الصفوف المجردة
Abstract classes

- يمتلك مفهوم الصف المجرد أهمية خاصة لأنه يوفر تقنية فعالة لتعريف صفوف عامة الأغراض يمكن تخصيصها فيما بعد باستخدام مفهوم الوراثة.
بكلمات أبسط، يعزز الصف المجرد مفهوم إعادة الاستخدام وذلك بكتابة صف يحتوي على الخصائص العامة التي تشترك فيها مجموعة من المسائل وترك التفاصيل الخاصة بكل مسألة لتعريفها من خلال الصفوف التي ترث هذا الصف المجرد.

- يتم التصريح عن الصف المجرد باستخدام الكلمة المفتاحية "abstract".
مثلاً: التصريح التالي يعرّف صفاً مجرداً "Shape":

```
abstract class Shape {
    String name;
}
```

- يمكن للصف المجرد أن يحوي طرقاً محققة Concrete Methods معرفة بشكل كامل ويمكن للصف المجرد أيضاً أن يعرّف طرقاً مجردة Abstract Methods تتألف فقط من ترويسة ونوع إرجاع، وفي هذه الحالة يجب كتابة الكلمة المفتاحية abstract في بداية التصريح عن هذه الطريقة.

- لا يمكن لصف غير مجرد أن يحوي طرقاً مجردة، أي أن الصف يصبح مجرداً إذا احوى طريقة مجردة أو أكثر.
ومنه يتألف الصف المجرد من معطيات أعضاء وطرق محققة وطرق مجردة.

/ /

مثال:

```
abstract class Shape{
    String name;
    void setname (String n)
    { name = n; }
    abstract double surface();
}
```

- نلاحظ أن الطريقة المجردة surface تم تعريفها فقط بالترويسة وأنها لا تملك أية تعليمات برمجية، أما الطريقة غير المجردة setname فهي تحوي الترويسة إضافة إلى التعليمات البرمجية.

- لا يمكن اشتقاق كائنات من الصف المجرد لأن الغاية من هذا النوع من الصفوف هو توفير صف عام يمكن تخصيصه لحل مسائل في مجال معين.

ولنفرض جدلاً أنه من الممكن إنشاء كائن من الصف المجرد Shape السابق:

```
Shape s;   ✓
s = new Shape();   X
```

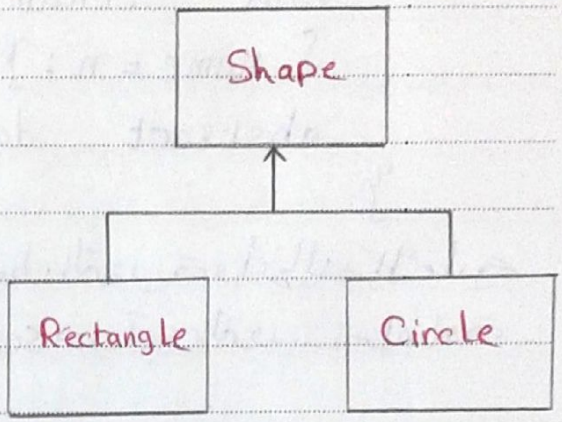
وفي هذه الحالة فالكائن s فيه مجموعة من الدوال ويمكن استدعاءها، ولكن استدعاء الطريقة surface هنا لا معنى له لأنها لا تحوي تعليمات برمجية أصلاً.
ولكن بالطبع يمكن التصريح عن كائن من نوع الصف المجرد دون إنشائه.

- يجب تخصيص الصف المجرد من خلال مفهوم الوراثة قبل استخدامه، ويجب على الصف الابن (الذي يخصص الصف المجرد) كتابة تعليمات تنفيذية لكل طريقة مجردة معرفة في الصف المجرد.

ملاحظة: يمكن للصف الوارث من صف مجرد ألا يقوم بإعادة تعريف طريقة مجردة ما ولكنه في هذه الحالة سيكون هو أيضاً على طريقة مجردة كونه يرث مكونات الصف الأب وبالتالي في هذه الحالة سيصبح الصف الوارث أيضاً "مجرداً".

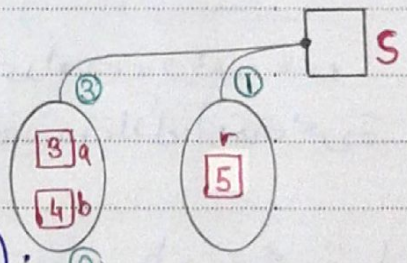
مثال: لنقم بتخصيص الصنف المكتوب في المثال السابق:

```
class Circle extends Shape{
   double r;
   Circle (double r) {
      this.r=r; }
   double surface () {
      return (3.14*r*r);
    }
  }
class Rectangle extends Shape{
   double a,b;
   Rectangle (double x, double y)
    {a=x ; b=y ;}
   double surface () {
      return (a*b);
    }
  }
```



- الآن يمكننا أن نكتب البرنامج التالي:

```
class UseShape{
  static void main (String args[])
   { Shape S;
     S=new Circle(5); ①
     System.out.print (S.surface()); ②
     S=new Rectangle(3,4); ③
     System.out.print (S.surface()); ④
  }}
```



- لقد قمنا في السطر ① بإنشاء كائن من النوع Circle ، وإسناده إلى المتحول s من النوع العام Shape.
- في السطر ② تم استدعاء الطريقة surface الموجودة في s ، ولكن s فعلياً الآن من النوع Circle ، لذا تم استدعاء الطريقة الموجودة في الصف Circle. وهنا ستكون نتيجة الطباعة هي قيمة 3.14*5*5.
- في السطر ③ تم إنشاء كائن من النوع Rectangle ، وإسناده إلى المتحول s من النوع العام Shape.
- في السطر ④ على نفس المنوال سنجد أنه سيتم استدعاء الطريقة surface الموجودة في الصف Rectangle ، وستكون نتيجة الطباعة هنا هي قيمة 3*4.

مفهوم تعدد الأشكال: Interchangeable Object with Polymorphism

عندما ننشئ هرماً وراثياً فإننا لا نريد استخدام متحولات من الأنواع المخصصة (الأبناء) بل نريد استخدام متحولات عامة من نوع الأب. لأن ذلك يعزز كتابة تعليمات برمجية عامة لا تتغير مع تغير الأنواع.

أي نريد للطرق التي تعمل على الأنواع العامة ألا تتغير بتغير النوع أو بإضافة نوع جديد إلى الهرم الوراثي. من شأن ذلك أن يسمح بكتابة طرق عامة تعمل على جميع الأنواع المخصصة دون الحاجة إلى إجراء أي تغيير فيها.

مثال: إذا كانت لدينا مثلاً الصفوف المكتوبة في المثال الأخير، ولنناقش المقطع البرمجي التالي:

```
void Redo(Shape l){
    System.out.print(l.surface());
}
```

عند ترجمة هذا المقطع البرمجي فإن المترجم لا يعرف مسبقاً أية surface() سيتم استدعاؤها.

بمعنى آخر لا توجد أية معلومات لدى المترجم أثناء مرحلة الترجمة فيما إذا كان سيتم استدعاء الدالة surface() من الصف Circle أو الصف Rectangle.

ولكن المهم بالنسبة إليه أن هذه الدالة ستأخذ وسيطاً وهو عبارة عن كائن من النوع Shape وتستدعي الدالة الموافقة لنوعه الفعلي.

/ /

تسمى القدرة على تحديد أية طريقة تستدعى أثناء التنفيذ وليس أثناء الترجمة خاصة تعدد الأشكال Polymorphism.

ومعنى هذه الخاصة أنه عند استدعاء الطريقة surface() على كائن من الصف العام Shape فإن هذا الصف يغير شكله اعتماداً على النوع المخصص الذي أسند إليه، أي إنه يتصرف تارة وكأنه من النوع المخصص Circle وتارة أخرى يتصرف وكأنه Rectangle.

- من المهم أن نلاحظ أن الاستدعاء l.surface() ليس من الشكل:
{ إذا كان l من نوع Circle عندئذٍ تصرف كـ Circle.
وإذا كان l من نوع Rectangle عندئذٍ تصرف كـ Rectangle.
بالواقع لو كان الأمر كذلك لما استطعنا كتابة الطريقة Redo بشكل عام، بل لتوجب علينا إضافة العبارة if كلما أضفنا نوعاً خاصاً جديداً.
ولكن خاصة تعدد الأشكال تجعل الطريقة Redo لا تحتاج إلى أي تغيير مهما أضفنا أنواعاً خاصة من النوع Shape.