Programmation orientée objet
Héritage

BTS SIO 1 – SLAM2

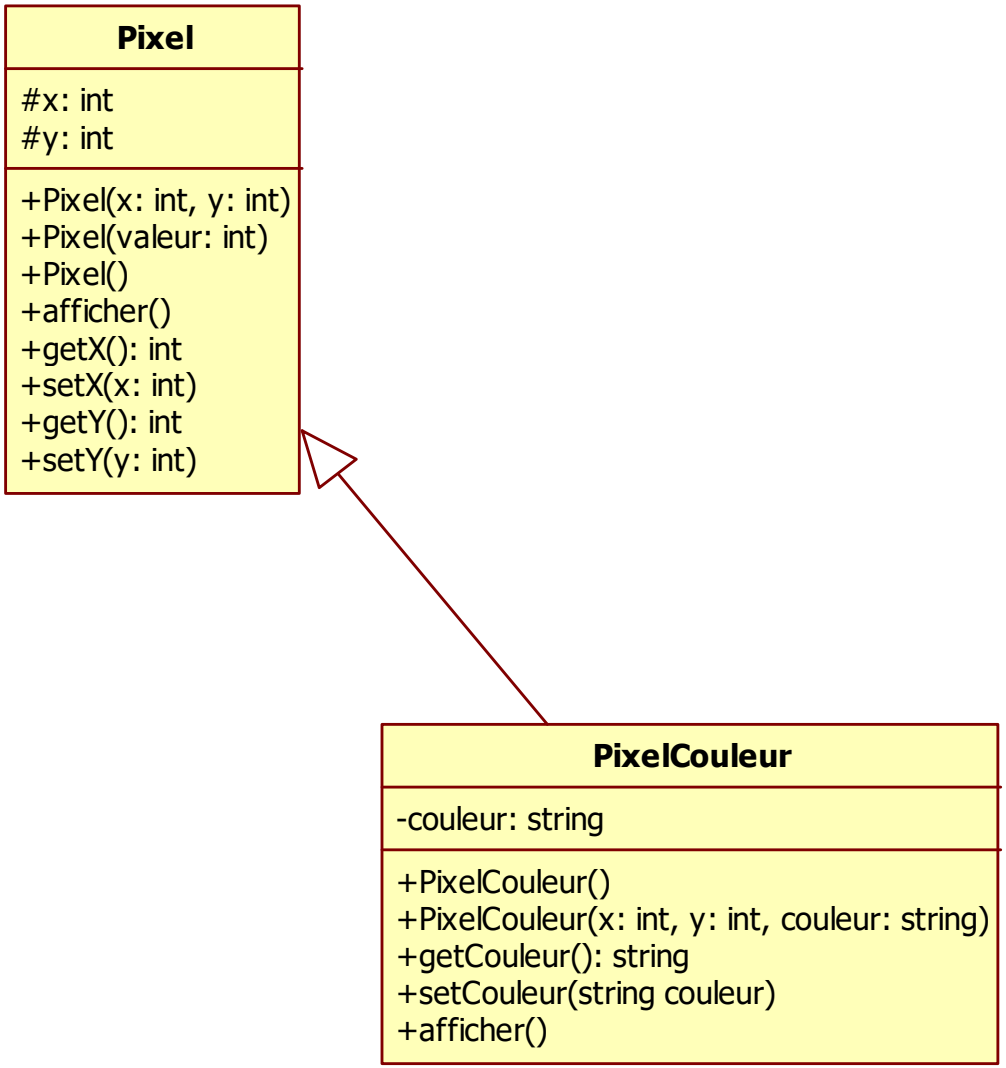
Programme

- Programmation Orientée Objet (POO)
  - Héritage
  - Polymorphisme
  - Transtypage

Héritage

- On peut hériter une classe d'une classe mère.
- La classe fille pourra utiliser les propriétés et méthodes de la classe mère comme ses propres propriétés.
- L'héritage permet de construire des hiérarchies de classe.

Exemple en UML



Exemple 1/2

```
class Pixel{
   protected int x , y ;
   public Pixel( int x , int y ) {
        this.x = x ;
        this.y = y ;
   }
   public void afficher () {
        System.out.println (“Pixel :(”+this.x+”,”+this.y+”)”);
   }
   public int getX() {
         return this.x;
   }
   public void setX(int x) {
         this.x = x;
   }…
}
```

→ Les attributs sont désormais protégés pour permettre à la classe fille d’y avoir accès.

Exemple 2/2

```
class PixelCouleur extends Pixel{
   private String couleur;
   public PixelCouleur( int x , int y, String couleur ) {
        super(x,y); // appel du constructeur de la classe mère
        this.couleur = couleur ;
   }
   public void afficher () {
        super.afficher(); //appel de la méthode de la mère
        System.out.println("et ma couleur est" + this.couleur);
   }
   public String getCouleur() {
         return this.couleur;
   }
   public void setCouleur(String couleur) {
         this.couleur = couleur;
   }…
}
```

Visibilité – Objets filles

- Les objets qui héritent d'une classe ont accès :
  - aux propriétés et méthodes public.
  - aux propriétés et méthodes protected.
- Et n'ont pas accès
  - aux propriétés et méthodes private.
- Le mot clef protected limite donc l'accès aux objets qui l'hérite.
- On peut noter que le modificateur protected (en plus de private et public) permet de rendre accessible les objets à d'autres classes au sein d'un héritage.

Visibilité – objets non liés

- Pour rappel, les objets qui n'héritent pas d'une classe ont accès :
  - aux propriétés et méthodes public.
- Et n'ont pas accès
  - aux propriétés et méthodes private.
  - aux propriétés et méthodes protected (sauf s'ils sont dans le même package).
- Le respect de l'encapsulation grâce au accesseurs et modificateurs implique que les propriétés sont déclarées à private.
- On ne peut donc plus accéder directement à celle-ci en dehors de la classe même avec l'héritage.

Héritage en java

- En java, toutes les classes sont dérivées de la classe Object.
- Elles héritent par défaut de cette classe sans spécifier « extends Object ».
- En java, on ne peut hériter que d'une seule classe.

Redéfinition de méthode

- La redéfinition est le fait de déclarer une méthode avec le même nom et les mêmes paramètres mais dans une classe qui l'hérite.

```
class PixelCouleur extends Pixel{
   …
   @Override // annotation pour valider la redéfinition
   public void afficher () {
        super.afficher(); //appel de la méthode de la mère
        System.out.println("et ma couleur est" + this.couleur);
   }
   …
}
```

La méthode `toString()` d'`Object`

```
class Pixel {
   …
   @Override // annotation pour valider la redéfinition
   public String toString () {
         return "Pixel :("+this.x+","+this.y+")";
   }
   …
}
```

```
class PixelCouleur extends Pixel{
   …
   @Override // annotation pour valider la redéfinition
   public String toString () {
        return super.toString()+ "et ma couleur est" + this.couleur;
   }
   …
}
```

Le mot clef `Final`

- Classe
  - Une classe final est une classe qui ne peut pas avoir de filles.
  - Une classe final ne peut pas être étendue: le mécanisme d'héritage est bloqué.
  - Mais une classe final peut évidemment être la fille d'une autre classe.

    Exemple : `public final class PixelCouleur{}`
- Méthode
  - Une méthode final est une méthode qui ne peut pas être redéfinie dans les sous-classes.
- Attribut
  - Le mot clef final pour un attribut ou une variable bloque la modification de sa valeur (constante).

    Exemple public final float PI =3.14;

Le mot clé abstract

- Méthode abstraite
  - Une méthode abstraite ne contient pas de définition, uniquement sa signature.
  - Une méthode abstraite ne peut pas être déclarée static ou private ou final.
  - Dès qu'une classe contient une méthode abstraite, elle doit elle aussi être déclarée abstraite, avec le modificateur abstract placé au début de son en-tête.

```
public abstract class ObjetGraphique{

   public abstract double getPerimetre() ;

}
```

Le mot clé `abstract`

- Classe abstraite
  - Une classe abstraite ne peut être instanciée.

```
public abstract class ObjetGraphique{}
…
public static void main(String[] args){
  ObjetGraphique og=new ObjetGraphique() ;
}
```

  - Si une classe l'étend, il faudra définir toutes les méthodes abstraites qu'elle contient pour pouvoir l'utiliser.
  - Une sous-classe d'une classe abstraite sera encore abstraite si elle ne définit pas toutes les méthodes abstraites dont elle hérite.

Le transtypage – type de base

- Le transtypage (conversion de type ou cast en anglais) consiste à modifier le type d'une variable ou d'une expression.
- Par exemple, il est possible transtyper un int en double :

```
double d; int i;
d=3.4 ;
i = (int) d; //transtypage d'un double en int
```

Le transtypage – référence d'objets

- Il est possible de convertir un objet d'une classe en un objet d'une autre classe si les classes ont un lien d'héritage
- Transtypage implicite
  - On imagine une classe Capitale qui hérite de Ville.

```
Ville v;
Capitale c = new Capitale("Paris", "France");
v = c; //transtypage implicite d'une Capitale en Ville
```

- Transtypage explicite
  - Toutefois, une ville n'est pas une capitale par défaut. Si on souhaite la transformer en capitale, il faut faire un transtypage ou un cast explicite.

```
Ville v = new Ville() ;
Capitale h = (Capitale)v; //transtypage d'une Ville en Capital
```

→ Attention, on peut déclencher un **`ClassCastException`** si la conversion explicite est incohérente (C'est le cas de l'exemple ci-dessus) ;-)