

الفصل السادس

٦

البرمجة الموجهة نحو الكائنات

Object Oriented Programming
OOP

مقدمة

تبنى البرمجة الموجهة نحو الكائنات أو العناصر Object Oriented Programming - OOP (قد تسمى بالبرمجة كائنية التوجه أو البرمجة الشيئية أو البرمجة الكائنية) على استخدام الكائن Object كوحدة برمجة حيث كانت البرمجة التقليدية توفر للمبرمج مكتبة من الدوال إضافة إلى تركيب تقليدي للبرنامج وعلى المبرمج أن يستخدم الدوال مع تركيب البرنامج لإنشاء التطبيقات مما يضطره لكتابة السطور الكثيرة أكثر من مرة

لقد كانت وحدة بناء البرنامج هي الدالة، ولكن جاءت البرمجة بواسطة الأهداف بفكرة جديدة هي إنشاء عناصر متكاملة تحتوي على بيانات ودوال هي أساس إنشاء البرنامج، وبالتالي أصبحت وحدة بناء البرنامج وحدة كبيرة هي الهدف Object أو الفصيلة Class مما سهل واختصر الكثير حيث يتم إعداد مجموعة من الفصائل العامة Classes التي تلبي معظم متطلبات إعداد برنامج والتي يحتاجها المبرمج

ولقد كانت برامج الحاسب التقليدية عبارة عن مجموعة من التعليمات التي يكتبها المبرمج في ملف ويقوم الحاسب بتنفيذها حسب الترتيب المحدد مسبقا ولكن في حالة OOP يكون البرنامج عبارة عن مجموعة من الكائنات المستقلة ويكون لكل كائن مهمة محددة

وللتوضيح أكثر نشبه الكائنات في لغات برمجة OOP مثل الجافا بالكائنات الحية حيث يستخدم الكائن الحي مكوناته لأداء المهام التي خلقت من أجلها فيستخدم الإنسان مثلا رجله للمشي ويديه للعمل ولسانه للكلام وأذنيه للسمع وهكذا

وحيث نحلل أو نجزأ برنامج الجافا إلى أجزاء لكل جزء أو مكون مهمة محددة فإننا حينها نكون في عملية برمجة OOP ، وكل برنامج في لغات برمجة OOP يكون عبارة عن مجموعة من الكائنات التي تعمل معا لتحقيق عمل ما ، وليست جميع الكائنات سواء بل تختلف عن بعضها في أمرين : الأول هو الصفة (خصائص الكائن نفسه) ، والثاني هو السلوك (يختلف سلوك كائن عن كائن آخر)

مزايا البرمجة بواسطة الأهداف OOP

- ١ - إمكانية استخدام كائن ما في أكثر من برنامج، فمثلا إذا كنت بصد كتابة برنامج لتسجيل الدخول على الإنترنت وتسجيل المواقع التي زارها المستخدم فإنك بحاجة إلى كائن مودم Modem يقوم بوظيفة محددة ، وعندما تكتب برنامجا آخر لطلب المكالمات التليفونية تلقائيا فلن تحتاج إلى كتابة برنامج للتعامل مع المودم حيث يمكنك وضع كائن المودم في البرنامج لكي يتعامل مع جهاز المودم لكي يطلب رقما معيناً ويبلغ برد الجهاز المطلوب
- ٢ - سهولة إزالة الأخطاء حيث أن اللغات غير OOP عبارة عن لائحة طويلة من الأوامر مرتبة على بعضها البعض وبالتالي توجد صعوبة في إزالة الأخطاء ولكن في حالة البرمجة OOP كل كائن مستقل بنفسه ويقوم بوظيفة محددة وعند وجود خطأ ما فإنها تكون مرتبطة بكائن واحد مستقل فيسهل إزالة هذا الخطأ

أهم مصطلحات OOP

١- الفصيلة أو الفئة Class

الفصيلة هي أساس البرمجة بواسطة الأهداف وهي التي يبني عليها البرنامج وهي عبارة عن البرنامج ككل أو الوصف الرئيسي الذي تنحدر منه Objects فكل برنامج نكتبه بلغة Java يسمى فئة Class، وأخذت فكرة الفصيلة من واقع الحياة فكل عنصر في الحياة عبارة عن فصيلة Class فمثلا يمكن أن نطلق على جميع السيارات إنها من فصيلة car مع بعض الاختلافات ، ويمكن أن نطلق على الطيور فصيلة Bird وهكذا

وتنتمي جميع العناصر الى فصول Classes وكل فصيلة نستطيع تمثيلها بعنصرين هما البيانات Data والدوال Methods فمثلا فصيلة الموظف Employee بياناتها هي بيانات الموظف العامة (كود الموظف -اسم الموظف -عنوان الموظف وباقي بياناته) ، وكذلك الدوال Methods هي دوال تحقيق العمليات التي يمكن أن تتم على الموظف مثل : عملية إضافة موظف جديد وحذف موظف وتعديل بيانات موظف وهكذا

والفصيلة في البرمجة عبارة عن مجموعة من السطور التي تمثل عنصرا تمثيل تام من حيث بيانات العنصر (والتي تسمى خصائص Properties) ودوال العنصر (والتي تسمى Methods) وبتقسيم البرنامج إلى فصول يصبح أكثر تنظيما وأسرع في إعداده

وتستخدم الفئة Class كقالب تصميمي لإنشاء الكائنات وبالتالي تعتبر الفئة نسخة أساسية للكائنات تحدد صفات وسلوك الكائن وعلى سبيل المثال كل برنامج يستعمل سلاسل الحروف يستخدم Classes String وعليه فإن هذا Classes لابد أن يحتوي على صفات وخواص تحدد ماهية الكائن String وفي نفس الوقت يحتوي هذا Classes على محددات سلوك للكائن String

ولقد قامت شركة Microsoft بتطوير لغة VB.NET وقامت بداخلها بإعداد مجموعة من الفصول التي تلبي متطلبات المبرمج حيث يجب على المبرمج عند إعداد برنامج ما أن يقوم بدراسة مكتبة الفصول Class Lib الموجودة للتعرف على الموجود بها والتعرف على بيانات (خصائص) Properties ودوال الفصيلة Methods()

الخلاصة

مفهوم الكلاس Class في الجافا عبارة عن حاوية كبيرة تحتوي على كل الكود من: متغيرات ودوال وكائنات إلخ..، أو هو مجموعة من البيانات Data items ، Methods أو operations أو Functions أي العمليات أو الوظائف التي سوف تشغل أو تعمل على تلك البيانات

بفرض اننا نريد عمل برنامج يحسب مساحة مستطيل فسوف نكون بحاجة الى بيانات من المستخدم مثل طول Length ضلع المستطيل والعرض Width ، وبالتالي فإن Length ، Width هي بيانات Data items سيتم تخزينها في البرنامج ثم يتم إجراء عملية حسابية لحساب المساحة area يتم وضعها على شكل method وهي وظيفة أو عملية يتم تنفيذها على البيانات وبالتالي فإن الكلاس يضم كل من Methods + Data item

وبالتالي فإن أي كود في الجافا لابد ان يكون داخل كلاس

```
public class Welcome {  
    public static void main(String[] args) {  
        // Display message Welcome to Java! on the console  
        System.out.println("Welcome to Java!");  
    }  
}
```

ولتعريف كلاس جديد يكفي فقط كتابة الكلمة **class** ثم وضع اسم له ثم فتح أقواس **Brackets** وهي:
قوس يحدد بداية أو فتح الكلاس ، وقوس يحدد نهاية أو غلق الكلاس

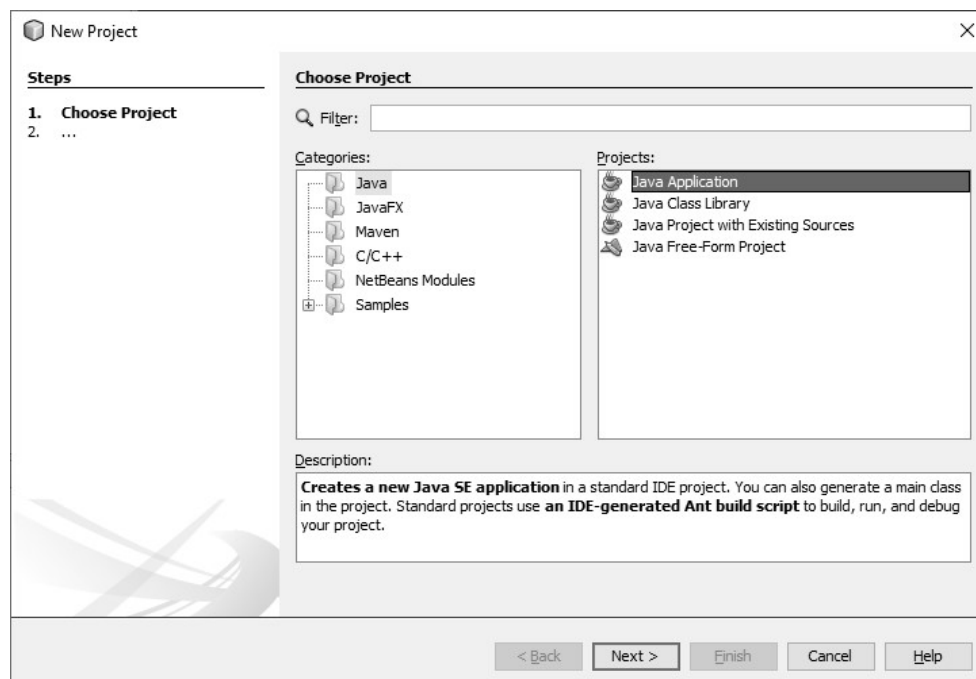
```
Public class ClassName {  
  
}
```

-أي برنامج جافا يحتوى على كلاس واحد على الأقل ، وعدد لا نهائي من الكلاسات

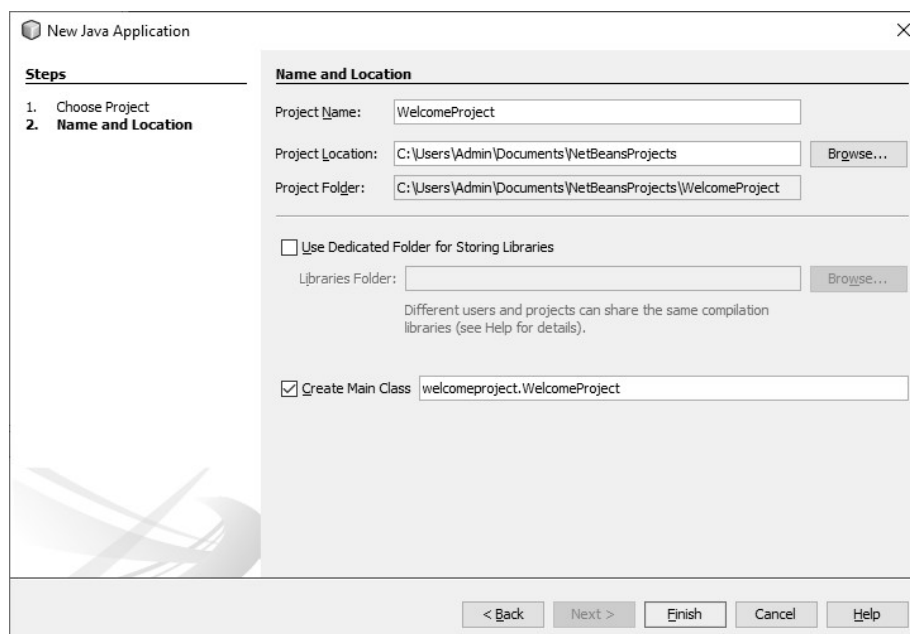
-أي برنامج جافا يحتوى على **method** واحدة فقط تسمى **main method** (الرئيسي أو الأساسي) وهي تمثل نقطة الدخول في البرنامج، فمثلا البرنامج الواحد يتضمن عشرات الكلاس ، ولكن هذا البرنامج عندما يبدأ في التشغيل يبدأ من أي كلاس بالضبط؟ الإجابة انه يبدأ يشتغل من **main** كلاس والذي يسمى **entrepont** أي النقطة المركزية أو نقطة الدخول في البرنامج أو بداية تنفيذ البرنامج تكون من هنا

مثال

نفتح برنامج **NetBeans IDE** ونفتح قائمة **File** ونختار **new project** (هذا المشروع **Project** يمكن اعتباره مثل مجلد **folder** يضم كل الكلاسات والملفات المساعده لها ، معنى ذلك ان المشروع الواحد **project** يمكن ان يتضمن أكثر من برنامج وأكثر من كلاس وملفات إضافية أخرى تستخدم في البرنامج الحالي فهو يمثل عملية تنظيمية) ثم نختار **java/java Application**

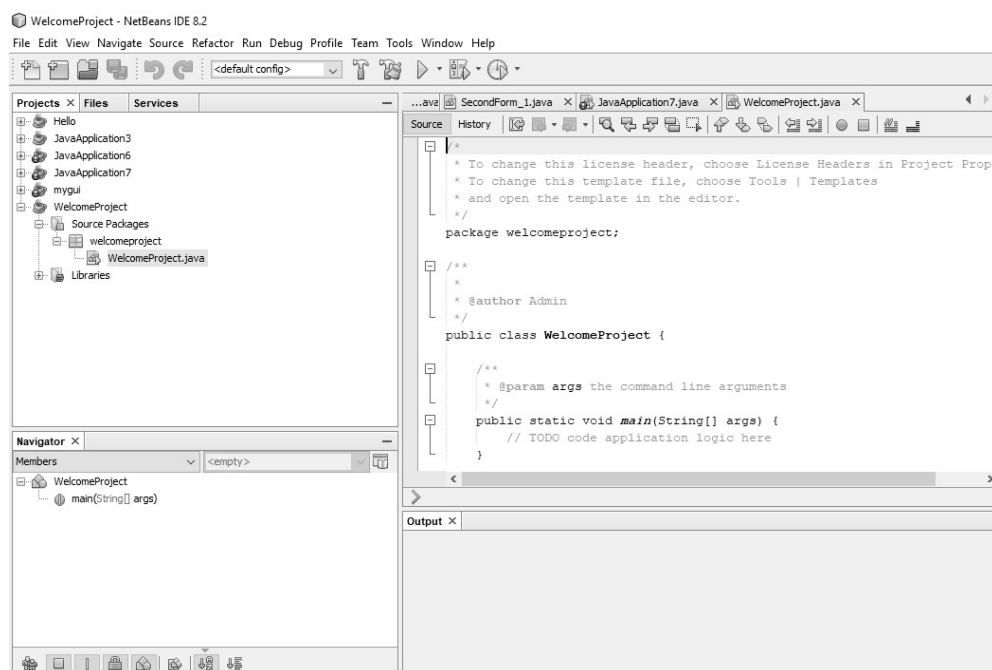


نضغط next عندئذ يظهر مربع حوارى يطلب اسم للمشروع ويمكن تسميته Welcome Project ، مع ملاحظة تنشيط الخيار Create Main Class



نضغط الزر Finish

الفصل السادس: البرمجة الموجهة نحو الكائنات OOP - Object Oriented Programming



يظهر في الاطار الایسر اسم المشروع `WelcomeProject.java` ، مع العلم بأن أي source code يتم كتابته في الجافا يتم تخزينه في ملف بالامتداد `.java`.

ويظهر في الجانب الأيمن كود الكلاس `class WelcomeProject` ، يليه قوس الفتح والغلق

كما يظهر كود `main method` والذي يمثل نقطة البداية لتنفيذ المشروع مع العلم إن أي `method` لابد ان يكون لها قوس بداية وقوس نهاية وبينهما الكود

كما نلاحظ وجود أكواد مكتوبة باللون الرمادي الفاتح فهي تسمح تعليقات `comments` وهي تعنى سطور يقوم المبرمج بكتابتها ولا يريد تنفيذها والغرض منها مجرد ملاحظات أو توثيق `Documentation` كتسجيل ملحوظة ما

ويمكن كتابه تلك التعليقات على سطر واحد وبالتالي يجب ان تبدأ بعلامتي `//` أو يمكن ان تكتب التعليقات على عدة سطور `Multi line comment` فيجب ان تكون التعليقات بين العلامتين `/*-----*/` (نلاحظ انه بمجرد كتابة الرمز `/*` فان برنامج `IDE` يقوم مباشرة بعمل رمزي الاغلاق `*/`)

نقوم بكتابة جملة الطباعة لرسالة الترحيب وهي كما يلي:

```
System.out.print("welcome to java");
```

ويجب ان ينتهي السطر بالفاصلة المنقوطة `Semi colon` والتي توضح للمترجم `Compiler` ان الجملة او العبارة `statement` انتهت عن ذلك الحد، وفي حالة عدم إضافة الفاصلة المنقوطة والنزول الى السطر التالي تظهر أيقونة حمراء اللون توضح ان هناك خطأ `error` وتوضح أيضا طبيعة الخطأ وهو ان البرنامج كان متوقع `expected` انك تقوم بعمل كذا

```

    /**
     * @param args the command line arguments
     */
    public void main(String[] args) {
        System.out.print("welcome to java");
    }

```

وبالتالي الكود الصحيح كما يلي:

```

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.print("welcome to java");
    }

```

كما يجب مراعاة ان الجافا Case Sensitive حساسة لحالة الاحرف، فمثلا يجب أن يكون الحرف الأول من كلمة System كبير وفي حالة كتابته صغير سوف تظهر رسالة خطأ كما يلي:

```

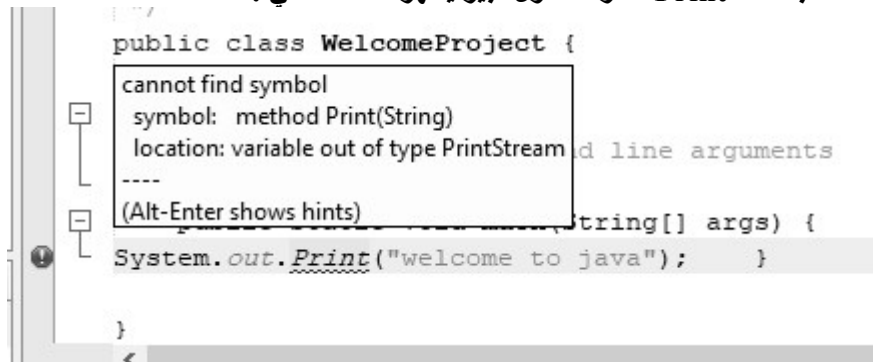
    /**
     * @param args the command line arguments
     */
    public void main(String[] args) {
        system.out.print("welcome to java");
    }

```

تتضمن رسالة الخطأ الكلمة package وتعني حزمة (كلمة System يطلق عليها package أي مجموعة أو حزمة من الكلاسات حيث ان كل مجموعة كلاسات لهم علاقه ببعضهما البعض يتم وضعهم في الجافا داخل حزمة package)

وبالتالي فعند القول ان الكلاس المسمى out الموجود في الحزمة package المسماه system نريد استخدام print method الخاصة به ، وبالتالي عندما يقوم البرنامج بالبحث عن الحزمة system فلا يجدها نظرا لان الحرف الأول منها صغير)

وأيضاً عندما نكتب كلمة **Print** الحرف الأول كبير يظهر الخطأ التالي :



توضح الرسالة السابقة ان **Print** method أو الوظيفة أو **function** أو **operation** أو **method** المسماة **Print** غير موجودة عندي في الكلاس المسمى **out**

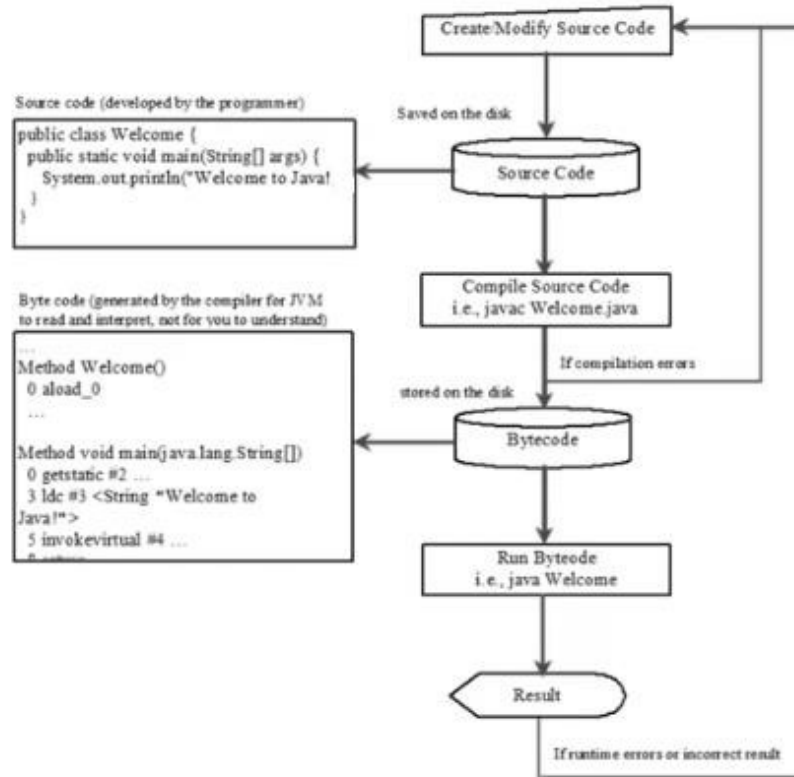
نقوم بتنفيذ البرنامج السابق بالضغط على الزر **Run** أو الضغط على مفتاح **F6** عندئذ تظهر نتيجة التنفيذ أسفل النافذة كما يلي:



خلاصه ما سبق

كتابة الكود المسمى **source code** - ترجمة الكود **compilation**

-تنفيذ الكود **run** والذي يعنى **execute to program**



المرحلة الأولى Create/Modify Source Code كتابة كود الجافا

المرحلة الثانية Source code

تخزين الكود السابق في ملف file امتداده java. هذا الكود مكتوب بلغة برمجة تسمى High level language وهي اللغة التي يفهما الانسان ولا تفهماها الاله والتي تفهم بما يسمى Machine language

المرحلة الثالثة Compile source code

يجب تحويل Source code الى اللغة التي يفهمها الحاسب لذا نحتاج الى ما يسمى compiler وهو برنامج software يقوم بالتعامل مع source code وتحويله الى لغة الحاسب

المرحلة الرابعة Bytecode

في لغة الجافا عند القيام بعملية التحويل فإن source code يتم تحويله الى ما يسمى Bytecode وهو ليس Machine language وإنما هو كود وسيط مخرجاته عبارته عن ملف file يسمى .class.

المرحلة الخامسة Run Bytecode

التنفيذ أي عمل Run للبرنامج وهذا التنفيذ يتم للـ Bytecode الناتج من خطوة Compilation وإظهار النتائج Results

٢- الخصائص Attributes

أي متغيرات يتم تعريفها داخل كلاس و خارج أي دالة تسمى خصائص وهذا يعني أن أي كائن من هذا الكلاس سيكون عنده هذه الخصائص ، ونستطيع التعامل مع هذه الخصائص من الكائن مباشرة ، بينما المتغيرات العادية لا يمكن التعامل معها من الكائن ، المتغيرات التي يتم وضعها كباراميترات أو التي يتم تعريفها بداخل الدوال تسمى متغيرات عادية (برنامج Netbeans يلون أسماء الخصائص باللون الأخضر لكي يساعد المبرمج على التفريق بين المتغيرات العادية والمتغيرات التي يتم اعتبارها خصائص علما بأن المتغيرات تسمى خصائص لأن أي كائن من هذا الكلاس سيملك نسخته الخاصة منها)

٣- الكائن Object

عبارة عن شيء له صفات وأساليب معينة خاصة به وهو نسخة مطابقة لكلاس معين وبالتالي لا يمكن إنشاء كائن إذا لم يكن هناك كلاس ، وفي مفهوم برمجة الكائنات يتم إنشاء كلاس معين يسمى blue print أي (النسخة الخام أو النسخة الأصلية) ثم ننشئ نسخة أو أكثر من هذا الكلاس و نفعل بها ما نريد بدون أن نغير محتويات الكلاس الأساسي وهكذا نكون حافظنا على أكواد الكلاس الأساسي لأننا نعدل على النسخ و ليس عليه مباشرة.

وبما أن الكائن عبارة عن نسخة من الكلاس لذا فلتعريف كائن من كلاس معين يجب وضع اسم الكلاس ثم وضع اسم للكائن.

مثال:

تعريف كائن من الكلاس Person اسمه elsaeed

```
Person elsaeed = new Person();
```

إذا الكائن elsaeed يكون عنده نسخة خاصة فيه من خصائص الكلاس Person

ملاحظة :

الكود new Person() هو الذي يقوم فعلياً بتوليد كائن من الكلاس، وهو يعطي قيم أولية للخصائص الموجودة فيه.

كتابة نفس الكود السابق على مرحلتين للحصول على كائن من الكلاس Person

1	<u>// Person سيمثل كائن من الكلاس</u>
2	<u>Person elsaeed;</u>
3	<u>// Person الكلاس elsaeed يمثل كائن من الكلاس</u>
4	<u>elsaeed = new Person();</u>

طريقة التعامل مع الكائنات

نقوم بإنشاء كائن من الكلاس ثم إدخال قيم لخصائصه واستدعاء دواله إلخ. ، ولا استدعاء أي شيء موجود في الكائن الذي تم إنشاؤه نضع اسم الكائن ثم نقطة ثم الشيء الذي نريد الوصول إليه (سواء اسم متغير أو دالة).

ملحوظات

-يفضل إنشاء كل كلاس في ملف جافا خاص.

-يبدأ اسم الكلاس دائماً بحرف كبير ، واسم الكائن دائماً بحرف صغير.

علاقة الكائن بالكلاس في الجافا

تساعد الكائنات المبرمج، فعند الرغبة في إنشاء برنامج لحفظ معلومات أشخاص لا يتم إنشاء كلاس لكل شخص بل يتم إنشاء كلاس واحد فقط يمثل شخص ونضع فيه الأشياء الأساسية التي نرغب أن تكون موجودة عند كل شخص، ثم ننشئ منه كائنات حيث يصبح كل كائن من هذا الكلاس عبارة عن شخص له معلوماته الخاصة

مثال:

إنشاء الكلاس Person وإنشاء كائنات منه في الكلاس الذي يحتوي على الدالة main() (يجب إنشاء الكلاس Person والكلاس Main في نفس المجلد حتى يعمل الكود بشكل صحيح)

Person.java	
1	public class Person {
2	<u>تعريف ٤ خصائص</u>
3	String name;
4	String sex;
5	String job;
6	int age;
7	<u>تعريف دالة تطبع محتوى كل خاصية عندما يتم استدعاءها</u>
8	void printInfo() {
9	System.out.println("Name: " +name);
10	System.out.println("Sex: " +sex);
11	System.out.println("Job: " +job);
12	System.out.println("Age: " +age);
13	System.out.println();
14	}
15	}
Main.java	
1	public class Main {
2	public static void main(String[] args) {
3	<u>إنشاء كائنات من الكلاس Person</u>
4	Person p1 = new Person();
5	Person p2 = new Person();
6	Person p3 = new Person();
7	Person p4 = new Person();
8	<u>تحديد خصائص الكائن p1</u>
9	p1.name = "Mostapha";
10	p1.sex = "Male";
11	p1.job = "Programmer";
12	p1.age = 31;
13	<u>تحديد خصائص الكائن p2</u>
14	p2.name = "reem";
15	p2.sex = "Female";
16	p2.job = "Secretary";
17	p2.age = 32;
18	<u>تحديد خصائص الكائن p3</u>
19	p3.name = "eslam";
20	p3.sex = "Male";
21	p3.job = "Doctor";
22	p3.age = 44;

```
23 // تحديد خصائص الكائن p4
24 p4.name = "remas";
25 p4.sex = "Female";
26 p4.job = "Engineer";
27 p4.age = 37;
28 // عرض خصائص كل كائن
29 p1.printInfo();
30 p2.printInfo();
31 p3.printInfo();
32 p4.printInfo();
33 }
34 }
```

نتيجة التنفيذ:

Name: Mostapha
Sex: Male
Job: Programmer
Age: 31

Name: reem
Sex: Female
Job: Secretary
Age: 32

Name: eslam
Sex: Male
Job: Doctor
Age: 44

Name: remas
Sex: female
Job: Engineer
Age: 37

٤-دوال البناء Constructors

من الخصائص الهامة في مفاهيم البرمجة بالأهداف OOP وجود دالة البناء Constructor داخل الفصيلة Class وهي دالة Method مثل أي دالة ولكن الفرق أنها دالة تنفذ تلقائياً بدون استدعاء عند استخدام الفصيلة Class (عند تعريف عنصر Object) والغرض من ذلك هو استخدام تلك الدالة في تنفيذ أي عمليات أولية للفصيلة class مثل إعطاء قيمة ابتدائية للمتغيرات ، وهي تشبه الحدث Form-Load الموجود في Form

وتستخدم تلك الدوال في تسجيل أي قيمة ابتدائية أو تعريف أي متغيرات أو أي شروط ابتدائية، فمن الضروري في OOP أن يكون في كل Class على الأقل أسلوب حتى نقوم بإنتاج object لهذا الـ Class ، والأساليب التي تقوم بعملية إنتاج Object للـ Class تسمى البناء Constructor ولا بد أن يكون اسمه هو نفس اسم الـ Class (حتى نستطيع استخدامه عند فحص الـ Class واستخدام الأساليب أو الصفات الموجودة فيه ، وفي كل Class في Java (حتى وان لم يضعه المبرمج) فإنها تحتوي على Constructor داخلي أو ما يسمى بـ Standard-Constructor

وبالتالي من أهم الأشياء التي يجب على المبرمج التفكير بها بعد إنشاء كلاس جديد هي تسهيل طريقة خلق كائنات من هذا الكلاس ، من هنا جاءت فكرة Constructor الذي هو عبارة عن دالة لها نوع خاص يتم استدعائها أثناء إنشاء كائن لتوليد قيم أولية للخصائص الموجودة فيه ، وحيث أنه لا يمكن إنشاء كائن من كلاس إلا من خلال Constructor سيقوم مترجم جافا بتوليد Constructor افتراضي فارغ إذا وجد أن الكلاس الذي تم تعريفه لا يحتوي على أي Constructor

مثال

بفرض أننا قمنا بتعريف كلاس اسمه Person ولم نقم بتعريف constructor له كما في الكلاس التالي:

```
class Person {
}
```

سيقوم المترجم بإنشاء constructor فارغ بشكل تلقائي كما يلي :

```
class Person {
public Person() {
}
}
```

ويتم تحديد دالة البناء Constructor داخل الفصيلة بإنشاء دالة بنفس اسم الفصيلة أي تصبح كما يلي:

```

1 Public class constructor{
2     Constructor( )
3     {
4         System.out.println("this is the constructor >>> it without calling");
5     }
6     public static void main(String[] args) {
7         Constructor obj=new constructor ( ) ;
8     }
9 }

```

ملاحظات حول constructor

- كل كلاس يتم إنشاؤه يحتوي على constructor واحد على الأقل حتى إن لم يتم تعريف أي constructor سيقوم المترجم بإنشاء واحد افتراضي.
- في كل مرة يتم إنشاء كائن جديد يجب استدعاء constructor حتى يتم إنشاء هذا الكائن.
- القاعدة الأساسية عند تعريف constructor هي أنه يجب أن يحمل نفس اسم الكلاس و يكون نوعه public.
- عند تعريف constructor لن يقوم المترجم بإنشاء واحد افتراضي.
- يمكن تعريف أكثر من constructor، ويمكن إنشاء constructor فارغ لاستخدامه في حاله إذا لم يرد المبرمج إعطاء قيم أولية محددة للخصائص عند إنشاء كائن.

أنواع المتغيرات التي يتم وضعها في الكلاس

- 1- Local Variables: هي المتغيرات التي يتم تعريفها بداخل أي دالة أو constructor أو بداخل block (مثل الحلقات، الجملة switch إلخ...)
- 2- Instance Variables: هي المتغيرات التي يتم تعريفها بداخل الكلاس وخارج حدود أي دالة أو constructor أو بداخل block (تسمى أيضاً Global Variables)
- 3- Class Variables: هي المتغيرات التي يتم تعريفها كـ static بداخل الكلاس وخارج حدود أي دالة أو constructor أو block

```

1 class VariablesTypes {
2     // block ( a, b, c, d ) تعتبر Instance Variables لأنه تم تعريفهم بداخل الكلاس و خارج أي دالة أو block
3     int a;
4     public int b;
5     protected int c;
6     private int d;
7     // المتغير e يعتبر Class Variable لأن نوعه static
8     static int e;
9     // المتغيرات ( x, y, z ) تعتبر Local Variables لأنه تم تعريفها بداخل الدالة
10    public int sum(int x, int y) {
11        int z = x + y;
12        return z;
13    }
14 }

```


٥- المحددات Modifiers

تمثل كلمات يمكن للمبرمج إضافتها عند تعريف أشياء جديدة (سواء كلاس أو متغير أو دالة إلخ..) لتحديد طريقة الوصول إليها

وغالبا يتم الاحتياج إليها عند العمل في برنامج كبير ضمن فريق من المبرمجين ولضمان عدم إساءة استخدام الأشياء التي قمت بتعريفها من قبل مبرمج آخر.

حيث ان استخدام Modifiers يساعد المبرمج في تحديد الأشياء التي يمكن لباقي المبرمجين الوصول إليها و الأشياء التي يريد التأكد من عدم التعديل عليها إلخ ، أي انها مجموعة من الكلمات المحجوزة Reserved Keywords التي تستخدم في تحديد درجة التوصل Access الى الفصيلة Class ودوالها ومتغيراتها وهذه الكلمات هي :

Static-public-protected, private, final, abstract, synchronized, volatile,...

ويمكن تقسيم المحددات Modifiers حسب الفئات التي تعمل معها كما يلي:

محددات Modifiers للسيطرة على الوصول Access Modifiers للدوال والمتغيرات والفصائل وهي : Public,protected,private

محددات Modifiers لإنهاء انشاء الفصائل والدوال والمتغيرات وهي : final

محددات Modifiers لإنهاء انشاء فصائل ودوال مجردة وهي : abstract

محددات Modifiers تستخدم مع عملية thread وهي : synchronized,volatile

وتنقسم Modifiers إلى نوعين : Access Modifiers ، Non Access Modifiers

مثال

- استخدام الكلمتين public ، private (تعتبران من Access Modifiers)
-استخدام الكلمتين final ، static (تعتبران من Non Access Modifiers)

1	public class Student {
2	private String firstName;
3	private String lastName;
4	private String specialization;
5	private int id;
6	private boolean isWork;
7	final String theAvgerageForSuccess = "50%";
8	static String CollegeName = "MIT";
9	public static void printFullName() {
10	System.out.println("Name: " +firstName+ " " +lastName);
11	}
12	}

محددات الوصول Access Modifiers

محددات الوصول عبارة عن Keyword توضع قبل الـ Method أو Object أو Constructor لتعريف مستوى الوصول access levels

Access modifiers used to help you to determine the level of access you want for classes as well as the variables, methods and constructors in your classes

ويوجد ٤ مستويات لـ Access Modifiers

- مرئية للـ package كلها وتعرف بـ default ولا تحتاج أي access modifier

- مرئية فقط للـ class وتعرف بـ private

- مرئية لكل البرنامج وتعرف بـ public

- مرئية للـ package & subclasses وتعرف بـ protected

1 - Default Access Modifier - No Keyword:

- Default Access Modifier تعني أنه لا يتم تعريف الـ access modifier صراحة لكل من class or method or etc.

- variable or method الذي يتم تعريفه من غير access control تكون متاحة لأي class في نفس الـ package

- لا يمكن استخدام default modifier في الـ method الموجودة داخل interface

Example:

Variables and methods can be declared without any modifiers, as in the following examples:

```
String version = "1.5.1";

boolean processOrder() {
    return true;
}
```

2- Private Access Modifier - private:

- methods , variables and constructors :الذي يتم تعريفه بـ private يتم استخدامه في الـ class نفسه فقط.

- class & interface :لا يمكن ان يكونوا private

- استخدام private modifier هي الطريقة التي تجعل object encapsulate itself ويخفي البيانات الموجودة فيه عن باقي البرنامج كله.

Example:

The following class uses private access control:

```
class Logger {
    private String format;
    public String getFormat() {
        return this.format;
    }
    public void setFormat(String format) {
        this.format = format;
    }
}
```

Here, the *format* variable of the *Logger* class is private, so there's no way for other classes to retrieve or set its value directly.

So to make this variable available to the outside world, we defined two public methods: *getFormat()*, which returns the value of *format*, and *setFormat(String)*, which sets its value.

3 - Public Access Modifier – public:

-عند تعريف أي من (class ,methods, variable and etc.) على أنه **public** فإنه يمكن استدعائه من أي **class** آخر.

-يمكن استدعاء **public class** في أي **package** أخرى عن طريق **import**

Example:

The following function uses public access control:

```
public static void main(String[] arguments) {
    // ...
}
```

The *main()* method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as *java*) to run the class.

4 - Protected Access Modifier - protected :

-أي **variable , method and constructor** يتم تعريفه على أنه **protected** في **superclass** يمكن استدعاؤه عن طريق ال **subclass** في **package** أخرى أو أي **class** آخر في نفس **package**

- لا يمكن استخدام **protected modifier** للـ **class** أو **interface** أو **methods** الموجودة داخل **interface**

Example:

The following parent class uses protected access control, to allow its child class override *openSpeaker()* method:

```
class AudioPlayer {
    protected boolean openSpeaker(Speaker sp) {
        // implementation details
    }
}

class StreamingAudioPlayer {
    boolean openSpeaker(Speaker sp) {
        // implementation details
    }
}
```

Here if we define *openSpeaker()* method as private then it would not be accessible from any other class other than *AudioPlayer*. If we define it as public then it would become accessible to all the outside world. But our intension is to expose this method to its subclass only, thats why we used *protected* modifier.

الجدول التالي يحتوي على الكلمات التي تنتمي للـ Access Modifiers

تعريفه	Modifiers
مستوى وصوله غير مقيد أي ان الكلاس أو الدالة أو المتغير الذي يتم تعريفه كـ public يمكن الوصول إليه مباشرة.	public
تمثل أعلى مستوى من حيث الحماية، المتغيرات و الدوال التي يتم تعريفها كـ private يمكن الوصول لها فقط من داخل الكلاس الذي تم تعريفها فيه ، ملاحظة: لا يمكن تعريف كلاس كـ private	private
الدالة أو المتغير الذي يتم تعريفه كـ protected يمكن الوصول إليه فقط من الكلاس الموجودة في نفس الـ package أو من الكلاس التي ترث منه ملاحظة: لا يمكنك تعريف كلاس كـ protected	protected
إذا لم تضع أي كلمة من الـ Access Modifiers عند تعريف كلاس أو دالة أو متغير سيتم وضع Modifier افتراضي يسمى package private وهذا يعني أنه يمكن الوصول إليه فقط من الكلاسات الموجودة في نفس الـ package	package private

كتابة الكود بشكل مثالي

- **Modifier** الافتراضي بشكل عام لا يستخدم في الغالب
- **public** : يستخدم مع الدوال التي تريد للجميع أن يصل إليها ويستخدم للمتغيرات التي لا تريد للكائنات والكلاسات التي ترث من الكلاس أن تصل إليها
- **protected** : يستخدم من أجل الكلاسات المرتبطة بالكلاس الذي تعمل عليه (فعلياً التي ترث منه) فمن خلاله ستكون البيانات متاحة أمام الكلاسات المرتبطة بالكلاس و لكنها غير متاحة أمام أي كلاس آخر.

خطوات التحكم بالكود وحمايته من المبرمجين الآخرين

- وضع **Modifier** ملائم لكل عنصر يتم تعريفه لحماية البيانات قدر المستطاع.
- المتغيرات التي تمثل الخصائص يجب أن لا تكون أبداً **public** ويجب وضعها **private** أو **protected** لتمنع الكلاسات الأخرى من الوصول المباشر إليهم.
- يجب تجهيز دوال نوعها **public** للتعامل مع هذه الخصائص ، والدوال التي نوعها **public** تسمح للمبرمجين (أو الكلاسات الأخرى) بالوصول إلى الخصائص ، هذه الدوال تسمح بإخفاء المتغيرات بالإضافة إلى التحكم بالخصائص.

القواعد التالية تم فرضها بالنسبة للدوال التي يرثها كلاس من كلاس آخر

- الدوال التي يتم تعريفها كـ **public** في الـ **Superclass** تعتبر **public** في جميع الـ **Subclasses**
- الدوال التي يتم تعريفها كـ **protected** في الـ **Superclass** تعتبر **protected** أو **public** في جميع الـ **Subclasses**
- الدوال التي يتم تعريفها كـ **private** لا يتم توريثها إلى أي كلاس ، لذلك لا يوجد قواعد من أجلهم.

مثال:

مطلوب اعداد شكل توضيحي يتضمن ما يلي:

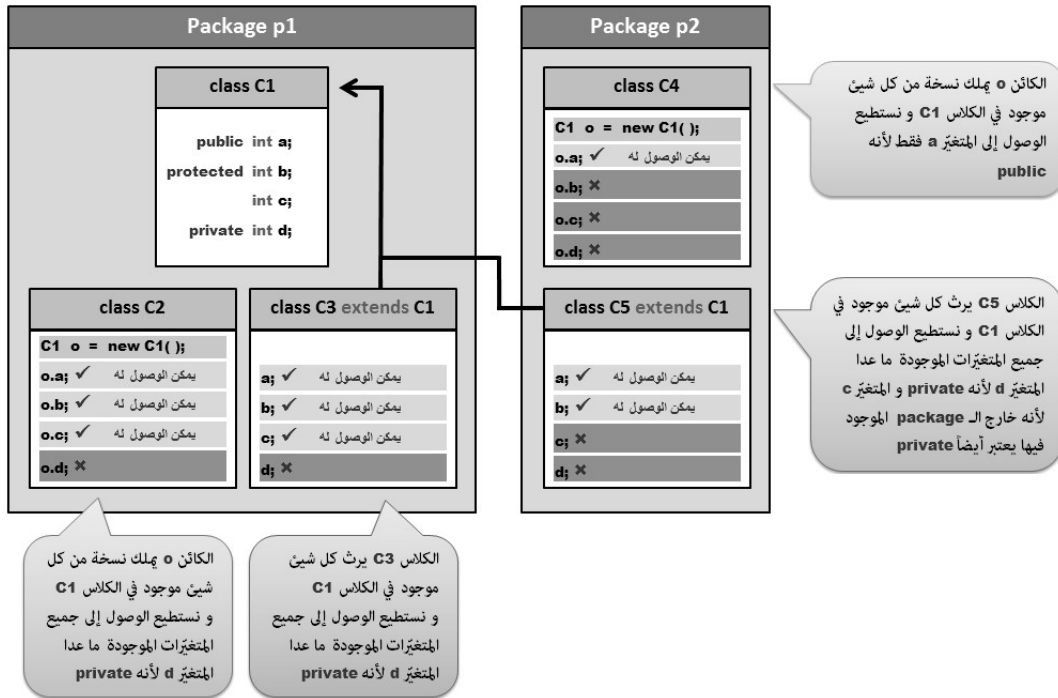
-إنشاء كلاس اسمه C1 يحتوي على المتغيرات (a, b, c, d)

-وضع Access Modifier لكل متغير.

-إنشاء كلاسات (C2 ، C3) في نفس الـ package والتي تحمل الاسم p1

-إنشاء كلاسات (C4 ، C5) في package أخرى بالاسم p2

- في كل كلاس نرغب في الوصول لجميع العناصر الموجودة في الكلاس C1



Non Access Modifiers

جميع الكلمات التي تنتمي إلى الـ Non Access Modifiers هي :

Static-final-abstract-synchronized-native-transient-volatile-strictfp

يتضمن الجدول التالي Non Access Modifiers الأكثر استخداما

تعريفه	Modifiers
يستخدم لتعريف كلاس أو متغير أو دالة مشتركة بين جميع الكائنات من كلاس معين	static
يستخدم لمنع الوراثة من الكلاس ، أو لمنع كتابة محتوى الدالة (أو تعديلها) في الكلاس الذي يرثها، أو لجعل قيمة المتغير غير قابلة للتغير بعد تحديدها.	final
يستخدم لإنشاء كلاس أو دالة مجردة (أي دالة لا تحتوي على كود) ، الهدف من هذا الـ Modifier تجهيز كلاس معين وجعل الكلاسات التي ترث من هذا الكلاس هي من تقوم بتعريف الأشياء الموجودة بداخله.	abstract

الكلمة static

حالات استخدام الكلمة static

- عند الرغبة في تعريف شيء ثابت لجميع الكائنات يتم تعريفه كـ static
- عند تعريف شيء بداخل كلاس معين ويراد الوصول إليه مباشرةً من الكلاس (بدلا من إنشاء كائن من الكلاس ثم استدعاء الشيء منه) يتم تعريفه كـ static

المتغيرات التي يتم تعريفها كـ static

- المتغير الذي يتم تعريفه كـ static يعتبر مشترك بين جميع الكائنات من نفس الكلاس ، بمعنى أن كل كائن يتم إنشاؤه من نفس الكلاس سوف يملك نفس هذا المتغير.
- المتغير سيتم تعريفه مرة واحدة في الذاكرة و جميع الكائنات من نفس الكلاس ستشير إليه بدلا من أن تملك نسخة خاصة منه ، أي أن static تعني نسخة واحدة من المتغير لجميع الكائنات.
- المتغير الذي يتم تعريفه كـ static يسمى أيضاً Class Variable ولا يمكن تعريف الـ Local Variables كـ static
- يمكن الوصول للمتغير الذي تم تعريفه كـ static بذكر اسم الكلاس الذي تم تعريفه فيه ثم وضع اسمه ، أو من أي كائن من الكلاس.

الدوال التي يتم تعريفها كـ static

- الدالة دائماً يتم تعريفها مرة واحدة في الذاكرة وجميع الكائنات من نفس الكلاس ستشير إليها، لكن الكلمة static تمكنك من الوصول إليها مباشرةً من الكلاس دون الحاجة لخلق كائن واستدعائها من خلاله.
- الدالة التي نوعها static يمكنها الوصول للمتغيرات المعرفة في الكلاس بشرط أن تكون هذه المتغيرات أيضاً static لكن بشكل عام الدوال التي نوعها static لا تستخدم المتغيرات الموجودة في الكلاس بل تستخدم المتغيرات التي يتم تعريفها كبارامترات لها أو المتغيرات التي يتم تعريفها بداخلها.
- يمكن الوصول للدالة التي تم تعريفها كـ static بذكر اسم الكلاس الذي تم تعريفها فيه ثم وضع اسمه، أو من أي كائن من الكلاس.

مثال حول الكلمة static

- تعريف كلاس اسمه Example يحتوي على الأشياء التالية:
- متغير اسمه a معرف كـ static دالة اسمها print تعرض قيمة المتغير a
 - دالة اسمها staticPrint تعرض أيضاً قيمة المتغير a لكنها معرفة كـ static بعدها تغيير وعرض قيمة المتغير a بعدة طرق.

Example.java	
1	public class Example {
2	
3	// يمكننا الوصول إليه من خلال كائن أو من أي مكان مباشرةً بهذا الشكل Example.a
4	تعريف المتغير a كـ static.
5	public static int a;
6	// هذه الدالة لا يمكن استدعائها إلا من خلال كائن من الكلاس Example
7	public void print() {
8	System.out.println("a: " +a);
9	}
10	// هذه الدالة يمكن استدعائها مباشرةً من أي مكان بهذا الشكل Example.staticPrint();
11	public static void staticPrint() {
12	System.out.println("a: " +a);
13	}
	}

Main.java	
1	public class Main {
2	public static void main(String[] args) {
3	<u>// إنشاء كائنين e1 و e2 من الكلاس Example</u>
4	Example e1 = new Example();
5	Example e2 = new Example();
6	<u>// إعطاء قيمة لـ a مباشرةً من الكائن الكلاس Example</u>
7	Example.a = 10;
8	<u>// عرض قيمة a من خلال الدالة staticPrint() التي يمكننا استدعائها مباشرةً من الكلاس لأن نوعها static</u>
9	Example.staticPrint();
10	<u>// عرض قيمة a من خلال الدالة staticPrint() التي يمكننا استدعائها من الكائن e1 أيضاً</u>
11	e1.staticPrint();
12	<u>// عرض قيمة a من خلال الدالة staticPrint() التي يمكننا استدعائها من الكائن e2 أيضاً</u>
13	e2.staticPrint();
14	<u>// تغيير قيمة a من خلال الكائن e1</u>
15	e1.a = 22;
16	<u>// العرض من خلال الدالة print() التي وصلنا إليها من خلال الكائن e1</u>
17	e1.print();
18	<u>// تغيير قيمة a من خلال الكائن e2</u>
19	e2.a = 75;
20	<u>// العرض من خلال الدالة print() التي وصلنا إليها من خلال الكائن e2</u>
21	e2.print();
22	}
23	}

ناتج التنفيذ كما يلي:

```
a: 10
a: 10
a: 10
a: 22
a: 75
```

ملحوظة:

لا يمكن وضع `this` عند استدعاء متغير نوعه `static` فهنا مثلاً لا يمكنك أن تكتب `this.a` بدل `a` لأن الكلمة `this` تستخدم للإشارة إلى كائن محدد على عكس مبدأ الـ `static`

الكلمة final

حالات استخدام الكلمة final

- عند إنشاء متغير يمكن تحديد قيمته مرة واحدة فقط.
- عند إنشاء دالة لا يمكن تعريفها من جديد في الكلاس الذي يرثها (أي لمنع الـ `override`)
- عند إنشاء كلاس لا يمكن الوراثة منه.

المتغيرات التي يتم تعريفها كـ final

- المتغير الذي يتم تعريفه كـ `final` يعني أنه بمجرد إعطائه قيمة لا يمكن تغييرها من جديد.
- عند إنشاء متغير نوعه `final` يجب تحديد قيمته مرة واحدة فقط إما عند تعريفه أو في `constructor`

المتغيرات التي يتم تعريفها كـ final static

- يمكن تعريف المتغير كـ `final`، `static` مع بعض ، وعندها يمكن الوصول للمتغير من الكلاس مباشرة أو من أي كائن من الكلاس مع عدم إمكانية تغيير قيمته بعد تحديدها.
- الـ `Math.PI` ، الـ `Math.E` هم من المتغيرات المعرفة كـ `final static` في الجافا ويمكن استخدامهم كما هم لكن لا يمكن تغيير قيمهم.

الدوال التي يتم تعريفها كـ final

- الدالة التي يتم تعريفها كـ `final` يعني أنه لا يمكن أن يتم تعريف محتواها في أي كلاس آخر ، أي الكلاس الذي يرثها لا يسمح له بأن يفعل لها `override`

الكلاسات التي يتم تعريفها كـ final

- الكلاس الذي يتم تعريفه كـ `final` يعني أنه لا يمكن الوراثة منه ، فمثلاً تم تعريف الكلاس `Math` في الجافا كـ `final static` حتى يكون متاح للاستخدام من أي مكان ، مع عدم القدرة على تعديل الأشياء التي تم تعريفها بداخله.

في المثال التالي سنقوم بتعريف 3 متغيرات نوعها `final` ، الهدف هنا معرفة الطرق المسموح فيها إعطاء قيمة للمتغير المعروف كـ `final`

Example.java	
1	<code>public class Example {</code>
2	<code>// تعريف متغير نوعه final و إعطاؤه قيمة مباشرة عند تعريفه</code>
3	<code>public final int a = 10;</code>
4	<code>// تعريف متغير نوعه final بدون تحديد قيمته</code>
5	<code>public final int b;</code>
6	<code>// تعريف متغير نوعه final بدون تحديد قيمته</code>
7	<code>public final int c;</code>
8	<code>public Example(int b) {</code>
9	<code>// تحديد قيمة المتغير b من الكائن، أي أن الكائن هو من سيقوم بتحديد لها</code>
10	<code>this.b = b;</code>
11	<code>// تحديد قيمة المتغير c مباشرة عند إنشاء كائن أي أن الكائن سيمثلها هكذا</code>
12	<code>c = 50;</code>
13	<code>}</code>
14	<code>}</code>

مثال:

إنشاء ٢ كلاس : الكلاس A ، الكلاس B الذي سيرث منه.
 -الكلاس A: سنقوم بتعريف دالة عادية و دالة نوعها `final`
 - الكلاس B : سنعمل `override` للدالة التي ليس نوعها `final`
 -إنشاء الكلاس Main لتجربة الكود.

الهدف هنا معرفة أن الدوال المعرفة كـ `final` لا يمكن تعريفها من جديد في الكلاس الذي يرثها (الكلمة `final` تمنع الـ `override`)

A.java	
1	<code>public class A {</code>
2	<code>// final أي كلاس سيرث من هذا الكلاس لا يمكنه أن يفعل Override لهذه الدالة لأنها معرفة كـ</code>
3	<code>public final void firstPrint() {</code>
4	<code>System.out.println("welcome to java");</code>
5	<code>}</code>
6	<code>// final أي كلاس سيرث من هذا الكلاس، يمكنه أن يفعل Override لهذه الدالة لأنها غير معرفة كـ</code>
7	<code>public void secondPrint() {</code>
8	<code>System.out.println("welcome to java");</code>
9	<code>}</code>
10	<code>}</code>

B.java	
1	// الكلاس B يرث من الكلاس A. أي يمكن للكلاس B استخدام الأشياء الموجودة في A و كأنها موجودة فيه تماماً
2	public class B extends A {
3	// إعادة كتابة محتوى الدالة secondPrint بالنسبة للكلاس B
4	@Override
5	public void secondPrint() {
6	System.out.println("class B override my content!");
7	}
8	}

Main.java	
1	public class Main {
2	public static void main(String[] args) {
3	// إنشاء كائن من الكلاس A اسمه a
4	A a = new A();
5	// إنشاء كائن من الكلاس B اسمه b
6	B b = new B();
7	// تنفيذ الدالة كما تم تعريفها في الكلاس A
8	a.firstPrint();
9	// تنفيذ الدالة كما تم تعريفها في الكلاس A
10	a.secondPrint();
11	// تنفيذ الدالة كما تم تعريفها في الكلاس A
12	b.firstPrint();
13	// تنفيذ الدالة كما تم تعريفها في الكلاس B
14	b.secondPrint();
15	}
16	}

ناتج التنفيذ كما يلي:

```
welcome to java
welcome to java
welcome to java
class B override my content
```

مثال

إنشاء كلاس نوعه **final** لجعل الكود غير قابل لأي تعديل خارجي ، ثم إنشاء الكلاس **Main** لتجربة الكود ، الهدف معرفة أن الكلاس المعرف كـ **final** يمكن إنشاء كائنات منه لكن لا يمكن الوراثة منه.

FatherOfJava.java	
1	// تعريف الكلاس كـ final لمنع الوراثة منه فقط
2	public final class FatherOfJava {
3	public String name = "James Arthur Gosling";
4	public String born = "May 19, 1955";
5	public void story() {
6	System.out.println(name+ " , born on " +born+ " . He is a Canadian computer scientist," + " best known as the father of the Java programming language");
7	}
8	
9	}

Main.java	
1	public class Main {
2	public static void main(String[] args) {
3	// إنشاء كائن من الكلاس FatherOfJava اسمه obj
4	FatherOfJava obj = new FatherOfJava();
5	// عرض قيمة المتغير name كما تم تعريفها في الكلاس الأساسي
6	System.out.println("Name of the father of java: "
7	+obj.name);
8	// تنفيذ الدالة كما تم تعريفها في الكلاس FatherOfJava
9	obj.story();
10	}
	}

ناتج التنفيذ كما يلي:

Name of the father of java: James Arthur Gosling

James Arthur Gosling, born on May 19, 1955. He is a Canadian computer scientist, best known as the father of the Java programming language

٦-دوال الهدم Destruction

هي دالة أو دوال تنفذ تلقائياً عند الانتهاء من استخدام الفصيلة Class ، وهي تشبه الحدث Form-unload الموجود في الـ Form وتستخدم للإنتهاء أو للتخلص من تعريف متغيرات أو إجراء أي عمليات قبل الخروج من البرنامج

٧-خاصية التوريث Inheritance

التوريث من مميزات لغات OOP ويعني توريث خواص وسلوك كائن لكائن آخر مماثل ، فمثلا عندما تبدأ في إنشاء كائن لكي تستعمله في برنامج ما تجد أن هذا الكائن الجديد يشبه كائن آخر قد صممته مسبقا

فالتوريث يعني توريث فصيلة قديمة Base Class موجودة بالفعل لفصيلة Class جديدة عند إنشائها بحيث يضاف تركيب الفصيلة القديمة Base Class من بيانات ودوال إلى الفصيلة الجديدة new class ثم نكمل عليها في الفصيلة الجديدة

وهذه الخاصية من أهم خصائص مفهوم OOP فعلى أساسه تبني مكتبات الفصائل Class Lib حيث يتم بناء فصيلة أساس Base Class ثم ترثها الفصيلة الثانية والثالثة وتأخذ الفصائل من بعضها البعض حتى تتكون مكتبة فصائل عبارة عن شجرة فصائل مثل مكتبة فصائل لغة Java المعروفة بالاسم JFC

كما تستخدم خاصية التوريث لتقليل إعادة كتابة الأوامر، وباستخدام خاصية التوريث نستطيع إنشاء فصيلة class تحتوي على خصائص properties ودوال method ثم استخدامها كأساس لفصائل أخرى

في الجافا يمكن للكلاس أن يرث من كلاس آخر وبالتالي يحصل على الدوال والمتغيرات الموجودة في هذا الكلاس، فمثلاً عند الرغبة في إنشاء كلاس جديد ولوحظ أنه يوجد كلاس جاهز يحتوي على أكواد قد تفيد فإنه يمكن استغلالها بدل من إعادة كتابتها من الصفر أي يمكن جعل الكلاس الذي قمت بتعريفه يرث هذا الكلاس وبعدها يمكن للمبرمج استخدام جميع المتغيرات والدوال التي ورثها الكلاس الجديد من الكلاس الجاهز.

الكلاس الذي يرث من كلاس آخر يسمى Subclass أو child class ، والكلاس الذي يورث محتوياته لكلاس آخر يسمى Superclass أو parent class

مثال

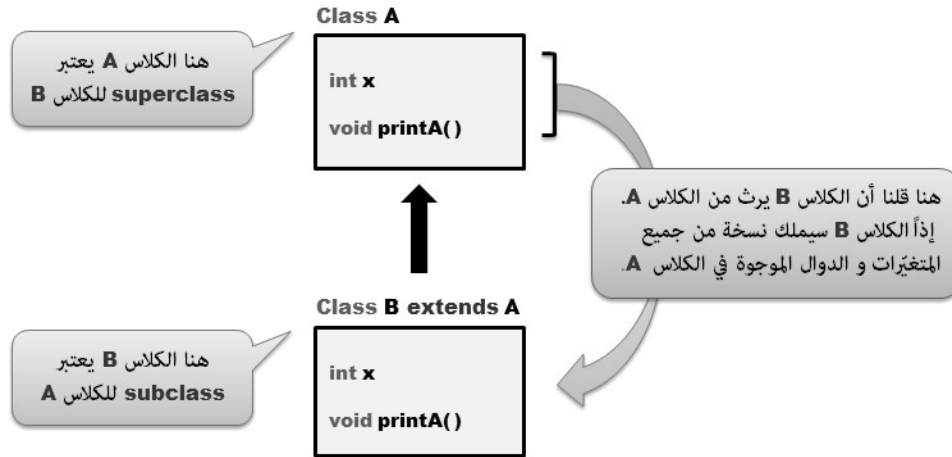
بفرض أننا نريد تعريف فصيلتين الأولى لتعريف بيانات الطالب والثانية لتعريف بيانات المعلم فبدون استخدام خاصية التوريث سنضطر لإنشاء الفصيلتين برغم من تشابه الفصيلتان واحتوائهما على بيانات متشابهة فمثلا فصيلة الطالب تحتوي على البيانات (الكود-الاسم-العنوان-التليفون-المؤهل-تاريخ الميلاد- بيانات أخرى.)، وتحتوي على الدوال التالية: دالة تسجيل بيانات الطالب، دالة حذف بيانات الطالب، دالة تعديل بيانات الطالب (ودوال أخرى)

وعند النظر إلى فصيلة المعلم سوف نجد أنها تشتمل على كثير من البيانات والدوال الأعضاء في فصيلة الطالب بالإضافة لبعض الأعضاء الجديدة وبالتالي باستخدام خاصية التوريث علينا توريث فصيلة الطالب لفصيلة المعلم وإضافة الجديد إليها مما يوفر علينا إعادة كل شيء ويمكن تكرار هذه الفكرة مع فصائل جديدة أخرى مما يوفر الكثير من الوقت بالإضافة أن هذا الأسلوب يسمح باستخدام فصائل classes تم إعدادها وتم اختبارها

والتوريث في لغة java يتم إنشاء فئات كاملة بكل دوالها ثم توريث هذه الفئات بكل دوالها ومتغيراتها لفئات جديدة وذلك يوفر علينا الكثير وهذا يعني أنه يمكن إنشاء form جديدة بناء على form قديمة لأن form في حد ذاتها فصلة

مثال

بفرض أننا قمنا بتعريف كلاس A يحتوي على متغير x و دالة printA() ، ثم قمنا بإنشاء كلاس جديد فارغ B واعتبرنا أنه يرث من الكلاس A ، وبالتالي فإن الكلاس B أصبح يملك نسخة من جميع المتغيرات والدوال الموجودة في الكلاس A



الكلمة extends

تستخدم لجعل الكلاس يرث من كلاس آخر ، ويتم وضع الكلمة extends بعد اسم الكلاس ثم نضع بعدها اسم الكلاس الذي نريد الوراثة منه.
الكود التالي يعني أن الكلاس B يرث من الكلاس A

```

class A {
}
class B extends A {
}
    
```

الكلمة super

تستخدم للحالات التالية:

-التمييز بين المتغيرات والدوال الموجودة في Superclass ، Subclass إذا كانت الأسماء مستخدمة في كلا الكلاسين.

-استدعاء constructor الموجود في الـ Superclass

طريقة استخدام الكلمة super لاستدعاء متغير من الـ Superclass من خلال وضع الكلمة super ثم نقطة ثم نضع اسم المتغير المطلوب استدعائه من Superclass

`super.variableName`

طريقة استخدام الكلمة `super` لاستدعاء دالة من الـ `Superclass` من خلال وضع الكلمة `super` ثم نقطة ثم نضع اسم الدالة المطلوب استدعائها من `Superclass`

`super.methodName();`

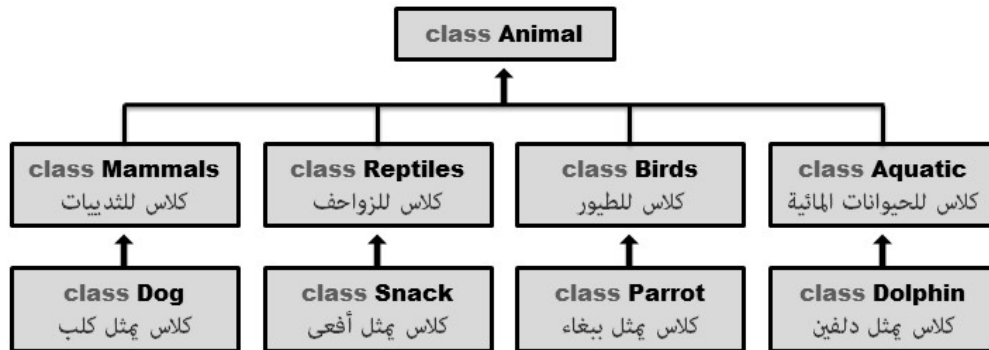
أشكال الوراثة

يوجد ٣ أشكال أساسية للوراثة وهي:

الكود	شكلها	إسمها
<pre>class A { } class B extends A { }</pre>	<pre> graph BT B[class B] --> A[class A] </pre>	Single inheritance وراثة فردية
<pre>class A { } class B extends A { } class C extends B { }</pre>	<pre> graph BT C[class C] --> B[class B] B --> A[class A] </pre>	Multi Level inheritance وراثة متتالية
<pre>class A { } class B extends A { } class C extends A { }</pre>	<pre> graph BT B[class B] --> A[class A] C[class C] --> A </pre>	Hierarchical inheritance وراثة هرمية

علاقة الكائنات مع بعضها

بفرض إنشاء برنامج لحفظ معلومات عن الحيوانات، يجب في البداية تجهيز `class` أساسي يمثل جميع الخصائص المشتركة بين الحيوانات، ثم تقسيم الحيوانات إلى ٤ فئات أساسية (ثدييات- زواحف- طيور- حيوانات مائية)، ثم تصنيف كل حيوان ضمن الفئة التي ينتمي لها.



تمثيل المطلوب بلغة الجافا

```
// تعريف الكلاس الأساسي لجميع الحيوانات
class Animal { }
// تعريف فئات الحيوانات
class Mammals extends Animal { }
class Reptiles extends Animal { }
class Birds extends Animal { }
class Aquatic extends Animal { }

// تعريف ٤ حيوانات مع وضع كل حيوان ضمن فئته
class Dog extends Mammals { }
class Snake extends Reptiles { }
class Parrot extends Birds { }
class Dolphin extends Aquatic { }
```

يتم قراءة الكود السابق كما يلي:

Dog يعتبر من Mammals ويعتبر Animal أيضاً ، Snake يعتبر من Reptiles ويعتبر Animal أيضاً ، Parrot يعتبر من Birds ويعتبر Animal أيضاً ، Dolphin يعتبر من Aquatic ويعتبر Animal أيضاً.

مثال :-

نكتب سطور تعريف فصيلة class بالاسم person كما يلي :

```
1 class Person
2 {
3     public String Pname, Paddress;
4     public void Set_Data(String Name_V,String address_V)
5     {
6         Pname=Name_V;
7         Paddress=address_V;
8     }
9
10    public String Get_Name()
11    {
12        Return Pname;
13    }
14    public String Get_Address()
15    {
16        Return Paddress;
17    }
18 }
19 Public class InhertApp1 {
20     public static void main(String[] args)
21     {
22     }
23 }
```


حيث تم ما يلي:

- ١- إنشاء فصيلة بالاسم person وتعريف متغيرات
- ٢- كتابة فصيلة متكاملة تحتوى على المتغيرات Pname ، Paddress وتحتوى على الدالة SetData() التي تقوم بتسجيل بيانات الفصيلة ، كما تحتوى على الدالة getName() التي تعيد قيمة المتغير pname الخاص بالفصيلة والخاص باسم الشخص ، وكذلك الدالة getAddress() التي تعيد قيمة المتغير paddress
- ٣- عند إنشاء فصيلة جديدة وتوريثها هذه الفصيلة person يتم استخدام متغيراتها ودوالها مباشرة دون الحاجة إلى إنشاؤها مرة أخرى ، ويتم توريث هذه الفصيلة إلى فصيلة جديدة كما يلي :

نكتب سطور فصيلة جديدة بالاسم student كما يلي :

1	class Student extends Person
2	{
3	Int degree;
4	public void Set_Degree(int degree_V)
5	{
6	Degree=degree_V;
7	}
8	Public int RetDegree()
9	{
10	Return degree;
11	}
12	}
13	}

حيث تم ما يلي:

- ١- إنشاء فصيلة بالاسم student

- ٢- استخدام العبارة extend person ومعناها قم بتوريث الفصيلة person بكل ما فيها للفصيلة الجديدة student وبالتالي أصبحت نفس أعضاء الفصيلة القديمة person أعضاء في الفصيلة الجديدة student وهذا يسمح باستخدامها مباشرة كما يلي:

نكتب في الدالة الرئيسية (main) سطور استخدام كل من الفصيلتين person ، student كما يلي :

```

1 public class InherApp1
2
3 public static void main(String[] args)
4 {
5     String vname,vaddress;
6     Person no=new Person( );
7     no.Set_Data("elsaeed","dameitta");
8     Vname=no.Get_Name( );
9     Vaddress=no.Get_Address( );
10    System.out.println(vname);
11    System.out.println(Vaddress)
12
13    Student st=new Student( );
14    St.Set_Degree(100);
15    Int vd;
16    String vaddress2;
17    Vd=st.RetDegree( );
18    St.set_Data("ali","cairo ");
19    Vname=st.Get_Name( );
20    vaddress =st.Get_Address( );
21
22    System.out.println("vd:" + vd);
23    System.out.println("vname:" + vname);
24    System.out.println("vaddress:" + vaddress);
25 }
26
27 }
```

حيث أن :

- السطر ٥ : تعريف المتغيرات vname ،vaddress
- السطر ٦ : تعريف المتغير no (هدف) في الفصيلة person
- السطور ٧،٨،٩ : التعامل مع هدف الفصيلة person باستدعاء الدوال والتعامل معها
- السطر ١٣ : تعريف المتغير st من الفصيلة student
- السطر ١٤ : استدعاء الدالة الجديدة المضافة للفصيلة بالاسم set_Degree
- السطرين ١٥ ، ١٦ : الإعلان عن متغيرات
- السطر ١٨ : استدعاء الدالة Set_Data() المعرفة داخل الفصيلة الأساسية person فبالرغم من عدم وجود الدالة داخل الفصيلة الجديدة إلا انه تم استخدامها لأنها معرفة في الفصيلة التي تم ارثها باستخدام الأمر Inherits
- السطرين ٢٠ ، ٢١ : استدعاء دوال الفصيلة الأساسية فبالرغم من عدم وجود هذه الدوال داخل الفصيلة إلا أننا استخدمناها لأنها معرفة في الفصيلة التي تم ارثها.
- نلاحظ مما سبق انه تم استخدام الخصائص التي تم ارثها من الفصيلة person مع الفصيلة student بالرغم من عدم تعريفها أو الإعلان عنها، والجدول التالي يوضح الكود كاملا :

```

1  class Person
2  {
3  public String Pname, Paddress;
4  public void Set_Data(String Name_V,String address_V)
5  {
6      Pname=Name_V;
7      Paddress=address_V;
8  }
9
10 public String Get_Name()
11 {
12     Return Pname;
13 }
14 public String Get_Address()
15 {
16     Return Paddress;
17 }
18 }
19 class Student extends Person
20 {
21     Int degree;
22 public void Set_Degree(int degree_V)
23 {
24     Degree=degree_V;
25 }
26 }
27     Public int RetDegree()
28 {
29     Return degree;
30 }
31 }
32 }
33 public class InhertApp1
34
35 public static void main(String[] args)
36 {
37     String vname,vaddress;
38     Person no=new Person();
39     no.Set_Data("elsaeed","dameitta");

```

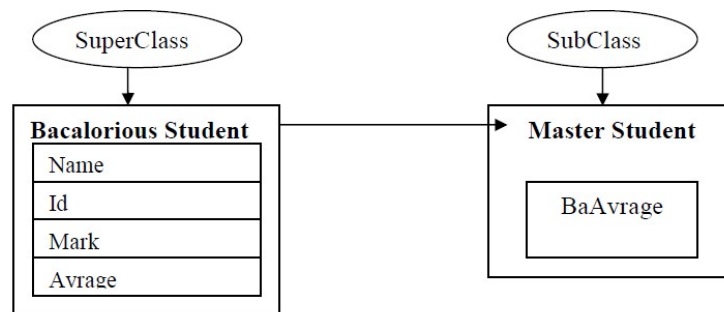
```

40 Vname=no.Get_Name( );
41 Vaddress=no.Get_Address( );
42 System.out.println(vname);
43 System.out.println(Vaddress)
44
45 Student st=new Student( );
46 St.Set_Degree(100);
47 Int vd;
48 String vaddress2;
49 Vd=st.RetDegree( );
50 St.set_Data("ali","cairo ");
51 Vname=st.Get_Name( );
52 vaddress =st.Get_Address( );
53 System.out.println("vd:" + vd);
54 System.out.println("vname:" + vname);
55 System.out.println("vaddress:" + vaddress);
56 }
58 }

```

مثال

أن التوريث يعني إنشاء class أو object فرعي من class آخر فمثلا لدينا طلاب بكالوريوس وطلاب دراسات عليا والاثنين في النهاية هم طلاب ولكن طلاب الماجستير لديهم بعض الخصائص والمؤهلات التي تميزهم عن طلاب الدراسات العليا لذلك يمكن عمل Object فرعي لطلاب الماجستير من object طلاب البكالوريوس حيث يسمى الكلاس الفرعي Sub Class والكلاس الأصلي يسمى Super Class



نلاحظ من الشكل السابق أن كل الطلاب لهم اسم ورقم جلوس وتخصص ومتوسط درجات تراكمي ولكن طلاب الماجستير لهم معدل إضافي لذلك أنشأنا لهم Sub class من الـ super class وهو الطلاب ، هذا وتستخدم الكلمة Extends لعمل Sub class كما يلي :

```
Class MasterStudent extends Student {
```

```
}
```

اسم الكلاس الرئيسي

إنشاء الكلاس الرئيسي Super Class

```
1 class Student{
2     private String name;
3     private double id;
4     private int mark;
5     /**Default Constructor of Student Object*/
6     public Student(){
7     }
8     /**Constructor of Student Object with parameter */
9     public Student(String aName,double aId,int aMark){
10    name = aName;
11    id = aId;
12    mark = aMark;
13    }
14    /**accessor to name*/
15    public String getName(){
16    return name;
17    }
18    /**mutator to name*/
19    public void setName(String aName){
20    name = aName;
21    }
22    /**accessor to Id*/
23    public double getId(){
24    return id
25    }
26    /**mutator to Id*/
27    public void setId(double aId){
28    id = aId;
29    }
30    /**accessor to Mark*/
31    public int getMark(){
32    return mark;
33    }
34    /**mutator to Mark */
35    public void setMark(int aMark){
36    mark = aMark;
37    }
38    }
```

-إنشاء الكلاس الفرعي sub Class

```

1 class MasterStudent extends Student{
2     double BaAvrage;
3     public MasterStudent(){
4     }
5     public MasterStudent(String aName,double aId,int aMark , double aBaAvr){
6         super(aName,aId,aMark);
7         BaAvrage = aBAvr;
8     }
9     public void setBaAvrage(double average){
10        BaAvrage = average;
11    }
12    public double getBaAvrage(){
13        return BaAvrage;
14    }
15    public String toString(){
16        return "Master Student :\n"+super.toString()+"BaAvrage= : "+ getBaAvrage;
17    }
18 }
19 class StudentTest{
20     public static void main(String[] args){
21         Student abdallah = new Student("abdallah ",20111017,80);
22         MasterStudent amjad = new MasterStudent ("amjad", 9711123 ,84,78);
23         System.out.print(abdallah.toString()+"\n"+amjad.toString());
24     }
25 }

```

ملحوظة:

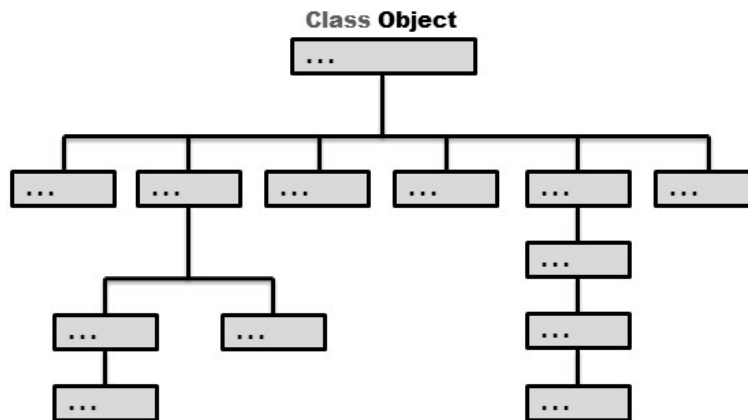
الدالة super تدل على ان الكلاس رئيسي أما الدالة this تدل على الكلاس التي هي موجودة به، ويمكن استخدامهم بطريقتين:

- super (المعاملات)

-super . method اسم

الكلاس Object

يوجد في الجافا كثير من الكلاسات الجاهزة التي تم بناءها بشكل هرمي كما يلي:



جميع الكلاسات في الجافا ترث بشكل تلقائي من الكلاس **Object** لأنه يأتي في رأس الهرم ، فالكلاس **Object** أعلى **Superclass** في الجافا ، ولقد لاحظنا مسبقا أن أي كلاس جديد كنا نستخدمه يحتوي على الدوال **equals()**, **hashCode()**, **toString()**.. ، سبب وجود هذه الدوال في كل كلاس يتم استخدامه أنه تم وراثتهم من الكلاس **Object**

٨- الخاصية Overloading

تعني إمكانية إنشاء أكثر من دالة **Method** بنفس الاسم مع تغيير عدد المعاملات **Parameters** وهذا يفيد بإنشاء أكثر من دالة لنفس الوظيفة بمعاملات مختلفة بنفس الاسم مثل إنشاء دالتين بالاسم بنفس الاسم إحداهما تأخذ معامل رقمي هو كود الموظف والثانية تأخذ عبارة حرفية وهي اسم الموظف وفي الحالتين تبحث الدالة وبالتالي يشعر المبرمج كأنها دالة واحدة ولكنها دالتين يتم استدعاء كل واحدة تلقائيا حسب المعامل المرسل لها

وبالتالي فإن مصطلح **Overloading** يعني تعريف أكثر من دالة أو **constructor** لهم نفس الاسم ولكنهم مختلفون في عدد أو نوع البارامترات ، تتمثل الفكرة من **Overloading** في تجهيز عدة دوال لهم نفس الاسم ، هذه الدوال تكون متشابهة من حيث الوظيفة ومختلفة قليلاً في الأداء ، فعلياً تحتوي كل دالة على ميزات إضافية عن الدالة التي أنشأت قبلها.

وتتمثل شروط **Overloading** فيما يلي : تطبق فقط على الدوال و **Constructors** ، يجب أن يملكوا نفس الاسم ، يجب أن يختلفوا في نوع أو عدد البارامترات

المثال الأول

تعريف دوال لها نفس الاسم وتختلف في نوع الباراميترات.
 بفرض تعريف كلاس اسمه **MyMethods** يحتوي على ٣ دوال **sum()** ، و نوعهم **public void**
 الدالة الأولى : الهدف منها جمع أي عددين ، نوعهما **int** ، ثم طباعة قيمتهم.
 الدالة الثانية: الهدف منها جمع أي عددين ، نوعهما **float** ، ثم طباعة قيمتهم.
 الدالة الثالثة : الهدف منها جمع أي عددين نوعهما **double** ، ثم طباعة قيمتهم.
 بعد إنشاء هذا الكلاس نقوم بإنشاء الكلاس **Main** واستدعاء الدوال الثلاثة فيه.

MyMethods.java	
1	public class MyMethods {
2	public void sum(int a, int b) {
3	System.out.println("First method is called ==> "+a+" + "+b+" = "+(a+b));
4	}
5	public void sum(float a, float b) {
6	System.out.println("Second method is called ==> "+a+" + "+b+" = "+(a+b));
7	}
8	public void sum(double a, double b) {
9	System.out.println("Third method is called ==> "+a+" + "+b+" = "+(a+b));
10	}
11	}

Main.java	
1	public class Main {
2	public static void main(String[] args) {
3	MyMethods m = new MyMethods(); // إنشاء كائن من الكلاس MyMethods لاستدعاء الدوال منه
4	m.sum(1000, 4000); // int باراميتر نوعهم int تأخذ ٢ استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم int
5	m.sum(10f, 40f); // float باراميتر نوعهم float تأخذ ٢ استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم float
6	m.sum(10d, 40d); // double باراميتر نوعهم double تأخذ ٢ استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم double
7	System.out.println();
8	m.sum(2000, -100); // int باراميتر نوعهم int تأخذ ٢ استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم int
9	m.sum(5.5, -3.3); // double لأن الأرقام تحتوي على فاصلة استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم double
10	m.sum(5.5d, -3.3d); // double باراميتر نوعهم double تأخذ ٢ استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم double
11	m.sum(5.5f, -3.3f); // float لأننا وضعنا f بعد كل رقم استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم float
12	System.out.println();
13	m.sum(6, 5.25); // double لأن النوع double يقبل كلا النوعين استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم double
14	m.sum(6, 5.25f); // float لأن النوع float يقبل كلا النوعين استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم float
15	m.sum(6, 5.25d); // double لأن النوع double يقبل كلا النوعين استدعاء الدالة التي تأخذ ٢ باراميتر نوعهم double
16	}
17	}

التركيب العام للفصيلة

```
class ClassName
{
    Constructor1
    Constructor2
    .....
    method1
    method2
    .....
    field1
    field2
    .....
}
```

حيث يتم تعريف الفصيلة باستخدام كلمة class ثم اسم الفصيلة ثم الاقواس ثم المحتويات

إنشاء واستخدام الفصائل Classes

نقوم بإنشاء تطبيق جديد ثم نكتب سطر الكود التالي :

1	public class FirstClass{
2	{ public static void main(String[] args
3	System.out.println("this the main class");
4	}
5	}

يتكون الكود السابق من فصيلة Class واحدة وهى الفصيلة الرئيسية وبالتالي لابد أن يحتوى برنامج Java على فصيلة واحدة على الأقل وبها الدالة (main) التي يبدأ منها تنفيذ البرنامج حيث أن:
السطر ١: تعريف الفصيلة Class بالاسم FirstClass واستخدام الكلمة المحجوزة class
السطر ٢: الدالة الرئيسية داخل الفصيلة

الكلمة Public

عند وضع الكلمة Public قبل الـ method أو الـ Field أو الـ Constructor نستطيع استخدامهم في أى Class آخر وفى هذه الحالة يجب كتابة الـ Documentation لهذه الأشياء وذلك يجعل أمر تحديثها أمر صعب لأن ممكن أي تحديث لها يغير مفهومها وبذلك يصبح التعليق الذي كتب عنها خطأ فلا يعرف المستخدم ماذا تفعل تلك الأشياء بالضبط

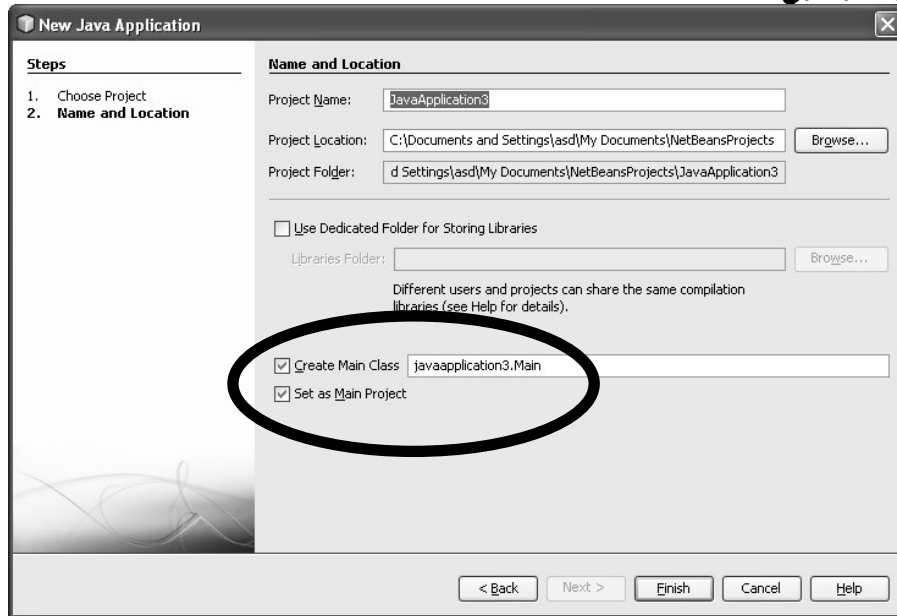
الكلمة Private

إذا كانت الأشياء السابقة (Method ، Field ،) لا نستطيع استخدامها إلا من خلال الـ Class الذي عرفت فيه ولا نحتاج إلى كتابة تعليق وبذلك يكون أمر تحديثها سهل ولذلك دائما نستخدم فى Object الكلمة Private ونستخدم فى Method الكلمة Public

ملحوظة

التركيز دائما في Java على Object حيث هو الذي يحتوى على المعلومات والـ Method المطلوبة لتعطينا نتائج هذه المعلومات ، والـ Method الرئيسي في الـ Java عبارة عن أوامر لتنفيذ الـ Method التي هي داخل الـ Object فقط

مثال يتناول موضوع الفصائل Classes



ملحوظة هامة

يجب بعد تحديد اسم المشروع تحديد اسم Main class (الفصيلة الرئيسية) كما بالنافذة السابقة

```
1 class MathClass {
2     int a,b,c ;
3     public void printABC()
4     {
5         a=10;
6         b=20;
7         c=30;
8         System.out.println("a="+a) ;
9         System.out.println("b="+b);
10        System.out.println("c="+c);
11    }
12 }
13 public class Class2{
14     public static void main(String[] args) {
15     }
16 }
```

حيث أن:
 السطر ١ : إنشاء فصيلة بالاسم Mathclass باستخدام الكلمة المحجوزة class حتى يمكن التعامل مع الفصيلة class1
 السطر ٢ : الإعلان عن المتغيرات a,b,c من النوع Integer مع وضع الكلمة Public حتى يمكن التعامل مع تلك المتغيرات مباشرة من خارج الفصيلة Class1
 السطر ٣ : إنشاء Method من النوع Sub إجراء بالاسم PrintABC مع وضع الكلمة Public قبلها حتى يمكن التعامل مع هذا الإجراء مباشرة من خارج الفصيلة Class1
 السطور ٨ ، ٩ ، ١٠ : استخدام الدالة println() ثلاث مرات لطباعة قيم المتغيرات a ، b ، c وبذلك تم إنشاء فصيلة جديدة Class بالاسم class1 مع تعريف متغيرات data (a,b,c) ودالة Printabc() كأعضاء لهذه الفصيلة

استخدام الفصيلة الجديدة

نعود للبرنامج السابق وتعديله كما يلي :

```

1  class MathClass {
2      public void printABC()
3  }
4      a=10;
5      b=20;
6      c=30;
7      System.out.println("a="+a) ;
8      System.out.println("b="+b);
9      System.out.println("c="+c);
10 {
11 {
12 public class class2 {
13     public static void main(String[] args ) {
14         MathClass obj1=new MathClass() ;
15         obj1.printABC ( );
16     }
17 }
```

شكل شاشة الكود

```

class MathClass {
    int a,b,c;
    public void printABC()
    {
        a=10;
        b=20;
        c=30;
        System.out.println("a="+a) ;
        System.out.println("b="+b) ;
        System.out.println("c="+c) ;
    }
}

public class class2 {
    public static void main(String[] args) {

        MathClass obj1=new MathClass ();
        obj1.printABC();
    }
}
```

حيث أن :
السطر ١٤ : تم الإعلان عن متغير بالاسم obj1 من نوع الفصيلة class1 فهو في هذه الحالة لا يسمى متغير بل يسمى هدف Object مع استخدام الكلمة new لإنشاء هدف object فعلى بالذاكرة

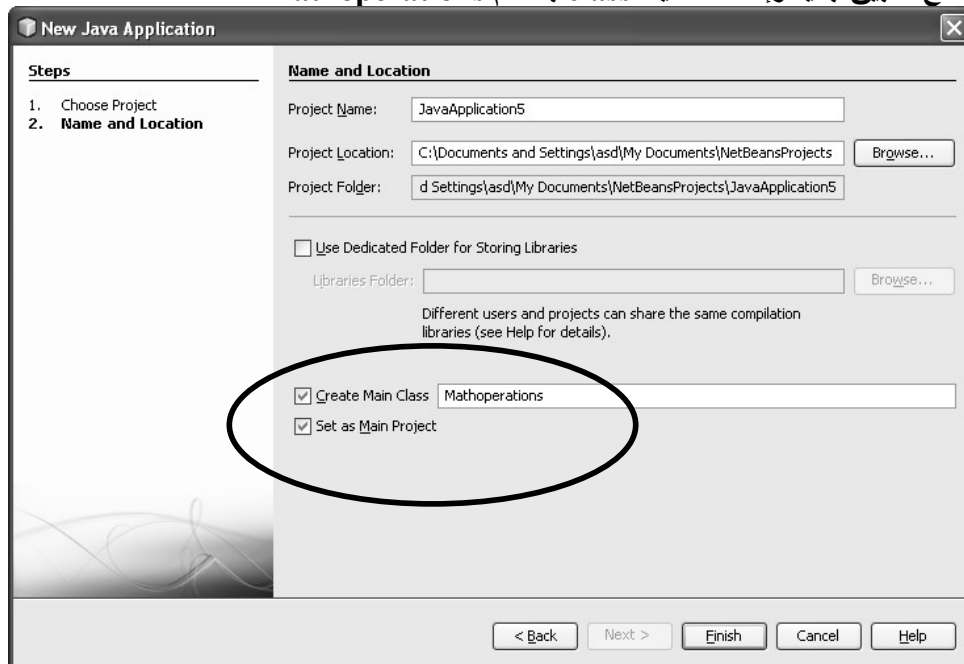
ملحوظة:
أي تعامل مع الفصيلة لا يتم مع اسمها بل يتم مع اسم الهدف المعرف منها مع ملاحظة أنه يمكن تعريف أكثر من هدف وبالتالي تغيير القيم في كل هدف
السطر ١٥ : تم استدعاء الدالة Printabc() ولكن مع اسم الهدف من الفصيلة وهو obj1 حيث لا يصلح - نلاحظ عند تنفيذ البرنامج استدعاء السطور المكتوبة داخل الدالة الرئيسية وهي تعريف هدف من الفصيلة وإعطاء قيمة لمتغيرات الفصيلة ثم طباعة هذه القيم باستدعاء الدالة printabc() كما يلي :

ويظهر التنفيذ كما يلي :

```
Output - JavaApplication4 (run)

run:
a=10
b=20
c=30
BUILD SUCCESSFUL (total time: 0 seconds)
```

مثال:فتح تطبيق جديد وإنشاء فصيلة class بالاسم Mathoperations



1	class operations
2	{
3	
4	{
5	
6	public class Mathoperations {
7	
8	public static void main(String[] args) {
9	}
10	}

-نقوم داخل سطور الفصيلة الجديدة **Mathoperations** بإنشاء دوال العمليات الحسابية حيث نقوم بتعريف دالة لعملية الجمع ودالة للطرح ودالة للضرب ودالة للقسمة لتصبح الفصيلة الجديدة **Mathoperations** كما يلي :

1	class operations
2	{
3	
4	{
5	
6	public class Mathoperations {
7	
8	public static void main(String[] args) {
9	}
10	}

-نقوم داخل الفصيلة الجديدة بتعريف متغيرات جديدة وهى **a ، b ، c**

- نقوم داخل الفصيلة الجديدة بإنشاء دوال العمليات الحسابية كما يلي:

```
1 class operations
2 {
3     public double a,b,c;
4     public double sum(double v1, double v2)
5     {
6         double res;
7         res =v1 + v2;
8         return res;
9     }
10    public double sub(double v1, double v2)
11    {
12        ouble res;
13        res =v1 - v2;
14        return res;
15    }
16    public double mul(double v1, double v2)
17    {
18        double res;
19        res =v1 * v2;
20        return res;
21    }
22    public double div(double v1, double v2)
23    {
24        double res;
25        res =v1 / v2;
26        return res;
27    }
28    {
29
30    }
31 }
32 public class Mathoperations {
33
34     public static void main(String[] args) {
35     }
36 }
```

-استخدام الفصيلة الجديدة حيث نعود للبرنامج السابق ونقوم بتعديله كما يلي :

```

1  class operations
2  {
3      public double a,b,c;
4      public double sum(double v1, double v2)
5      {
6          double res;
7          res =v1 + v2;
8          return res;
9      }
10     public double sub(double v1, double v2)
11     {
12         ouble res;
13         res =v1 - v2;
14         return res;
15     }
16     public double mul(double v1, double v2)
17     {
18         double res;
19         res =v1 * v2;
20         return res;
21     }
22     public double div(double v1, double v2)
23     {
24         double res;
25         res =v1 / v2;
26         return res;
27     }
28 }
29 public class Mathoperations {
30     public static void main(String[] args) {
31         operations obj=new operations( );
32         double sumRes,subRes,mulRes,divRes;
33         sumRes=obj.sum(100, 200);
34         subRes=obj.sub(300, 100);
35         mulRes=obj.mul(10,20);
36         divRes=obj.div(200, 5);
37         System.out.println("sumRes="+sumRes);
38         System.out.println("subRes="+subRes);
39         System.out.println("mulRes="+mulRes);
40         System.out.println("divRes="+divRes);
41     }
42 }
```

```

class operations
{
    public double a,b,c;
    public double sum(double v1, double v2)
    {
        double res;
        res =v1 + v2;
        return res;
    }
    public double sub(double v1, double v2)
    {
        double res;
        res =v1 - v2;
        return res;
    }
    public double mul(double v1, double v2)
    {
        double res;
        res =v1 * v2;
        return res;
    }
    public double div(double v1, double v2)
    {
        double res;
        res =v1 / v2;
        return res;
    }
}

public class Mathoperations {
    public static void main(String[] args) {
        operations obj=new operations();
        double sumRes,subRes,mulRes,divRes;
        sumRes=obj.sum(100, 200);
        subRes=obj.sub(300, 100);
        mulRes=obj.mul(10,20);
        divRes=obj.div(200, 5);
        System.out.println("sumRes="+sumRes);
        System.out.println("subRes="+subRes);
        System.out.println("mulRes="+mulRes);
        System.out.println("divRes="+divRes);
    }
}

```


يظهر ناتج التنفيذ كما يلي:

```
Output - JavaApplication5 (run)

run:
sumRes=300.0
subRes=200.0
mulRes=200.0
divRes=40.0
BUILD SUCCESSFUL (total time: 2 seconds)
```

الفصلية object

تعد تلك الفصلية هي الفصلية الأب لجميع الفصائل وهي توجد على رأس شجرة الفصائل وتحتوي تلك الفصلية على مجموعة الدوال التالية :

-clone() - equals(Object obj) - Finalize()- getClass()
-hashCode() - notify()- notifyAll()- toString()- wait()

وبالطبع جميل الفصائل بشكل أو بآخر ترث هذه الدوال

مثال:

يوضح كيفية استخدام الفصلية object الفصلية الأب لجميع الفصائل classes لتوضيح نظرية تعدد صور الدوال polymorphism بين الفصائل كما يلي :

```

1  class A extends Object
2      {
3
4      {
5      Class B extends A
6      {
7      public String toString() {
8      return "toString in class B:"
9      }
10     }
11     Class C extends B
12     {
13     public String toString() {
14     return "toString in class C:"
15     }
16     }
17     Public class Polymorphism04
18     {
19     public static void main(String[] args ) {
20     Object varA = new A( );
21     String v1=varA.toString( );
22     System.out.println(v1);
23     Object varB = new B();
24     String v2=varB.toString( );
25     System.out.println(v2);
26     Object varC = new C();
27     String v3=varC.toString( );
28     System.out.println(v3);
29     }//end main
30 }

```

حيث أن:

- السطر ١ : تعريف فصيلة class جديدة بالاسم A ترث الفصيلة الأب للفصائل Object
- السطر ٧ : تعريف فصيلة class جديدة بالاسم B ترث الفصيلة A ولكن تقوم بإعادة كتابة الدالة toString باستخدام الخاصية Function overriding
- السطر ١٣ : تعريف فصيلة class جديدة بالاسم C ترث الفصيلة B ولكن تقوم بإعادة كتابة الدالة toString باستخدام الخاصية Function overriding
- السطر ١٩ : تبدأ الفصيلة الأساسية وبها الدالة الرئيسية (main) التي تستخدم الفصائل السابقة

وعند تنفيذ الكود السابق نحصل على النتيجة التالية :

```
A @111f71
toString in class B
toString in class C
```

ملحوظات

overriding: يقصد بها الهيمنة حيث نستطيع كتابه نفس الـ Method وله نفس المعاملات في الـ Sub class الموجود في الـ main class

Polymorphism : يقصد بها التعددية حيث يمكن في الـ Java عمل مرجعية للـ Object فمثلا يمكن عمل مساواة بين object من نوع الـ main class بـ object من نوع Sub class كما يلي :

```
MasterStudent y= new MasterStudent( );
Student x=y;
```

هنا لا نستطيع استخدام الـ Method داخل الـ Sub class للـ Object المسمى x لأنه من نوع الـ main class ، نستفيد من تلك الخاصية في عمل مصفوفة من الطلاب

مثال

```
Student [ ] std = new MasterStudent [2] ;
Std [ 0] = new Student ( ) ;
Std [1] = new MasterStudent( ) ;
```

إنشاء أكثر من دالة باسم واحد

من الخصائص المتوافرة في مفاهيم البرمجة بالأهداف OOP مفهوم إنشاء أكثر من دالة باسم واحد ويطلق عليه method overloading حيث يسمح هذا المفهوم بإنشاء أكثر من دالة بنفس الاسم إذا دعت الحاجة لذلك مثل إنشاء مجموعة دوال رسم بالاسم draw() نستطيع من خلالها رسم خط line ومربع box ودائرة circle وغيره ولكن يتوقف ذلك على معاملات الدالة

كما يمكن إنشاء أكثر من دالة بحث بالاسم search() واحدة للبحث عن موظف بمعلومية رقم الموظف والثانية للبحث عن الموظف باسم الموظف ويتم استدعاء الدالة الأولى أو الثانية حسب المعامل الذي تم إرساله

```

1  class shapes
2  {
3      Public void drawline( )
4  {
5      For ( int i=0;i< 10;i++)
6      System.out.print( "*"");
7      System.out.println( " " );
8  }
9      Public void drawline( int n )
10 {
11     For ( int i=0;n< 10;i++)
12     System.out.print( "*"");
13     System.out.println( " " );
14 }
15     Public void drawline( int n , char ch)
16 {
17     For ( int i=0;i< n;i++)
18     System.out.print( ch);
19     System.out.println( " " );
20 }
21 }
22 Public class shapesApp {
23
24     public static void main(String[] args) {
25     Shapes obj=new shapes( ) ;
26     Obj.drawline( );
27     Obj.drawline(20);
28     Obj.drawline(30,"-");
29     }
30 }
```

مثال

```

1  class Student{
2  /**String النوع وهو من الطالب وهو من النوع */
3  public String name;
4  /**double النوع وهو من الجامعي وهو من النوع */
5  public double id;
6  /**int النوع وهي العلامة وهي من النوع */
7  public int mark ;
8  /**Default Constructor of Student Object*/
9  public Student(){
10 }
11 /**Constructor of Student Object with parameter */
12 public Student(String aName,double aId,int aMark){
13 name = aName;
14 id = aId;
15 mark = aMark;
16 }
17 }
18 class StudentTest{
19 public static void main(String[] args){
20 Student Mohamed = new Student("Mohamed",02345754,55);
21 System.out.print(Mohamed.name+ Mohamed.id+"["+
22 Mohamed.mark+"]");
24 }
}

```

يظهر ناتج التنفيذ كما يلي:

```

Output - JavaApplication11 (run)
run:
Mohamed642028.0[55]BUILD SUCCESSFUL (total time: 1 second)

```

mutator ، Accessor

في الـ Object دائما يوجد لكل field معرف عدد (٢) Method وهما:

- Accessor: يستخدم للوصول للقيمة المخزنة داخل الـ Field ويبدأ اسمه دائما بالكلمة get

ويقوم بإرجاع قيمة الـ Field ولا يوجد له باراميتير

- mutator: يستخدم لتغيير قيمة field معين ويبدأ اسمه دائما بالكلمة set وله باراميتير ولا

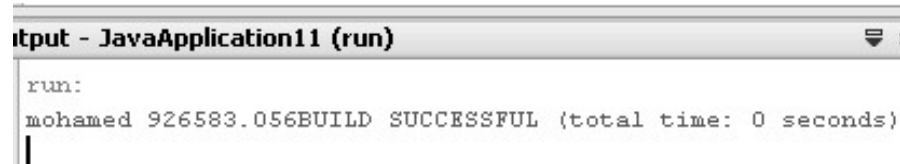
يقوم بإرجاع قيمة

ملحوظة : عند وضع كلمة Void قبل الـ Method هذا يعنى انه لا يرجع قيمة

```
1  class Student{
2  private String name;
3  private double id;
4  private int mark;
5  /**Default Constructor of Student Object*/
6  public Student(){
7  }
8  /**Constructor of Student Object with parameter */
9  public Student(String aName,double aId,int aMark){
10 name = aName;
11 id = aId;
12 mark = aMark;
13 }
14 /**accessor to name*/
15 public String getName( ){
16 return name;
17 }
18 /**mutator to name*/
19 public void setName(String aName){
20 name = aName;
21 }
22 /**accessor to Id*/
24 public double getId( ){
25 return id;
26 }
27 /**mutator to Id*/
28 public void setId(double aId){
29 id = aId;
30 }
31 /**accessor to Mark*/
32 public int getMark( ){
33 return mark;
34 }
35 /**mutator to Mark */
36 public void setMark(int aMark){
37 mark = aMark;
38 }
```

```
39 }
40 class StudentTest{
41     public static void main(String[] args){
42         Student mohamed = new Student( );
43         mohamed.setName("mohamed ");
44         mohamed.setId(03421567);
45         mohamed.setMark(56);
46         System.out.print(mohamed.getName()+mohamed.getId( )+
47         mohamed.getMark( ));
48     }
49 }
```

ويظهر ناتج التنفيذ كما يلي :



```
Output - JavaApplication11 (run)
run:
mohamed 926583.056BUILD SUCCESSFUL (total time: 0 seconds)
```