Introduction à la programmation orientée objet

BTS SIO 1 – SLAM2

Programme

- Programmation Orientée Objet (POO)
  - Langage Java
  - Formalisme UML
- L'objet vs la classe
  - Attributs et méthodes
  - Constructeur
  - Encapsulation

Programmation orientée objet (Poo)

- Les objectifs :
  - Faciliter le développement et l'évolution des applications;
  - Permettre le travail en équipe;
  - Augmenter la qualité des logiciels (moins de bugs).
- Solutions proposées :
  - Découpler (séparer) les parties des projets;
  - Limiter (et localiser) les modifications lors des évolutions;
  - Réutiliser facilement du code.

Java

- Le langage Java :
  - est un langage de programmation orienté objet
  - créé par James Gosling et Patrick Naughton (Sun)
  - présenté officiellement le 23 mai 1995.
- Les objectifs de Java :
  - simple, orienté objet et familier;
  - robuste et sûr;
  - indépendant de la machine employée pour l'exécution;
  - très performant;
  - interprété, multitâches et dynamique.

Autres langages orienté objet

- C++ : très utilisé
- C# : langage de Microsoft (appartient à .NET)
- Objective C : langage utilisé par Apple
- PHP : langage très utilisé sur le Web
- Python
- Ruby
- Ada
- Smalltalk
- ...

La syntaxe change mais le concept objet est le même!

UML

- UML = Unified Modeling Language.
- UML est un langage de modélisation graphique.
- UML est apparu dans le cadre de la “conception orientée objet”.
- UML propose 13 types de diagrammes qui permettent la modélisation d’un projet durant tout son cycle de vie.
- Nous allons nous intéresser aux diagrammes de classes.

Diagramme de classes

- Schéma utilisé pour représenter les classes et les interfaces.
- On représente également les interactions entre les classes.
- On peut écrire notre programme à partir du diagramme de classes.
- Il permet de réfléchir à la structure du programme.
- Il permet de décrire la structure du programme.

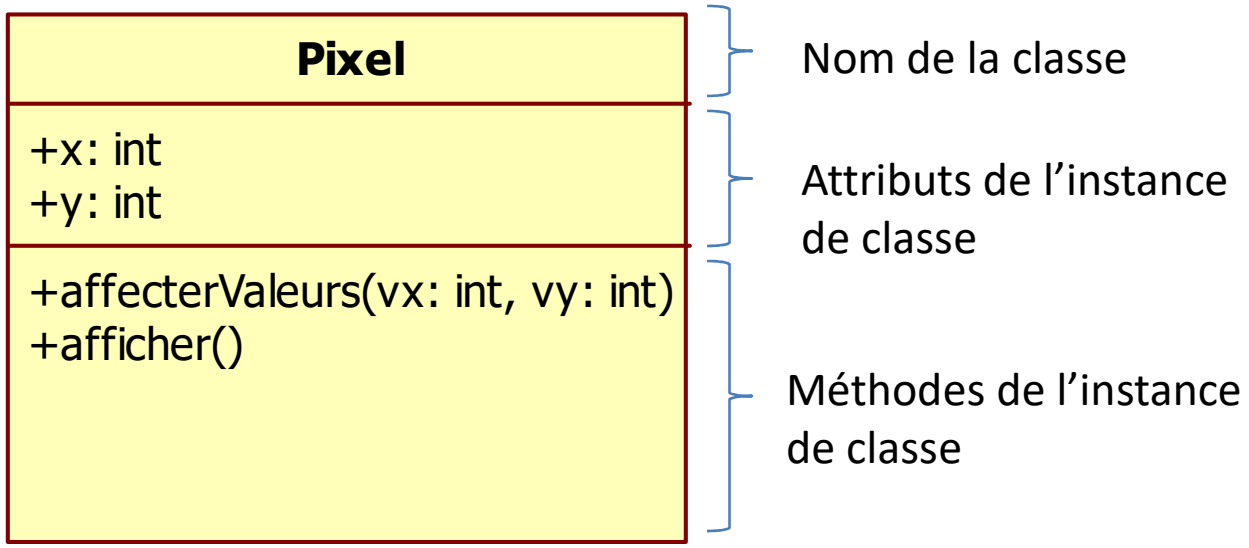
L'Objet vs la Classe

- Un objet :
  - rend un ensemble de services (interface de l'objet)
  - contient des données (état de l'objet)

- Une classe est un « moule » pour fabriquer des objets. Elle :
  - définit les méthodes (c'est-à-dire les services)
  - décrit la structure des données avec les attributs

→ Un objet créé à partir d'une classe A est une instance de la classe A.

Représenter une classe en UML



Définir une classe en java

```
class Pixel{
   int x , y ; // attributs de l'objet
   // méthode de l'objet
   void affecterValeurs ( int vx, int vy ) {
       x = vx ;
       y = vy ;
   }
   //autre méthode de l'objet
   void afficher () {
       System.out.println ("Pixel :("+x+","+y+")");
   }
}
```

Créer une instance en java

- Création d'une instance avec le mot-clé ***new*** :

```
new Pixel ();
```

- Cette expression renvoie une référence vers l'instance créée par ***new*** . Elle peut être affectée à une variable de type **`Pixel`**.

- Déclaration d'une variable de type **`Pixel`**:

```
Pixel point;
```

- Affectation de la référence à la variable :

```
point = new Pixel ();
```

Utilisation des méthodes et accès aux données

- On accède aux données et aux méthodes avec l'opérateur"."(point) :

```
class ProgrammePrincipal {
   public static void main( String arg [ ] ) {
       Pixel point = new Pixel ();
       point.affecterValeurs (10 ,30);
       point.afficher();
       System.out.println (point.x ); // affiche 10.
       System.out.println (point.y ); // affiche 30.
   }
}
```

→ En Java, le compilateur vérifie que **le membre** (méthode ou attribut) existe en utilisant le type de la référence.

Référence **`null`**

- Les variables de type référence contiennent la valeur `null` par défaut.
- La valeur `null` signifie que la référence ne pointe vers aucune instance.

- L'affectation de la valeur `null` à une variable :

```
Pixel point = null ;
```

- L'utilisation d'une méthode (ou d'une donnée) à partir d'une variable de type référence ayant une valeur `null` provoque une erreur à l'exécution :

```
Pixel point = null ;
point.affecterValeurs (10 ,30);
```

→ Exception in thread "main" java.lang.NullPointerException at ProgrammePrincipal.main(ProgrammePrincipal.java:x)

Destruction d'une instance

- En Java, la destruction des instances est assurée par le Garbage collector.
- Lorsqu'une instance n'est plus accessible à partir des variables, elle peut être détruite automatiquement par le Garbage collector.

```
class ProgrammePrincipal {
   public static void main( String arg [ ] ) {
       Pixel point = new Pixel();
       point.affecterValeurs (10 ,30);
       point.afficher();
       point = null ; // L'instance créée par le new
                         peut être détruite
       …
   }
}
```

→ conclusion: une référence contient "null" ou désigne un objet

Constructeur

```
class Pixel{
   int x , y ;
   Pixel( int vx , int vy ) {
        x = vx ;
        y = vy ;
   }
   Pixel ( int valeur ) {
        x = valeur ;
        y = valeur ;
   }
   Pixel () { // constructeur par défaut !!!
        x = 0 ;
        y = 0 ;
   }

   void afficher () {
        System.out.println (“Pixel :(”+x+”,”+y+”)”);
   }
}
```

Utilisation du constructeur

```
class ProgrammePrincipal {
  public static void main( String arg [ ] ) {
      Pixel point1 = new Pixel ();
      Pixel point2 = new Pixel (10);
      Pixel point3 = new Pixel (12,75);
      point1.afficher(); //affiche “Pixel (0,0)”
      point2.afficher(); //affiche “Pixel (10,10)”
      point3.afficher(); //affiche “Pixel (12,75)”
  }
}
```

→ Si une classe ne définit pas de constructeurs, seul le constructeur par défaut sera disponible pour initialiser l’objet

Mot-clé **this**

```
class Pixel{
   int x , y ;
   Pixel( int x , int y ) {
        this.x = x ;
        this.y = y ;
   }
   …

   void afficher () {
        System.out.println (“Pixel :(”+this.x+”,”+this.y+”)”);
   }
}
```

→ Le mot clé **this** sert à référencer l'objet courant de l'intérieur de la classe

Encapsulation

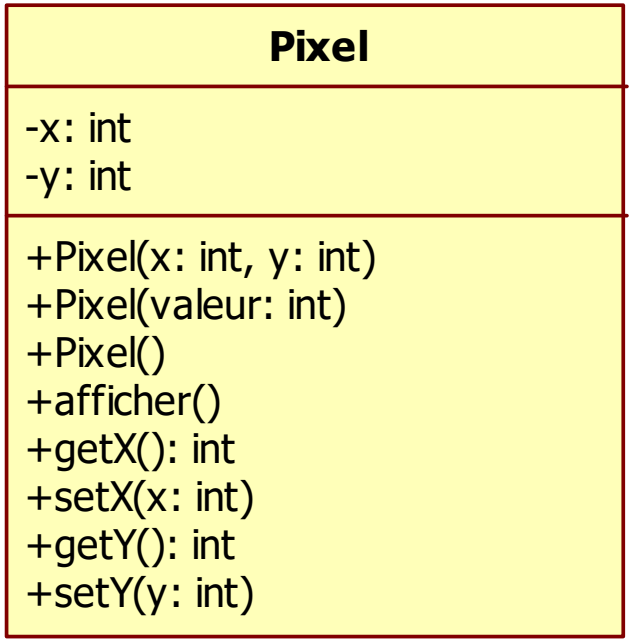
- L'encapsulation consiste à gérer la classe comme une boite noire responsable de sa propre cohérence.
- Pour y parvenir, il est possible de définir la visibilité de chacun des membres d'une classe.
- Un membre peut être :
  - `public` (+): visible par tous
  - `private` (-): visible uniquement des membres de la classe
  - `protected` (#) : utile lors de l'héritage
  - `rien` : (~) visibilité par défaut, uniquement au niveau package

Encapsulation

```
class Pixel{
   private int x , y ;
   public Pixel( int x , int y ) {
        if(x>=0) this.x = x ; // on préserve la cohérence
        if(y>=0) 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) {
         if(x>=0) this.x = x; // on préserve la cohérence
   }…
}
```

→ Les attributs étant désormais privés, nous devons ajouter des méthodes publiques pour accéder et modifier leur contenu.

Pixel en UML


Pixel
-x: int
-y: int
+Pixel(x: int, y: int)
+Pixel(valeur: int)
+Pixel()
+afficher()
+getX(): int
+setX(x: int)
+getY(): int
+setY(y: int)

Exercice

- Ajouter les deux fonctionnalités suivantes à la classe Pixel :
  - Déplacer en x,
  - Déplacer en y,

  → Ecrire une fonction main pour tester les nouvelles méthodes via un objet Pixel.