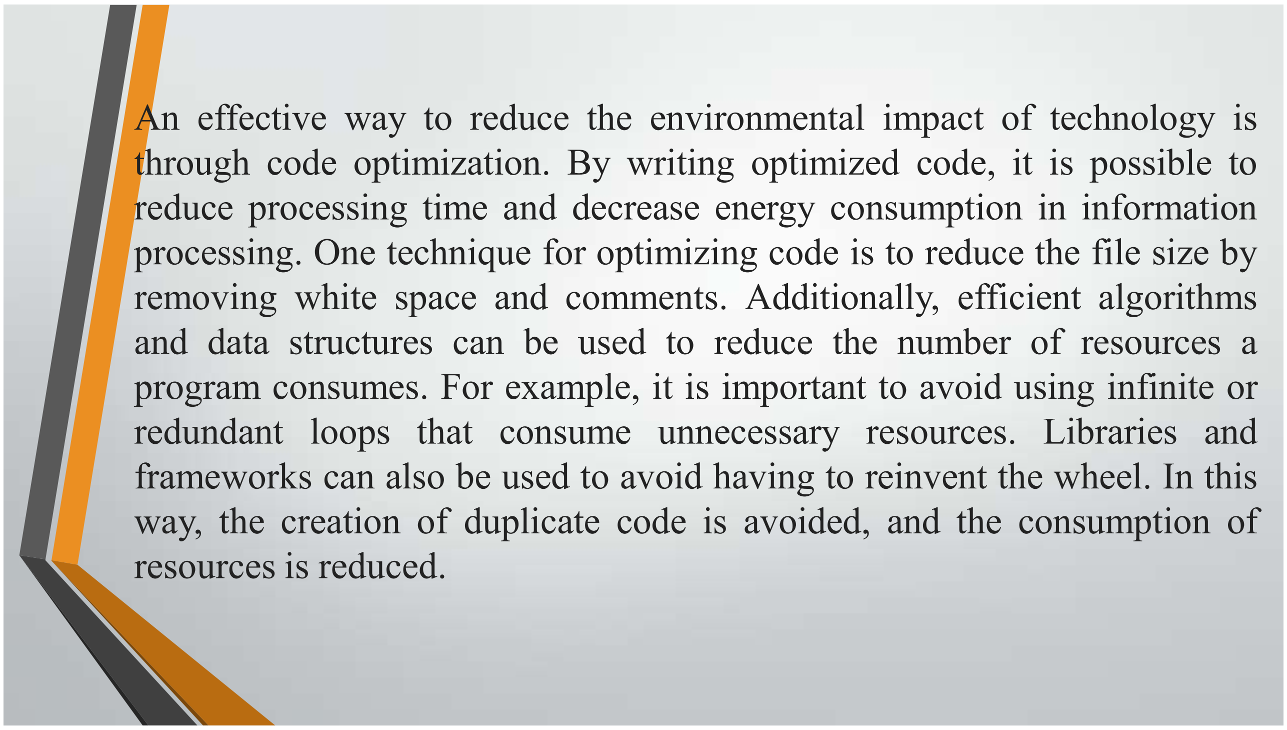
Code Optimization

Improvements can be made at various phases:-

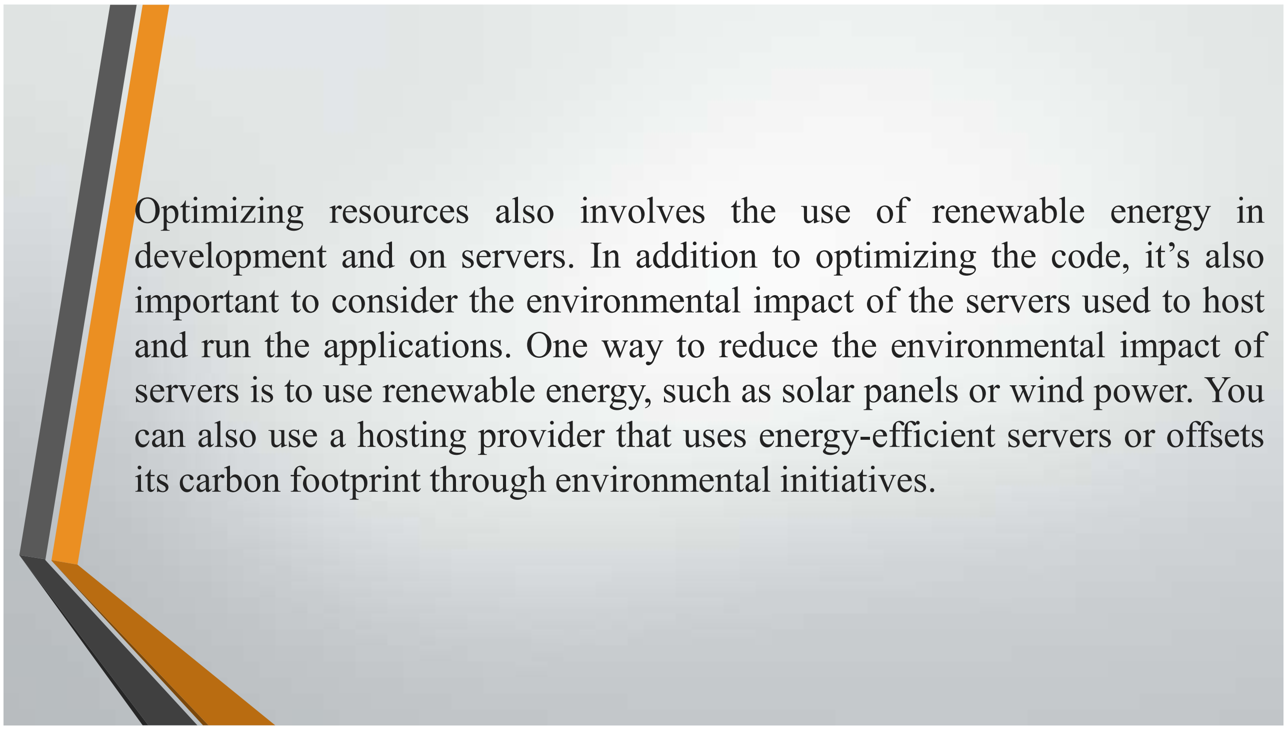
- ***Source Code***: Algorithm's transformation can produce spectacular improvements
- ***Intermediate Code***: Compiler can improve loops, procedure calls and address calculations.

  -Typically only optimizing compilers include this phase
- ***Target Code***: Compilers can use registers efficiently

[redacted]



An effective way to reduce the environmental impact of technology is through code optimization. By writing optimized code, it is possible to reduce processing time and decrease energy consumption in information processing. One technique for optimizing code is to reduce the file size by removing white space and comments. Additionally, efficient algorithms and data structures can be used to reduce the number of resources a program consumes. For example, it is important to avoid using infinite or redundant loops that consume unnecessary resources. Libraries and frameworks can also be used to avoid having to reinvent the wheel. In this way, the creation of duplicate code is avoided, and the consumption of resources is reduced.



Optimizing resources also involves the use of renewable energy in development and on servers. In addition to optimizing the code, it's also important to consider the environmental impact of the servers used to host and run the applications. One way to reduce the environmental impact of servers is to use renewable energy, such as solar panels or wind power. You can also use a hosting provider that uses energy-efficient servers or offsets its carbon footprint through environmental initiatives.

Hardware Recycling

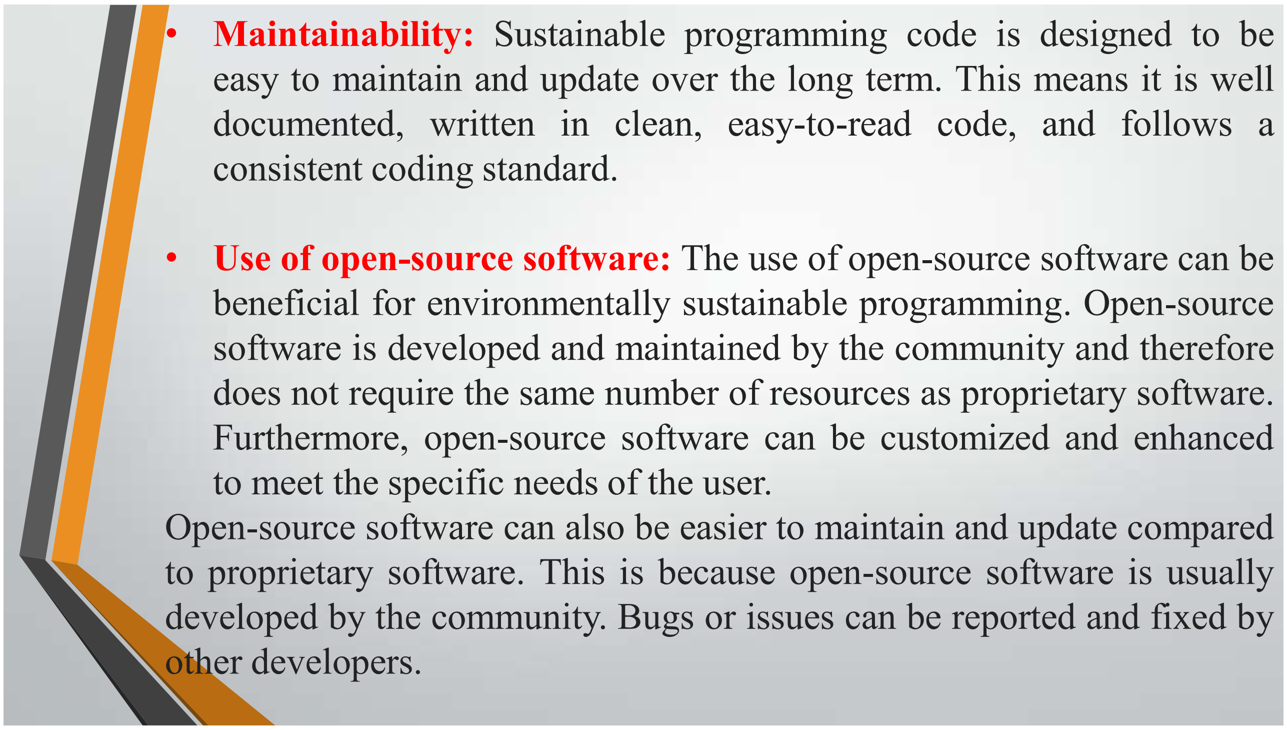
Hardware recycling is an important practice in environmentally sustainable programming. Instead of throwing away old hardware, it can be reused or recycled. Reusing old hardware can reduce the need to build new hardware and therefore reduce the carbon footprint of programming.
Additionally, hardware recycling can help reduce the amount of e-waste that accumulates in landfills. Electronic components such as batteries and circuitry can be highly toxic if disposed of incorrectly. Therefore, it is important to follow proper hardware recycling procedures.

Sustainable Programming

To achieve sustainable programming, developers can use certain practices and techniques, such as:

- **Modular design:** Sustainable programming code is written with a modular design, which means that it is broken down into small, manageable pieces that can be separately updated and maintained. This makes it easier to maintain and update software and reduces the amount of energy and resources required to do so.

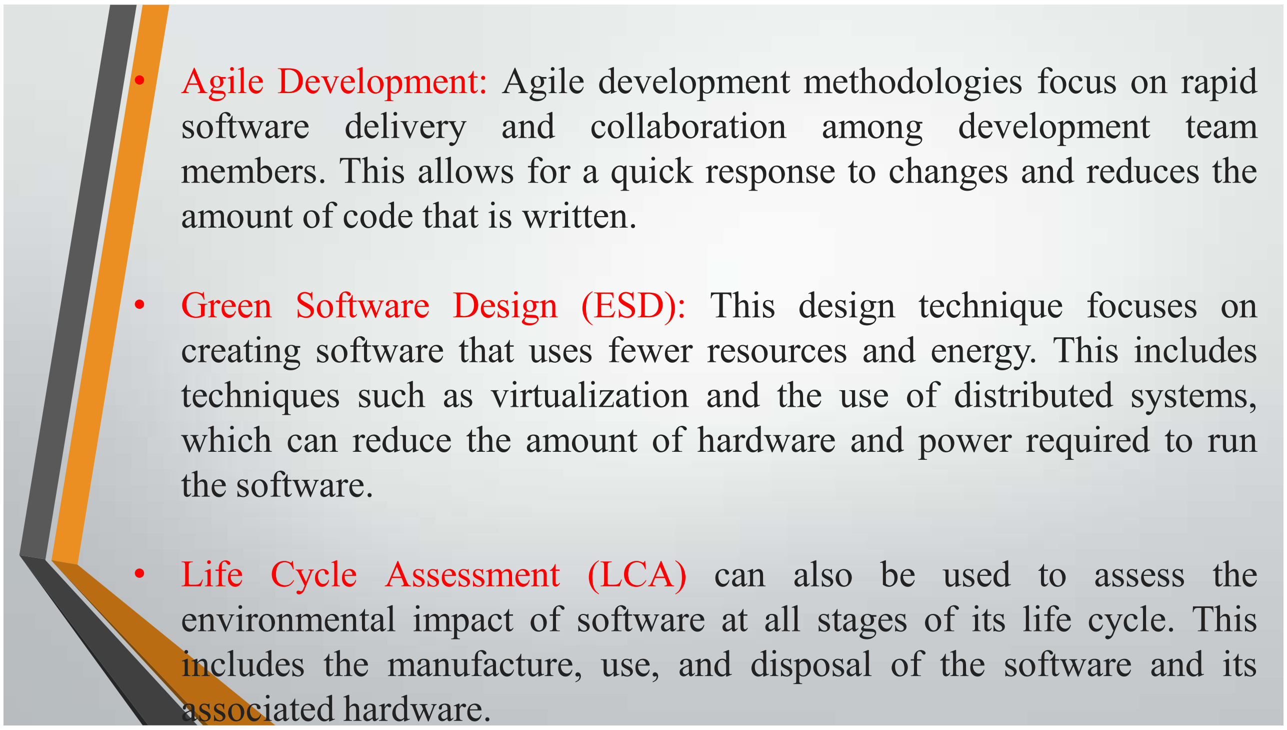
- **Social Responsibility:** The sustainable programming code considers social needs and concerns, such as accessibility and privacy. This means that the software is designed to be accessible to everyone, regardless of disability, and that as little personal data as possible is collected and used.



- **Maintainability:** Sustainable programming code is designed to be easy to maintain and update over the long term. This means it is well documented, written in clean, easy-to-read code, and follows a consistent coding standard.

- **Use of open-source software:** The use of open-source software can be beneficial for environmentally sustainable programming. Open-source software is developed and maintained by the community and therefore does not require the same number of resources as proprietary software. Furthermore, open-source software can be customized and enhanced to meet the specific needs of the user.

Open-source software can also be easier to maintain and update compared to proprietary software. This is because open-source software is usually developed by the community. Bugs or issues can be reported and fixed by other developers.

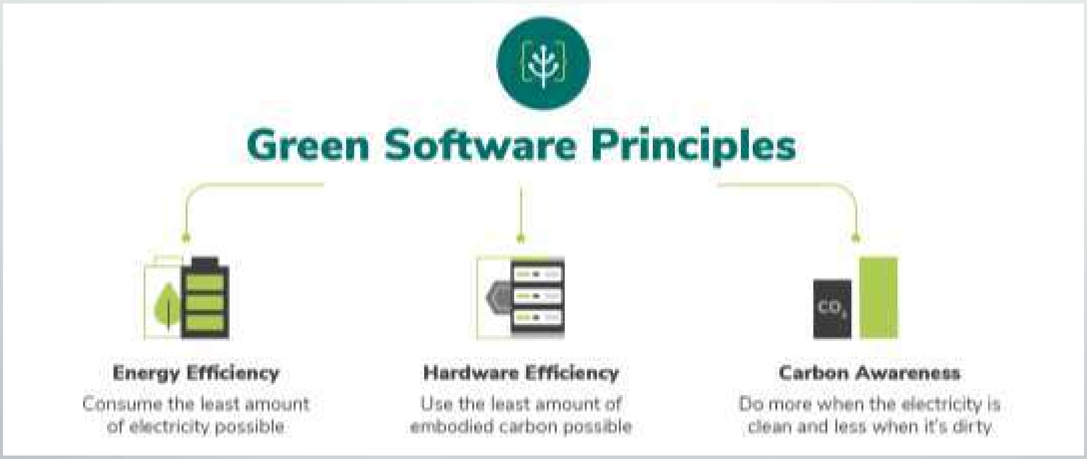


- Agile Development: Agile development methodologies focus on rapid software delivery and collaboration among development team members. This allows for a quick response to changes and reduces the amount of code that is written.

- Green Software Design (ESD): This design technique focuses on creating software that uses fewer resources and energy. This includes techniques such as virtualization and the use of distributed systems, which can reduce the amount of hardware and power required to run the software.

- Life Cycle Assessment (LCA) can also be used to assess the environmental impact of software at all stages of its life cycle. This includes the manufacture, use, and disposal of the software and its associated hardware.

[redacted]


Green Software Principles
Energy Efficiency
Consume the least amount of electricity possible
Hardware Efficiency
Use the least amount of embodied carbon possible
Carbon Awareness
Do more when the electricity is clean and less when it's dirty
CO2

You can integrate the **Sustainable Development Goals (SDGs)** into programming by designing software solutions that address global challenges such as climate change, resource efficiency, education, and social equity. Here are some ways to incorporate SDGs into your programming projects:

1. Sustainable Software Development (General)
•Write **energy-efficient code** to reduce CPU and memory usage.
•Optimize algorithms to **minimize processing power and storage**.
•Use cloud computing and **green hosting solutions**.

2. Object-Oriented Programming (OOP) in Java with SDGs
•**Encapsulation & Modularity** → Build reusable components for sustainability-related applications.
•**Inheritance & Polymorphism** → Design flexible, extendable green technology frameworks.
•**Design Patterns** → Use patterns like MVC (Model-View-Controller) to separate concerns in sustainability applications.

[redacted]

3. Project Ideas (Based on SDGs)

🌱 **SDG 13: Climate Action**

•**Carbon Footprint Tracker** → A Java app that calculates and visualizes CO_2 emissions.

•**Green Energy Optimizer** → Software that suggests energy-efficient alternatives for users.

💧 **SDG 6: Clean Water & Sanitation**

•**Smart Water Management** → A system that monitors and optimizes water consumption using Java-based IoT.

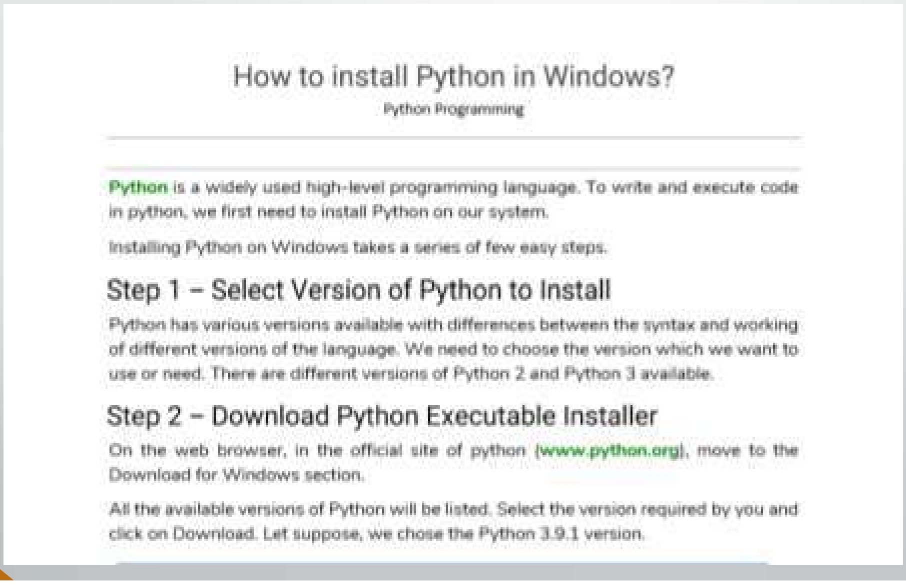
🏙 **SDG 11: Sustainable Cities & Communities**

•**Smart Traffic System** → Use Java to build a system that reduces traffic congestion and emissions.

📚 **SDG 4: Quality Education**

•**E-Learning Platform** → An OOP-based Java learning system for sustainability education.

OOP in Python



How to install Python in Windows?

Python Programming

Python is a widely used high-level programming language. To write and execute code in python, we first need to install Python on our system.

Installing Python on Windows takes a series of few easy steps.

Step 1 – Select Version of Python to Install

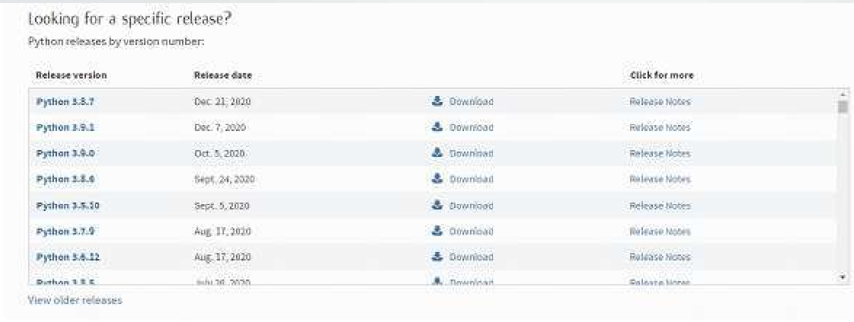
Python has various versions available with differences between the syntax and working of different versions of the language. We need to choose the version which we want to use or need. There are different versions of Python 2 and Python 3 available.

Step 2 – Download Python Executable Installer

On the web browser, in the official site of python (www.python.org), move to the Download for Windows section.

All the available versions of Python will be listed. Select the version required by you and click on Download. Let suppose, we chose the Python 3.9.1 version.

[redacted]



Looking for a specific release?

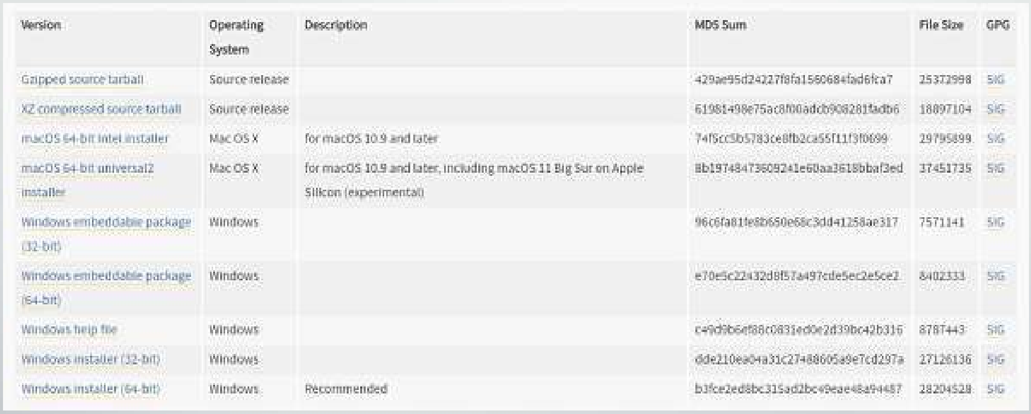
Python releases by version number:

Release version	Release date		Click for more
Python 3.8.7	Dec. 21, 2020	Download	Release Notes
Python 3.9.1	Dec. 7, 2020	Download	Release Notes
Python 3.9.0	Oct. 5, 2020	Download	Release Notes
Python 3.8.6	Sept. 24, 2020	Download	Release Notes
Python 3.5.10	Sept. 5, 2020	Download	Release Notes
Python 3.7.9	Aug. 17, 2020	Download	Release Notes
Python 3.6.12	Aug. 17, 2020	Download	Release Notes
Python 3.8.5	July 20, 2020	Download	Release Notes

View older releases

On clicking download, various available executable installers shall be visible with different operating system specifications. Choose the installer which suits your system operating system and download the installer. Let suppose, we select the Windows installer (64 bits).

The download size is less than 30MB.



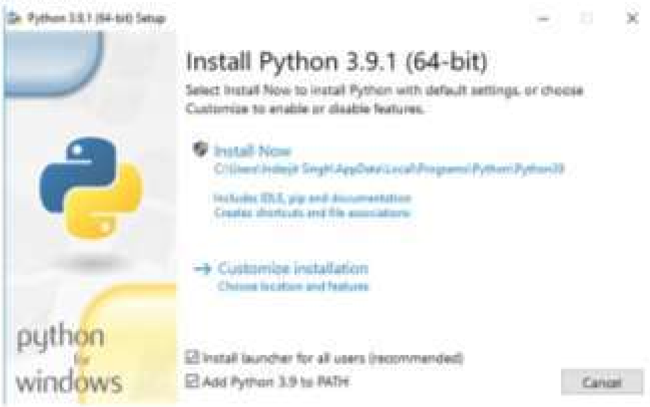
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		429ae95d24227f8fa1560684fad6fca7	25372998	SIG
XZ compressed source tarball	Source release		61981498e75ac8f00adcb908281fadb6	18897104	SIG
macOS 64-bit Intel installer	Mac OS X	for macOS 10.9 and later	74f5cc5b5783ce8fb2ca55f11f3f0699	29795899	SIG
macOS 64-bit universal2 installer	Mac OS X	for macOS 10.9 and later, including macOS 11 Big Sur on Apple Silicon (experimental)	8b19748473609241e60aa3618bbaf3ed	37451735	SIG
Windows embeddable package (32-bit)	Windows		96c6fa81fe8b650e68c3dd41258ae317	7571141	SIG
Windows embeddable package (64-bit)	Windows		e70e5c22432d8f57a497cde5ec2e5ce2	8402333	SIG
Windows help file	Windows		c49d9b6ef88c0831ed0e2d39bc42b316	8787443	SIG
Windows installer (32-bit)	Windows		dde210ea04a31c27488605a9e7cd297a	27126136	SIG
Windows installer (64-bit)	Windows	Recommended	b3fce2ed8bc315ad2bc49eae48a94487	28204528	SIG

[redacted]

Step 3 – Run Executable Installer

We downloaded the Python 3.9.1 Windows 64 bit installer.

Run the installer. Make sure to select both the checkboxes at the bottom and then click Install Now.


Python 3.9.1 (64-bit) Setup

Install Python 3.9.1 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

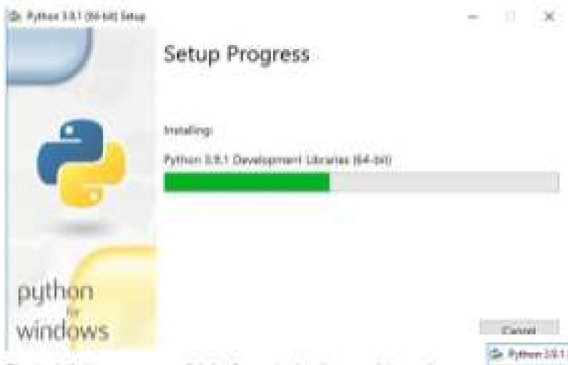
Install Now
C:\Users\Indrajit Singh\AppData\Local\Programs\Python\Python39

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.9 to PATH

On clicking the Install Now, The installation process starts.

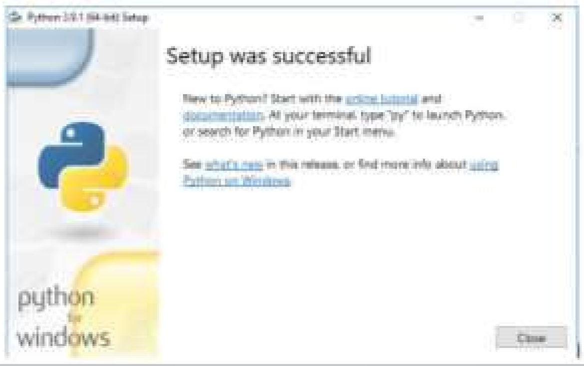

Python 3.9.1 (64-bit) Setup

Setup Progress

Installing:

Python 3.9.1 Development Libraries (64-bit)

The installation process will take few minutes to complete and on successful, the following screen is displayed.


Python 3.9.1 (64-bit) Setup

Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

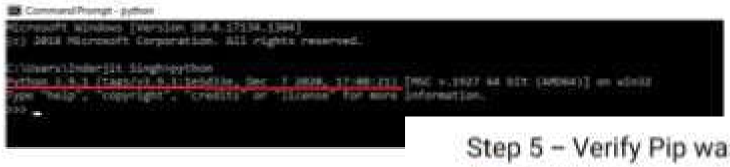
See what's new in this release, or find more info about using Python on Windows.

[redacted]

Step 4 – Verify Python is installed on Windows

To ensure if Python is successfully installed on your system. Follow the given steps –

- Open the command prompt.
- Type 'python' and press enter.
- The version of the python which you have installed will be displayed if the python is successfully installed on your windows.



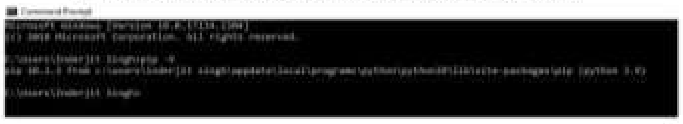
```
Microsoft Windows [Version 10.0.17134.1304]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Inderjit Singh>python
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Step 5 – Verify Pip was installed

Pip is a powerful package management system for Python software packages. Thus, make sure that you have it installed.

To verify if pip was installed, follow the given steps –

- Open the command prompt.
- Enter pip –V to check if pip was installed.
- The following output appears if pip is installed successfully.



```
Microsoft Windows [Version 10.0.17134.1304]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Inderjit Singh>pip -V
pip 20.2.3 from c:\users\inderjit singh\appdata\local\programs\python\python39\lib\site-packages\pip (python 3.9)

C:\Users\Inderjit Singh>
```

We have successfully installed python and pip on our Windows system.

Python OOPs Concepts

Object Oriented Programming is a fundamental concept in Python, empowering developers to build modular, maintainable, and scalable applications. By understanding the core OOP principles (classes, objects, inheritance, encapsulation, polymorphism, and abstraction), programmers can leverage the full potential of Python OOP capabilities to design elegant and efficient solutions to complex problems.

OOPs is a way of organizing code that uses objects and classes to represent real-world entities and their behavior. In OOPs, object has attributes thing that has specific data and can perform certain actions using methods.

OOPs Concepts in Python

•Class in Python

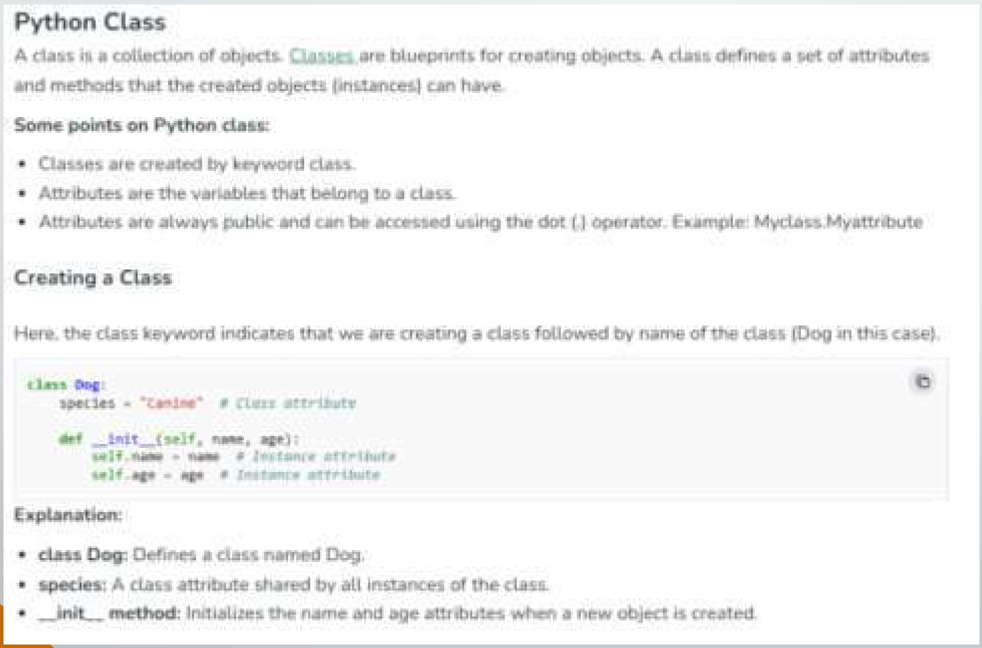
•Objects in Python

•Polymorphism in Python

•Encapsulation in Python

•Inheritance in Python

•Data Abstraction in Python



Python Class

A class is a collection of objects. Classes are blueprints for creating objects. A class defines a set of attributes and methods that the created objects (instances) can have.

Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Example: Myclass.Myattribute

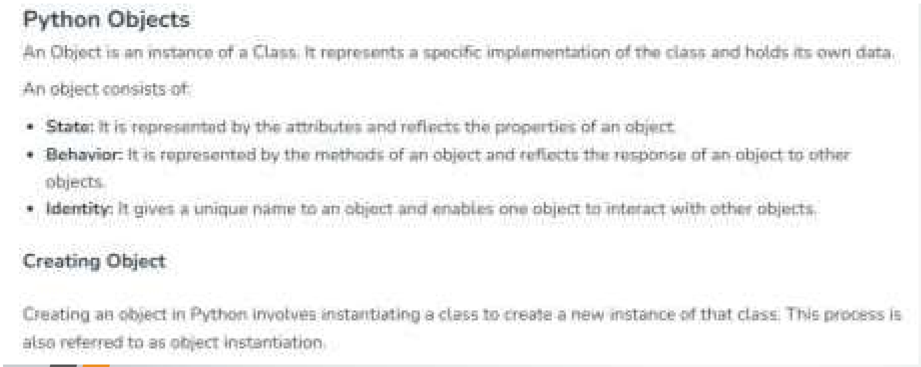
Creating a Class

Here, the class keyword indicates that we are creating a class followed by name of the class (Dog in this case).

```
class Dog:
    species = "Canine"  # Class attribute

    def __init__(self, name, age):
        self.name = name  # Instance attribute
        self.age = age  # Instance attribute
```

Explanation:

- **class Dog:** Defines a class named Dog.
- **species:** A class attribute shared by all instances of the class.
- **__init__ method:** Initializes the name and age attributes when a new object is created.



Python Objects

An Object is an instance of a Class. It represents a specific implementation of the class and holds its own data.

An object consists of:

- **State:** It is represented by the attributes and reflects the properties of an object.
- **Behavior:** It is represented by the methods of an object and reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

Creating Object

Creating an object in Python involves instantiating a class to create a new instance of that class. This process is also referred to as object instantiation.

```
class Dog:
    species = "Canine"  # Class attribute

    def __init__(self, name, age):
        self.name = name  # Instance attribute
        self.age = age  # Instance attribute
# Creating an object of the Dog class
dog1 = Dog("Buddy", 3)

print(dog1.name)
print(dog1.species)
```

Output

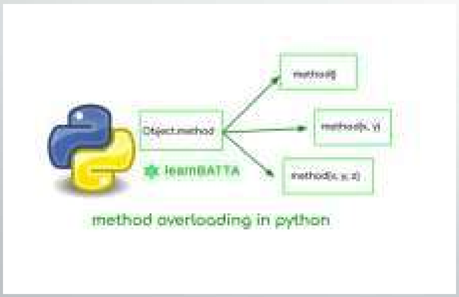
```
Buddy
Canine
```

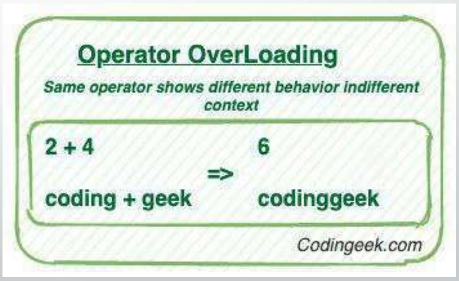
Explanation:
- **dog1 = Dog("Buddy", 3):** Creates an object of the Dog class with name as "Buddy" and age as 3.
- **dog1.name:** Accesses the instance attribute name of the dog1 object.
- **dog1.species:** Accesses the class attribute species of the dog1 object.

[redacted]

method overloading

The problem with method overloading in Python is that we may overload the methods but can only use the latest defined method.



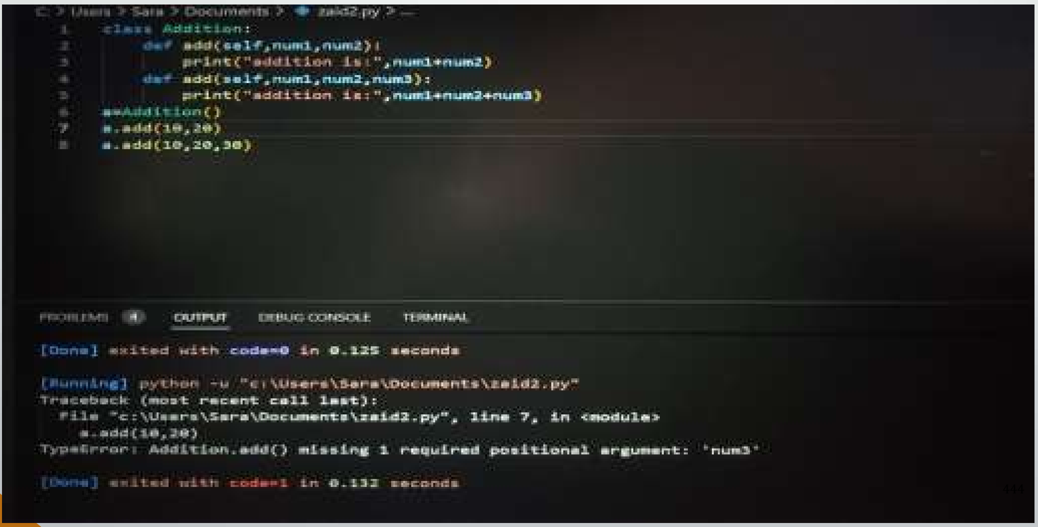

Operator OverLoading

Same operator shows different behavior indifferent context

```
2 + 4              6
          =>
coding + geek      codinggeek
```

Codingeek.com

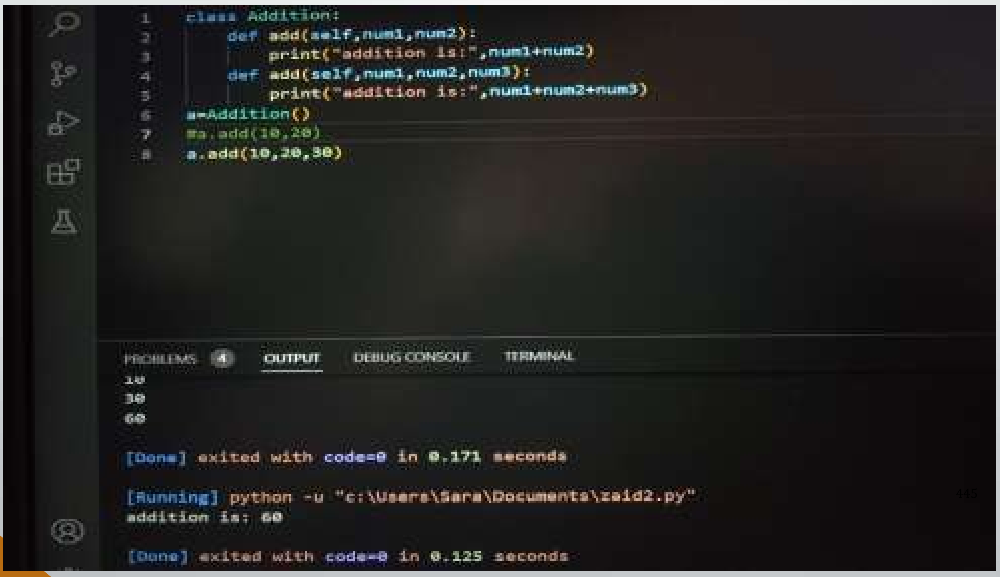
Example 1- Wrong



```
class Addition:
    def add(self,num1,num2):
        print("addition is:",num1+num2)
    def add(self,num1,num2,num3):
        print("addition is:",num1+num2+num3)
a=Addition()
a.add(10,20)
a.add(10,20,30)
```

```
[Done] exited with code=0 in 0.125 seconds

[Running] python -u "c:\Users\Sara\Documents\zaid2.py"
Traceback (most recent call last):
  File "c:\Users\Sara\Documents\zaid2.py", line 7, in <module>
    a.add(10,20)
TypeError: Addition.add() missing 1 required positional argument: 'num3'

[Done] exited with code=1 in 0.132 seconds
```

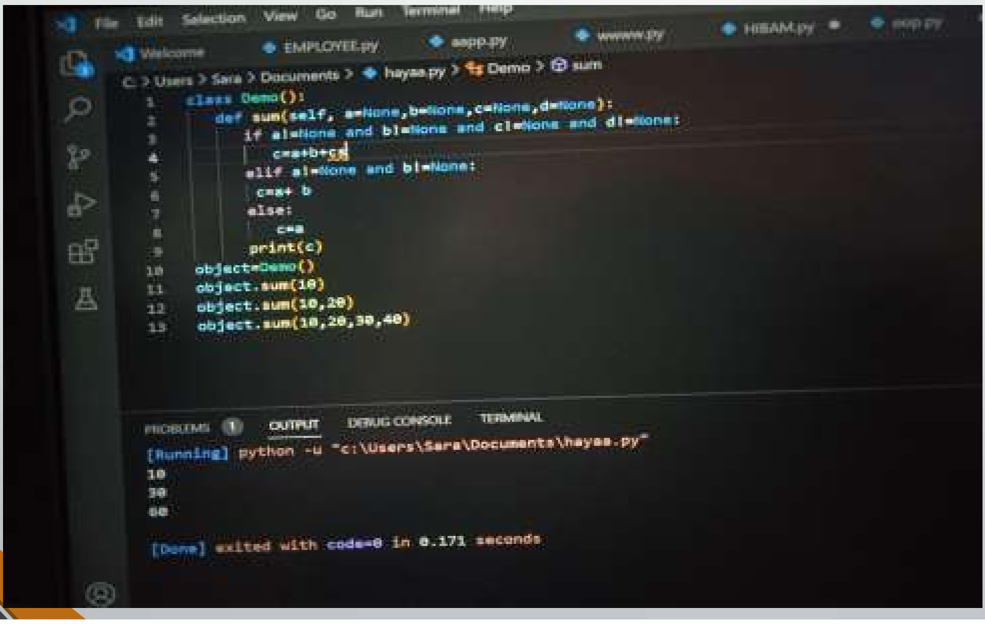
Example 1 correct



```
class Addition:
    def add(self,num1,num2):
        print("addition is:",num1+num2)
    def add(self,num1,num2,num3):
        print("addition is:",num1+num2+num3)
a=Addition()
#a.add(10,20)
a.add(10,20,30)
```

```
10
30
60

[Done] exited with code=0 in 0.171 seconds

[Running] python -u "c:\Users\Sara\Documents\zaid2.py"
addition is: 60

[Done] exited with code=0 in 0.125 seconds
```

Example 2



File Edit Selection View Go Run Terminal Help
Welcome
EMPLOYEE.py
aapp.py
wwww.py
HIBAM.py
aap.py
C: > Users > Sara > Documents > hayaa.py > Demo > sum

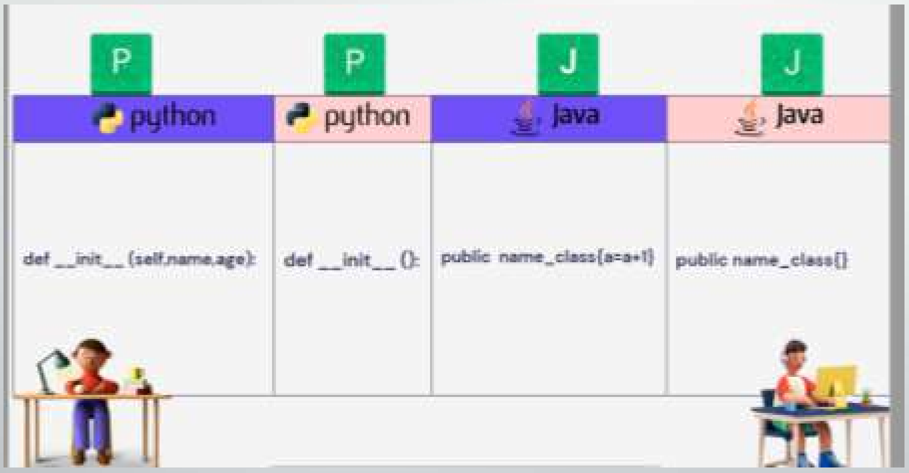
```
class Demo():
    def sum(self, a=None,b=None,c=None,d=None):
        if a!=None and b!=None and c!=None and d!=None:
            c=a+b+c
        elif a!=None and b!=None:
          c=a+ b
        else:
            c=a
        print(c)
object=Demo()
object.sum(10)
object.sum(10,20)
object.sum(10,20,30,40)
```

PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL

```
[Running] python -u "c:\Users\Sara\Documents\hayaa.py"
10
30
60

[Done] exited with code=0 in 0.171 seconds
```

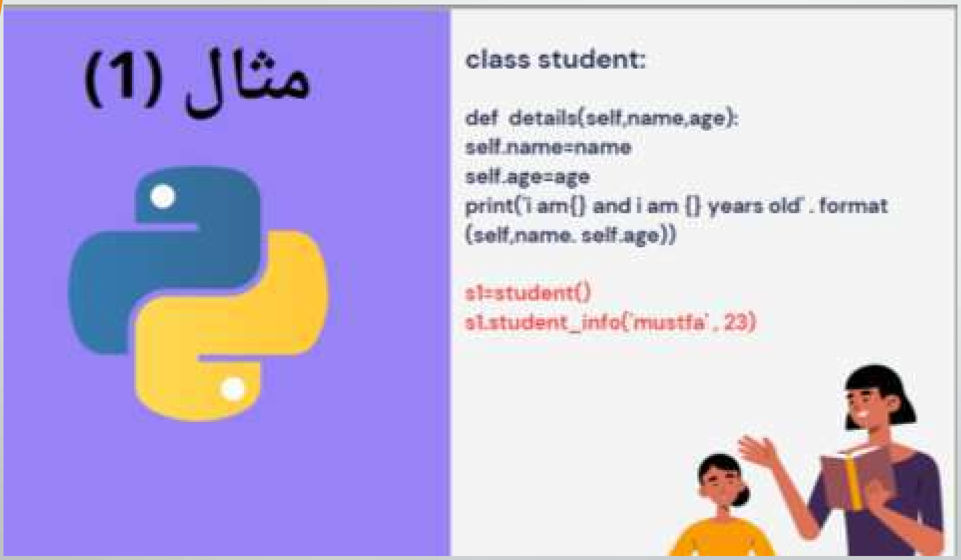
A constructor in Python is a special method used to initialize objects of a class. It is automatically called when an object is created. The constructor's name is always `__init__`. It sets up the initial state of the object by assigning values to the object's attributes.

```
class Dog:
    def __init__(self, name, breed):
   self.name = name
  self.breed = breedmy_dog = Dog("Buddy", "Golden Retriever")
print(my_dog.name)
# Output: Buddyprint(my_dog.breed)
# Output: Golden Retriever
```



P	P	J	J
python	python	Java	Java
def __init__ (self,name,age):	def __init__ ():	public name_class(a=a+1)	public name_class()

[redacted]



```
class student:

def  details(self,name,age):
self.name=name
self.age=age
print('i am{} and i am {} years old' . format
(self,name. self.age))

s1=student()
s1.student_info('mustfa' , 23)
```

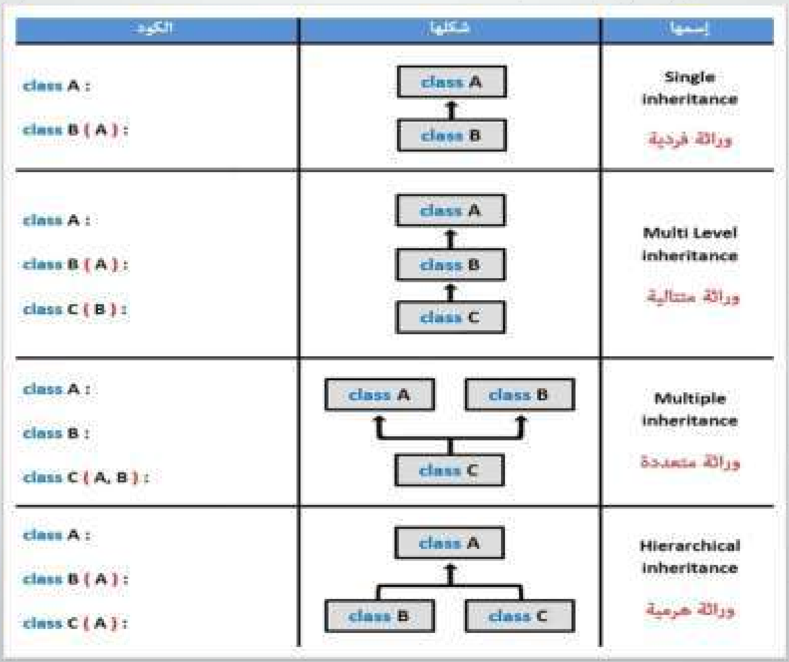
Inheritance in python

- Inheritance means including the content of a class in another class.
- In Python, a class can inherit from another class in order to obtain the functions and variables in it.
- A class that inherits from another class is called a child class. It is called a Subclass, and it is also called (Derived Class, Extended Class or Child Class).
- The class that inherits its contents to another class is called the parent class, and it is called the Superclass and it is also called (Base Class or Parent Class).

- The subclass directly inherits everything in the superclass except for the properties that are defined as parameters inside the __init__() function.

- If we want to call the __init__() function in the Superclass, we use a ready-made function called super().

Forms of inheritance in Python



الكود	شكلها	إسمها

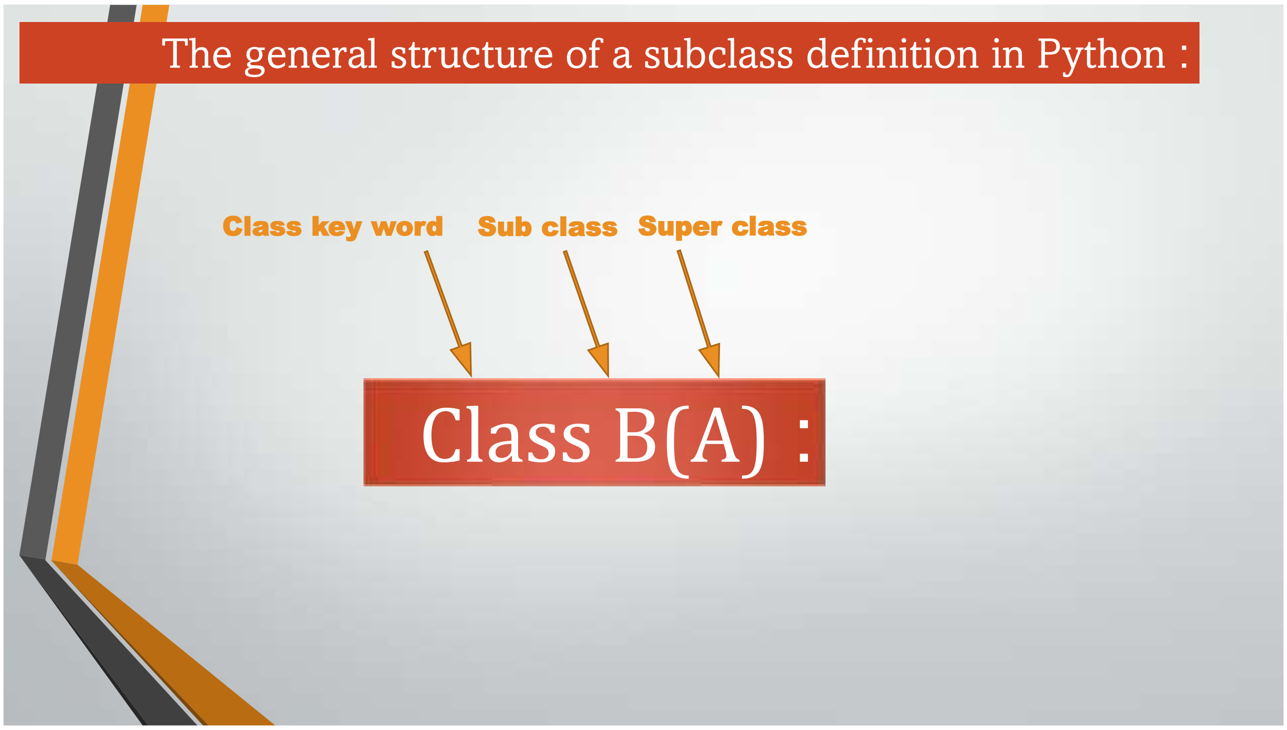
class A :
class B (A) :
	$\begin{tikzcd} \text{class A} \\ \text{class B} \arrow[u] \end{tikzcd}$	Single inheritance
		وراثة فردية

class A :
class B (A) :
class C (B) :
	$\begin{tikzcd} \text{class A} \\ \text{class B} \arrow[u] \\ \text{class C} \arrow[u] \end{tikzcd}$	Multi Level inheritance
		وراثة متتالية

class A :
class B :
class C (A, B) :
	$\begin{tikzcd} \text{class A} & & \text{class B} \\ & \text{class C} \arrow[ul] \arrow[ur] & \end{tikzcd}$	Multiple inheritance
		وراثة متعددة

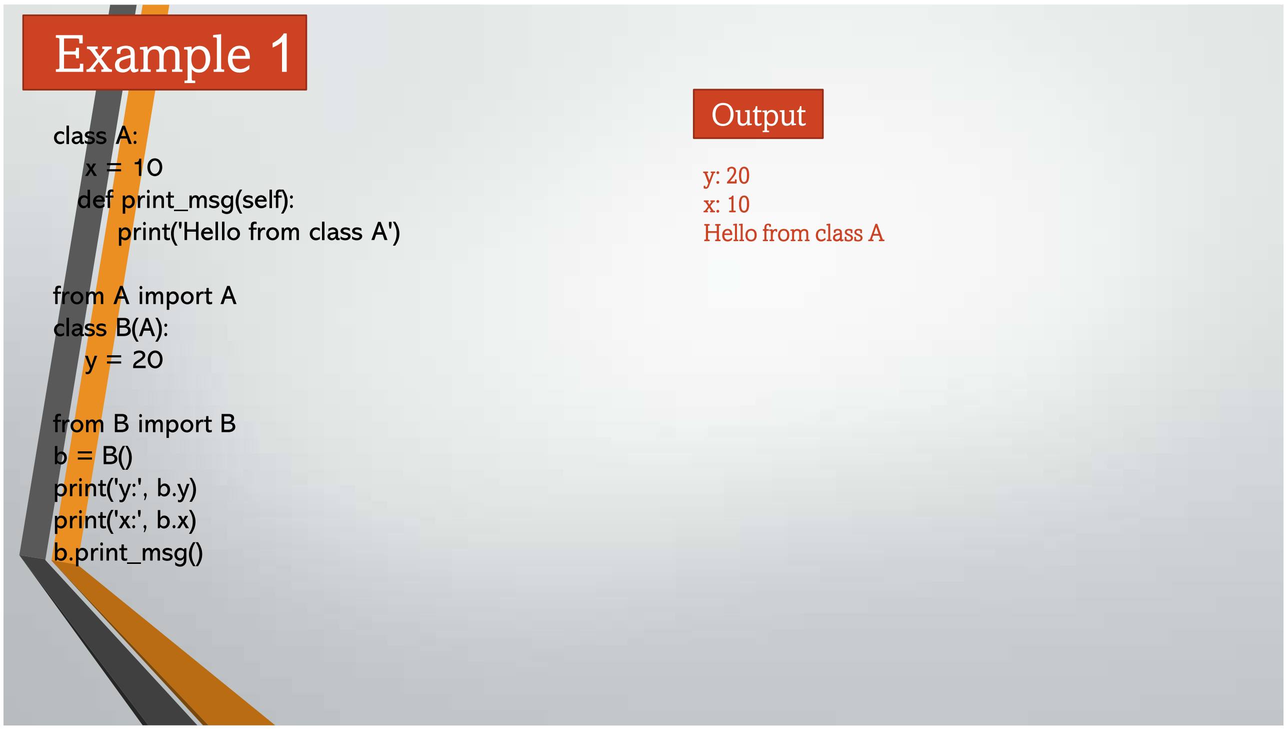
class A :
class B (A) :
class C (A) :
	$\begin{tikzcd} & \text{class A} & \\ \text{class B} \arrow[ur] & & \text{class C} \arrow[ul] \end{tikzcd}$	Hierarchical inheritance
		وراثة هرمية

[redacted]



The general structure of a subclass definition in Python :

```
Class B(A) :
```



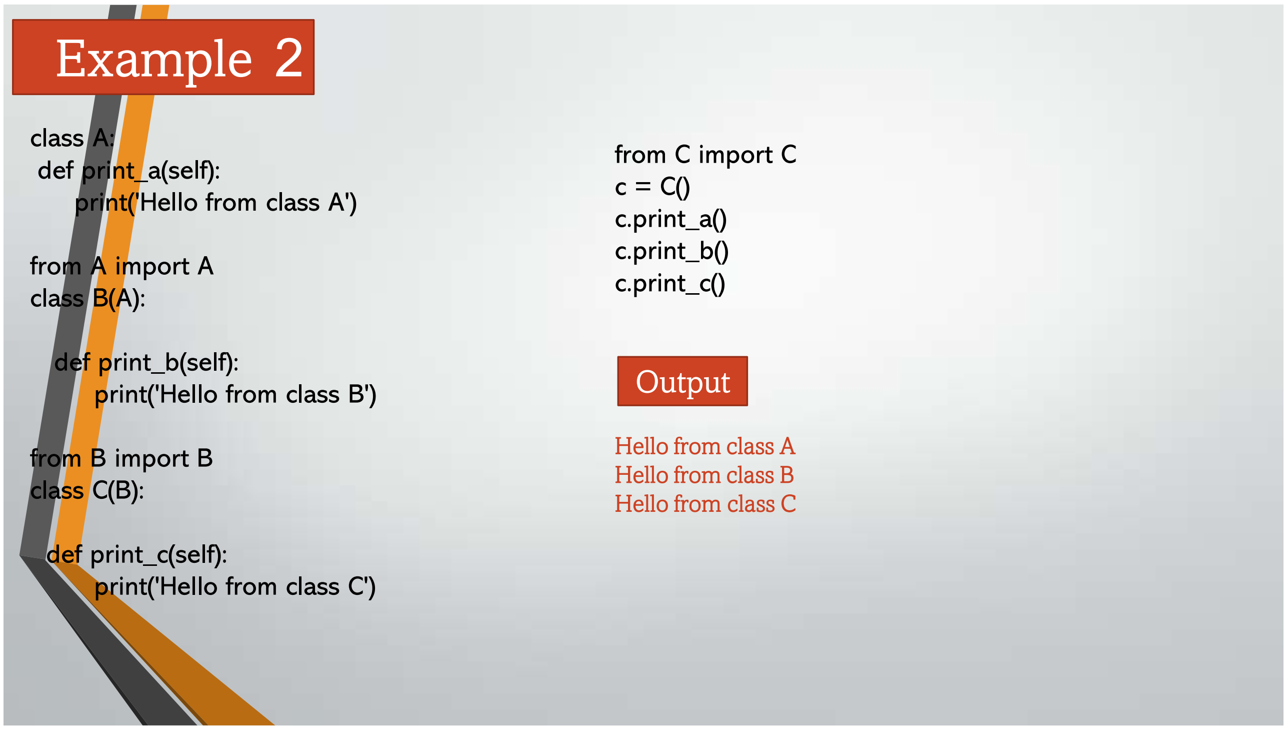
Example 1

```
class A:
    x = 10
   def print_msg(self):
        print('Hello from class A')

from A import A
class B(A):
    y = 20

from B import B
b = B()
print('y:', b.y)
print('x:', b.x)
b.print_msg()
```

Output

```
y: 20
x: 10
Hello from class A
```

[redacted]



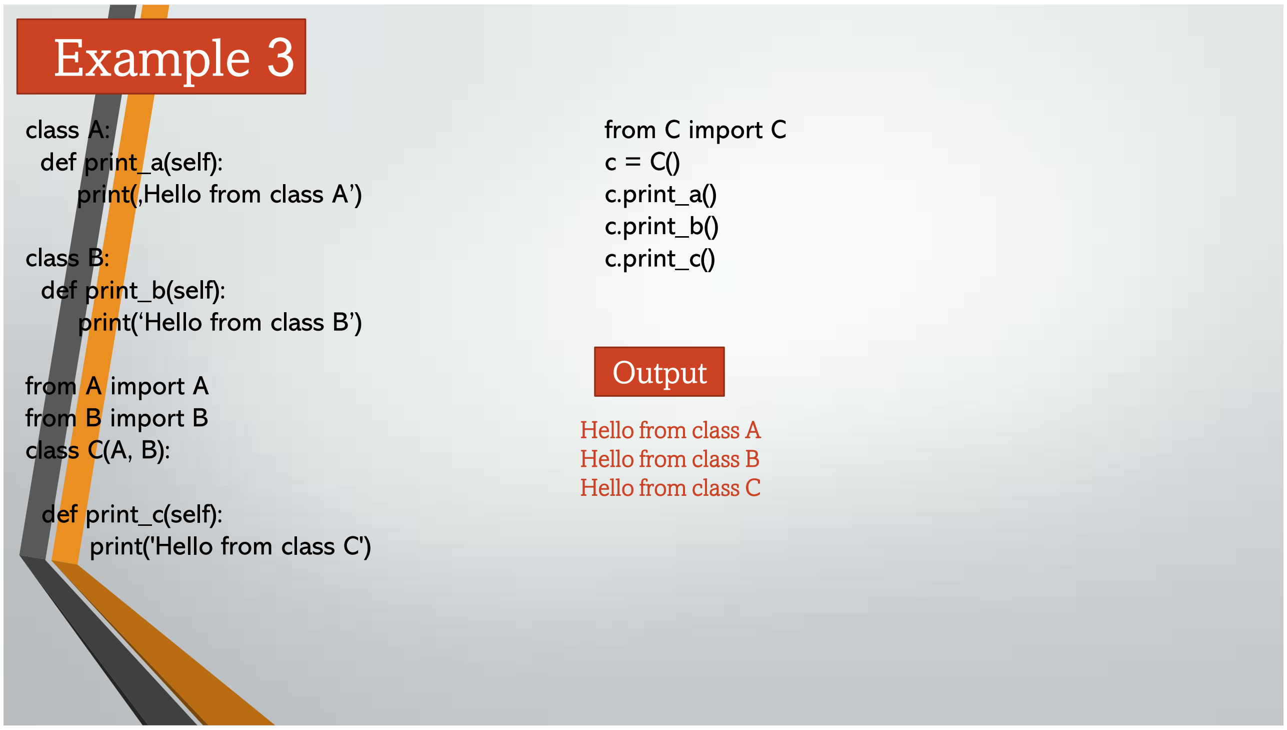
Example 2

```
class A:
 def print_a(self):
      print('Hello from class A')

from A import A
class B(A):

   def print_b(self):
        print('Hello from class B')

from B import B
class C(B):

  def print_c(self):
        print('Hello from class C')
```

```
from C import C
c = C()
c.print_a()
c.print_b()
c.print_c()
```

Output

```
Hello from class A
Hello from class B
Hello from class C
```



Example 3

```
class A:
  def print_a(self):
       print(‚Hello from class A’)

class B:
  def print_b(self):
       print(‘Hello from class B’)

from A import A
from B import B
class C(A, B):

  def print_c(self):
       print('Hello from class C')
```

```
from C import C
c = C()
c.print_a()
c.print_b()
c.print_c()
```

Output

```
Hello from class A
Hello from class B
Hello from class C
```

[redacted]

Example 4

```
class A:
   def print_a(self):
          print(‚Hello from class A’)

from A import A
class B(A):

   def print_b(self):
          print(‚Hello from class B’)

from A import A
class C(A):

   def print_c(self):
          print(‚Hello from class C’)
```

```
from B import B
from C import C
c = C()
c.print_a()
c.print_c()

b = B()
b.print_a()
b.print_b()
```

Output

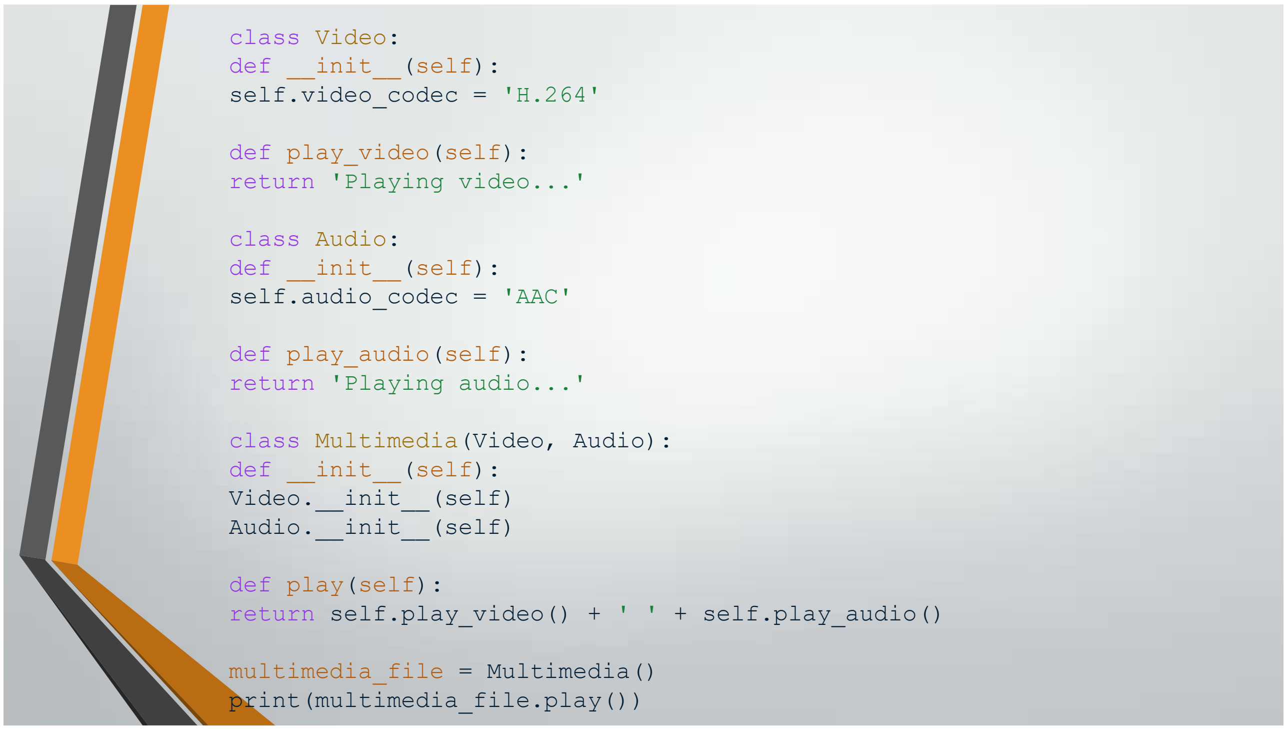
```
Hello from class A
Hello from class C
Hello from class A
Hello from class B
```

Multiple inheritance in python

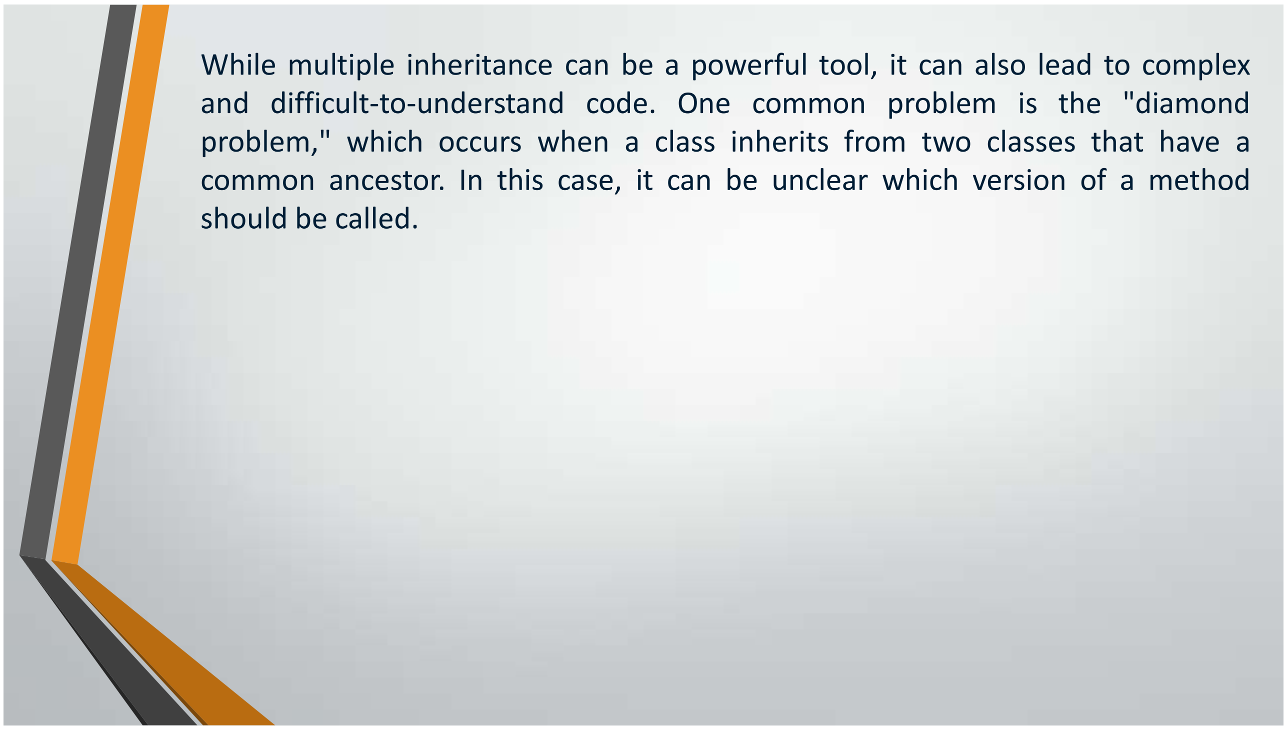
Multiple inheritance is a feature in Python where a class can inherit attributes and methods from more than one parent class. This allows for the creation of complex class hierarchies and the reuse of code across different parts of an application.

When a class inherits from multiple parent classes, Python needs a way to determine the order in which methods are searched for in the inheritance hierarchy. This is known as the Method Resolution Order (MRO). Python uses the C3 linearization algorithm to determine the MRO, which ensures that the search order is consistent and predictable.

[redacted]



```
class Video:
def __init__(self):
self.video_codec = 'H.264'

def play_video(self):
return 'Playing video...'

class Audio:
def __init__(self):
self.audio_codec = 'AAC'

def play_audio(self):
return 'Playing audio...'

class Multimedia(Video, Audio):
def __init__(self):
Video.__init__(self)
Audio.__init__(self)

def play(self):
return self.play_video() + ' ' + self.play_audio()

multimedia_file = Multimedia()
print(multimedia_file.play())
```



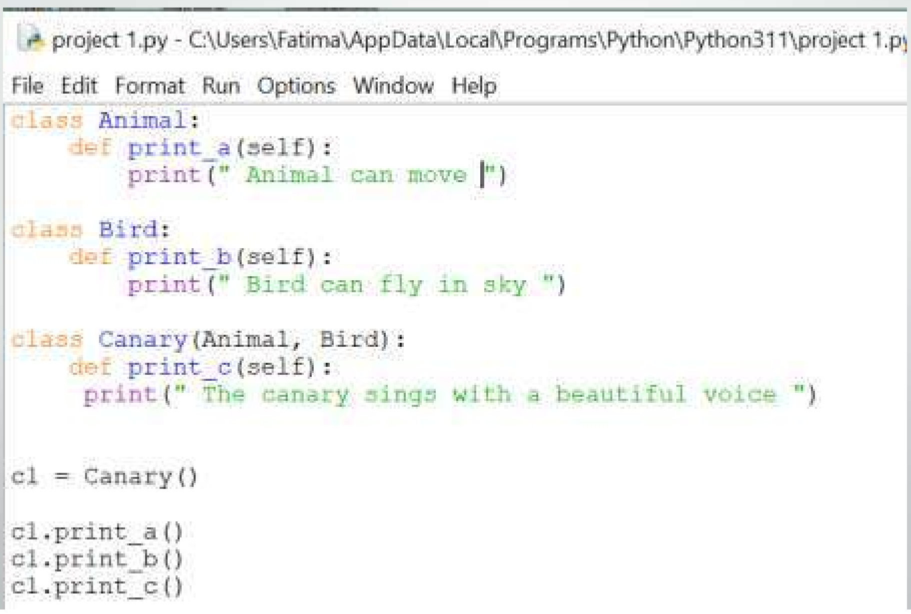
While multiple inheritance can be a powerful tool, it can also lead to complex and difficult-to-understand code. One common problem is the "diamond problem," which occurs when a class inherits from two classes that have a common ancestor. In this case, it can be unclear which version of a method should be called.

[redacted]

```
class A:
def method(self):
print("Method A")

class B(A):
def method(self):
print("Method B")

class C(A):
def method(self):
print("Method C")

class D(B, C):
pass

d = D()
d.method()
print(D.mro())
```

In this example, class D inherits from both B and C, which both inherit from A. When `d.method()` is called, Python uses the MRO to determine that `B`'s method should be called first.

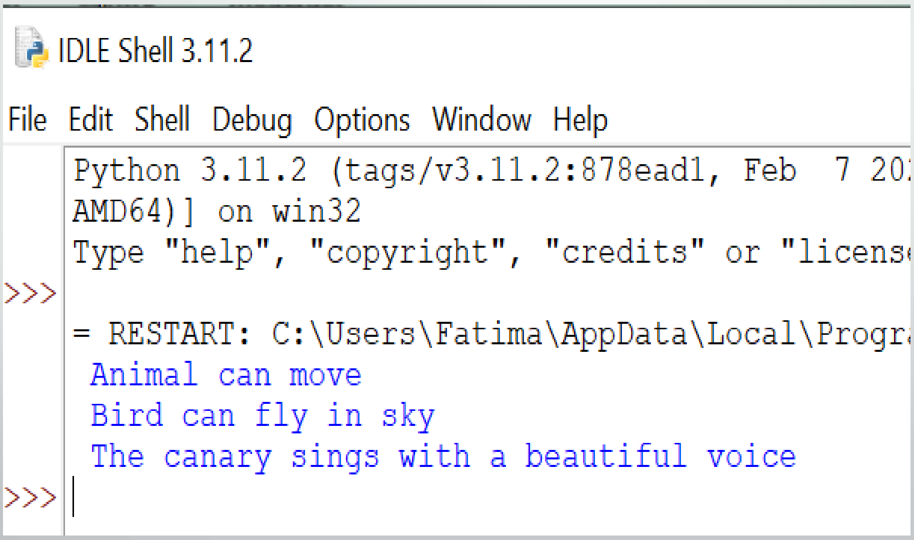
Example



project 1.py - C:\Users\Fatima\AppData\Local\Programs\Python\Python311\project 1.py

```
class Animal:
    def print_a(self):
        print(" Animal can move ")

class Bird:
    def print_b(self):
        print(" Bird can fly in sky ")

class Canary(Animal, Bird):
    def print_c(self):
     print(" The canary sings with a beautiful voice ")


c1 = Canary()

c1.print_a()
c1.print_b()
c1.print_c()
```

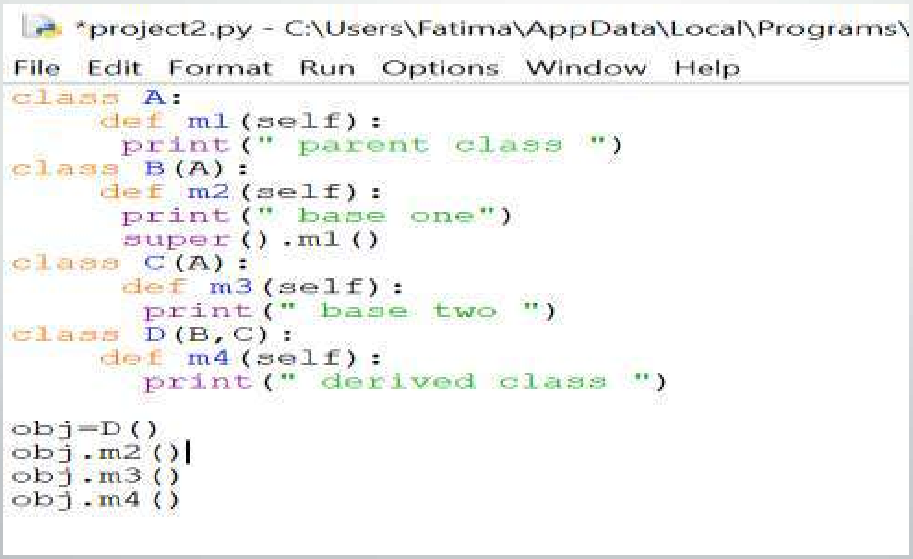
Output



IDLE Shell 3.11.2

```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 20
AMD64)] on win32
Type "help", "copyright", "credits" or "licens
>>>
= RESTART: C:\Users\Fatima\AppData\Local\Progr
 Animal can move
 Bird can fly in sky
 The canary sings with a beautiful voice
>>>
```

Example

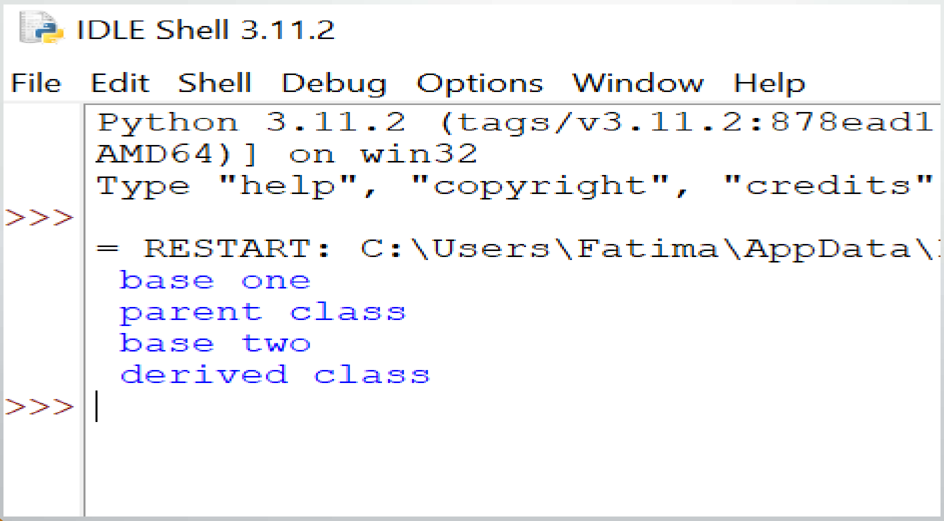


*project2.py - C:\Users\Fatima\AppData\Local\Programs\

```
class A:
    def m1(self):
     print(" parent class ")
class B(A):
    def m2(self):
     print(" base one")
     super().m1()
class C(A):
     def m3(self):
      print(" base two ")
class D(B,C):
    def m4(self):
      print(" derived class ")

obj=D()
obj.m2()
obj.m3()
obj.m4()
```

[redacted]

Output



IDLE Shell 3.11.2

```
    Python 3.11.2 (tags/v3.11.2:878ead1
    AMD64)] on win32
    Type "help", "copyright", "credits"
>>>
    = RESTART: C:\Users\Fatima\AppData\
     base one
     parent class
     base two
     derived class
>>> |
```

What is Interface in Python?

An abstract class is a class which may contain some abstract methods as well as non-abstract methods also. Imagine there is an abstract class

which contains only abstract methods and doesn't contain any concrete methods, such classes are called Interfaces.

Therefore, an interface is nothing but an abstract class which can contains only abstract methods.

كما نعرف ان الـ Abstract class هو كلاس يحتوي على Abstract methods وايضا ممكن لا يحتوي. وهناك ايضاً Abstract class الذي يحتوي على Abstract methods فقط والتي تدعى interface.

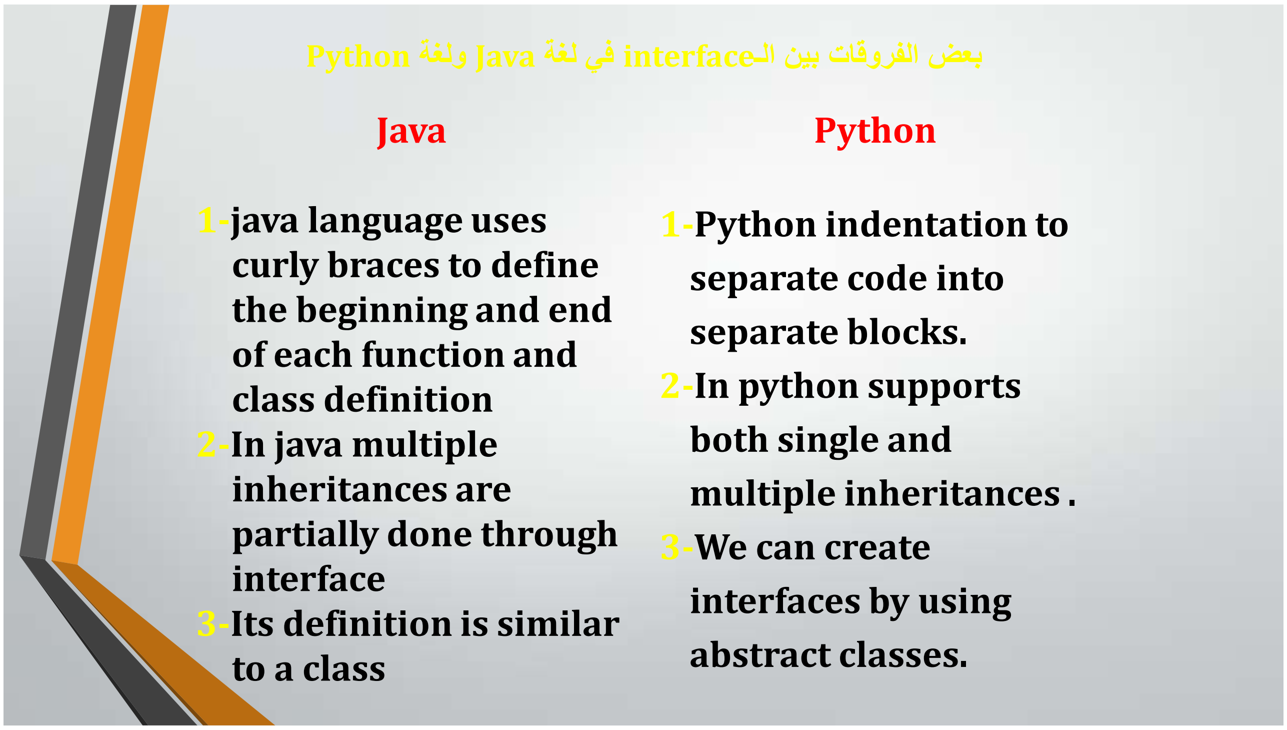
لذلك نستطيع ان نقول ان الواجهة في python تحتوي على Abstract class الذي يحتوي على Abstract method فقط .

Points to Remember:

In python there is no separate keyword to create an interface. We can create interfaces by using abstract classes which have only abstract methods.

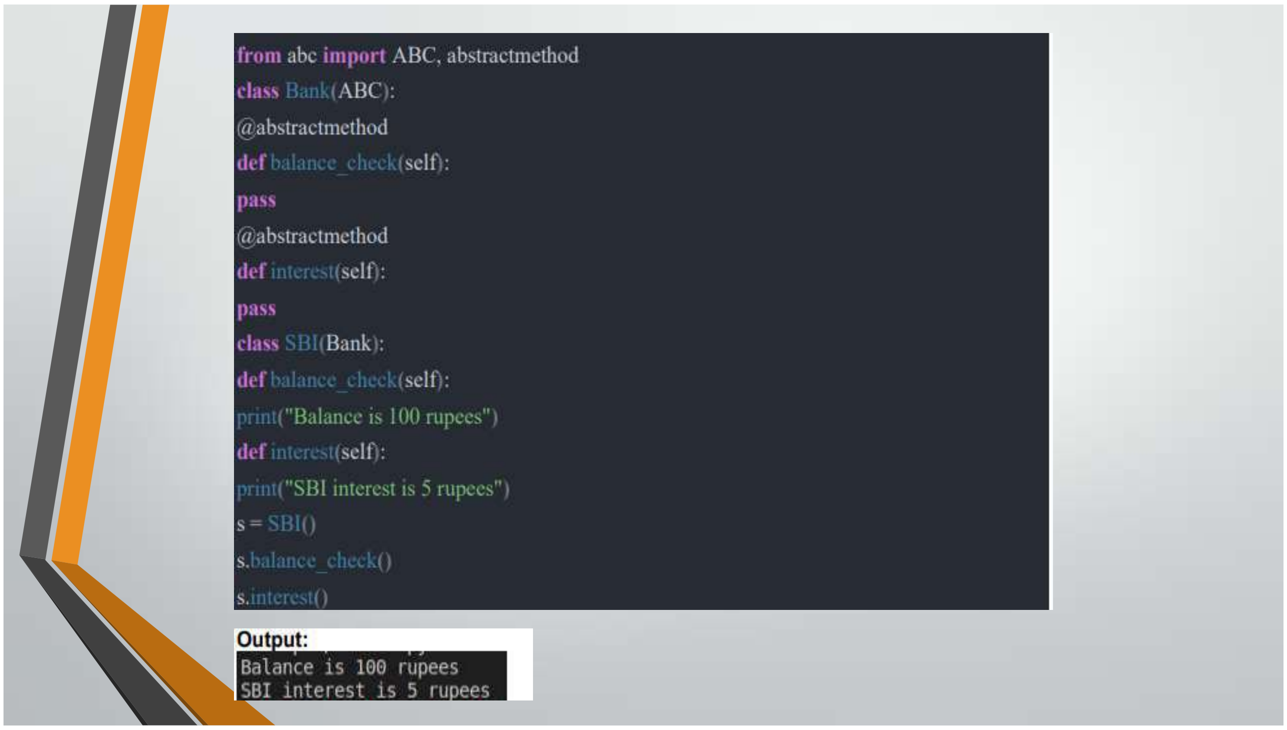
ملاحظة: في بايثون لا نحتاج لكلمة معينة حتى نكون واجهة نحن نستطيع تكوين واجهة باستخدام الـ abstract class الذي يحتوي على abstract methods فقط.

[redacted]



بعض الفروقات بين الـinterface في لغة Java ولغة Python

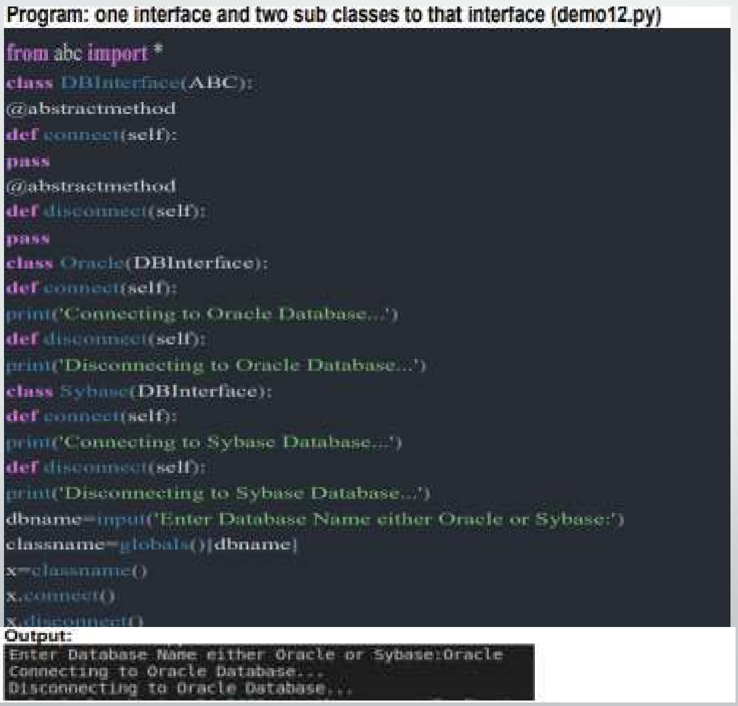
Java	Python
1-java language uses curly braces to define the beginning and end of each function and class definition	1-Python indentation to separate code into separate blocks.
2-In java multiple inheritances are partially done through interface	2-In python supports both single and multiple inheritances .
3-Its definition is similar to a class	3-We can create interfaces by using abstract classes.



```
from abc import ABC, abstractmethod
class Bank(ABC):
@abstractmethod
def balance_check(self):
pass
@abstractmethod
def interest(self):
pass
class SBI(Bank):
def balance_check(self):
print("Balance is 100 rupees")
def interest(self):
print("SBI interest is 5 rupees")
s = SBI()
s.balance_check()
s.interest()
```

Output:

```
Balance is 100 rupees
SBI interest is 5 rupees
```



Program: one interface and two sub classes to that interface (demo12.py)

```
from abc import *
class DBInterface(ABC):
@abstractmethod
def connect(self):
pass
@abstractmethod
def disconnect(self):
pass
class Oracle(DBInterface):
def connect(self):
print('Connecting to Oracle Database...')
def disconnect(self):
print('Disconnecting to Oracle Database...')
class Sybase(DBInterface):
def connect(self):
print('Connecting to Sybase Database...')
def disconnect(self):
print('Disconnecting to Sybase Database...')
dbname=input('Enter Database Name either Oracle or Sybase:')
classname=globals()[dbname]
x=classname()
x.connect()
x.disconnect()
```

Output:

```
Enter Database Name either Oracle or Sybase:Oracle
Connecting to Oracle Database...
Disconnecting to Oracle Database...
```

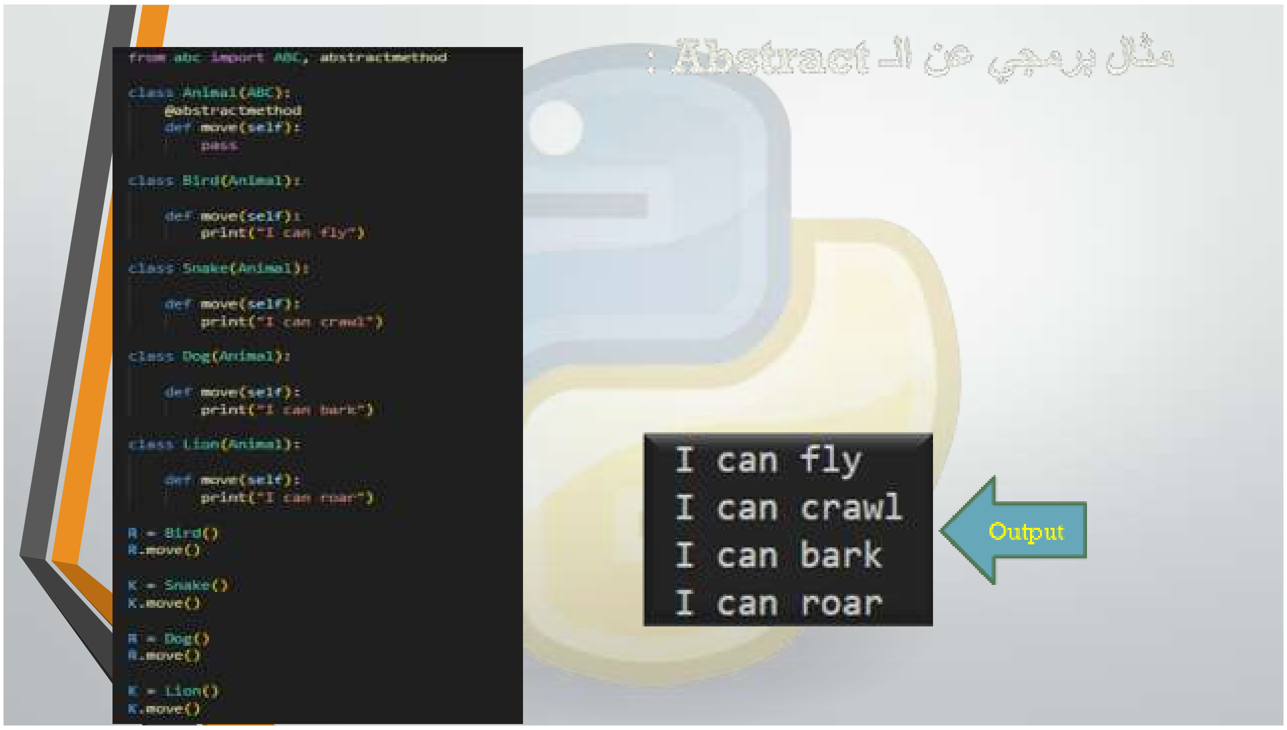
❖**In Python, an abstract class** is a class that cannot be instantiated on its own and is meant to be subclassed by other classes. Abstract classes are created using the abc (Abstract Base Classes) module.

Example: Abstraction in Python in Action

Let's take a practical example of creating an abstract class Shape to represent geometric shapes. We'll define two concrete subclasses: Rectangle and Circle. Both shapes need to compute their area and perimeter, but the implementation of these methods will differ for each shape

https://www.youtube.com/watch?v=97V7ICVeTJc

[redacted]

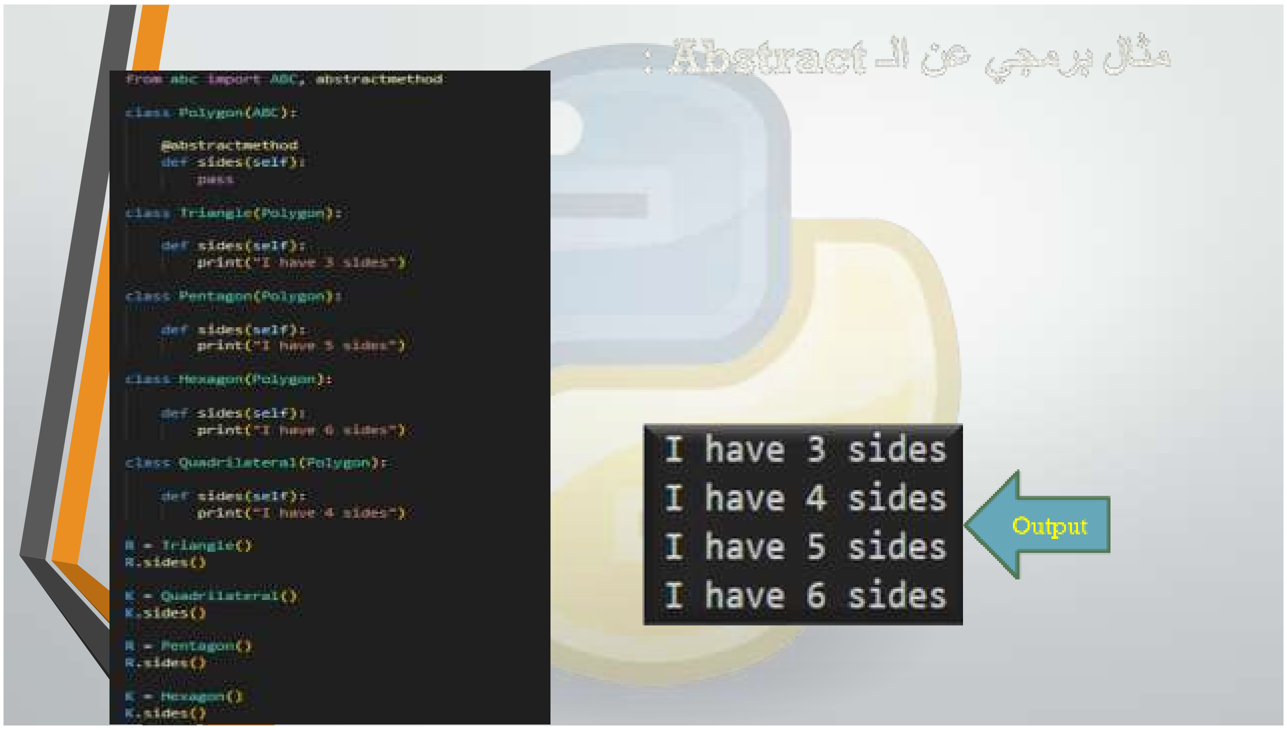


مثال برمجي عن الـ Abstract :

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def move(self):
        pass

class Bird(Animal):

    def move(self):
        print("I can fly")

class Snake(Animal):

    def move(self):
        print("I can crawl")

class Dog(Animal):

    def move(self):
        print("I can bark")

class Lion(Animal):

    def move(self):
        print("I can roar")

R = Bird()
R.move()

K = Snake()
K.move()

R = Dog()
R.move()

K = Lion()
K.move()
```

```
I can fly
I can crawl
I can bark
I can roar
```

Output



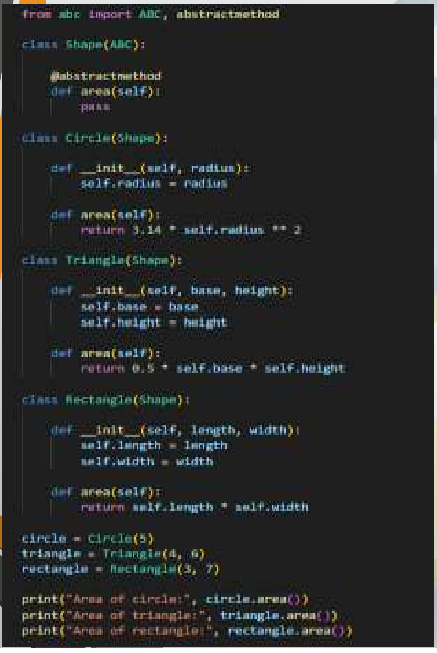
مثال برمجي عن الـ Abstract :

```
from abc import ABC, abstractmethod

class Polygon(ABC):

    @abstractmethod
    def sides(self):
        pass

class Triangle(Polygon):

    def sides(self):
        print("I have 3 sides")

class Pentagon(Polygon):

    def sides(self):
        print("I have 5 sides")

class Hexagon(Polygon):

    def sides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):

    def sides(self):
        print("I have 4 sides")

R = Triangle()
R.sides()

K = Quadrilateral()
K.sides()

R = Pentagon()
R.sides()

K = Hexagon()
K.sides()
```

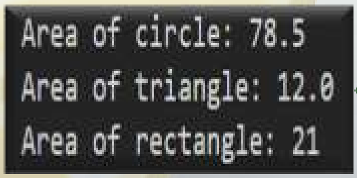
```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

Output

مثال برمجي عن الـ Abstract :



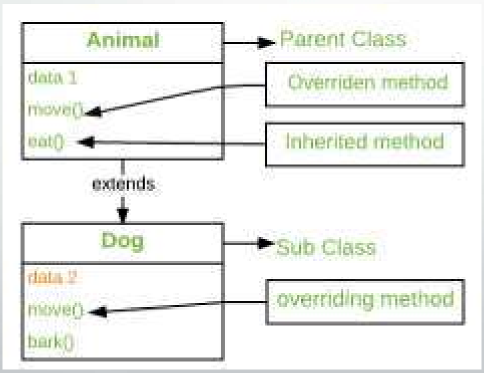
```python
from abc import ABC, abstractmethod

class Shape(ABC):

    @abstractmethod
    def area(self):
        pass

class Circle(Shape):

    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

class Triangle(Shape):

    def __init__(self, base, height):
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height

class Rectangle(Shape):

    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

circle = Circle(5)
triangle = Triangle(4, 6)
rectangle = Rectangle(3, 7)

print("Area of circle:", circle.area())
print("Area of triangle:", triangle.area())
print("Area of rectangle:", rectangle.area())
```



```
Area of circle: 78.5
Area of triangle: 12.0
Area of rectangle: 21
```

Output

Method overriding is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, the same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.



The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

https://www.youtube.com/watch?v=4Y83cUbDKZ8

```
class Parent():

    # Constructor
    def __init__(self):
        self.value = "Inside Parent"
            # Parent's show method
    def show(self):
        print(self.value)
        # Defining child class
class Child(Parent):
        # Constructor
    def __init__(self):
        super().__init__()  # Call parent constructor
        self.value = "Inside Child"

    # Child's show method
    def show(self):
        print(self.value)
        # Driver's code
obj1 = Parent()
obj2 = Child()
obj1.show()  # Should print "Inside Parent"
obj2.show()  # Should print "Inside Child"
```

Output:

```
Inside Parent
Inside Child
```

[redacted]