Contenu du cours

1. Introduction à la POO
2. Une **classe** : définition de nouveaux objets
3. **Instanciation** et utilisation d'objets
4. Création des objets : les **constructeurs**
5. **Références**, visibilité des variables
6. **Encapsulation** et masquage des données
7. **Statique**, ou d'instance ?
8. **Héritage**
9. **Polymorphisme**
10. **Classes abstraites** et **interfaces**
11. **Introduction aux types génériques**
12. **Exceptions** en java
13. *Compléments syntaxiques*

Introduction aux types génériques

Notion de « programmation générique »

- **Programmation générique**

On parle de programmation générique lorsqu'un langage permet d'écrire un code source unique utilisable avec des objets ou des variables de types quelconques.

- **Plusieurs interprétations possibles**
  1. **Le type est effectivement quelconque**

Utilisation du type java Object (car toute classe hérite de Object)

  2. **Le type n'est pas connu à l'écriture de la classe ou de la méthode**

Précision du type différée au moment de l'instanciation de la classe, ou de l'appel de la méthode

Cette dernière possibilité a été introduite depuis java 1.5 : on parle de classes ou méthodes **génériques**

Notion et création de classe générique

- **Paramètrer** par un type
  - Le **type** est considéré comme un **paramètre de la classe**
  - Nommé de manière abstraite (ex. **T**, **E**), et noté entre chevrons (ex. **<T>**)
- Exemple
  - Un couple d'objets quelconques, l'un est le **premier**, l'autre est le **second**

On ne sait pas ce qu'est T

T est utilisé dans le code syntaxiquement comme un type normal

```
public class Pair<T> {
    private T e1, e2;

    Pair(T e1, T e2) {
        this.e1 = e1; this.e2 = e2;
    }
    public T getFirst() {
        return e1;
    }
    public T getLast() {
        return e2;
    }
}
```

Utilisation d'une classe générique

- **Déclaration / instanciation**
  - **On précise entre les chevrons le type que l'on souhaite utiliser**
  - **Par exemple : Pair<Integer>, ou Pair<NombreComplexe>, etc.**

Le type précisé doit obligatoirement être un type objet

- **Exemple de programme**

```
public class Main {
    public static void main(String[] args) {
        Pair<Integer> pi = new Pair<Integer>(5, 12);
        System.out.println("Premier : "+pi.getFirst()+"Second : "+pi.getLast());

        Pair<NombreComplexe> pc = new Pair<NombreComplexe>(
                new NombreComplexe(1, 1),
                new NombreComplexe(1, -1));
        System.out.println("Premier : "+pc.getFirst()+"Second : "+pc.getLast());
    }
}
```

On peut utiliser plusieurs types génériques

- Nommer un type générique
  - Tout identificateur convient
  - En pratique, souvent une seule lettre (majuscule, ex <T> ou <E>), ou une lettre et un chiffre (ex <T1> ou <T2>
- Exemple : un couple d'objets mixtes

```
public class MixedPair<T1, T2> {
    private T1 e1; private T2 e2;

    MixedPair(T1 e1, T2 e2) {
        this.e1 = e1; this.e2 = e2;
    }
    public T1 getFirst() {
        return e1;
    }
    public T2 getLast() {
        return e2;
    }
}
```

Utilisation : exemple

- **Exemple : couple à deux types**

```
public class GoGenerics {
    public static void main(String[] args) {
        Pair<Integer> pi = new Pair<Integer>(5, 12);
        System.out.println("Premier : "+pi.getFirst()+"Second : "+pi.getLast());

        Pair<NombreComplexe> pc = new Pair<NombreComplexe>(
                new NombreComplexe(1, 1),
                new NombreComplexe(1, -1));
        System.out.println("Premier : "+pc.getFirst()+"Second : "+pc.getLast());

        MixedPair<Integer, NombreComplexe> mp =
                    new MixedPair< Integer, NombreComplexe> (5, new NombreComplexe(2, 1));
        System.out.println("Premier : "+mp.getFirst()+"Second : "+mp.getLast());

    }
}
```

- **Les génériques évitent beaucoup de transtypage**
- **Très utilisés avec les Collections**

Exercice (non corrigé) : entraînez-vous à décrire une pile (LIFO) et une file (FIFO) génériques

Reprendre l'exemple d'une méthode compareTo() en version générique ou non