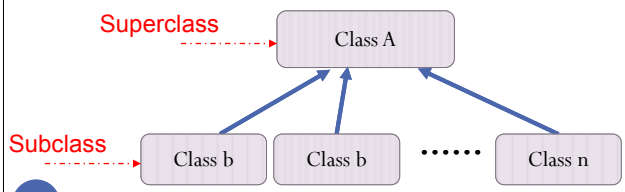
الوراثة Inheritance

- مفهوم الوراثة في الحياة هو أن يستمد نوع معين صفات وسلوك من نوع آخر، فمثلا يستمد الطفل من صفات أبيه لون العينين، الطول ومن سلوك أبيه مهارة الرسم مثلاً وهكذا... وبالطبع قد يتمتع الابن بصفات جديدة لم تكن موجودة في أبويه. (**أي أن الوراثة هي نقل الصفات**).
- يعتبر التوريث من مفاهيم البرمجة الشيئية OOP الأساسية، إذ تتيح البرمجة الشيئية للفئات Classes ***وراثة*** الخصائص والسلوكيات المشتركة من فئات classes أخرى بالاضافة لصفات أخرى.
- حيث تمكّنك الوراثة من إنشاء فئات جديدة تعيد استخدام وتوسّع وتعدّل السلوك (method) الذي تم تعريفه في الفئات الأخرى. مما يقلل إعادة كتابة الأوامر وذلك بإنشاء class يحتوي على خصائص ودوال ثم يتم إستعمالها كأساس لكائنات objects آخرى، وبالتالي لانحتاج لكتابة ماكتبناه في تعريف الكائن الأولي.

2

- البرمجة الشيئية OOP تتيح للفئات Classes ***وراثة*** الخصائص والسلوكيات المشتركة من فئات classes أخرى. حيث تمكّنك الوراثة من إنشاء فئات جديدة تعيد استخدام وتوسّع وتعدّل السلوك (method) الذي تم تعريفه في الفئات القديمة.
- الـ class المُورث الذي يقوم بتوريث صفاته للآخر يُسمى أي:
  - Superclass ✓
  - Parent class ✓
  - Base class ✓
- بينما الـclass المُشتق الذي يرث صفات وسلوك من الآخرين فيُسمى:
  - Subclass ✓
  - Child class ✓
  - Derived class ✓



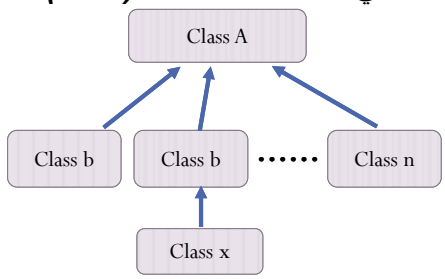
3

أنواع الوراثــة

- **وراثة فردية الأب (Single inheritance):** وفيها يرث الابن صفات وسلوك من أب واحد فقط وهذا النوع هو النوع الذي تدعمه لغة Java.
- **وراثة فيها أكثر من أب (Multiple inheritance):** وفيها قد يرث الابن صفات وسلوك أكثر من أب، وهذا النوع لا تدعمه لغة Java بشكل مباشر بل يمكن أن يتم بشكل غير مباشر وذلك من خلال التوريث مرتين (اب، ابن، حفيد) وفي هذه الحالة يكون الحفيد قد ورث من أكثر من أب. (مع فقدان الصفات التي كانت Protected في الاب لانها تصبح Private في الابن و بالتالي الحفيد لا يراها)

4

- في لغة Java على وجه الخصوص، يسمح لكل Class بأن يرث من فئة أب Superclass واحدة ((منعاً للغموض)).
- يُسمح لكل Superclass بعدد لا منتهي من ***الفئات الفرعية (الابناء)*** Subclass



ولكن هل يرث الـSubclass كل الصفات (المتغيرات) والسلوك (الدوال)؟
طبعاً **لا**، بل يُسمح فقط بوراثة المتغيرات والدوال (Class members) التي:

- ✓ Public
- ✓ Protected.

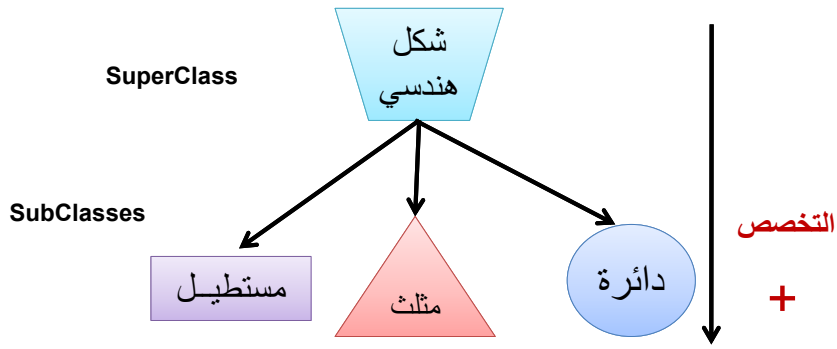
5

مفهـوم الـ Protected access modifiers

- معنى أن المتغير Protected محمي أي لايمكن استخدامه إلا في:-
  1. نفس الـClass المعرف فيه هذا المتغير.
  2. أي class مشتق Subclass من الـclass المعرف فيه هذا المتغير.
  3. جميع الـclasses التي في نفس الحزمة package التي بها الـClass الذي يحوي هذا المتغير المحمي.
- للعلم الـProtected access modifiers يستعمل فقط مع المتغيرات والدوال methods ولايستخدم مع الـclass (لا يمكن القول عن Class انه protected) بعكس Public التي يمكن استخدامها الـclass.
- جميع المتغيرات والدوال التي يتم تعريفها ضمن الـ SuperClass من النوع Protected يمكن للـ Subclass أن يرثها ويراها ويتعامل معها.

6

العلاقة بين الـ SuperClass و الـ SubClass



7

7

مفهـوم الـ Protected access modifiers

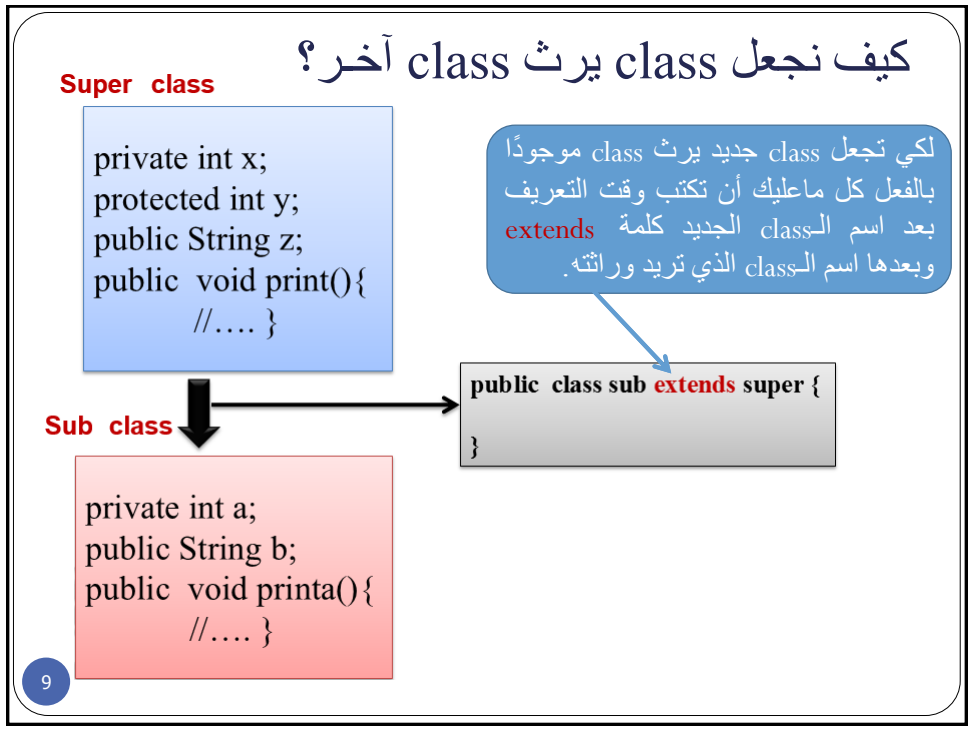
Super class

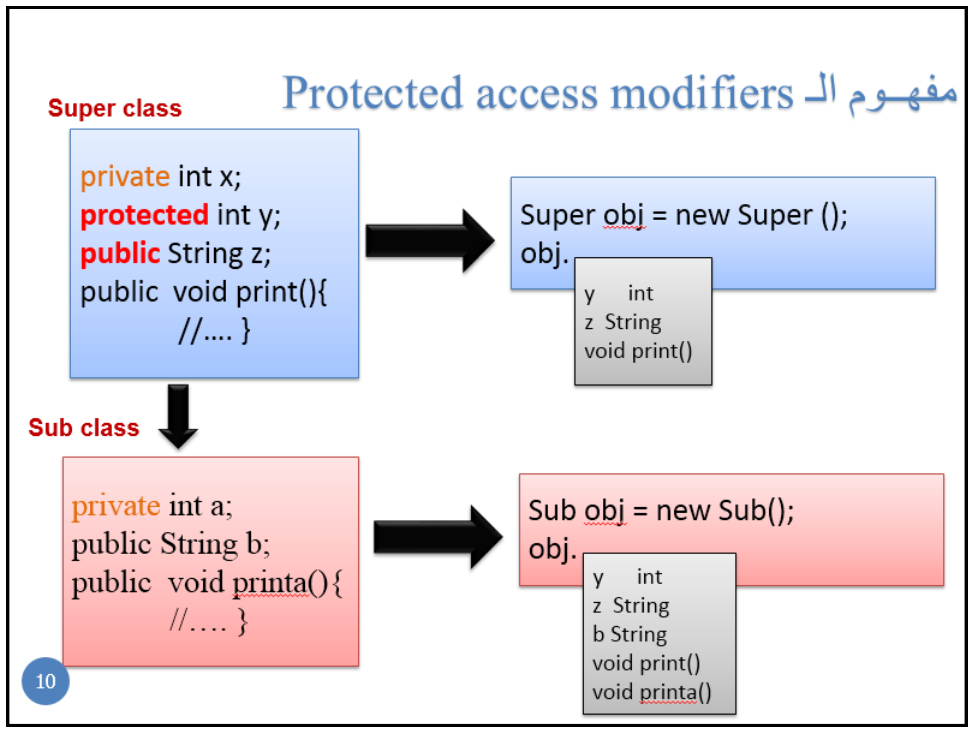
```
private int x;
protected int y;
public String z;
public  void print(){
        //.... }
```

```
Super obj = new Super ();
obj.
```

y int
z String
void print()

8


كيف نجعل class يرث class آخر؟
Super class
private int x;
protected int y;
public String z;
public void print(){
//…. }
لكي تجعل class جديد يرث class موجودًا بالفعل كل ماعليك أن تكتب وقت التعريف بعد اسم الـclass الجديد كلمة extends وبعدها اسم الـclass الذي تريد وراثته.
public class sub extends super {
}
Sub class
private int a;
public String b;
public void printa(){
//…. }
9


Protected access modifiers مفهـوم الـ
Super class
private int x;
protected int y;
public String z;
public void print(){
//…. }
Super obj = new Super ();
obj.
y int
z String
void print()
Sub class
private int a;
public String b;
public void printa(){
//…. }
Sub obj = new Sub();
obj.
y int
z String
b String
void print()
void printa()
10

Access Control Levels

private: accessible from within this class only
- should be standard practice on instance fields to support OO encapsulation

protected: accessible from any subclass or any class in the package

public: accessible from any class anywhere

none (if you don't specify): accessible from any class in the package

access modifier	class	subclass	package	world
private	x			
default	x		x	
protected	x	x	x	
public	x	x	x	x

11

استعمال الـ class first and class second

مثال:

```
class first{
   int x,y;
   void show( ){
     System.out.println("x=   "+x+"\t y=  "+y);
   }
}
```

Class first

```
class second extends first{
   int z;
   void shows( ){
      System.out.println("z=  "+z);
   }
}
```

Class second ويرث من Class first الـ

12

تابع المثال في الشريحة السابقة: (استعمال الـ class first and class second)

```
package student;
public class Student {
   public static void main(String[] args) {
      first f= new first( );
      second s= new second( );
      f.x=1;
      f.y=2;
      f.show();
      s.x=3;
      s.y=4;
      s.z=5;
      s.show();
      s.shows()
   }
}
```

Output:

x= 1 y= 2

x= 3 y= 4

z= 5

13

طبيعة التعامل مع الـ Constructor

A class

```
protected int y;
public String z;
public A(){
        //…. }
```

B class

```
private int a;
public String b;
public  B(){
 Super();
        //…. }
```

لاحظ

عندما يرث class جديد class آخر فإنه يرث كل ماهو public, protected باستثناء الدوال Constructor إلا أن الـ class B يقوم **ضمنيًا** باستدعاء الـConstructor الخاص بالـ class A

كما يمكن أن يستدعيه بشكل **صريح** من داخل الـ Constructor الخاص به بجملة برمجية واحدة كالتالي:

super();

وهي تعني استدعاء للـConstructor الخاص بالأب وهي خطوة تتم بشكل ضمني وبالتالي لا داعي لكتابتها بشكل صريح

14

طباعة مضروب عدد باستخدام مفهوم الوراثة ودالة البناء الـ Constructor

```
package inheritance2;
public class Inheritance2 {
   public static void main(String[] args) {
      setnumber result = new setnumber(5);
      result.findfact (result.no);
   }
}
class fact {
   int fact=1;
   void findfact(int x){
      for (int i=1;i<=x;++i)
         fact=fact*i;
      System.out.println ("The factorial =  "+fact);
   }
}
class setnumber extends fact {
    int no;
    setnumber ( int y){
       no=y;
    }
}
```

هنا استدعاء الـ object (الصورة) من الابن لدالة موجودة في الكلاس الأب

ناتج البرنامج
The factorial = 120

الكلاس الأب

الكلاس الأبن

15

```
package inheritance3;
public class Inheritance3 {
   public static void main(String[] args) {
   A x= new A();
   B z= new B();
   System.out.println(x.val);
   System.out.println(z.val);
     z.printA();
     z.printB();
}
}
class A{
   public int val;
   public A()
   {
    z.val=20;
   }
   public void printA(){
      System.out.println("class A");
}
class B extends A{
   public B()
   {
   }
   public void printB(){
      System.out.println("class B");
   }
}
```

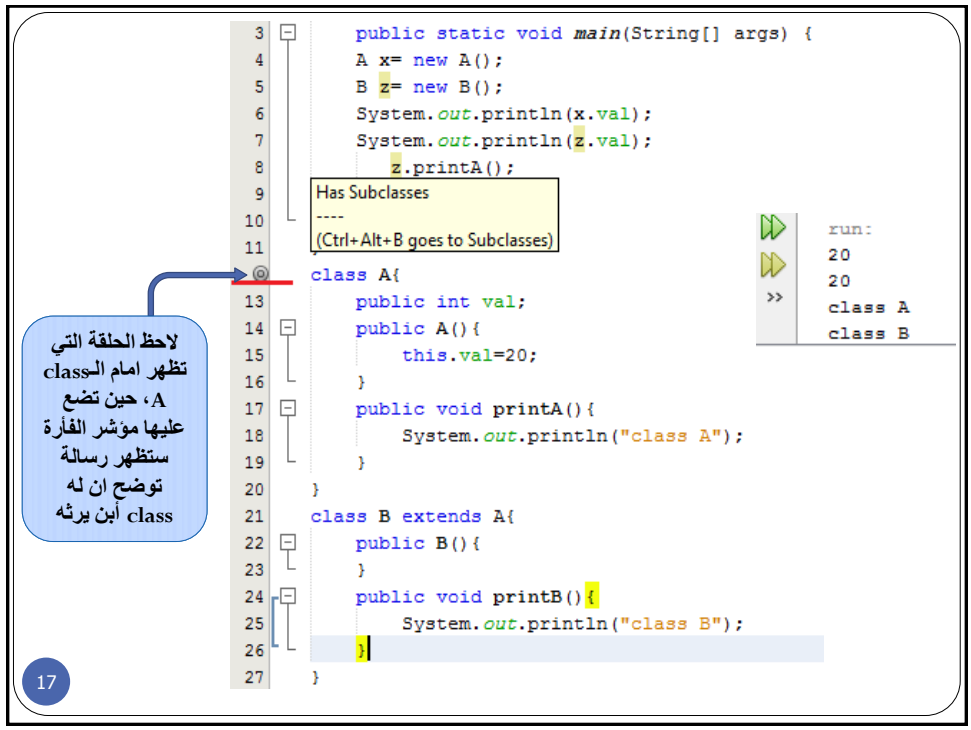
هنا بمجرد تكوين الكائن x فإنه يستبدل الكلمة this في الكلاس A بـ x

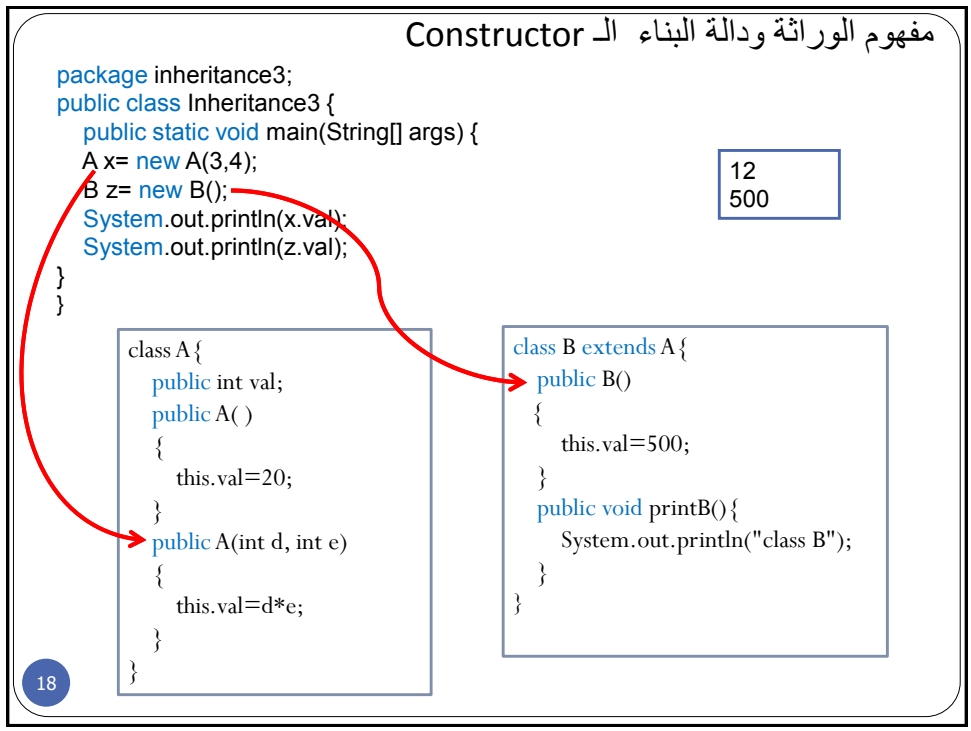
هنا بمجرد تكوين الكائن z فإنه يستبدل الكلمة this في الكلاس B بـ z

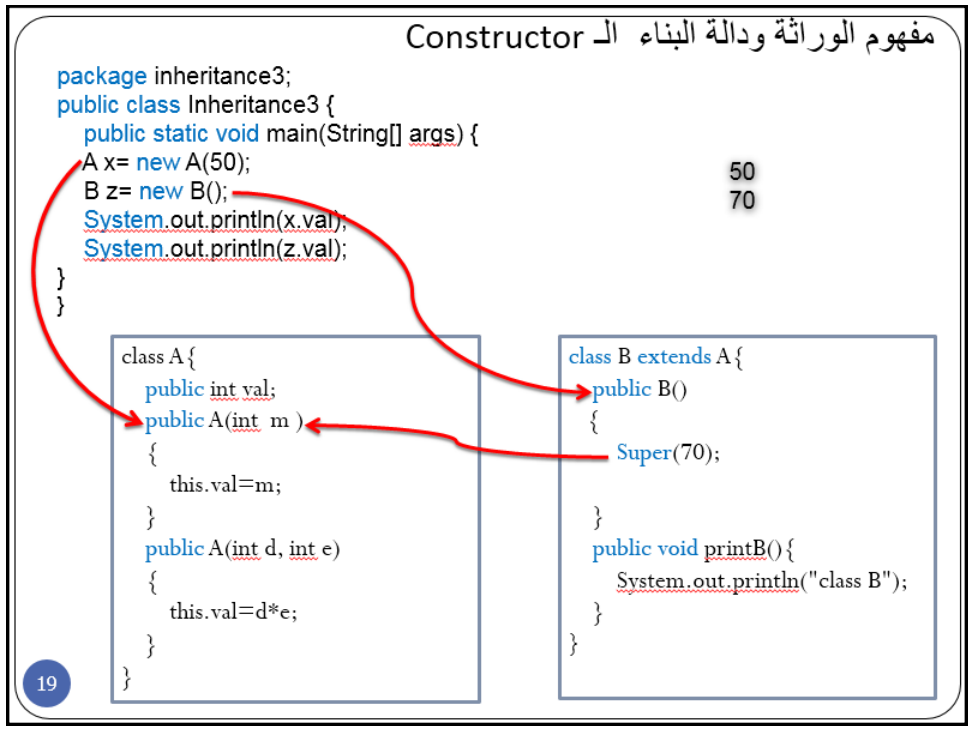
```
20
20
class A
class B
```

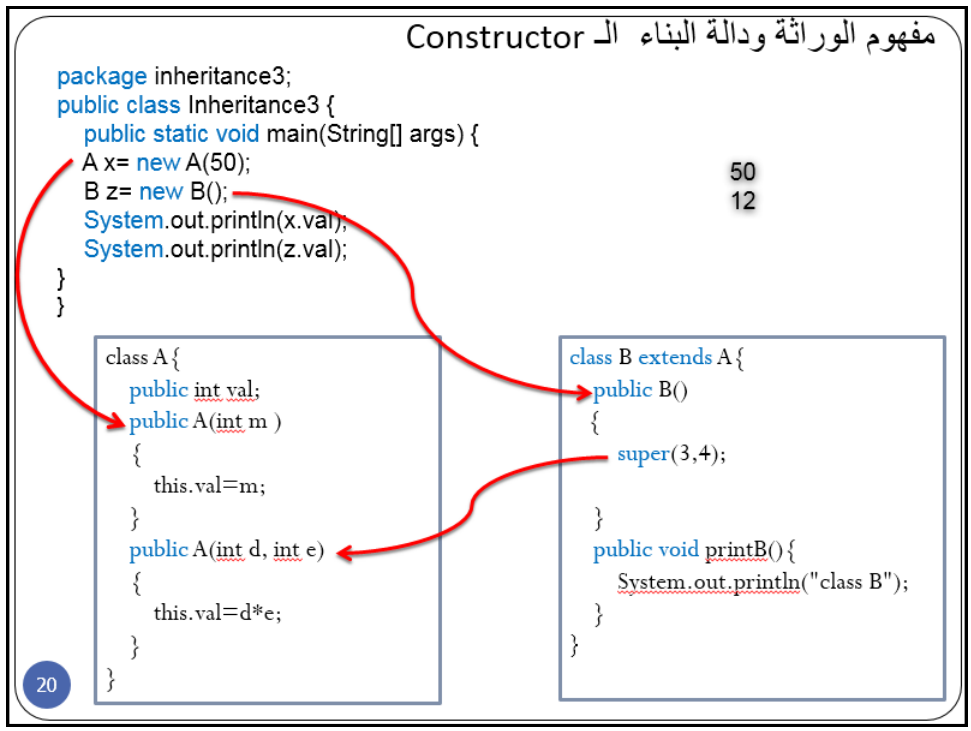
this متغير مجهول يمثل الكائن الذي سوف يتم إنشاؤه وتستخدم للوصول إلى محتويات الـ class داخليا ولا تستخدم خارج الـ class.

16


لاحظ الحلقة التي تظهر امام الـclass A، حين تضع عليها مؤشر الفأرة ستظهر رسالة توضح ان له class أبن يرثه
Has Subclasses
(Ctrl+Alt+B goes to Subclasses)
run:
20
20
class A
class B
17


مفهوم الوراثة ودالة البناء الـ Constructor
12
500
18


مفهوم الوراثة ودالة البناء الـ Constructor
package inheritance3;
public class Inheritance3 {
public static void main(String[] args) {
A x= new A(50);
B z= new B();
System.out.println(x.val);
System.out.println(z.val);
}
}
50
70
class A{
public int val;
public A(int m)
{
this.val=m;
}
public A(int d, int e)
{
this.val=d*e;
}
}
class B extends A{
public B()
{
Super(70);
}
public void printB(){
System.out.println("class B");
}
}
19


مفهوم الوراثة ودالة البناء الـ Constructor
package inheritance3;
public class Inheritance3 {
public static void main(String[] args) {
A x= new A(50);
B z= new B();
System.out.println(x.val);
System.out.println(z.val);
}
}
50
12
class A{
public int val;
public A(int m)
{
this.val=m;
}
public A(int d, int e)
{
this.val=d*e;
}
}
class B extends A{
public B()
{
super(3,4);
}
public void printB(){
System.out.println("class B");
}
}
20

تركيب الدوال Method overriding

- من الإمكانيات المتاحة في الـــ OOP خاصية overriding وهذه الخاصية تعني إمكانية كتابة دوال جديدة في الـ class الجديدة بنفس اسم الدوال الموجودة في الـ class الأساسية التي تم توريثها أي أن الدالة موجودة في الـ class الأساسية ومع ذلك تم إنشائها مرة أخرى بنفس الاسم في الـ class الجديدة، وبالتالي يتم إلغاء الدالة من الـ class الأساسية واعتماد الدالة الجديدة وهذا ما يسمى دالة على أخرى overriding.

21

تركيب الدوال Method overriding

```
class one {
   void say (){
       System.out.println(" First class ");
    }
}

class two extends one {
  void say (){
    System.out.println(" Second  class ");
   }
}
```

الـ class two يرث من class one وفيه دالة باسم say () وهو نفس اسم الدالة الموجودة في الـ class one

22

تركيب الدوال Method overriding

```
public class overriding1
 {
public static void main(String[] args)  {
     one o=new one();
     o. say();
     two t=new two();
     t. say();
}
}
```

Output

```
First class
Second class
```

23

منع التوريث ومنع overriding على الدوال باستخدام final

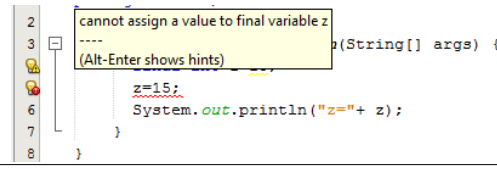
```
public class Final1 {
public static void main(String[] args)  {
    final int z=10;
     System.out.println("z="+ z);
}
}
```

Z=10

ماذا لو حاولنا تغيير قيمة الثابت كالتالي؟

```
public class Final1 {
public static void main(String[] args)  {
    final int z=10;
     z=15;
     System.out.println("z="+ z);
}
}
```

لم يتم تنفيذ العملية لأن المتغير z (ثابت) final
ولا نستطيع اسناد قيمة جديدة لنفس المتغير.



24

استخدام final في الـ class لمنع overriding على الدوال

```
class one {
  final void say (){
    System.out.println(" First class ");
   }
}
class two extends one
{
  void say ()
{
  System.out.println(" Second  class ");
}
}
```

الـ class two يرث من class one وفيه دالة باسم say وهو نفس اسم الدالة الموجودة في الـ class one ولكن في هذه الحالة الدالة هنا final فلذلك سيمنع الـ overriding .

```
13  class one {
14      final void say (){
15          System.out.println("F
16      }
17
18
20      void say(){
21          System.out.println("Seco
        }
```

say() in two cannot override say() in one
overridden method is final

(Alt-Enter shows hints)

25

استخدام final لمنع توريث الـ class >> Not inheritable

```
final class one {
  void say (){
    System.out.println(" First class ");
}
}
class two extends one {
 void say (){
   System.out.println(" Second  class ");
}
}
```

الـ class two يرث من class one ولكن لـن تتم الوراثة لكون الـ class one تم الإعلان على أنه final.
وسوف تظهر رسالة خطأ تفيد بعدم امكانية الوراثة .

```
13  final class one {
14      void say (){
15          System.out.println("First class");
16      }
    class two extends one {
```

cannot inherit from final one

(Alt-Enter shows hints)

26

Example :

قم بإنشاء class باسم x1 يحتوي على دالة باسم max تعمل على تحديد أكبر رقم من ثلاث أرقام.
قم بإنشاء class باسم x2 ومنع توريثها لـ class آخر وترث الـ x1 وتحتوي على دالة باسم numbers ولا ترجع أي قيمة وتعمل على تحديد إدخال 3 أرقام .
أكتب البرنامج الرئيسي بحيث يتم طباعة قيمة العدد الأكبر وذلك من خلال استدعاء الدالة max , numbers

27

```
import java.util.Scanner;
public class Myclass {
    public static void main(String[] args) {
        x2 z= new x2();
        z.numbers();
        System.out.println("max is : " + z.max(z.n1,z.n2,z.n3));
    }
}
```

```
class x1{
   int max (int a,int b ,int c)
   {
      int result ;
      if (a>b && a>c)
           result  = a;
      if(b>a && b>c)
           result =b;
      else
           result =c;
      return result ;
   }}
```

```
Final class x2 extends x1 {
        int n1;
        int n2;
        int n3;
   void numbers(){
      Scanner in = new Scanner (System .in);
       n1= in.nextInt();
       n2= in.nextInt();
       n3= in.nextInt();
   }
}
```

28

شكراً لإنصاتكم

أي سؤال ؟

فالفائدة تبدأ بالسؤال ثم النقاش ..

لنبدأ النقاش فنستفيد ...

29