Class Diagram

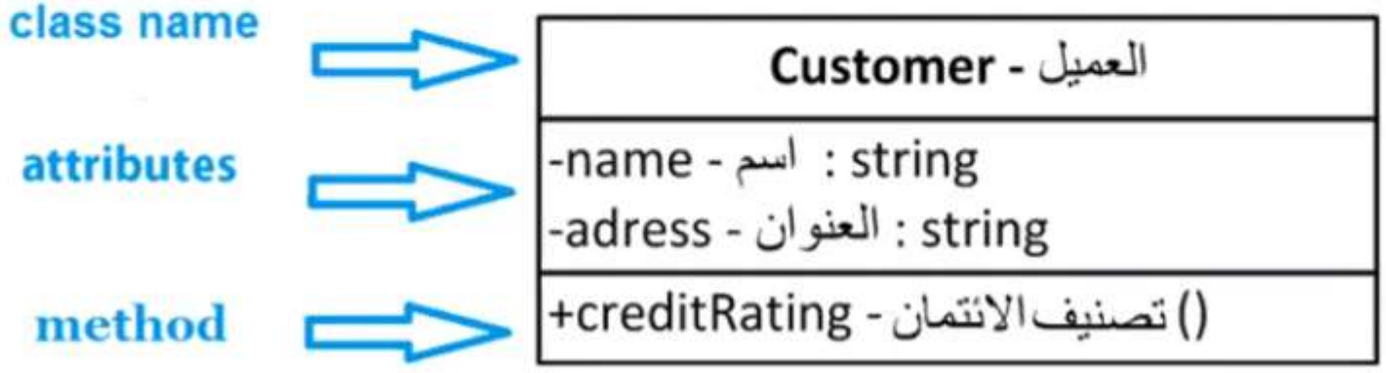
(مخطط الصفوف) و هو المخطط الأساسي الذي يعتمد عليه المبرمج أثناء تصميمه بالمنهجيات غرضية التوجه ، حيث تتم عملية التطوير كالتالي:

1- تحليل غرضي التوجه :

نركز في هذه المرحلة على فهم المشكلة و تحليل و نمذجة المتطلبات عن طريق مخطط UseCase Diagram .

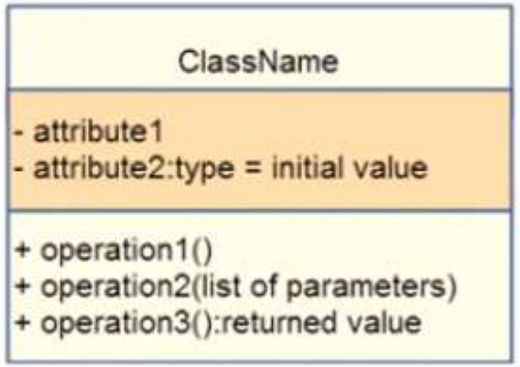
أي أننا نركز في هذه المرحلة على ما هو المطلوب من النظام (نمذجة للمتطلبات الوظيفية) كذلك نبدأ بالبحث عن ال classes (الصفوف) الأساسية التي يتكون منها

و بنهاية مرحلة التحليل سنصل إلى ما يعرف ب (الصفوف التحليلة) و هي على الشكل التالي :



مكونات مخطط الأصناف:

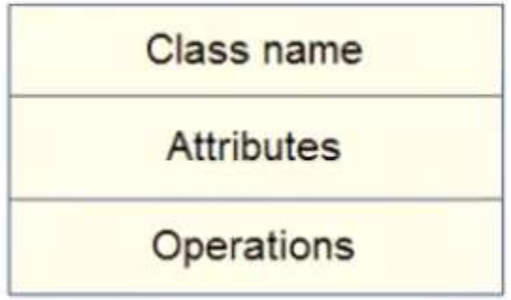
- الاصناف Classes
- الصفات Attributes
- الطرق Methods
- العلاقات Relationships

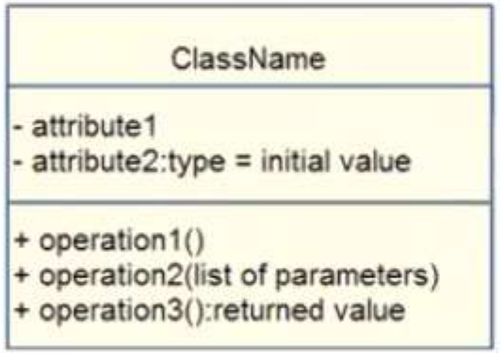


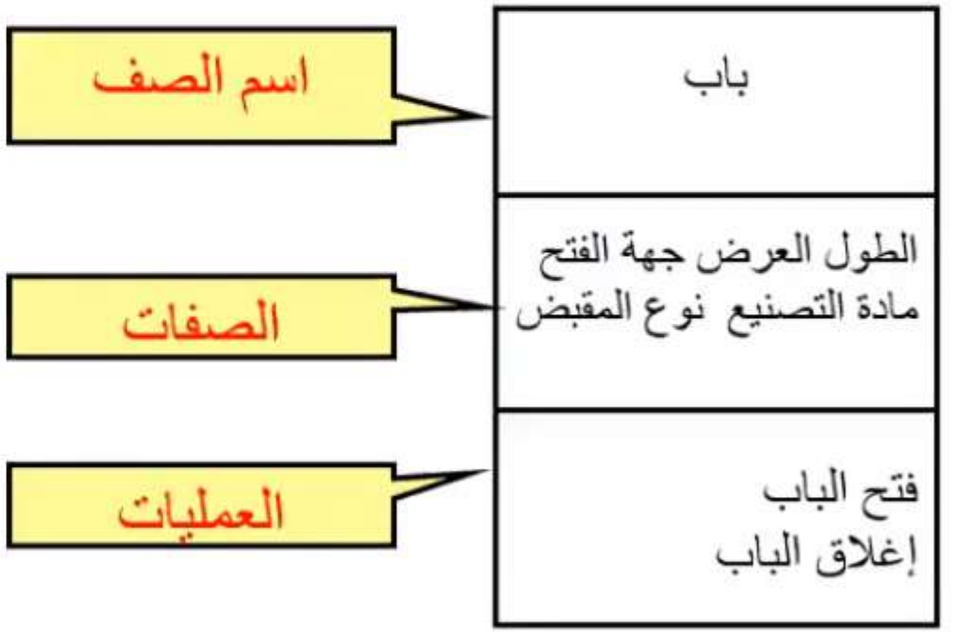
الصف	الطالب
الصفات	رقم الطالب
	اسم الطالب
	الهاتف
	العنوان
	معدل الثانوية
الطرق	- اضافة ()
	- حذف ()
	- تعديل ()
	- تخصص ()

الأصناف (الصفوف) Classes

الصنف Class هو تجريد لمجموعة من الأشياء تشترك بمزايا و مهام مشتركة.
مثال : باب – دجاجة – امتحان – مدرس
الكائن Object هو عبارة عن تحقيق للصنف المجرد على أرض الواقع فهو مثال حقيقي للصنف المجرد سواء كان هذا التحقيق مادياً أو معنوياً
مثال : باب الغرفة التي نحن فيها – امتحان المقرر نهاية الفصل - محمد الرسلان





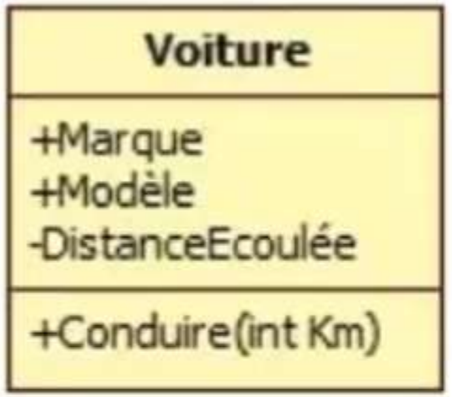


صفات الصنف Attributes هي صفات عامة للصنف وهي بمجموعها تغطي المزايا المشتركة للكائنات التي تنتمي لهذا الصنف. إن كل كائن من كائنات الصنف يجب أن يكون له قيمة معينة من أجل كل صفة من صفات الصنف الذي ينتمي له.
أما العمليات فهي ما يستطيع كائن من هذا الصنف أن يقوم به.

خصائص المتغيرات

Visibility

تصف الرؤية إمكانية الوصول إلى متغير الصنف



المعنى	الشكل المختصر	الاسم
يمكن الوصول إليها من كافة الأنواع الأخرى.	+	Public
يمكن الوصول إليها فقط من التعريف الداخلي لهذا النوع.	-	Private
يمكن الوصول إليها فقط ضمن الحزمة التي تتضمن هذا النوع، وفي أي حزم تقوم باستيرادها بشكل صريح. راجع تعريف مساحات الأسماء و الحزم.	~	الحزمة
يمكن الوصول إليها فقط من هذا النوع و الأنواع التي ترث منه. راجع الوراثة .	#	محمي

إذن نبدأ في مرحلة التصميم باكتشاف العلاقات مابين ال classes، فما هي هذه العلاقات ؟

إن العلاقات المستخدمة في ال class Diagram على نوعين :

1) علاقات تربط بين الصفوف مباشرة .
2) علاقات تربط بين ال objects من هذه الصفوف .

و سنتحدث عن هذه العلاقات بالتفصيل :

سنبدأ أولاً بالعلاقات التي **تربط بين الصفوف** و هي على نوعين :

1- الوراثة (generalization) .
2- التحقيق (realization) .

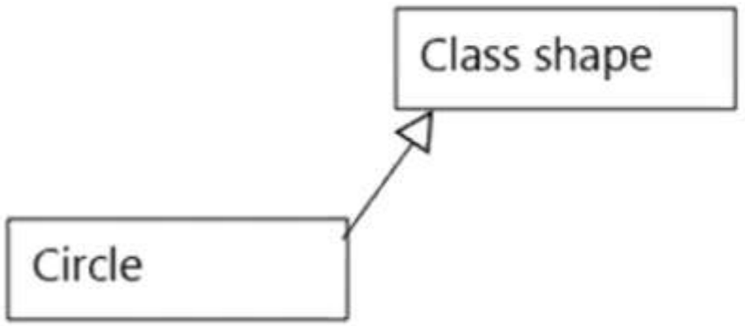
أولاً : الوراثة (generalization)

نستخدم هذه العلاقة عندما يكون لدينا صف لديه صفات عامة و صف آخر لديه تلك الصفات العامة بالإضافة إلى صفات أخرى خاصة به ، و عندها نجعل الصف الذي يحوي الصفات العامة (صف أب) و الصف الذي يحوي الصفات الخاصة (صف ابن) .

مثلاً :

ليكن لدينا صف shape (شكل) يحوي بعض الصفات العامة مثل المساحة و المحيط و لدينا صف آخر هو circle يحوي الصفات العامة المساحة و المحيط بالإضافة إلى واصفة خاصة هي (نصف القطر R)

هنا علينا أن نجعل ال class الذي يسمى shape أب و نجعل له class ابن هو ال circle كالشكل التالي :

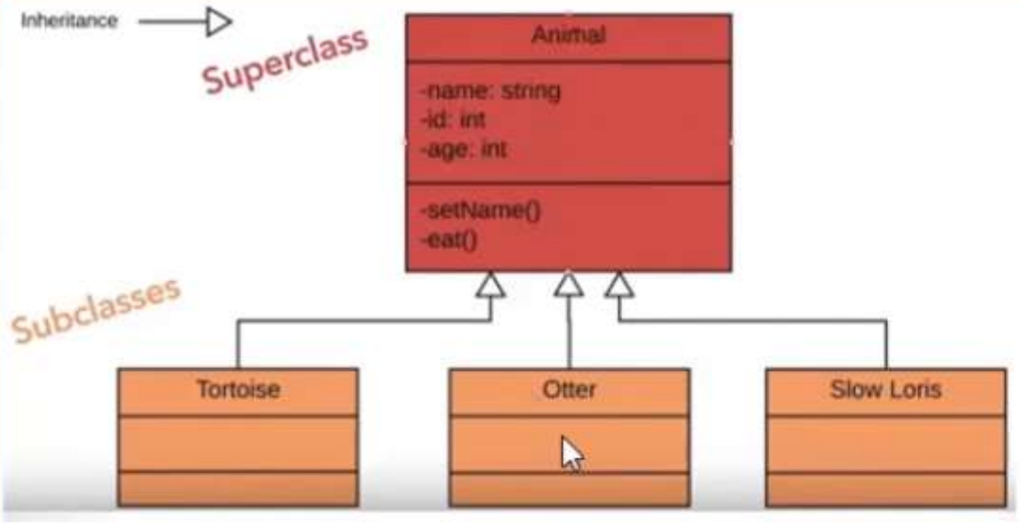


و كما نلاحظ تمثل الوراثة بسهم يتجه من الصف الابن إلى الصف الأب.

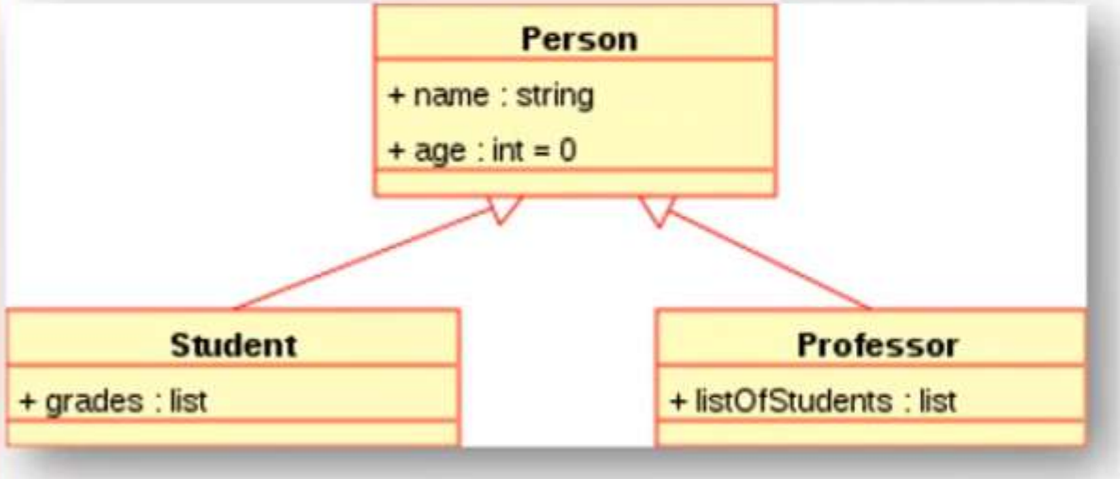
لاكتشاف علاقة الوراثة بين الأصناف نتبع أسلوب is a

Circle is a shape

علاقة الوراثة (التعميم Generalization)



- علاقة التعميم تمكن المحلل من انشاء اصناف تستطيع توارث Inheritance صفات و عمليات اصناف اخرى
  - Super class
  - Sub class



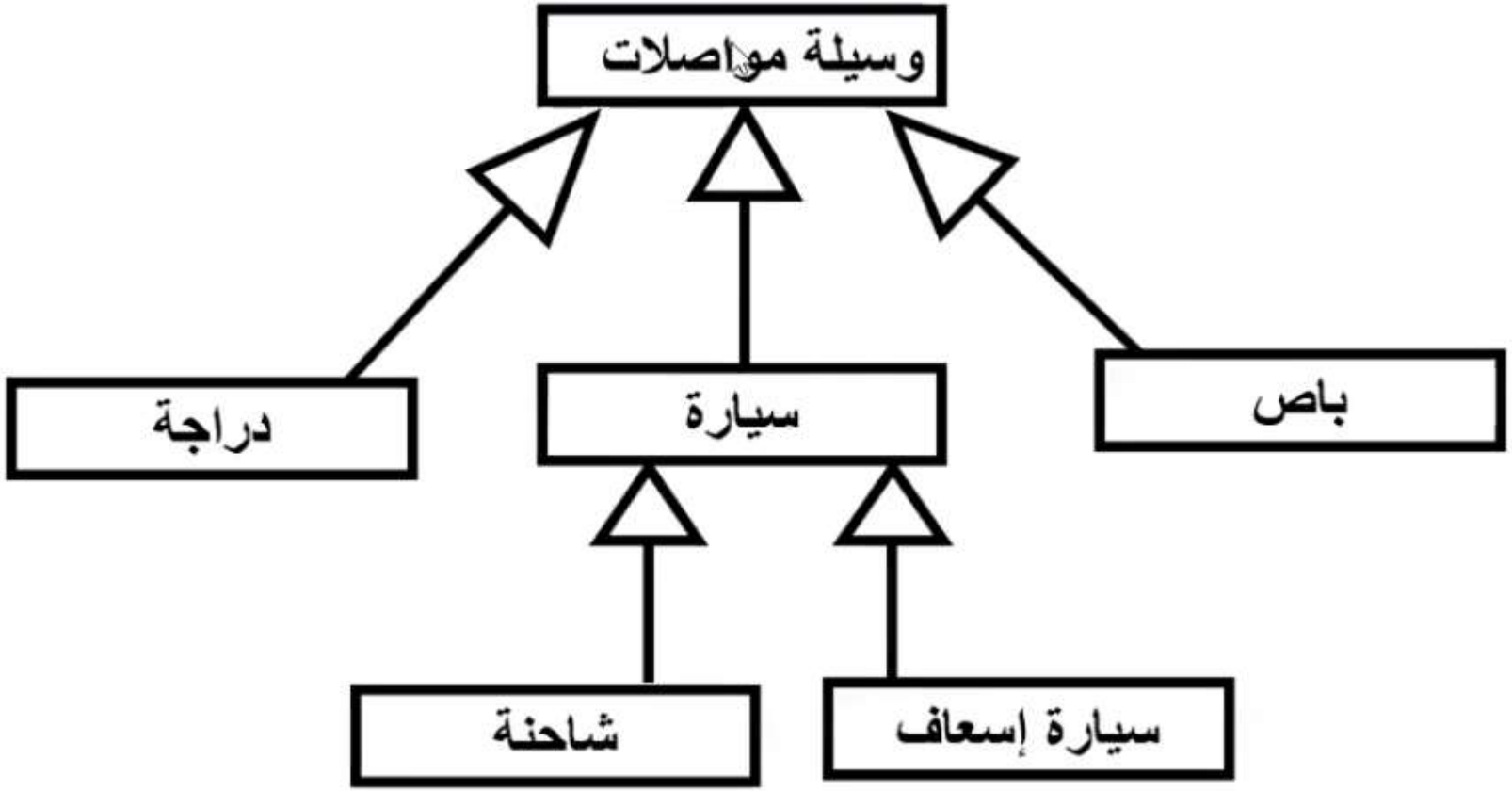
ملاحظة :

تعتبر الوراثة أهم مبدأ من مبادئ البرمجة غرضية التوجه و ذلك لأنها تحقق أهم مبدأ من مبادئ هندسة البرمجيات .

و هو مبدأ ال reusability ، فلو اتبعنا المنهجية غرضية التوجه في تطوير نظامنا و لم نقم باستخدام علاقة الوراثة ما بين ال classes يعتبر كل شغلنا بلا طعمة .

علاقة الوراثة (التعميم Generalization)

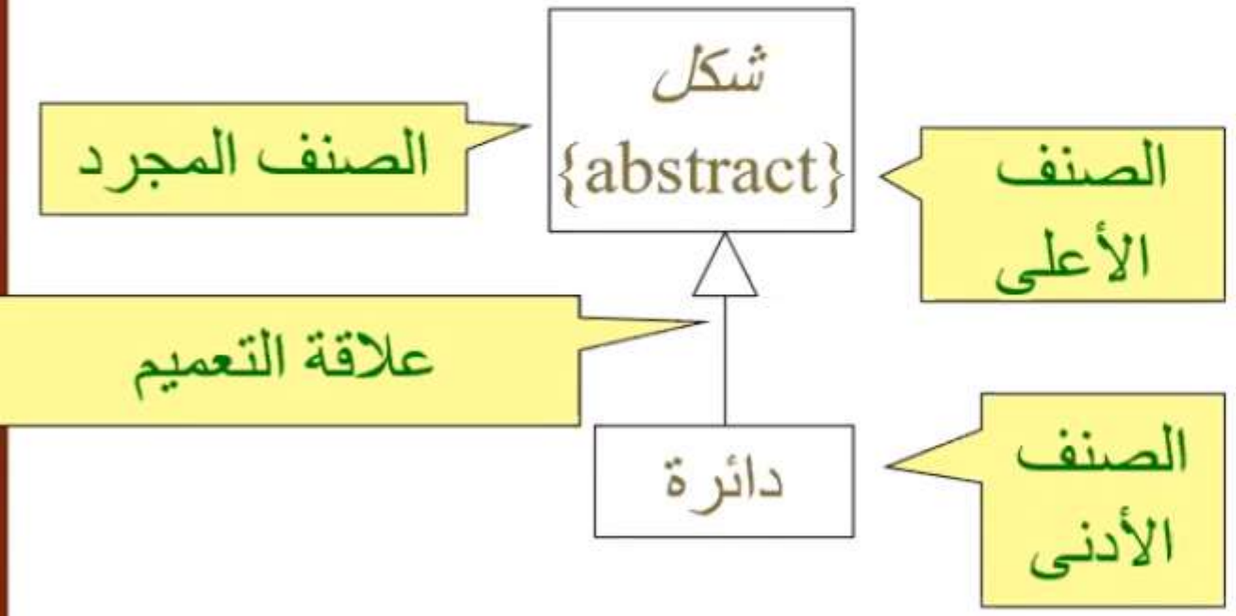
علاقة الوراثة متعددة المستويات



علاقة الوراثة (التعميم Generalization)

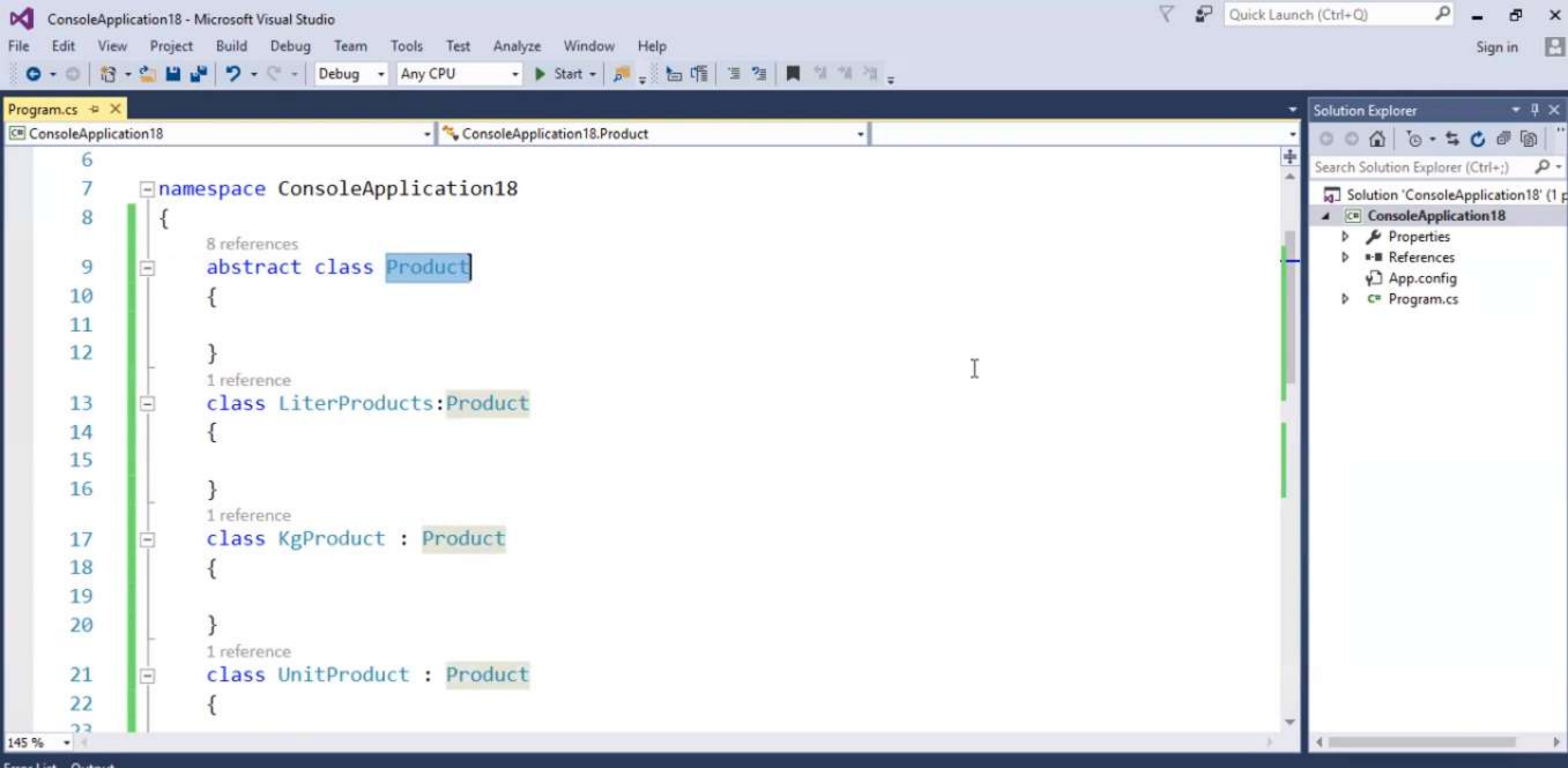
الصنف المجرد: الفئات المجردة هي فئات غير موجودة على أرض الواقع وإنما تمثل مفهوما عاما لشيء ما، وهي فئات لا يمكن استنساخ كائنات منها.

يتميز بأن اسمه مائل و بالكلمة abstract إلى جانبه



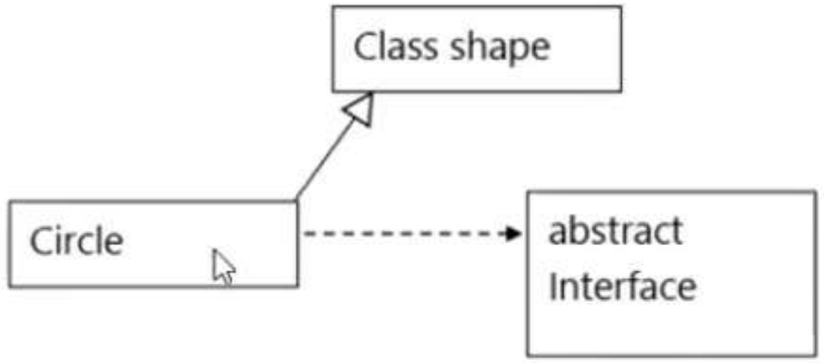
قلنا أن البرمجة كائنية التوجه أتت لتجعل من مفهوم البرمجة أقرب مايكون للحياة الواقعية، والتي يعتبر كل مافيها كائنات. في الحياة الواقعية هناك كائنات غير موجودة فعليا لكنها تعتبر نوعا لكائنات أخرى، فوسائط النقل هي كائن لكنه غير موجود، أما السيارات والطائرات والسفن والقطارات هي كائنات مشتقة من الكائن وسيطة نقل ولها مميزات خاصة بها تميزها عن الكائنات الأخرى من نفس النوع. بمعنى أنه لا يمكنك أن ترى وسيطة نقل أو تستخدمها لكن بإمكانك استخدام ورؤية السيارات، فوسيطة النقل

عبارة عن فكرة مجردة أما السيارة فهي كائن موجود!


ConsoleApplication18 - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Sign in
Debug
Any CPU
Start
Program.cs
ConsoleApplication18
ConsoleApplication18.Product
6
7 namespace ConsoleApplication18
8 {
8 references
9 abstract class Product
10 {
11
12 }
1 reference
13 class LiterProducts:Product
14 {
15
16 }
1 reference
17 class KgProduct : Product
18 {
19
20 }
1 reference
21 class UnitProduct : Product
22 {
145 %
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApplication18' (1
ConsoleApplication18
Properties
References
App.config
Program.cs

```
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Product p1 = new LiterProducts();
        Product p2 = new KgProduct();
        Product p3 = new UnitProduct();
        List<Product> bill = new List<Product>();
        bill.Add(p1);
        bill.Add(p2);
        bill.Add(p3);
    }
}
```

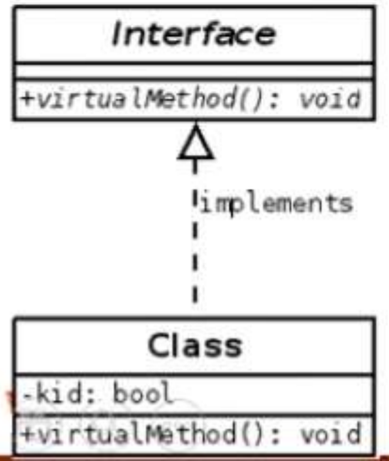
ثانياً : التحقيق (realization) :



لدينا بعض لغات البرمجة غرضية التوجه تمنع ال Multi Inheritance و لذلك إذا أردنا أن نجعل class ما يرى أكثر من صف ماذا علينا أن نفعل ؟؟

نقوم ببساطة بتعريف Interface و تكون abstract أي نعرف لها سلوك معين لكن لا نستطيع أن ننشئ غرضاً من تلك ال Interface أي لانستطيع أن نقوم بعملية تحقيق لهذه ال Interface ، و نجعل ال class الذي نريده أن يرث من أكثر من صف أن يرث من هذه ال Interface .

و بالتالي class circle قام بالوراثة من shape و بتحقيق ل interface معينة ، و كما نلاحظ تمثل علاقة التحقيق بسهم منقط من الاين إلى ال Interface.



هنالك علاقة مختلفة في UMLللواجهات ((interfaces، فالوراثة من واجهة تسمى " "implementationالتي هي علاقة "تطبيق" ((realizationفي مخططات .UMLتمثيلها الشكلي مشابه للوراثة، إلا أنَّ الخط مقطّع ((dashed، ويجب تحديد أنَّ الواجهة هي «مجرَّدة» (– (abstractأي أنَّ اسمها مكتوبٌ بخطٍ مائل؛ كما هو مبيّن في هذا الرسم.

تدعم بعض لغات البرمجة مبدأ الوراثة المتعددة، أي إمكانية وراثة فئة ما لأكثر من فئة أخرى. فيC # لا يمكن للفئة أن ترث إلا من فئة واحدة..

وبالتالي للوراثة من أكثر من فئة فنستخدم مفهوم الواجهات.

يشابه مفهوم الواجهات بشكل كبير مفهوم الفئات المجردة، والتي لا يمكن استساخ كائنات منها، بالإضافة لعدم إمكانية كتابة أي كود عملي داخل الواجهات وإنما فقط الوظائف والخصائص التي ستستعمل هذه الواجه

يمكن لفئة واحدة أن ترث من أكثر من واجهة، وهو الأمر غير الممكن بالنسبة للوراثة من الفئات، وتسمى عملية الوراثة من واجهة بـ Implementation.

```
public interface ICar
{
  int carMaxSpeed { get; set; }
  void AddItem(string Item);
}

public class BMW : ICar
{
}
public class BMW2 : ICar, ITruck
{
}
```

إن العلاقات المستخدمة في ال class Diagram على نوعين :

1) علاقات تربط بين الصفوف مباشرة .
2) علاقات تربط بين ال objects من هذه الصفوف .

ما بين ال objects لدينا علاقة واحد تربط بينهم و هي علاقة **association (الاقتران)** ، و هذه العلاقة تقسم لنوعين :

1) Aggregation (علاقة تركيب) .
2) Composition (علاقة تشكيل) .

رمز العلاقات في مخطط الأصناف