Java Avancé

Cours 4 : Généricité

Arsène Lapostolet & Nada Nahle

Code générique

Concept

Prendre des types en paramètre.

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

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

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

Classe Générique

```java
public class Pair<L, R> {
    private final L left;
    private final R right;

    public Pair(L left, R right){
        this.left = left;
        this.right = right;
    }

    public L getLeft(){ return left;}
    public R getRight(){ return right;}
}
```

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

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

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

Des questions ?

Contraintes de paramètres de types

Contraintes de paramètre de type

```java
public abstract class Animal {

    public abstract void eat();
}

class Dog extends Animal {
    public void eat() { }
}

class Cat extends Animal {
    public void eat() { }
}
```

Contraintes de paramètre de type

Méthode polymorphe :

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

❌ Ne compile pas :

```java
List<Cat> cats = new ArrayList<Cat>();
cats.add(new Cat());
addAnimals(cats);
```

Contraintes de paramètre de type

En changeant la méthode avec une contrainte :

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

Des questions ?