ENSET
ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Programmation Orientée Objet

La généricité en Java

Mme Hassna BENSAG
Email: **h.bensag@gmail.com**

Qu'est-ce que la généricité ?

- La généricité est un principe java qui permet de créer des classes des méthodes, et des interfaces qui fonctionnent avec différents types de données.
- Une entité telle qu'une classe, une interface ou une méthode qui opère sur un type paramétré est appelée une entité générique.
- La généricité consiste à structurer les méthodes et l'information de manière à ce qu'elle soit indépendante d'un type. Par exemple, créer un vecteur pouvant contenir des valeurs réelles ou complexes.
- La généricité est introduite en Java 1.5.
- Avantage de de la généricité et la réutilisation du code : nous pouvons écrire une méthode/classe/interface une fois et l'utiliser pour n'importe quel type que nous voulons.

Définir une classe générique

- Afin de définir un type générique pour une classe il faut suivre son nom par un identifiant qui doit être placé entre "<" et ">".

Pair.java

```java
package generic;

public class Pair <T>{  5 usages
    private T premier;  3 usages
    private T second;  3 usages
    public Pair(T premier, T second){  2 usages
        this.premier=premier;
        this.second=second;
    }

    public T getPremier() {  2 usages
        return premier;
    }

    public T getSecond() {  2 usages
        return second;
    }

    public void setSecond(T second) {  no usages
        this.second = second;
    }

    public void setPremier(T premier) {  no usages
        this.premier = premier;
    }

}
```

Instancier une classe générique

- Instancier une classe générique consiste à donner une valeur à la (ou les) variable(s) de type :

Main.java

```java
package app;

import generic.Pair;

//TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or
// click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.
public class Main {
    public static void main(String[] args) {
        Pair <Integer> p1= new Pair<>( premier: 2, second: 13);
        System.out.println(p1.getPremier()+" , "+p1.getSecond());
        Pair <String> p2= new Pair<>( premier: "Bonjour", second: " tout le monde");
        System.out.println(p2.getPremier()+" , "+p2.getSecond());
    }
}
```

Étendre une classe générique

- On peut créer une classe qui hérite d'une classe générique. Dans ce cas, nous avons deux scénarisons, soit la classe fille est aussi générique (exemple 1) ou non générique (exemple 2).

Triplet.java

```java
package generic;

public class Triplet <T> extends Pair{  3 usages
    private T troisieme;  3 usages

    public Triplet(T premier, T second,T troisieme){  1 usage
        super(premier,second);
        this.troisieme=troisieme;
    }

    public T getTroisieme() {  1 usage
        return troisieme;
    }

    public void setTroisieme(T troisieme) {  no usages
        this.troisieme = troisieme;
    }
}
```

Exemple 1

Triplet.java

```java
package generic;

public class Triplet extends Pair{  3 usages
    private Double troisieme;  3 usages

    public Triplet(String premier, Integer second,Double troisieme){  1 usage
        super(premier,second);
        this.troisieme=troisieme;
    }

    public Double getTroisieme() {  1 usage
        return troisieme;
    }

    public void setTroisieme(Double troisieme) {  no usages
        this.troisieme = troisieme;
    }
}
```

Exemple 2

Définir des contraintes sur un type générique

- On peut définir des contraintes sur le type génique. on peut préciser qu'un type générique hérite d'une classe ou d'une (ou plusieurs) interface(s).
- Dans l'exemple suivant on précise que le type T doit hériter de la classe Document et implémente l'interface Empruntable.

GestDictionnaire.java

```
1  package generic;
2
3  public class GestDictionnaire <T extends Document & Empruntable>{
4      private T dictionnaire;  4 usages
5
6      public GestDictionnaire(T dictionnaire){  no usages
7          this.dictionnaire=dictionnaire;
8      }
```

Définir une méthode générique

- Une méthode peut être paramétrée par un type, qu'elle soit dans une classe générique ou non.

Calcul.java

```java
package generic;

public class Calcul {  no usages

    public <T> boolean comparer(T a, T b){  no usages
        return  a.equals(b);
    }
}
```

```java
    Calcul calcul= new Calcul();
    int a=3;
    int b=3;
    System.out.println(calcul.comparer (a,b));
```

Définir une interface générique

- En java on peut définir une Interface qui a des paramètres génériques

Comparateur.java

```
1  package generic;
2
3  public interface Comparateur<T> {  1 usage  1 implementation
4
5          int comparer(T a, T b);  no usages  1 implementation
6
7  }
8
```

Exemple d'interface générique

ComparateurEntier.java

```
1  package generic;
2
3  public class ComparateurEntier implements Comparateur <Integer>{  2 usa
4      @Override  1 usage
5      public int comparer(Integer a, Integer b) {
6          return a.compareTo(b);
7      }
8  }
9
```

Exemple d'implémentation de l'interface générique