Lec 0

- Syllabus
- Books
- Java

- 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

- Syllabus
  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

Lec1

- **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.

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

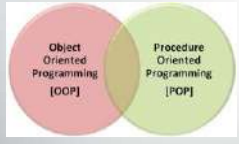
- It can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other.
- It doesn't support the possibility of jumping from one instruction to some other with the help of any statement like 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#

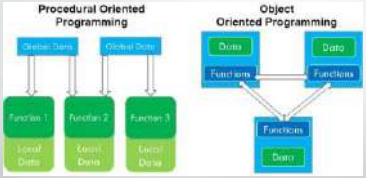
Example structured vs unstructured

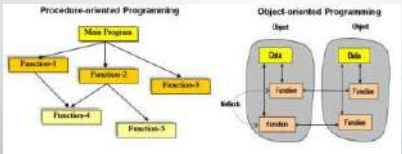
program fragments, the first is structured, while the second

```
Structured:                    Unstructured:
IF x<=y THEN                   IF x>y THEN GOTO 2;
  BEGIN                        z := y-x;
     z := y-x;                 q := SQRT(z);
     q :=SQRT(z);              GOTO 1;
  END                          2: z:= x-y;
ELSE                           q:=-SQRT(z);
  BEGIN                        1: writeln(z,q);
     z := x-y;
     q := -SQRT(z)
  END;
WRITELN(z,q);
```

The main two approaches are:

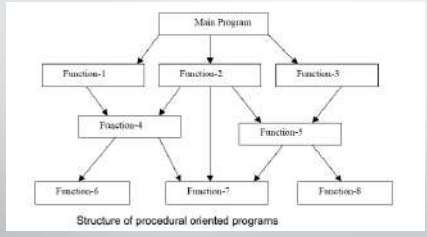


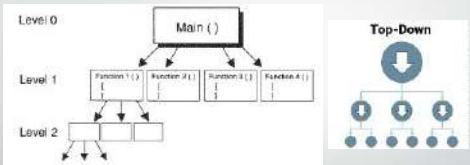




- **Procedural /Functional Programming**

- 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.



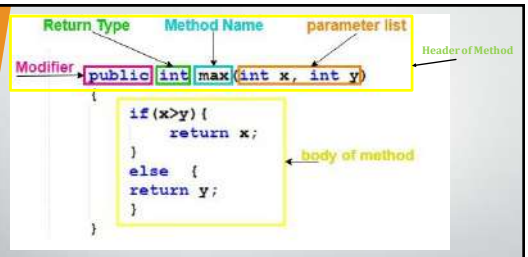


Example

greatest common divisor (Python)

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

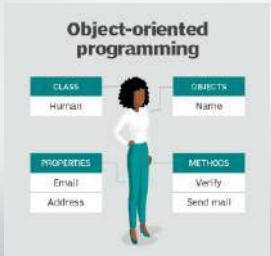
```
gcd(9702, 945)
-> gcd(945, 252)
   -> gcd(252, 189)
      -> gcd(189, 63)
         -> gcd(63, 0)
            -> 63
         -> 63
      -> 63
   -> 63
-> 63
```

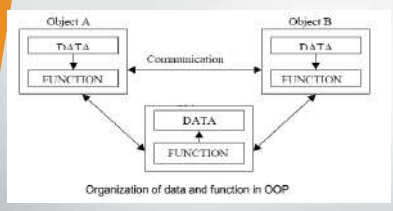


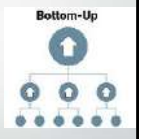
- OOP Object Oriented Programming

- 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).

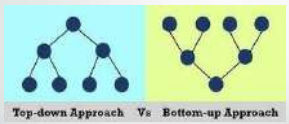
Example







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




Any questions
GOODBYE
THANK YOU

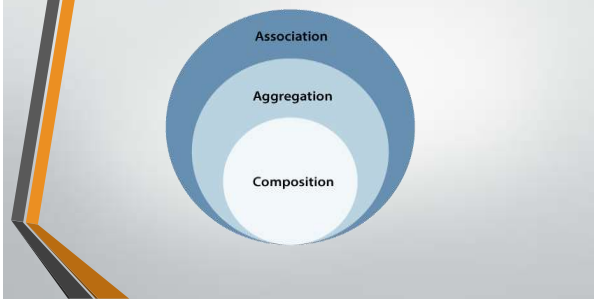
Lec2

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

- **OOP Relationships**

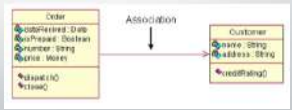
One of the advantages of Object-Oriented programming language is code reuse. This reusability is possible due to the relationship between the classes. Object oriented programming generally support 4 types of relationships that are: inheritance , association, composition and aggregation. All these relationship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



- **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.

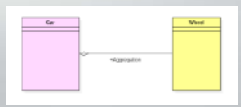


2- Aggregation (التجميع)

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

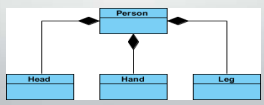
A car needs a wheel to function correctly, but a wheel doesn't always need a car. It can also be used with the bike, bicycle, or any other vehicles but not a particular car. Here, the wheel object is meaningful even without the car object. Such type of relationship is called UML Aggregation relation.



3- Composition (التكوين)

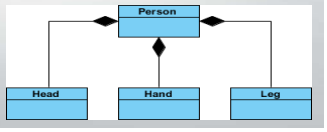
- Composition is a "part-of" relationship. In composition relationship both entities are interdependent of each other for example "heart is part of body. Heart is a part of each body and both are dependent on each other. (ownership).
- Composition implies a relationship where the child cannot exist independent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

Example2

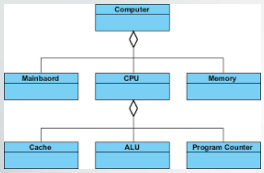


Example3

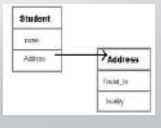
Example4



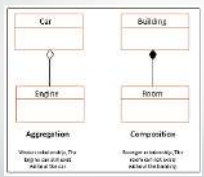
Example5



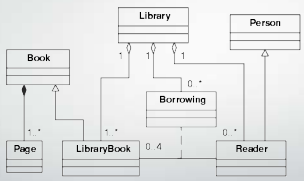
Example6



Example7



Example8

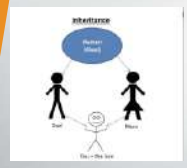


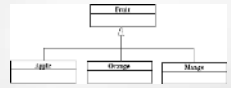
Lec3

- **OOP Relationships cont.**
  - Inheritance (التوارث)
  - Relation types
- **OOP Concepts (مفاهيم) part I**
  - What are the main concepts of OOP
    - Object and Class
    - State and behavior
    - Examples

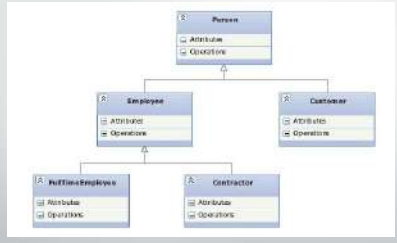
- **Inheritance (التوارث)**

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

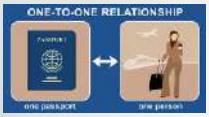




Example1

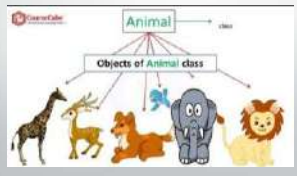


- **Relation Types**



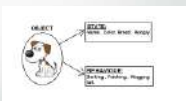
- **Classes and Objects**

- 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.



Syntax

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

Write examples of classes from Real World

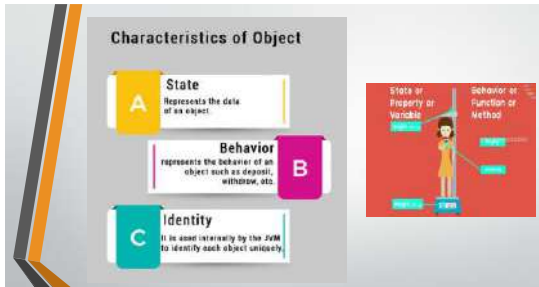
- **What is an Object?**

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



- **What is the Difference Between Object & Class?**

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

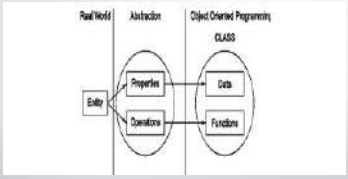
Let's see some real-life examples of class and object in java to understand the difference well:

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

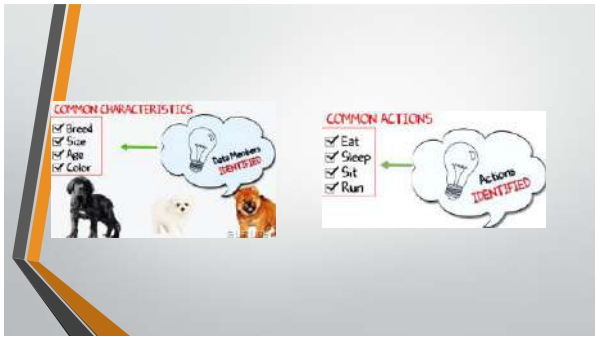
- **The differences between object and class:**

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.

- **Different point of views**



How to convert real life entity to object?

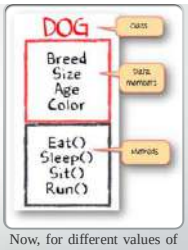


So far, we have defined following things,

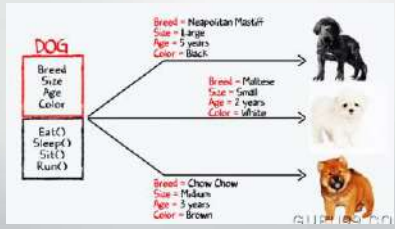
Class - Dogs

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

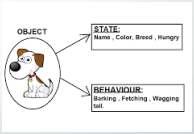
Methods- eat, sleep, sit and run.



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



State and behavior are the basic properties of an Object

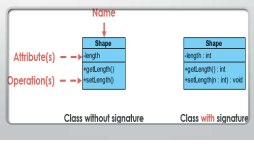


- **State** tells us about the type or the value of that object whereas **behavior** tells us about the operations or things that the object can perform.

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




Thank you!
Do you have any Questions?
BYE BYE

Lec4

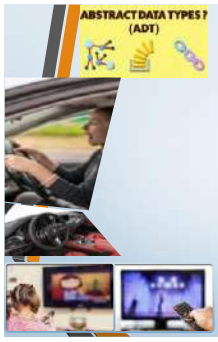
1. Defining a class

- 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
- Inheritance
- Polymorphism

- **Defining a class**
  - How is a class defined?
    - How is a variable member declared?
    - How is a method (function) defined?
  - How is members accessed?
  - How is an object created?
  - what does message pass mean?
  - Applying practical examples using NetBeans (Lab)



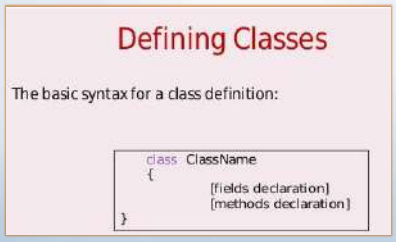
- The class in general is an **A**bstract **D**ata **T**ype (**ADT**).
- **Example1**

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**

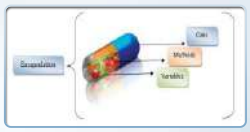
We all know how to turn the TV on, but we don't need to know how it works to enjoy it.

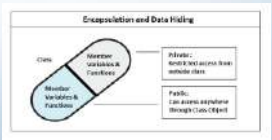
- **Defining a class**



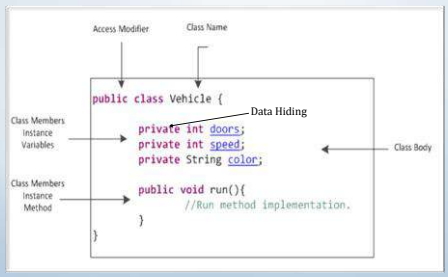
- **Encapsulation**

- 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.



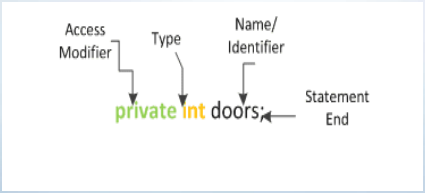


Example 3

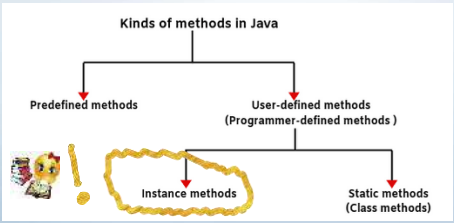


- **Access Modifiers in Java**

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

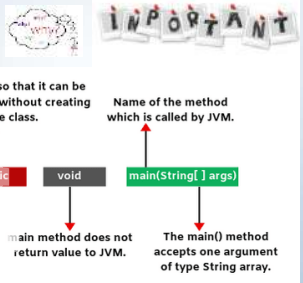


- **Methods Types (kinds)**



- **Special method main**

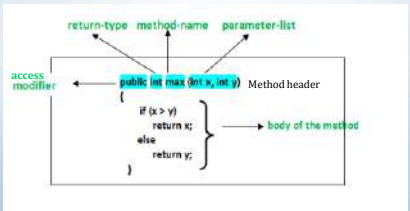
It is a java main method



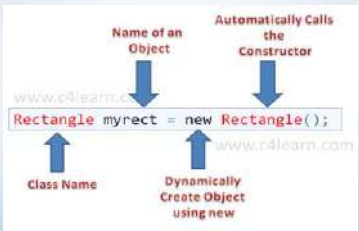
- **Declaration of Methods**

Example 4

•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.



- **Creating an object**



- **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

- **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.

Abstraction .. Encapsulation

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;*


Thank you!
Do you have any Questions?


BYE BYE

Lec 5

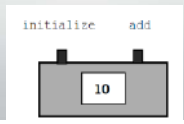
- Examples for defining a class
  - Examples for crating an object
- behavior and state
- defining more than one object
- defining array of objects

Example 1

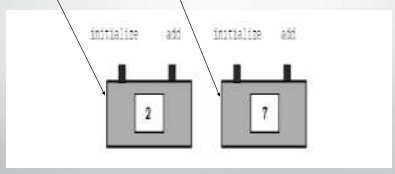
let us consider the example of counters (المعدادات).

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.

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

2

memory

Object	number
c1	02

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());
}
}
```

2
1
3

Object	number
c1	0
c2	0
c3	0

Object	number
c1	02
c2	01
c3	0123

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	0 1 2
c2	0 1
c3	0 1 2 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	0 1 2 3 4
c2	0 1 2

4

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	0 1 2

Object	number
c1	0 1 2
c2	0 1 2

1
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

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;}  }
```

- 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());
 }  }
```

Any question?

Lec 6

- **Polymorphism I (Static Polymorphism)-Method overloading**
  - What is method overloading?
  - What are method overloading types?
  - How is method overloading implemented in Java?

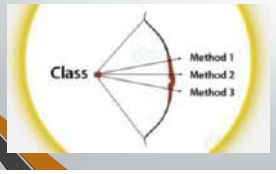
- Examples

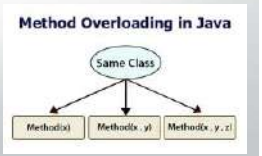
- Applying practical examples using NetBeans

• 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);

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.

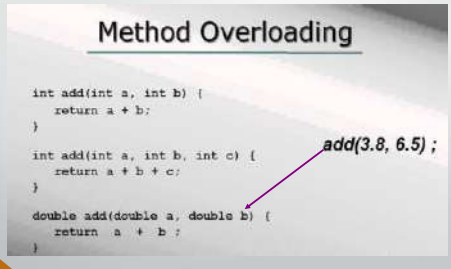
Example 1

```
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



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

```
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.

Lec7

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

• Constructor Definition

- **Constructor** in Java is a block of code like a method that's called when an **instance of an object is created**. 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:

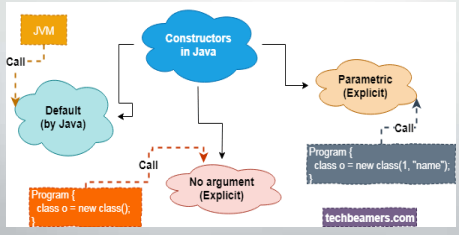
public ClassName (parameter-list) {
Statements
}

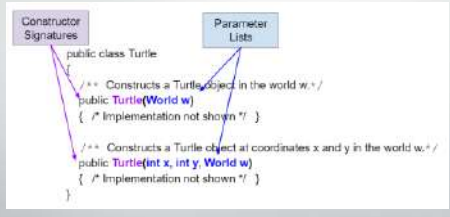
CONSTRUCTOR	METHODS
It is a block of code which instantiate a newly created object.	It is a collection of statements, always return a value.
It does not have any return type.	It may return a value.
It's name should be same as the class name.	It's name should not be same as the class name.

• 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.

• Constructor Types





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.

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();

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();
        ...
    }
}
```

Beginnersbook.com

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();

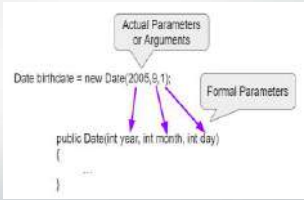
- **H.W. what will be the output when there is no constructor in class Student?**
- **Rule: If there is no constructor in a class, compiler automatically creates a default constructor.**

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.



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();     } }
```

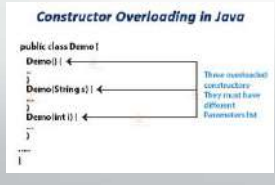
The Output is:
111 true
222 false

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



Example 4 (Trace)

```
1.  public class Student{
2.    private int id;
3.    private   boolean state;
4.    private int age;
5.     public Student(int i,boolean  b){      id = i;      state = b;      }
6.     public  Student(int i,boolean b,int a){      id = i;       state = b;       age=a;      }
7.      public void display(){System.out.println(id+"   "+ state +"  "+age);}    }
8.      public class Main{
9.      public static void main(String args[]){
10.     Student s1 = new Student(111,true);
11.     Student s2 = new Student(222,false,25);
12.     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; }
}
```

Lec8 and Lec9

- **Advanced Examples**
  - Taking advanced examples

- **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
```

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

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	

```
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());}}}
```

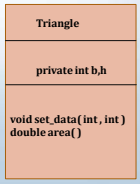
```
The screen
2
8
18
32
50
```

```
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.



1- Solution

Object	b	h	set and area
t	10	20	

```
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());  }}
```

The screen
100.0

The order of project(programs)execution

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

2- Solution

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

```
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 t1=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.     }}
```

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.**

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); }}
```

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

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

```
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());}}
```

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
```

```
1.  public class Number {
2.  private int x,y;
3.  public Number(int n1,int n2){  x=n1; y=n2; }
4.  public  int add(){ return(x+y); }
5.  public  int sub(){return (x-y);}
6.  public int mul(){ return x*y;}
7.  public int div(){ return x/y;}
8.  public void print(){
9.  System.out.println("x="+x+" y="+y);}
10.  }
```

```
1.  public class Main {
2.     public static void main(String[] args) {
3.        Number n;
4.        n=new Number(50,100);
5.        n.print();
6.        System.out.println(n.add());
7.        System.out.println(n.sub());
8.        System.out.println(n.mul());
9.        System.out.println(n.div());
10.  } }
```

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.??????

- **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+"$"); } }
```

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

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

```
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. }}
```

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

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+"$"); }}
```

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

```
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	

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);}}
```

```
123.5
```

```
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	

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

- **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.

```
1.  public class Employee {
2.  private int number;
3.  private int step;
4.  private double salary;
5.  public Employee(int no,int st, double sal) {
6.  number=no;   step=st;
7.  if (sal > 0.0)
8.  salary=sal;
9.  else salary=0.0; }
10. public void setNo(int no){ number=no; }
11. public void setSt(int st){ step=st; }
12. public void setSalary(double sal){
13. if (sal > 0.0)
14. salary = sal;
15. else
16. salary = 0.0; }
17. public int getNo(){  return (number);   }
18. public int getSt(){  return (step); }
19. public double getSalary(){ return salary; }}
```

```
1.  public class EmployeeTest {
2.  public static void main (String args[]){
3.  Employee e1=new Employee (1,3,20000.00);
4.  Employee e2=new Employee (2,5,50000.00);
5.  System.out.println("1) "+e1.getNo()+“ "+e1.getSt()+“ "+e1.getSalary()*12);
6.  System.out.println("2) "+e2.getNo()+e2.getSt()+e2.getSalary()*12);
7.  e1.setSalary(0.1*e1.getSalary()+e1.getSalary());
8.  e2.setSalary(0.1*e2.getSalary()+e2.getSalary());
9.  System.out.println("3) "+e1.getNo()+”  +e1.getSt()+” “+e1.getSalary()*12);
10. System.out.println("4) "+e2.getNo()+” “+ e2.getSt()+” “+e2.getSalary()*12);
11.                                                    }
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.

```
1.  public class Date {
2.  private int month;
3.  private int day;
4.  private int year;
5.  public Date(int m, int d, int y) {
6.  month = m;  day = d; year =y;}
7.  public void setMonthDate(int m) { month = m;}
8.  public int getMonthDate() { return (month);}
9.  public void setDayDate(int d) { day = d;}
10. public int getDayDate() {  return day;}
11. public void setYearDate(int y) { year = y;}
12. public int getYearDate() { return year;}
13. public void displayDate(){  System.out.println(month+"/"+day+"/"+year);}
14. }
```

```
1.  public class DateTest {
2.  public static void main(String[] args) {
3.  Date myDate = new Date(19,6,1977);
4.  myDate.displayDate();
5.  int myMonth = 5;
6.   myDate.setMonthDate(myMonth);
7.  int myDay = 7;
8.   myDate.setDayDate(myDay);
9.  int myYear = 1974;
10.  myDate.setYearDate(myYear);
11. myDate.displayDate();
12. }}
```

The output will be:

6/19/1977
5/7/1974

Example7 (Implementing a complex number operations)

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

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

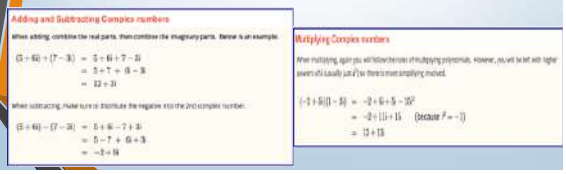
- To add or subtract two **complex numbers**, just add or subtract the corresponding real and **imaginary** parts.
- For instance, the sum of

  5 + 3i

  4 + 2i

  is 9 + 5i.

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




```
1.  public class Complex {
2.  private float real,img;
3.  public Complex(float r,float i){ real=r;  img=i;  }
4.  public Complex(){  }
5.  public float get_real(){ return(real); }
6.  public float get_img(){ return(img); }
7.  public Complex add (Complex c2){
8.  Complex c3=new Complex();
9.   c3.real=c2.real+real;
10.  c3.img=c2.img+img;
11.  return(c3);}
12. public void print(){    System.out.println(real+"+"+img+"i" );
13. }}
```

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

3.       Complex no1=new Complex( 5.0f,4.0f);
4.       Complex no2=new Complex(8.0f,9.0f);
5.       Complex no3=new Complex();

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

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

11.      no4.print();
12.      no5.print();
13.      no6.print();
14.  }}
```

1. The results add operation

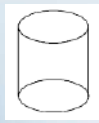
5.0+4.0i
8.0+9.0i
13.0+13.0i

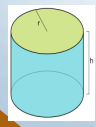
2. The results without add operation

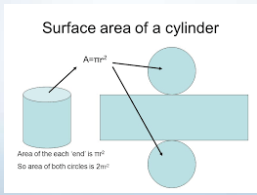
1.0+2.0i
3.0+4.0i
4.0+6.0i

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

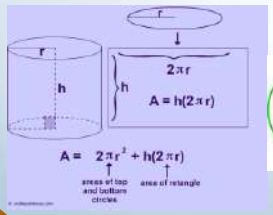
Does circle is a part of cylinder?

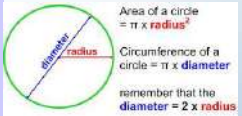






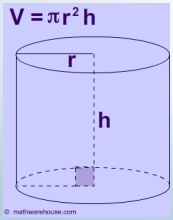
How to find out the surface area of a cylinder?

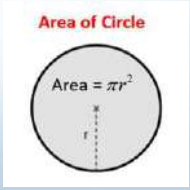




https://www.youtube.com/watch?v=9ez0Vw9Ddws

How to find out the volume of a cylinder?





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.

Lec10

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

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

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.

H.W.

Write either True or False

1. `5>x>=0`
   `int x= (int)(Math.random()*(5-0))+0;`
2. `5=>x>=0`
   `int x= (int)(Math.random()*(5-0+1))+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;`

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(اقرء):

```
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(اقرء):

```
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);}
```

- **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; }
 }
```

```
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. ");
} }
```

Lec11

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

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!

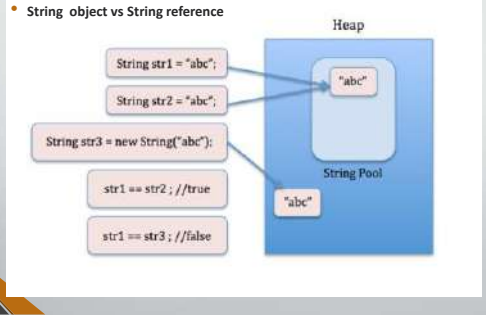
2. Constructor with Literal the string could be created using the new keyword and a constructor.

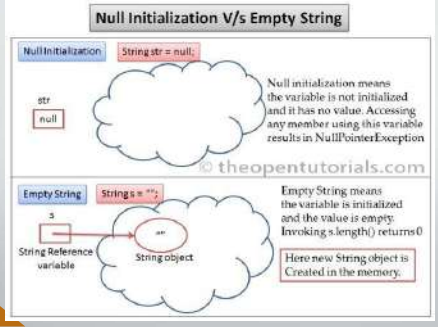
```
String s=new String("Welcome");  System.out.println(s);
String ss=new String("Raheeq"); System.out.println(ss);
```

3. Constructor with array of char: As with any other object, you can create String objects by using the new keyword 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





Example 1:
Trace the following java code:

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

The output

```
java
strings
example
```

Example 2:
Trace the following java code:

The output

ERROR

```
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);    } }
```

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:

```
1. public class Example3{
2. public static void main(String args[]{
3. String s1="hello";
4. String s2="whatsup";
5. System.out.println("String length is: "+s1.length());
6. 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:

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

The output

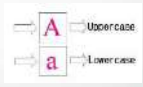
```
hellohow are you
```

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

```
String s1="ssss";
String s2="ddddd";
String s3=s1+s2;                    s3  sssssddddd

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

String s1="sss";
String s2="bbbb";
String s3="dddd";
String s5=s1+s2+s3;                          s5  sssbbbbddddd


String s6=s1.concat(s2.concat(s3));          s6  sssbbbbddddd
```

- **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.



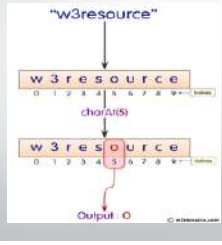
Example5

```
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

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);

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

The output
A

The output
I
R
A
Q

Example7: (Trace)

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

ERROR …Why?

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".

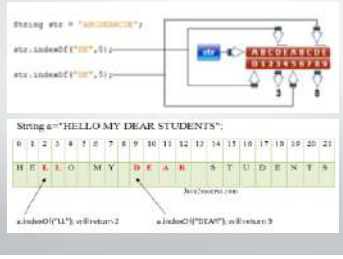
Example8
Trace the following java code:

```
1. package javaapplication34;
2. class Example8 {

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

The output:
0
5

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".



Example9

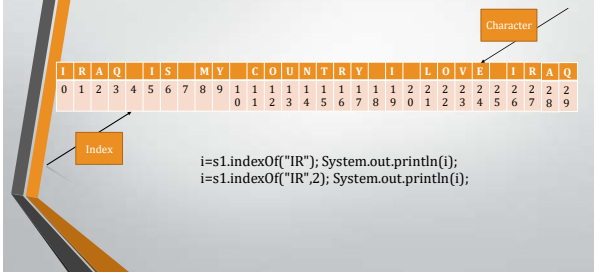
Trace the following java code:

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

The output:

0
26

Explanation (توضيح) Example9



Thank you for listening

Any questions?

Lec12

- **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:**

Substring Method

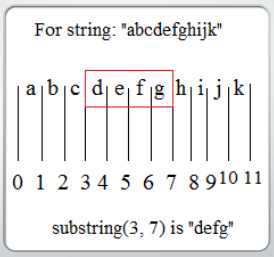
Substring is a subset of another string. Note: Index starts from 0.
You can get substring from the given string object by one of the two methods:

•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

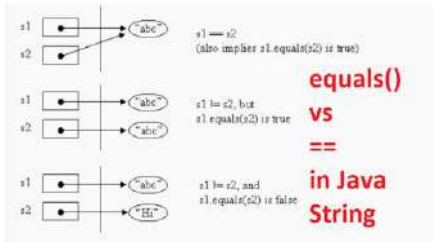
The **String equalsIgnoreCase()** method compares the two given strings on the basis of content of the string irrespective of case of the string. It is like equals() method but doesn't check case. If any character is not matched, it returns false otherwise it returns true.

- **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
```



- **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
```

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
```

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

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

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?

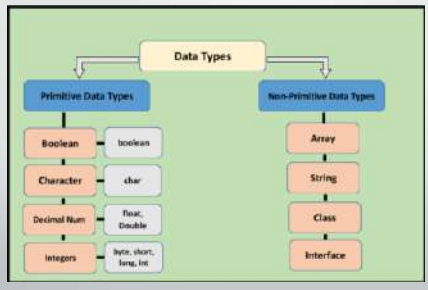
```
str1 & str2 comparison: -16
str1 & str3 comparison: 0
str2 & string argument comparison: 0
```

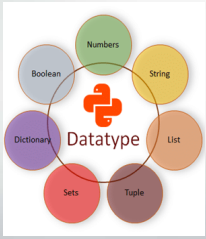
Thank you for listening

Any questions?

Lec13

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





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
```


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 ✖

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
```

3- Converting from any data type to String

3. The toString method

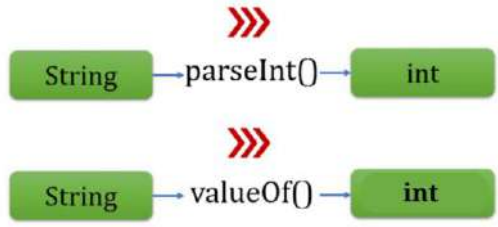
It could be converted any data type to string using toSring method, for example to convert integer (the same to double, long, float …etc.) to string is shown below:

Exampl 4:

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

The output is:

```
42
42
```



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 Lec11-13

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)

14

Goodbye

Thank you for listening

Any questions?

15

Lec14 Inheritance in OOP – Java

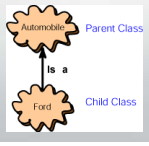
- Examples
- Practical Part (trace)
- Writing programs
- Tracing programs

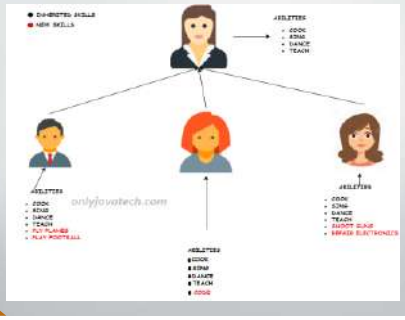
Inheritance def.

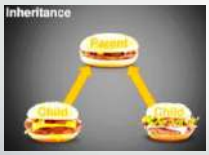
Inheritance is a mechanism in which one class acquires (يكتسب) the property of another class. For example, a child inherits the traits (سمات) of his/her parents.

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.







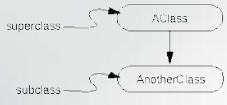
Important terminology:

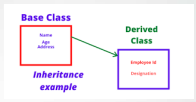
- **Super Class:** The class whose features are inherited is known as super class(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in 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.

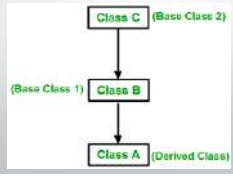
Extends keyword

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

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

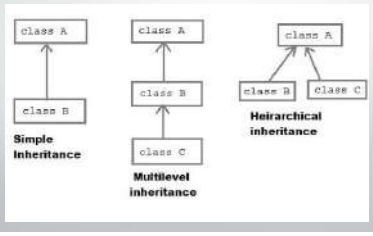






Example1

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



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

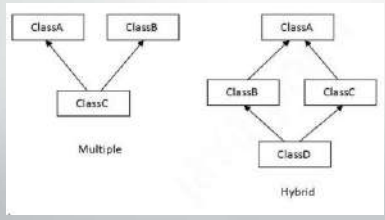
```
public class A{
.
.
.
.}
```

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

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

Example2

- Define a java structure for the following two figures



13

Multiple Inheritance

Error
Not allowed in java

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

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

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

14

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

15

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";}
```

obj

designation	collegeName	mainSubject	does()
Teacher	Beginners book	Physics	

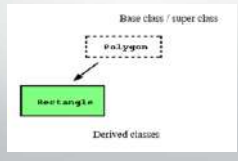
```
public class Main{
  public static void main(String args[]){
    PhysicsTeacher obj = new PhysicsTeacher();
    System.out.println(obj.collegeName);
    System.out.println(obj.designation);
    System.out.println(obj.mainSubject);
     obj.does(); }}
```

Output:
Beginners book
Teacher
Physics
Teaching

16

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.



17

```
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

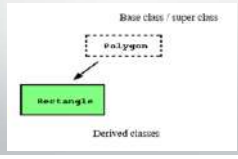
Screen

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.



19

```
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

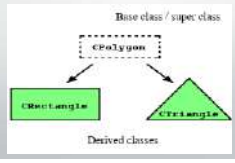
Object	width	height	area	set
rect1	4	5		
rect2	6	7		
rect3	8	9		

memory

20

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.



21

```
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

Screen

Object	width	height	set area
rect	4	5	
trg	5	6	

memory

22

- **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, 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.

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.

23

Example4 (Tracing a program)

```
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();    }}
```

The output :
5 6
10 20
100 200

Scree

24

Example5 (Tracing a program)

```
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();     }}
```

The output :
5 6
10 20
15 50

Scree

25

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

26

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?

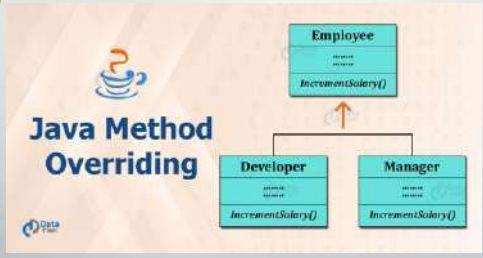
Scree

27

28

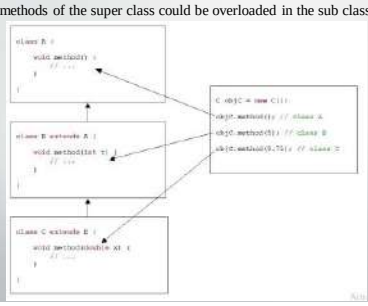
Lec15 Inheritance in OOP – Java Cont. (overloading)

- 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…



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

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

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

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?

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
```

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.


```
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
```


Lec16Constructors in base and sub-classes

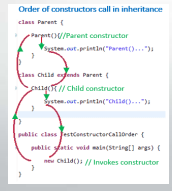
- **Constructor definition in subclass**
- **Calling a base class constructor in subclass**
- **Examples (tracing and writing programs)**

Constructors in Subclasses

- **Constructors are not inherited**. 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!

2

- 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**.



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

4

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

5

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

6

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? ⊗

7

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?

8

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,

9

```
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);   }}
```

10

```
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);
   }}
```

11

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

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

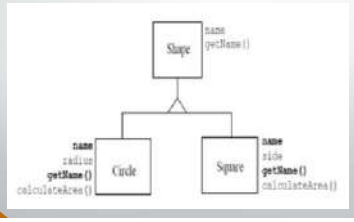
The output will be:

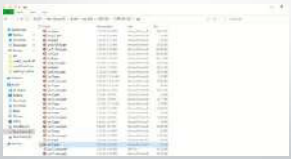
Esteemed Yasser Mohammed,
FIRE near School HIGH

12

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.





Lec17 "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)

Java Programming

Understanding the Java Class

Superclass Constructors

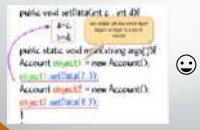
'this' keyword ←

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

Usage of java this keyword

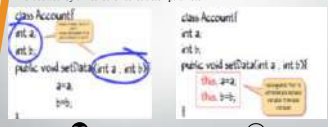
Using this with a Field

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



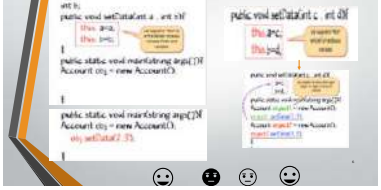
Using this with a Field

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



QUIZ

Choose the suitable face for each case



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;     }}
```

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).

- **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 Maip {
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);
```


- 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.

- **The Special Variable super**

- **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 **superx** 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 (ابعد) 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.

- **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
```


Lec18 final keyword in Java

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

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

- **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; }

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 = 3.14159265358979323;

 //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;    }
}...}
```

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 = 20;  // Error because i is final.
      PI=2;  // Error because i is final.    } }
```

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() {...}}
```

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
```

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 (الحماية والمنفعة) 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.**

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.

- Give a programming example with its execution that describes g the following :

- 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.

10

Lec19 Packages

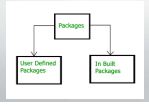
Package def. and types

- Creating a package
- Importing a package
- Access to classes within the same package
- Examples

11

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.



12

13

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.

14

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");
  }
}
```

15

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();  }}
```

16

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();   } }
```

17

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();   } }
```

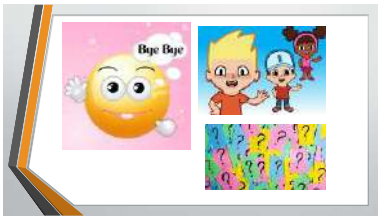
18

- **Sub - package**

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

- Access Modifiers

Keyword	Visibility
private	Access to a private variable or method is only allowed within the code fragments of a class.
protected	Access to a protected variable or method is only allowed within the code fragments of a class and its subclasses
(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



Lec20 Override method

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

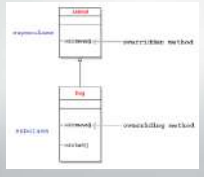
Overriding Method (المهيمنة)(dynamic polymorphism)

- 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.

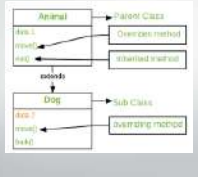
Rules for Method Overriding

- 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.

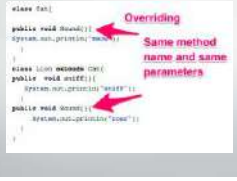
Graphical example 1



Graphical example 2



Example 3



Example 4 (Explanation

The differences between method overloading and overriding

Method Overloading	Method Overriding
1. Defining multiple 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 (ahronosphism)/Early binding/Static binding	4. Run time polymorphism/Late binding/Dynamic binding
5. Must or may not need Inheritance.	5. Must need Inheritance.

Are Same

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 ☺

- 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 public, protected or the default 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(this point will be explained later).

- 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

- 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?

- Example 9 (Tracing) Test2

```
public class that {
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?

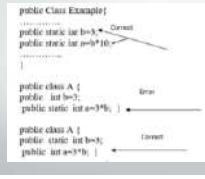
Lec21 static keyword in Java

- **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.
- It is used with static -variables nly.

Example1 (Explanation)



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.

Example 2 (Explanation

```
public class C {
private int n=0;
public static int s=0;
public int gets() { return (n); //OK }
public int gets() { return s; //OK }
public static int get2s() { return n; //ERROR!! } }
```

WHY?!!!

Example 3 (Explanation)

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= 1;
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	

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 }
}
```

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)

The output:
zzz

```
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();  } }
```

- QUIZ

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

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;
   public static void f1(){ System.out.println("static  f1 from class One");}
   public void f1(){  System.out.println("f1 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 f1(){System.out.println("method overriding from class Two");
    super.f1();   } }

public class Main {
public static void main(String[] args) {
Two t=new Two();
t.f1();   t.f0();
One.f1();   Two.f1();   } }
```

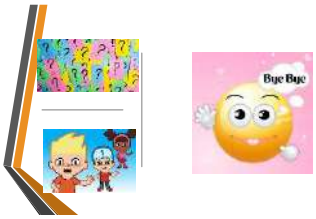
Does the output correct or not and why?

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

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[1].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



Lec22 Nested classes

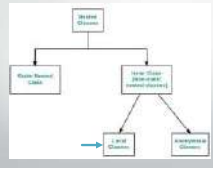
- **Outer class**
- **Inner class**
- **Static class**
- **Examples (tracing programs)**

- **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 { ... }
}

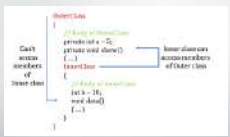
- **Very important note (Terminology):**
Nested classes are divided into two categories



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.

- Access modifiers



- **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

```
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

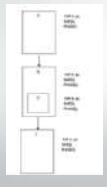
```
public class Outer {
    private int x;
    inner in=new inner();
    public void set()  {
x=100;
y=200;
    in.set();   }
public void print(){
    System.out.println("x="+x);
System.out.println("y="+in.get()); }
    public  class inner   {
        private int y;
        public int get(){return (y);}
        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();
Top.Nested1.Nested2 nested2 = new Top.Nested1.Nested2(t, nested1);
 System.out.println(t);
System.out.println(nested1);
}
```

Lec23 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).*

* 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.

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):

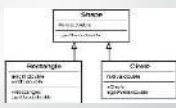
```
public class Animal{
  public void move(){ System.out.println("Animals can move");  } }

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
```

Case I

Case II



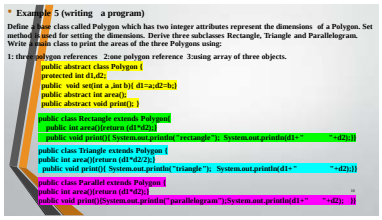
- **Abstract Class**

* If a class is abstract and cannot be instantiated, the class does not have much use unless it has subclasses.

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);
```

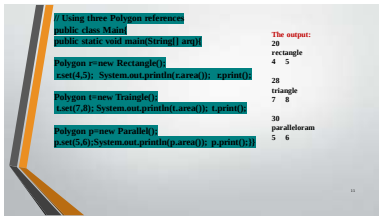


- **Example 5 (writing a program)**

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:

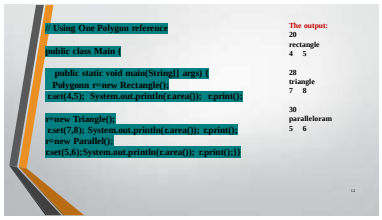
1: three polygon references 2:one polygon reference 3:using array of three objects.

```
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); }}
```



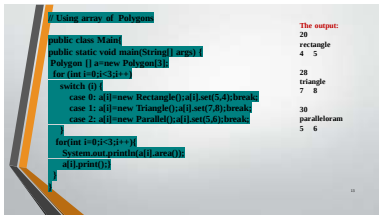
```
// Using three Polygon references
public class Main{
public static void main(String[] arg){

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
```

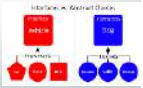


```
// Using array of Polygons

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
```

Lec24 Interface

- **Multiple inheritance**
- **Interface declaration**
- **Default method and tag interface**
- **Examples (tracing and writing programs)**



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:

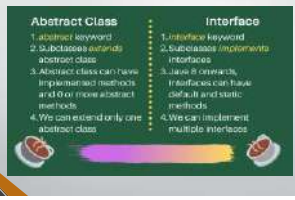
A class describes the attributes and behaviors of an object while, an interface contains behaviors that a class implements. **Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.**

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

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.



Declaring Interfaces

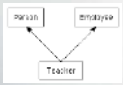
The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface. Let us look at an example that depicts encapsulation:
Interfaces have the following properties:
•An interface is implicitly (ضمنيا) abstract. You do not need to use the **abstract** keyword when 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.

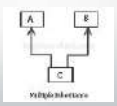
Implementing interface

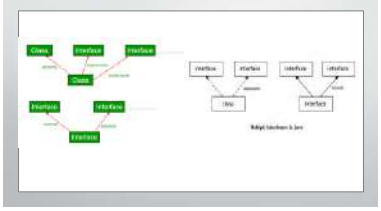
- **A class uses the implements keyword to implement an interface.**
- **A class can implement more than one interface at a time.**
- **A class can extend only one class but implement many interfaces.**
- **An interface itself can extend another interface**

- **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.**

Multiple Inheritance







- 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 8;  }}

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.

- 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.**

- 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

- 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 Vehicle();
        System.out.println("Vehicle");
        obj.distance();
        obj.speed();        }}
```

The output :
Vehicle
distance travelled is 9000

- 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
```

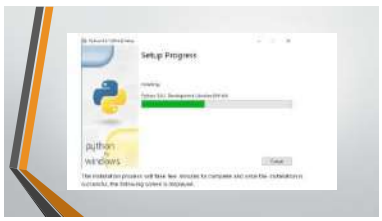
H.W.
Defines Java structure for the following Hierarchy diagram

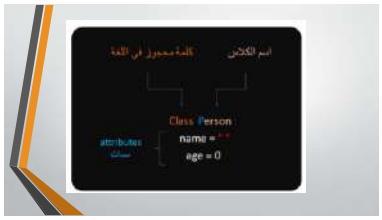
- Quiz

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

Lec26-30 Python

- **Examples**




اسم الكلاس
كلمة محجوز في اللغة
Class Person
attributes
name = ""
age = 0


class Car:
def __init__(self, make, model, year):
self.make = make
self.model = model
self.year = year
my_car = Car("Toyota", "Corolla", 2021)
print(my_car.make) # يطبع Toyota


نفترض player هو اسم أن كلاس، ثم يمكننا انشاء كائنات من أن كلاس مثل:Player2 ،Player1
هنا انشاء الكائن عن طريق اسم أن كلاس:
Player1 = Player ()
Player2 = Player ()
مثال توضيحي
Class Player :
Energy = 100
Player1 = Player ()
Player2 = Player ()
Print (Player1.energy)
Print (Player2.energy)
Out put:
100
100


Method overloading in python

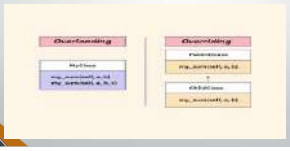

method python overloading
طريقة التحميل الزائد في بايثون
طريقتان أو أكثر لهما نفس الاسم ولكن تختلف فيها أما نوع المتغيرات أو عدد و نوع المتغيرات.
*في البايثون لا يدعم طريقة التحميل الزائد فلو اضفنا الكثير من الطرق فان بايثون سوف يأخذ الطريقة الأخيرة الا في طرق خاصة.
The problem with method overloading in Python is that we may overload the methods but can only use the latest defined method.
المشكلة في طريقة التحميل الزائد في بايثون أننا قد نفرط في استعمال التحميل لكنه لا يأخذ الا اخر طريقة .


فوائد استخدام طريقة التحميل الزائد في بايثون
على نقة البرمجة بايثون توفر طريقة التحميل الزائد العديد من الفوائد، منها:
1. إعادة استخدام الكود: يمكن استخدام الطريقة لتعريف دالة واحدة يمكن استخدامها لأنواع مختلفة من البيانات أو الحجج، مما يوفر الوقت والجهد في كتابة الكود.
2. تحسين قابلية القراءة والصيانة: يمكن استخدام الطريقة لتعريف دوال مختلفة تتبع نفس الاسم المنطقي، مما يجعل من السهل فهم الغرض من كل دالة واستخدامها بشكل صحيح.
3. تقليل التعقيد: يمكن استخدام الطريقة لتعريف دوال مختلفة تتعامل مع أنواع مختلفة من البيانات أو الحجج، مما يقلل من الحاجة إلى كتابة شروط معقدة للتحقق من نوع البيانات.
4. تحسين الأداء: يمكن استخدام الطريقة لتحسين أداء البرنامج عن طريق تحديد الدالة الأكثر كفاءة للاستخدام بناءً على نوع البيانات أو الحجج.
5. توفير المرونة: يمكن استخدام الطريقة لتعريف دوال مختلفة تتعامل مع أنواع مختلفة من البيانات أو الحجج، مما يتيح للمبرمجين تخصيص سلوك الدالة بناءً على احتياجاتهم المحددة.
ومع ذلك، يجب أن يتم استخدام طريقة التحميل الزائد بحذر، حيث يجب على المبرمج التأكد من أن الدوال المختلفة المعرفة بنفس الاسم تؤدي إلى نفس الغرض وتتوافق مع بعضها البعض، وأنه يتم استخدام الدالة المناسبة في السياق المناسب.


مساوئ استخدام طريقة التحميل الزائد في بايثون
على الرغم من الفوائد التي توفرها طريقة التحميل الزائد في لغة البرمجة بايثون، إلا أنها قد تنطوي على بعض المساوئ. ومن بين هذه المساوئ:
1. صعوبة في فهم الكود: يمكن أن يؤدي استخدام طريقة التحميل الزائد في تعريف دوال بنفس الاسم إلى صعوبة في فهم الكود، خاصة إذا لم يتم التعامل معه بشكل صحيح.
2. احتمال في الأخطاء: قد يؤدي استخدام طريقة التحميل الزائد إلى تشفير الأداء في بعض الأحيان، خاصة إذا كان هناك العديد من الدوال المعرفة بنفس الاسم.
3. صعوبة في تحديد الدالة الصحيحة: في بعض الأحيان، يمكن أن يكون من الصعب تحديد الدالة الصحيحة التي يجب استخدامها، خاصة إذا كانت جميع الدوال المعرفة بنفس الاسم تؤدي نفس الوظيفة.
4. صعوبة في تتبع الأخطاء: قد حدوث خطأ في الكود المستخدم لطريقة التحميل الزائد، يمكن أن يكون من الصعب تحديد الدالة التي تسببت في الخطأ، لأنه يمكن أن يؤدي ذلك إلى صعوبة في تتبع وإصلاح الأخطاء.
لتجنب هذه المشاكل، يجب على المبرمج استخدام طريقة التحميل الزائد بحذر، وتأكد من تحديد الدالة الصحيحة في كل مرة يتم فيها استدعاء الدالة، وتأكد من أن الدوال المختلفة المعرفة بنفس الاسم تؤدي إلى نفس الغرض وتتوافق مع بعضها البعض.

مثال اول
البرنامج خاطئ

الفرق بين overriding ,overloading



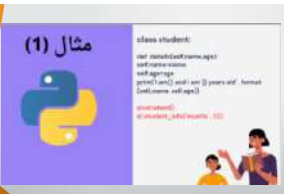
مخطط يوضح التحميل الزائد للدوال في بايثون

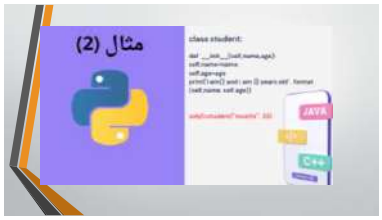
مثال اول
البرنامج صحيح

مثال الثاني

البنائة

هي نوع من انواع الدوال تستدعى و تنفذ لحظة تكوين الكائن و يمكنها استقبال متغيرات او لا تستقبل اي متغيرات ويمكن لجسم الدالة تنفيذ اي عملية يقوم المبرمج بكتابتها [داخلها] او تكون فارغة لا تؤثر.

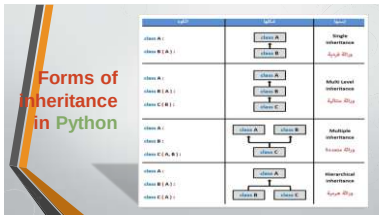




Inheritance in python

- 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().



The general structure of a subclass definition in Python :

Class key word Sub class Super class

```
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
```

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
```

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
الوراثة المتعددة في بايثون

Inheritance (الوراثة) ميزة قوية في البرمجة الشيئية OOP

يشير الى تعريف class جديد مع تعديل بسيط او بدون تعديل على class الموجود سابقاً.

1. ال class الرئيسية هي الفئة الموروثة , وتسمى ايضا الفئة الاساسية او parent.
2. ال class الجديدة فئة مشتقة او child وهي الفئة التي ترث من فئة اخرى.

في بايثون يوجد 4 اشكال لتوراثة وهي:

1. وراثة فردية **Single inheritance** :- هي أن كلاس يرث من كلاس واحد فقط.
2. وراثة متتابعة **Multi level inheritance** :- و تعني بأن كلاس يرث من كلاس واحد و هذا الكلاس كان في الأصل يرث من كلاس آخر.
3. وراثة متعددة **Multiple inheritance** :- تعني أن الكلاس يرث من أكثر من كلاس.
4. وراثة هرمية **Hierarchical inheritance** :- تعني أن الكلاس موروث من قبل أكثر من كلاس.

Multiple inheritance in python
الوراثة المتعددة في بايثون

الوراثة المتعددة" او "الإرث المتعدد" هي مفهوم في البرمجة الشيئية يستخدم في لغة بايثون وغيرها من لغات البرمجة الشائعة مثل

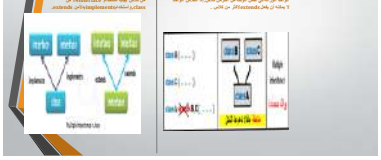
java , c#, c++ ... php و غيرها

الكلاس يمكنه ان يرث من كلاس آخر حتى يحصل على الدوال و المتغيرات الموجودة فيه. في حال كان الكلاس يرث من أكثر من كلاس، يجب وضع فاصلة بين كل كلاسين تضعهما بين القوسين.

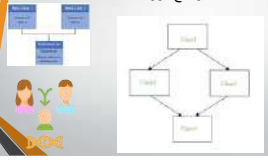
ويتم ذلك برمجياً في python

```
Class A:                    (نقوم بتعريف كلاس باسمه A)
    Body of the class
Class B:                    (نقوم بتعريف كلاس باسمه B )
    Body of the class
Class C(A,B):               ( نقوم بتعريف كلاس باسمه يرث من الكلاس A و الكلاس B )
    Body of the class
```

اما في لغة java



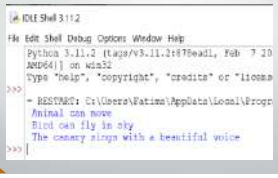
مخطط يوضح الوراثة المتعددة



مثال الاول(البرنامج)

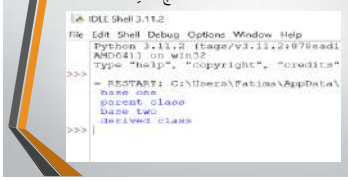
```
class Animal:
    def print_a(self):
        print(" Animal can write ")

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 ")

cl = Canary()

cl.print_a()
cl.print_b()
cl.print_c()
```

ناتج التنفيذ



مثال الثاني(البرنامج)

ناتج التنفيذ



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.

عندما نعرف ان الـ Abstract class هو كلاس يحتوي على Abstract methods وايضا ممكن لا يحتوي. وهناك ايضا Abstract class الذي يحتوي على Abstract methods فقط والذي ندعى بـ 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.

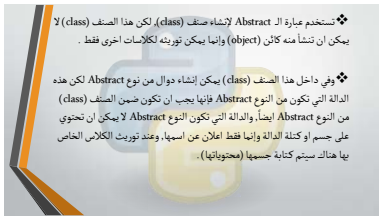
ملحوظة: في بايثون لا نحتاج لكلمة معينة حتى تكون واجهة نحن نستطيع تكوين واجهة باستخدام الـ abstract class الذي يحتوي على abstract methods فقط.

الـ interface ممكن ان يحتوي على:

- Constructor ←
- Variables ←
- Abstract Methods ←
- Sub Class ←

بعض الفروقات بين الـinterfaces 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.

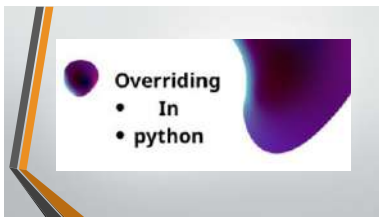

❖ تستخدم عبارة الـ Abstract لإنشاء صنف (class) لكن هذا الصنف (class) لا يمكن ان تنشأ منه كائن (object) وإنما يمكن توريثه لكلاسات اخرى فقط .
❖ وفي داخل هذا الصنف (class) يمكن إنشاء دوال من نوع Abstract لكن هذه الدالة التي تكون من النوع Abstract فإنها يجب ان تكون ضمن الصنف (class) من النوع Abstract ايضاً, والدالة التي تكون النوع Abstract لا يمكن ان تحتوي على جسم او كتلة الدالة وإنما فقط اعلان عن اسمها, وعند توريث الكلاس الخاص بها هنالك سيتم كتابة جسمها (محتوياتها) .


❖ ولاحظ ايضاً الدالة التي تكون من النوع Abstract يجب ان تذكر وتكتب في الصنف (class) الذي سيرثها واذا كانت اكثر من دالة يمكن كتابتهم جميعاً .
Class Shape
abstract Area();
Class Rectangle
area=?*?
Class Triangle
area=0.5*?*?
Class Circle
area=3.14*?*?


I can fly
I can crawl
I can bark
I can roar


I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides


Area of circle: 78.5
Area of triangle: 12.0
Area of rectangle: 21


Overriding
In
python


Override


شروط الـ
Overriding

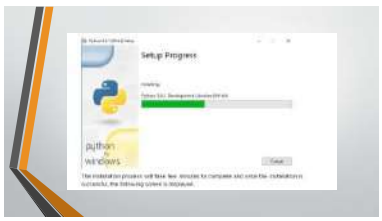

PrintInfo.py هنا نحن نستطيع الوراثة عله الموجود في الملف
بتضمين
الكلاس PrintInfo
from CountryInfo import CountryInfo
class Australia(CountryInfo):
pass
class Lebanon(CountryInfo):
def print_language(self):
print('Arabic')
class Spain(CountryInfo):
def print_language(self):
print('Spanish')

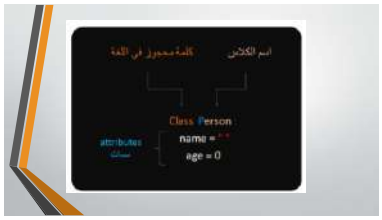

English
Arabic
Spanish
from Australia import Australia
from Lebanon import Lebanon
from Spain import Spain
au = Australia()
lb = Lebanon()
sp = Spain()
au.print_language()
lb.print_language()
sp.print_language()


THE
END

Lec26-30 Python

- **Examples**




اسم الكلاس
كلمة محجوز في اللغة
Class Person
attributes
name = ""
age = 0


class Car:
def __init__(self, make, model, year):
self.make = make
self.model = model
self.year = year
my_car = Car("Toyota", "Corolla", 2021)
print(my_car.make) # يطبع Toyota


نفترض player هو اسم أن كلاس، ثم يمكننا انشاء كائنات من أن كلاس مثل:Player2 ،Player1
هنا انشاء الكائن عن طريق اسم أن كلاس:
Player1 = Player ()
Player2 = Player ()
مثال توضيحي
Class Player :
Energy = 100
Player1 = Player ()
Player2 = Player ()
Print (Player1.energy)
Print (Player2.energy)
Out put:
100
100


Method overloading in python

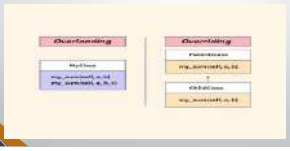

method python overloading
طريقة التحميل الزائد في بايثون
طريقتان أو أكثر لهما نفس الاسم ولكن تختلف فيها أما نوع المتغيرات أو عدد و نوع المتغيرات.
*في البايثون لا يدعم طريقة التحميل الزائد فلو اضفنا الكثير من الطرق فان بايثون سوف يأخذ الطريقة الأخيرة الا في طرق خاصة.
The problem with method overloading in Python is that we may overload the methods but can only use the latest defined method.
المشكلة في طريقة التحميل الزائد في بايثون أننا قد نفرط في استعمال التحميل لكنه لا يأخذ الا اخر طريقة .


فوائد استخدام طريقة التحميل الزائد في بايثون
على نقة البرمجة بايثون توفر طريقة التحميل الزائد العديد من الفوائد، منها:
1. إعادة استخدام الكود: يمكن استخدام الطريقة لتعريف دالة واحدة يمكن استخدامها لأنواع مختلفة من البيانات أو الحجج، مما يوفر الوقت والجهد في كتابة الكود.
2. تحسين قابلية القراءة والصيانة: يمكن استخدام الطريقة لتعريف دوال مختلفة تتبع نفس الاسم المنطقي، مما يجعل من السهل فهم الغرض من كل دالة واستخدامها بشكل صحيح.
3. تقليل التعقيد: يمكن استخدام الطريقة لتعريف دوال مختلفة تتعامل مع أنواع مختلفة من البيانات أو الحجج، مما يقلل من الحاجة إلى كتابة شروط معقدة للتحقق من نوع البيانات.
4. تحسين الأداء: يمكن استخدام الطريقة لتحسين أداء البرنامج عن طريق تحديد الدالة الأكثر كفاءة للاستخدام بناءً على نوع البيانات أو الحجج.
5. توفير المرونة: يمكن استخدام الطريقة لتعريف دوال مختلفة تتعامل مع أنواع مختلفة من البيانات أو الحجج، مما يتيح للمبرمجين تخصيص سلوك الدالة بناءً على احتياجاتهم المحددة.
ومع ذلك، يجب أن يتم استخدام طريقة التحميل الزائد بحذر، حيث يجب على المبرمج التأكد من أن الدوال المختلفة المعرفة بنفس الاسم تؤدي إلى نفس الغرض وتتوافق مع بعضها البعض، وأنه يتم استخدام الدالة المناسبة في السياق المناسب.


مساوئ استخدام طريقة التحميل الزائد في بايثون
على الرغم من الفوائد التي توفرها طريقة التحميل الزائد في لغة البرمجة بايثون، إلا أنها قد تنطوي على بعض المساوئ. ومن بين هذه المساوئ:
1. صعوبة في فهم الكود: يمكن أن يؤدي استخدام طريقة التحميل الزائد في تعريف دوال بنفس الاسم إلى صعوبة في فهم الكود، خاصة إذا لم يتم التعامل معه بشكل صحيح.
2. احتمال في الأخطاء: قد يؤدي استخدام طريقة التحميل الزائد إلى تشفير الأداء في بعض الأحيان، خاصة إذا كان هناك العديد من الدوال المعرفة بنفس الاسم.
3. صعوبة في تحديد الدالة الصحيحة: في بعض الأحيان، يمكن أن يكون من الصعب تحديد الدالة الصحيحة التي يجب استخدامها، خاصة إذا كانت جميع الدوال المعرفة بنفس الاسم تؤدي نفس الوظيفة.
4. صعوبة في تتبع الأخطاء: قد حدوث خطأ في الكود المستخدم لطريقة التحميل الزائد، يمكن أن يكون من الصعب تحديد الدالة التي تسببت في الخطأ، لأنه يمكن أن يؤدي ذلك إلى صعوبة في تتبع وإصلاح الأخطاء.
لتجنب هذه المشاكل، يجب على المبرمج استخدام طريقة التحميل الزائد بحذر، وتأكد من تحديد الدالة الصحيحة في كل مرة يتم فيها استدعاء الدالة، وتأكد من أن الدوال المختلفة المعرفة بنفس الاسم تؤدي إلى نفس الغرض وتتوافق مع بعضها البعض.

مثال اول
البرنامج خاطئ

الفرق بين overriding ,overloading



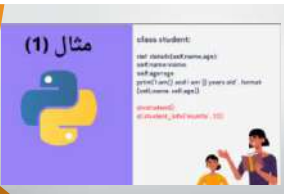
مخطط يوضح التحميل الزائد للدوال في بايثون

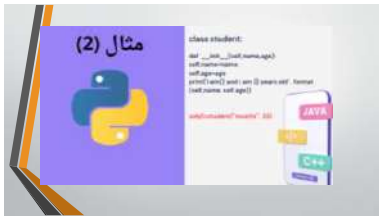
مثال اول
البرنامج صحيح

مثال الثاني

البنائة

هي نوع من انواع الدوال تستدعى و تنفذ لحظة تكوين الكائن و يمكنها استقبال متغيرات او لا تستقبل اي متغيرات ويمكن لجسم الدالة تنفيذ اي عملية يقوم المبرمج بكتابتها [داخلها] او تكون فارغة لا تؤثر.

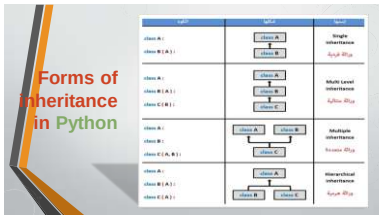




Inheritance in python

- 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().



The general structure of a subclass definition in Python :

Class key word Sub class Super class

```
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
```

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
```

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
الوراثة المتعددة في بايثون

Inheritance (الوراثة) ميزة قوية في البرمجة الشيئية OOP

يشير الى تعريف class جديد مع تعديل بسيط او بدون تعديل على class الموجود سابقاً.

1. ال class الرئيسية هي الفئة الموروثة , وتسمى ايضا الفئة الاساسية او parent.
2. ال class الجديدة فئة مشتقة او child وهي الفئة التي ترث من فئة اخرى.

في بايثون يوجد 4 اشكال لتوراثة وهي:

1. وراثة فردية **Single inheritance** :- هي أن كلاس يرث من كلاس واحد فقط.
2. وراثة متتابعة **Multi level inheritance** :- و تعني بأن كلاس يرث من كلاس واحد و هذا الكلاس كان في الأصل يرث من كلاس آخر.
3. وراثة متعددة **Multiple inheritance** :- تعني أن الكلاس يرث من أكثر من كلاس.
4. وراثة هرمية **Hierarchical inheritance** :- تعني أن الكلاس موروث من قبل أكثر من كلاس.

Multiple inheritance in python
الوراثة المتعددة في بايثون

الوراثة المتعددة" او "الإرث المتعدد" هي مفهوم في البرمجة الشيئية يستخدم في لغة بايثون وغيرها من لغات البرمجة الشائعة مثل

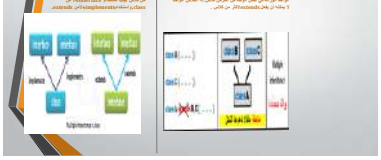
java , c#, c++ ... php و غيرها

الكلاس يمكنه ان يرث من كلاس آخر حتى يحصل على الدوال و المتغيرات الموجودة فيه. في حال كان الكلاس يرث من أكثر من كلاس، يجب وضع فاصلة بين كل كلاسين تضعهما بين القوسين.

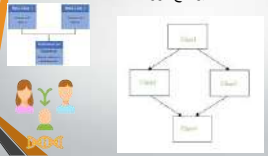
ويتم ذلك برمجياً في python

```
Class A:                    (نقوم بتعريف كلاس باسمه A)
    Body of the class
Class B:                    (نقوم بتعريف كلاس باسمه B )
    Body of the class
Class C(A,B):               ( نقوم بتعريف كلاس باسمه يرث من الكلاس A و الكلاس B )
    Body of the class
```

اما في لغة java



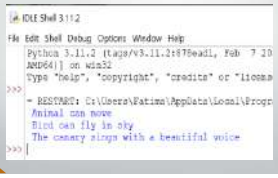
مخطط يوضح الوراثة المتعددة



مثال الاول(البرنامج)

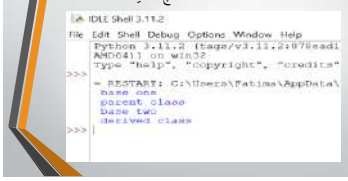
```
class Animal:
    def print_a(self):
        print(" Animal can write ")

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 ")

cl = Canary()

cl.print_a()
cl.print_b()
cl.print_c()
```

ناتج التنفيذ



مثال الثاني(البرنامج)

ناتج التنفيذ



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.

عندما نعرف ان الـ Abstract class هو كلاس يحتوي على Abstract methods وايضا ممكن لا يحتوي. وهناك ايضا Abstract class الذي يحتوي على Abstract methods فقط والذي ندعى بـ 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.

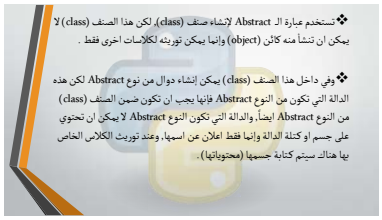
ملحوظة: في بايثون لا نحتاج لكلمة معينة حتى تكون واجهة نحن نستطيع تكوين واجهة باستخدام الـ abstract class الذي يحتوي على abstract methods فقط.

الـ interface ممكن ان يحتوي على:

- Constructor ←
- Variables ←
- Abstract Methods ←
- Sub Class ←

بعض الفروقات بين الـinterfaces 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.

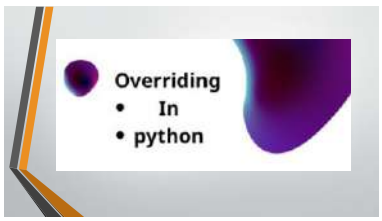

❖ تستخدم عبارة الـ Abstract لإنشاء صنف (class) لكن هذا الصنف (class) لا يمكن ان تنشأ منه كائن (object) وإنما يمكن توريثه لكلاسات اخرى فقط .
❖ وفي داخل هذا الصنف (class) يمكن إنشاء دوال من نوع Abstract لكن هذه الدالة التي تكون من النوع Abstract فإنها يجب ان تكون ضمن الصنف (class) من النوع Abstract ايضاً, والدالة التي تكون النوع Abstract لا يمكن ان تحتوي على جسم او كتلة الدالة وإنما فقط اعلان عن اسمها, وعند توريث الكلاس الخاص بها هنالك سيتم كتابة جسمها (محتوياتها) .


❖ ولاحظ ايضاً الدالة التي تكون من النوع Abstract يجب ان تذكر وتكتب في الصنف (class) الذي سيرثها واذا كانت اكثر من دالة يمكن كتابتهم جميعاً .
Class Shape
abstract Area();
Class Rectangle
area=?*?
Class Triangle
area=0.5*?*?
Class Circle
area=3.14*?*?


I can fly
I can crawl
I can bark
I can roar


I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides


Area of circle: 78.5
Area of triangle: 12.0
Area of rectangle: 21


Overriding
In
python


Override


شروط الـ
Overriding


PrintInfo.py هنا نحن نستطيع الوراثة عله الموجود في الملف
بتضمين
الكلاس PrintInfo
from CountryInfo import CountryInfo
class Australia(CountryInfo):
pass
class Lebanon(CountryInfo):
def print_language(self):
print('Arabic')
class Spain(CountryInfo):
def print_language(self):
print('Spanish')


English
Arabic
Spanish
from Australia import Australia
from Lebanon import Lebanon
from Spain import Spain
au = Australia()
lb = Lebanon()
sp = Spain()
au.print_language()
lb.print_language()
sp.print_language()


THE
END