
Java Avancé

Cours 3 : Qualité Logicielle

Arsène Lapostolet & Nada Nahle

Code générique

Concept

Prendre des types en paramètre.

- Avoir une vérification plus robuste du typage à la compilation
- Éviter à avoir à faire des casts explicites
- Implémenter des algorithmes génériques

```
1 var strings = new ArrayList<String>();
```

Java

Liste de String, la classe ArrayList est paramétrisée par le type String

Classe Générique

```
1 public class Pair<L, R> {
2     private final L left;
3     private final R right;
4
5     public Pair(L left, R right){
6         this.left = left;
7         this.right = right;
8     }
9
10    public L getLeft(){ return left;}
11    public R getRight(){ return right;}
12 }
```

Java

```
1 final var pair = new Pair<String, Integer>("test", 42);
```

 Java

Les types primitifs ne peuvent pas être utilisés comme paramètre de type, il faut utiliser leur équivalent en type référence (ex ici : Integer pour int)

Méthode générique

```
1 public <T> List<T> fromArrayToList(T[] array) {  
2     return Arrays.stream(array).collect(Collectors.toList());  
3 }
```

Java



Des questions ?

Contraintes de paramètres de types

Contraintes de paramètre de type

```
1 public abstract class Animal {  
2  
3     public abstract void eat();  
4 }  
5  
6 class Dog extends Animal {  
7     public void eat() { }  
8 }  
9  
10 class Cat extends Animal {  
11     public void eat() { }  
12 }
```

Java

Contraintes de paramètre de type

Méthode polymorphe :

```
1 public static void addAnimals(Collection<Animal> animals) Java
```

✗ Ne compile pas :

```
1 List<Cat> cats = new ArrayList<Cat>();  
2 cats.add(new Cat());  
3 addAnimals(cats); Java
```

Contraintes de paramètre de type

En changeant la méthode avec une contrainte :

```
1 public static <T extends Animal> void  
   addAnimals(Collection<T> animals)
```

Java



Des questions ?