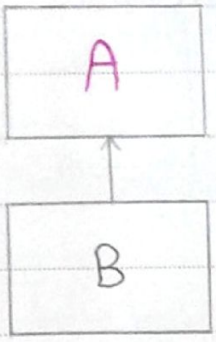
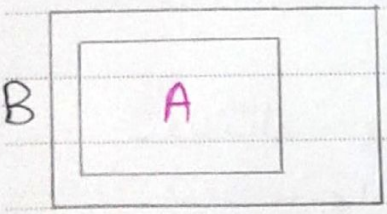
الوراثة في لغة الجافا
Inheritance

مقدمة:

- كما رأينا سابقاً إن فكرة الكائن مفيدة لأنها تتيح لنا تغليف البيانات والطرق في قالب واحد يسمى صفاً class.
لكن يبدو أنه من غير المنطقي أن ننشئ صفاً ثم نجبر أن ننشئ صفاً جديداً يمكن أن يملك الصفات نفسها أو حالات خاصة منها.
الأفضل من ذلك هو أن نأخذ صفاً موجوداً ونأخذ نسخة منه ثم نضيف أو نجري تغييرات إلى هذه النسخة. وهذا بالضبط ما تفعله الوراثة.
- إذا كان الصف B هو نسخة عن الصف A فإن أي تغيرات تحدث على الصف A تنعكس على B.
نسمي بمصطلحات الوراثة الصف A بالصف الأب Base class
بينما ندعو الصف B بالصف الابن Derived Class
ونقول في هذه الحالة إن الصف B يرث الصف A.



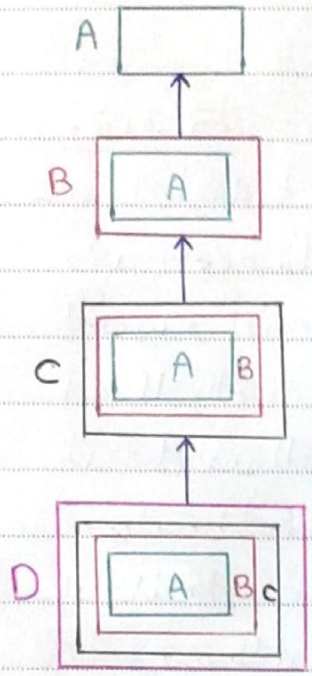
- إن الصف B يحوي جميع مكونات الصف A المسموح لها بالانتقال بعلاقة الوراثة (protected أو public)
أي أن كل ما هو خاص private في A لن يكون محتوى في B.
كما أن الصف B يحوي الحقول والطرق الإضافية المعرفة فيه.
- يمكن للصف B أن يعيد تعريف طرق الصف A لتغيير عملها وذلك بأن يعرّف دالة لها نفس الاسم ونفس أنواع وأعداد وترتيب الوسطاء ويكتب التعليمات البرمجية التي يريد.
وهذا الأمر يختلف عن التحميل الزائد للدوال الذي هو ممكن هنا أيضاً، حيث يمكن للصف B أن تحمّل أية دالة في A تحميلاً زائداً ما لم تكن محددة بـ final.



- إن وراثة B من A يتم التعبير عنها عند تعريف الصف B بالكلمة المفتاحية extends
وذلك كما يلي:

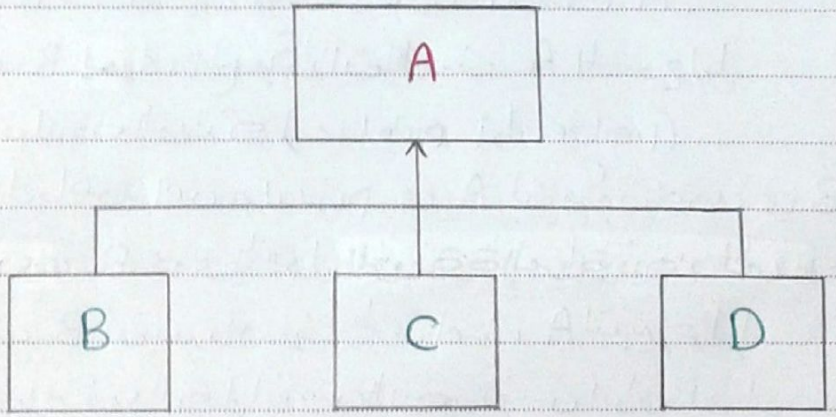
```
class B extends A{
    ≡≡≡
}
```

/ /

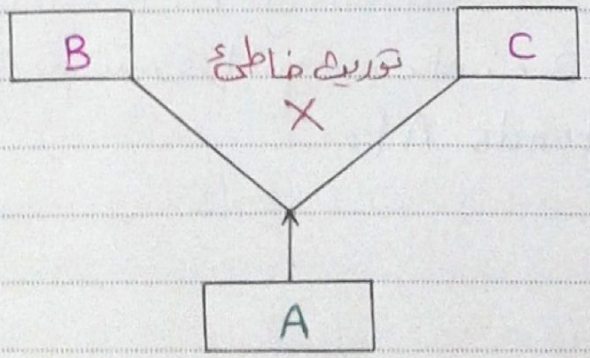


- يمكن للوراثة بين الصفوف أن تذهب إلى أي عمق نريد طالما أن لكل صف أباً واحداً مباشراً.
- في الشكل المجاور نلاحظ أن:
أي كائن من B هو كائن من A
وأي كائن من C هو كائن من A, B بنفس الوقت.
وكذلك أي كائن من D هو كائن من A, B, C.
وبشكل عام نقول إن الابن هو أب ولكن العكس ليس صحيح.
ومنه يمكن دوماً التصريح عن كائن من صف ما على أنه كائن من نوع الأب.

- ويمكن أيضاً للصف الواحد أن يمتلك أي عدد من الأبناء.



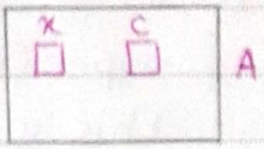
- ولكن لا تسمح لغة الجافا بالوراثة المتعددة.
أي لا يسمح للصف الواحد أن يرث من أكثر من أب مباشر واحد.



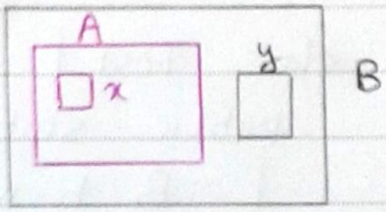
/ /

مثال:

```
class A{
  public int x;
  private char c;
  public A(int x) {this.x = x;}
  public void print()
  {System.out.print("I am an object of A");}
}
```




```
class B extends A{
  int y;
  B (int y) {this.y=y;}
  void print () {System.out.print("I am an object of B");}
}
```



إن إنشاء كائن من B يعني ضمنياً إنشاء كائن من A.
ولكن وبما أننا لم نقم باستدعاء باني A ضمن باني B بشكل صريح فستقوم اللغة تلقائياً باستدعاء الباني الخالي لـ A.
ولكن بما أننا كتبنا بانياً في الصف A فإن اللغة لن تنشئ بانياً افتراضياً خالياً وبالتالي لن تستطيع اللغة إنشاء كائن من A افتراضياً وهنا وقعنا في إشكال.
يمكن حل هذا الإشكال بإحدى طريقتين:

1) إما أن نضيف الباني الخالي إلى الصف A. `A() {}`
2) أو أن نضيف استدعاءً لباني A في باني B مع إضافة وسيط آخر لباني B نمرّره كوسيط لباني A.

- توفر لغة الجافا الكلمة المفتاحية super التي تعبّر عن الصف الأب للصف الذي نحن فيه، وتسمح لنا هذه الكلمة باستدعاء باني الأب ضمن باني الابن بشرط أن يكون هذا الاستدعاء هو التعليمة الأولى من التعليمات في باني الابن.

/ /

أي أن باني B في المثال السابق يجب أن يكون لهذا الشكل:

```
B (int y, int n)
{ super(n);
  this.y=y;
}
```

- لنناقش الآن بعض الحالات على المثال نفسه:

```
class Test {
  public static void main (String arg[])
  { B b=new B(2,3);
    b.x=4;          ✓
    b.c='X';        X        لأن c خاص بـ A
    b.print();
```

هنا سوف يستخدم الدالة الموجودة في الصف B أي يطبع:

((I am an object of B))

```
    A aa=new B(5,2);   ✓
```

إن aa فنياً هو كائن من النوع B ولكن يمكن التصريح عنه على أنه من نوع الأب A

```
    aa.print();
```

أي إذا كتبنا:

((I am an object of B)) فإنه سوف يطبع:

```
    B bb=new A(7);   X
```

خطأ لأنه لا يمكن التصريح عن كائن من نوع الأب A على أنه كائن من نوع الابن B

```
}}
```

* وبشكل عام في حال تعدد الصفوف المتوارثة يمكن بناء كائن من النوع الأب عن طريق أي باني لأي صف من الأبناء.

ملاحظة: يوجد صف اسمه object ترث منه كل الصفوف التي نكتبها.