

Java : généricité



Plan

- 1 Introduction
- 2 Premier exemple
- 3 Exemple avec deux types génériques
- 4 Exemple avec plusieurs attributs avec le même type générique
- 5 Exemple avec l'héritage

Le concept objet

- Une classe est composée d'un ensemble d'attributs + un ensemble de méthodes
- Les attributs d'une classe ont **forcément** un nom et un type (*Java est fortement typé*)
- Le type d'attribut ne change pas pour toutes les instances d'une classe
- Les méthodes permettent généralement d'effectuer des opérations sur les attributs tout en respectant leurs spécificités (type, taille...)

Problématique

- Et si on a besoin d'une classe dont les méthodes effectuent les mêmes opérations qu'un type d'attributs
 - **somme** pour **entiers** ou **réels**,
 - **concaténation** pour **chânes de caractères**,
 - **ou logique** pour **booléens**...
 - ...
- **Impossible sans définir plusieurs classes (une pour chaque type) et une interface ou en imposant le type `Object` et en utilisant plusieurs cast...**

Depuis Java 5, une solution plus élégante avec la généricité

- Ne pas définir de type pour les attributs
- Définir un type générique qui ne sera pas précisé à la création de la classe
- d'instanciation de la classe, on précise le type à utiliser par cette classe
- **On peut donc choisir pour chaque instance le type que l'on souhaite utiliser.**

Java

Une première classe avec un type générique

```
public class Exemple <T> {  
    private T var;  
  
    public T getVar() {  
        return var;  
    }  
  
    public void setVar(T var) {  
        this.var = var;  
    }  
}
```

Java

Créons des instances de la même classe avec des types différents

```
public static void main(String[] args) {  
    Exemple<Integer> entier = new Exemple<Integer>();  
    entier.setVar(10);  
    System.out.println( entier.getVar().getClass().  
        getTypeName() + " " + entier.getVar());  
    Exemple<String> chaine = new Exemple<String>();  
    chaine.setVar("Bonjour");  
    System.out.println( chaine.getVar().getClass().  
        getTypeName() + " " + chaine.getVar());  
}
```

Java

Créons des instances de la même classe avec des types différents

```
public static void main(String[] args) {  
    Exemple<Integer> entier = new Exemple<Integer>();  
    entier.setVar(10);  
    System.out.println( entier.getVar().getClass().  
        getTypeName() + " " + entier.getVar());  
    Exemple<String> chaine = new Exemple<String>();  
    chaine.setVar("Bonjour");  
    System.out.println( chaine.getVar().getClass().  
        getTypeName() + " " + chaine.getVar());  
}
```

Le résultat affiché :

```
java.lang.Integer 10  
java.lang.String Bonjour
```


Java

Exemple avec deux types génériques

```
public class Exemple <T,S> {  
    private T var1;  
    private S var2;  
    public T getVar1() {  
        return var1;  
    }  
    public void setVar1(T var1) {  
        this.var1 = var1;  
    }  
    public S getVar2() {  
        return var2;  
    }  
    public void setVar2(S var2) {  
        this.var2 = var2;  
    }  
}
```

Java

Testons ça

```
public static void main(String[] args) {  
    Exemple<Integer,String> couple = new Exemple<  
        Integer,String>();  
    couple.setVar1(10);  
    couple.setVar2("Bonjour");  
    System.out.println(couple.getVar1().getClass().  
        getTypeName() + " " + couple.getVar1());  
    System.out.println(couple.getVar2().getClass().  
        getTypeName() + " " + couple.getVar2());  
}
```

Java

Testons ça

```
public static void main(String[] args) {  
    Exemple<Integer,String> couple = new Exemple<  
        Integer,String>();  
    couple.setVar1(10);  
    couple.setVar2("Bonjour");  
    System.out.println(couple.getVar1().getClass().  
        getTypeName() + " " + couple.getVar1());  
    System.out.println(couple.getVar2().getClass().  
        getTypeName() + " " + couple.getVar2());  
}
```

Le résultat affiché :

```
java.lang.Integer 10  
java.lang.String Bonjour
```

Java

Exemple avec plusieurs attributs

```
public class Operation <T>{  
    private T var1;  
    private T var2;  
    public Operation(T var1, T var2) {  
        this.var1 = var1;  
        this.var2 = var2;  
    }  
    public void plus () {  
        if (var1.getClass().getSimpleName().equals("Double"))  
            System.out.println(Double.parseDouble(var1.toString()) + Double.  
                parseDouble(var2.toString()));  
        else if (var1.getClass().getSimpleName().equals("Integer"))  
            System.out.println(Integer.parseInt(var1.toString()) + Integer.  
                parseInt(var2.toString()));  
        else if (var1.getClass().getSimpleName().equals("Boolean"))  
            System.out.println( Boolean.parseBoolean(var1.toString())  
                ||  
                Boolean.parseBoolean(var2.toString()));  
        else  
            System.out.println( var1.toString() + var2.toString());  
    }  
}
```

Java

Testons cela

```
public static void main(String[] args) {  
    Operation <Integer> operation1 = new Operation<Integer>(5,3);  
    operation1.plus();  
    Operation <String> operation2 = new Operation<String>("bon","jour");  
    operation2.plus();  
    Operation <Double> operation3 = new Operation<Double>(5.2,3.8);  
    operation3.plus();  
    Operation <Boolean> operation4 = new Operation<Boolean>(true,false);  
    operation4.plus();  
}
```

Java

Testons cela

```
public static void main(String[] args) {  
    Operation <Integer> operation1 = new Operation<Integer>(5,3);  
    operation1.plus();  
    Operation <String> operation2 = new Operation<String>("bon","jour");  
    operation2.plus();  
    Operation <Double> operation3 = new Operation<Double>(5.2,3.8);  
    operation3.plus();  
    Operation <Boolean> operation4 = new Operation<Boolean>(true,false);  
    operation4.plus();  
}
```

Le résultat affiché :

```
8  
bonjour  
9.0  
true
```

Considéons la classe `Personne` suivante

```
package org.eclipse.mod

; public class Personne {
    private String nom;
    private String prenom;

    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    @Override
    public String toString() {
        return "Personne [nom=" + nom + ", prenom=" + prenom + "];"
    }
}
```

Java

La classe `Etudiant` qui hérite de la classe `Personne`

```
package org.eclipse.mod    ;

public class Etudiant extends Personne {

    private String niveau;

    public String getNiveau() {
        return niveau;
    }
    public void setNiveau(String niveau) {
        this.niveau = niveau;
    }
    @Override
    public String toString() {
        return "Etudiant [niveau=" + niveau + ", toString()=" +
            super.toString() + "]\n";
    }
}
```


Java

La classe Vehicule

```
package org.eclipse.mod    ;

public class Vehicule {

    private int marque;

    public int getMarque() {
        return marque;
    }

    public void setMarque(int marque) {
        this.marque = marque;
    }
    @Override
    public String toString() {
        return "Vehicule [marque=" + marque + "]";
    }
}
```

Java

La classe Humain

```
package org.eclipse.mod    ;

public class Humain <T> {

    private T var;

    public T getVar() {
        return var;
    }

    public void setVar(T var) {
        this.var = var;
    }
}
```

Java

En testant le `main` suivant, aucun message d'erreur n'est signalé

```
package org.eclipse.test;

import org.eclipse.mod
.Etudiant; import org.eclipse.mod
.Humain; import org.eclipse.mod
.Personne; import org.eclipse.mod
.Vehicule;

public class Main {
    public static void main(String[] args) {
        Humain<Personne> humain = new Humain();
        Humain<Etudiant> humain2 = new Humain();
        Humain<Vehicule> humain3 = new Humain();
    }
}
```

Java

Modifions la classe Humain

```
package org.eclipse.mod    ;

public class Humain <T extends Personne> {

    private T var;

    public T getVar() {
        return var;
    }

    public void setVar(T var) {
        this.var = var;
    }
}
```

Java

On ne change rien dans le main

```
package org.eclipse.test;

import org.eclipse.mod
.Etudiant; import org.eclipse.mod
.Humain; import org.eclipse.mod
.Personne; import org.eclipse.mod
.Vehicule;

public class Main {
    public static void main(String[] args) {
        Humain<Personne> humain = new Humain();
        Humain<Etudiant> humain2 = new Humain();
        Humain<Vehicule> humain3 = new Humain();
    }
}
```

Java

On ne change rien dans le main

```
package org.eclipse.test;

import org.eclipse.mod
.Etudiant; import org.eclipse.mod
.Humain; import org.eclipse.mod
.Personne; import org.eclipse.mod
.Vehicule;

public class Main {
    public static void main(String[] args) {
        Humain<Personne> humain = new Humain();
        Humain<Etudiant> humain2 = new Humain();
        Humain<Vehicule> humain3 = new Humain();
    }
}
```

La dernière instruction sera soulignée en rouge car `Vehicule` n'hérite pas de la

Pour les collections

- `<?>` autorise tout
- `<? extends Personne>` autorise seulement les objets de la classe `Personne` ou ceux dont la classe mère est `Personne`
- `<? super Personne>` autorise seulement les objets de la classe `Personne` ou ceux qui ont la classe `Personne` comme classe fille

S1- step 1: creation de repo github (avec ReadMe file et une structure de projet maven)

S2-step2: initialization du code sur git

S3-step3: commit N1 (l'etulisation de javadoc)

S4-step4: commit N2 (l'etulisation de javadoc)

S5-step5: commit N3 (l'etulisation de javadoc)

S6-step6: commit N4 (l'etulisation de javadoc)

S7-step7: commit N5 (l'etulisation de javadoc)

S8-step8: commit N6 (l'etulisation de javadoc)

S9-step9: commit N7 (l'etulisation de javadoc)

S10-step10: commit N8 (l'etulisation de javadoc)

S11-step11: commit N9 (l'etulisation de javadoc)

S12-step12: commit N10 (l'etulisation de javadoc)

S13- livraison du projet (l'etulisation de javadoc)