محاضرات العملي لمقرر

البرمجة الشيئية

ITGS211

الدرس العملي رقم (4)

إعداد الأستاذة :ملاك ددش

المتغيرات التي يتم وضعها في الكلاس تقسم إلى ثلاث فئات أساسية ذكرناها في الجدول التالي

بالإنجليزية	معناها
Local Variables	هي المتغيرات التي يتم تعريفها بداخل أي دالة, كونستركتور أو بداخل block (مثل الحلقات, الجملة `switch` إلخ..).
Instance Variables	هي المتغيرات التي يتم تعريفها بداخل الكلاس و خارج حدود أي دالة, كونستركتور, أو block. تسمى أيضاً Global Variables.
Class Variables	هي المتغيرات التي يتم تعريفها كـ `static` بداخل الكلاس و خارج حدود أي دالة, كونستركتور, أو block.

مثال

```
class VariablesTypes {

    // المتغيرات ( a, b, c, d ) تعتبر Instance Variables لأنه تم تعريفهم بداخل الكلاس و خارج أي دالة أو block
    // الكلمات ( public, protected, private ) ستفهم معناها في الدرس التالي, لكننا وضعناها فقط لتفهم الأسماء المستخدمة
    int a;
    public int b;
    protected int c;
    private int d;

    // المتغير e يعتبر Class Variable لأن نوعه static
    static int e;

    // المتغيرات ( x, y, z ) تعتبر Local Variables لأنه تم تعريفها بداخل الدالة
    public int sum(int x, int y) {
        int z = x + y;
        return z;
    }

}
```

الكلمة thisفي جافا:

الكلمة this هي كلمة محجوزة في لغة جافا, و هي تستخدم للإشارة إلى الـ Global Variables, و تستخدم أيضاً للإشارة إلى الكائن الحالي. و يمكن استخدامها في أماكن عديدة

في هذا الدرس سنستخدمها للتفرقة بين المتغيرات التي تم تعريفها بداخل الدوال Local Variables و بين المتغيرات التي تم تعريفها بداخل الكلاس و خارج الدوال Global Variables.

سنرجع إلى الكلاس Person و سنقوم باستخدام الكلمة this عدة مرات لمعرفة تأثيرها على الكود.

في هذا المثال لم نغير أي كود كان موجود, لكننا أضفنا كلمة this في كل مكان كنا نقصد فيه أننا نريد الوصول للخصائص

```
public class Person {

    String name;
    String gender;
    String job;
    int age;

    public Person() {

    }

    // هنا لا يوجد داعي لاستخدام الكلمة this لأن أسماء الباراميترات الموضوعة ليست نفسها أسماء الخصائص
    public Person(String n, String s, String j, int a) {
        this.name = n;
```

```
        this.gender  = s;
        this.job  = j;
        this.age  = a;
    }


    void printInfo() {
        System.out.println("Name: " +this.name);
        System.out.println("Gender: "  +this.gender);
        System.out.println("Job: "  +this.job);
        System.out.println("Age: "  +this.age);
        System.out.println();
    }

}
```

سنحصل على نفس النتيجة السابقة عند التشغيل

```
Name: Mhamad
Gender: Male
Job: Programmer
Age: 21

Name: Rose
Gender: Female
Job: Secretary
Age: 22

Name: Ahmad
Gender: Male
Job: Doctor
Age: 34

Name: Rabih
Gender: Male
Job: Engineer
Age: 27

Name: Lina
Gender: Female
Job: Graphic Designer
Age: 24
```

في هذا المثال استخدمنا الكلمة this عند الحاجة لها فقط.

فعلياً، قمنا بوضع أسماء للباراميترات كأسماء الخصائص في الكونستركتور فكان يجب وضع الكلمة this عند الحاجة للتفرقة بين الباراميترات و الخصائص حتى لا تحدث أخطاء منطقية عند تشغيل البرنامج.

Modifiersفي جافا:

مفهوم الـ Modifiersفي جافا:

الـ Modifiers هم كلمات يمكنك إضافتهم عند تعريف أشياء جديدة (سواء كلاس, متغير, دالة إلخ..) لتحديد طريقة الوصول إليها.
ستحتاجهم في الغالب إن كنت تعمل في برنامج كبير ضمن فريق من المبرمجين.

فإذا كنت تعمل على إنشاء برنامج معين ضمن فريق من المبرمجين, و تريد ضمان عدم إساءة إستخدام الأشياء التي قمت بتعريفها من قبل مبرمج آخر.
الـ Modifiers سيساعدوك في ذلك, فباستخدامهم يمكنك تحديد الأشياء التي يمكن لباقي المبرمجين الوصول إليها و الأشياء التي تريد التأكد من عدم التعديل عليها إلخ..

الـ Modifiers ينقسمون إلى نوعين:

- **Access Modifiers**
- **Non Access Modifiers**

في المثال التالي قمنا باستخدام الكلمتين public وprivate اللتين تعتبران من الـ Access Modifiers.
و قمنا باستخدام الكلمتين final وstatic اللتين تعتبران من الـ Non Access Modifiers

```
public class Student {

    private String firstName;
    private String lastName;
    private String specialization;
    private int id;
    private boolean isWork;
    final String theAvgerageForSuccess = "50%";
    static String CollegeName = "MIT";

    public void printFullName() {
        System.out.println("Name: " + firstName + " " + lastName);
    }

    public static void printCollegeName() {
        System.out.println("College: " + CollegeName);
    }

}
```

الـ Access Modifiersفي جافا:

Modifier	تعريفه
`public`	الكلاس أو الدالة أو المتغير الذي يتم تعريفه كـ `public` يمكن الوصول إليه مباشرة.
`protected`	الدالة أو المتغير الذي يتم تعريفه كـ `protected` يمكن الوصول إليه فقط من الكلاسات الموجودة في نفس الـ **package** أو من الكلاسات التي ترث منه. **ملاحظة:** لا يمكنك تعريف كلاس كـ `protected` .
	إذا لم تضع أي كلمة من الـ **Access Modifiers** عند تعريف كلاس أو دالة أو متغير سيتم وضع **Modifier** إفتراضي عنك يسمى **package private**. و هذا يعني أنه يمكن الوصول إليه فقط من الكلاسات الموجودة في نفس الـ **package**.
`private`	الـ `private` هو أعلا مستوى من حيث الحماية. المتغيرات و الدوال التي يتم تعريفها كـ `private` يمكن الوصول لها فقط من داخل الكلاس الذي تم تعريفها فيه. ملاحظات لا يمكنك تعريف كلاس كـ `private` . لا تقم بتعريف دالة كـ `private` إذا كان نوعها أيضاً `abstract` لأنك لن تستطيع أن تفعل لها **override**.

الـ Access Modifiersهم الأشياء الأساسية التي تسمح لك بتطبيق مبدأ الـ Encapsulationالذي يمكنك من إخفاء البيانات الأساسية في الكود التي لا تريد لأحد من المبرمجين الآخرين أن يعرف بتفاصيلهم

كتابة الكود بشكل مثالي:

بشكل عام, الـ Modifierالإفتراضي لا يستخدم في الغالب.
الـ publicيستخدم مع الدوال التي تريد للجميع أن يصل إليها.
الـ privateهو للمتغيرات التي لا تريد للكائنات و الكلاسات التي ترث من الكلاس أن تصل إليها.
الـ protectedيستخدم 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.

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

- **الدوال التي يتم تعريفهاكـ 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 كان, لذلك لا يوجد قواعد من أجلهم.**

مثال:

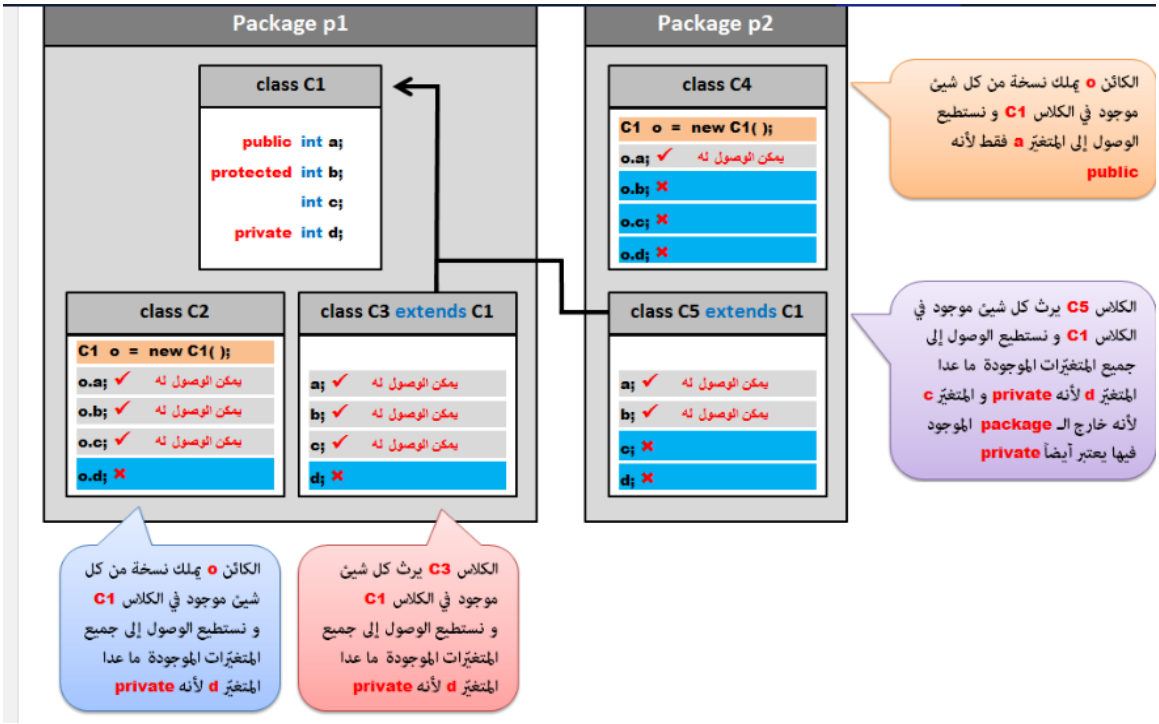
في الصورة التالية قمنا بإنشاء كلاس إسمه `C1` يحتوي على المتغيرات `a`, `b`, `c` و `d`

وضعنا **Access Modifier** لكل متغير.

ثم قمنا بإنشاء الكلاسات `C2` و `C3` في نفس الـ **package** و التي إسمها `p1`.

ثم قمنا بإنشاء الكلاسات `C4` و `C5` في **package** ثانية إسمها `p2`.

و في كل كلاس حاولنا الوصول لجميع العناصر الموجودة في الكلاس **C1**



Non Access Modifiers في جافا :

Modifier	تعريفه
`static`	يستخدم لتعريف كلاس أو متغير أو دالة مشتركة بين جميع الكائنات من كلاس معين.
`final`	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.
`abstract`	يستخدم لإنشاء كلاس أو دالة مجردة (أي دالة لا تحتوي على كود), الهدف من هذا الـ Modifier تجهيز كلاس معين و جعل الكلاسات التي ترث من هذا الكلاس هي من تقوم بتعريف الأشياء الموجودة بداخله.

الكلمة static في جافا :

ما الحاجة إلى تعريف شيء كـ static ؟

- **إن أردت تعريف شيء ثابت لجميع الكائنات, قم بتعريفه كـ static**
- **إن أردت تعريف شيء بداخل كلاس معين, و تريد الوصول إليه مباشرةً من الكلاس بدل إنشاء كائن من الكلاس ثم استدعاء الشيء منه, قم بتعريفه كـ static**

المتغيرات التي يتم تعريفها static

المتغير الذي يتم تعريفه كـ static يعتبر مشترك بين جميع الكائنات من نفس الكلاس. بمعنى أن كل كائن يتم إنشاءه من نفس الكلاس سيملك نفس هذا المتغير.
فعلياً المتغير هنا سيتم تعريفه مرة واحدة في الذاكرة و جميع الكائنات من نفس الكلاس ستشير إليه بدل أن تملك نسخة خاصة منه. إذاً static تعني نسخة واحدة من المتغير لجميع الكائنات.

المتغير الذي يتم تعريفه كـ static يسمى أيضاً Class Variable. لا يمكن تعريف الـ Local Variables كـ static.

يمكن الوصول للمتغير الذي تم تعريفه كـ static بذكر إسم الكلاس الذي تم تعريفه فيه ثم وضع إسمه, أو من أي كائن من الكلاس

الدوال التي يتم تعريفها static

في البداية, الدالة دائماً يتم تعريفها مرة واحدة في الذاكرة و جميع الكائنات من نفس الكلاس ستشير إليها.
الكلمة static تمكنك من الوصول إليها مباشرةً إلى الدالة من الكلاس دون الحاجة لخلق كائن و استدعائها من خلاله.

الدالة التي نوعها static يمكنها الوصول للمتغيرات المعرفة في الكلاس بشرط أن تكون هذه المتغيرات أيضاً static.

بشكل عام الدوال التي نوعها static لا تستخدم المتغيرات الموجودة في الكلاس, بل تستخدم المتغيرات التي يتم تعريفها كباراميترات لها أو المتغيرات التي يتم تعريفها بداخلها.

يمكن الوصول للدالة التي تم تعريفها كـ static بذكر إسم الكلاس الذي تم تعريفها فيه ثم وضع إسمه, أو من أي كائن من الكلاس

مثال حول الكلمة staticفي جافا :

```
public class Example {

    // Example.a قمنا بتعريف المتغير a كـ static. إذاً يمكننا الوصول إليه من خلال كائن أو من أي مكان مباشرة بهذا الشكل
    public static int a;

    // Example هذه الدالة لا يمكن استدعاءها إلا من خلال كائن من الكلاس
    public void print() {
        System.out.println( "a: " +a );
    }

    // Example.staticPrint(); هذه الدالة يمكن استدعائها مباشرة من أي مكان بهذا الشكل
    public static void staticPrint() {
        System.out.println( "a: " +a );
    }

}
```

```
public class Main {

    public static void main(String[] args) {

        // Example هنا قمنا بإنشاء كائنين e1 و e2 من الكلاس
        Example e1 = new Example();
        Example e2 = new Example();

        Example.a = 10;             // Example هنا قمنا بإعطاء قيمة لـ a مباشرة من الكلاس
        Example.staticPrint();      // static التي يمكننا استدعائها مباشرة من الكلاس لأن نوعها staticPrint() هنا بعرض قيمة a من خلال الدالة
        e1.staticPrint();           // هنا قمنا بعرض قيمة a من خلال الدالة staticPrint() التي يمكننا استدعائها من الكائن e1 أيضاً
        e2.staticPrint();           // هنا قمنا بعرض قيمة a من خلال الدالة staticPrint() التي يمكننا استدعائها من الكائن e2 أيضاً

        e1.a = 22;      // هنا قمنا بتغير قيمة a من خلال الكائن e1
        e1.print();     // ثم قمنا بعرضها من خلال الدالة print() التي وصلنا إليها من خلال الكائن e1

        e2.a = 75;      // هنا قمنا بتغير قيمة a من خلال الكائن e2
        e2.print();     // ثم قمنا بعرضها من خلال الدالة print() التي وصلنا إليها من خلال الكائن e2

    }

}
```

سنحصل على النتيجة التالية عند التشغيل.

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

الكلمة finalفي جافا :

ما الحاجة إلى تعريف شيء كـ final ؟

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

المتغيرات التي يتم تعريفها final

المتغير الذي يتم تعريفه كـ finalيعني أنه بمجرد إعطاءه قيمة, لا يمكن تغييرها من جديد.
عند إنشاء متغير نوعه finalيجب تحديد قيمته مرة واحدة فقط إما عند تعريفه أو في الكونستركتور.

المتغيرات التي يتم تعريفها final static

يمكن تعريف المتغير كـ finalو staticمع بعض, و عندها يمكن الوصول للمتغير من الكلاس مباشرةً أو من أي كائن من الكلاس, مع عدم إمكانية تغيير قيمته بعد تحديدها.
الثابت Math.PIو الثابت Math.Eهم من المتغيرات المعرفة كـ final staticفي جافا حيث يمكنك استخدامهم كما هم لكن لا يمكنك تغيير قيمهم.

الدوال التي يتم تعريفها final

الدالة التي يتم تعريفها كـ finalيعني أنه لا يمكن أن يتم تعريف محتواها في أي كلاس آخر.
أي الكلاس الذي يرثها لا يسمح له بأن يفعل لها override

الكلاس التي يتم تعريفها final

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

أمثلة شاملة حول أماكن وضع الكلمة finalفي جافا :

```
public class Example {

    public final int a = 10;    // هنا قمنا بتعريف متغير نوعه final و أعطيناه قيمة مباشرة عند تعريفه
    public final int b;         // هنا قمنا بتعريف متغير نوعه final بدون تحديد قيمته
    public final int c;         // هنا قمنا بتعريف متغير نوعه final بدون تحديد قيمته

    public Example(int b) {
        this.b = b;             // هنا سيتم تحديد قيمة المتغير b من الكائن, أي أن الكائن هو من سيقوم بتحديدها
        c = 50;                 // هنا سيتم تحديد قيمة المتغير c مباشرة عند إنشاء كائن, أي أن الكائن سيملكها هكذا
    }

}
```

في المثال التالي سنقوم بإنشاء 2كلاس, الكلاس Aو الكلاس Bالذي سيرث منه.
في الكلاس Aسنقوم بتعريف دالة عادية و دالة نوعها final
في الكلاس Bسنفعل overrideللدالة التي ليس نوعها final
بعدها سنقوم بإنشاء الكلاس Mainلتجربة الكود.
الهدف هنا معرفة أن الدوال المعرفة كـ finalلا يمكن تعريفها من جديد في الكلاس الذي يرثها. إذاً الكلمة finalتمنع الـ override

```
public class A {

    // final لهذه الدالة لأنها معرفة كـ Override أي كلاس سيرث من هذا الكلاس, لا يمكنه أن يفعل
    public final void firstPrint() {
        System.out.println("welcome to java");
    }

    // final لهذه الدالة لأنها غير معرفة كـ Override أي كلاس سيرث من هذا الكلاس, يمكنه أن يفعل
    public void secondPrint() {
        System.out.println("welcome to java");
    }

}
```

```
// هنا قلنا أن الكلاس B يرث من الكلاس A. أي يمكن للكلاس B إستخدام الأشياء الموجودة في A و كأنها موجودة فيه تماماً
public class B extends A {

    // B هنا قمنا بإعادة كتابة محتوى الدالة secondPrint بالنسبة للكلاس
    @Override
    public void secondPrint() {
        System.out.println("class B override my content!");
    }

}
```

```
public class Main {

    public static void main(String[] args) {

        A a = new A();          // a هنا قمنا بإنشاء كائن من الكلاس A إسمه
        B b = new B();          // b هنا قمنا بإنشاء كائن من الكلاس B إسمه

        a.firstPrint();         // A هنا سيتم تنفيذ الدالة كما تم تعريفها في الكلاس
        a.secondPrint();        // A هنا سيتم تنفيذ الدالة كما تم تعريفها في الكلاس

        b.firstPrint();         // A هنا سيتم تنفيذ الدالة كما تم تعريفها في الكلاس
        b.secondPrint();        // B هنا سيتم تنفيذ الدالة كما تم تعريفها في الكلاس

    }

}
```

سنحصل على النتيجة التالية عند التشغيل

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

المثال الثاني :

في المثال التالي سنقوم بإنشاء كلاس نوعه final من أجل جعل الكود غير قابل لأي تعديل خارجي.
بعدها سنقوم بإنشاء الكلاس Main لتجربة الكود.
الهدف هنا معرفة أن الكلاس المعرف كـ final يمكن إنشاء كائنات منه, لكن لا يمكن الوراثة منه.

```
// هنا قمنا بتعريف الكلاس كـ final لمنع الوراثة منه فقط
public final class FatherOfJava {

    public String name = "James Arthur Gosling";
    public String born = "May 19, 1955";

    public void story() {
        System.out.println(name+ ", born on " +born+ ". He is a Canadian computer scientist,"
                                + " best known as the father of the Java programming language");
    }

}
```

```
public class Main {

    public static void main(String[] args) {

        // هنا قمنا بإنشاء كائن من الكلاس FatherOfJava إسمه obj
        FatherOfJava obj = new FatherOfJava();

        // هنا سيتم عرض قيمة المتغير name كما تم تعريفها في الكلاس الأساسي
        System.out.println("Name of the father of java: " +obj.name);

        // هنا سيتم تنفيذ الدالة كما تم تعريفها في الكلاس FatherOfJava
        obj.story();

    }

}
```

سنحصل على النتيجة التالية عند التشغيل

```
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
```