



Généricité ?!

Conteneur générique

- On a parler de conteneur générique
 - Un Vector par exemple, typer par une classe
 - `Vector<T> v`: est une liste d'objets (dérivés) de T
 - Avec l'héritage `class T extends U`
 - `Vector<U> w = v`; // interdit
 - On peut déclarer un Vector de ? extends T
 - En lecture seule, pour le passage aux méthodes

Généricité

- Même syntaxe que pour les collections

```
public class Pair<T> {  
    private T first;  
    private T second;  
    public Pair(T first) { super(); this.first = first;}  
    public T getFirst() {return first;}  
    public void setFirst(T first) {this.first = first;}  
    //...  
}
```

```
}  
//...  
public class Main {  
    public static void main(String [] args){  
        Pair<Integer> t = new Pair<Integer>(10);  
        t.setFirst(15);  
    }  
}
```

? Wildcard encore

- Pour l'instanciation (new) il faut bien spécifier le type : `new Pair<MyClass>`
 - Pas de `new Pair<?>`
- Déclaration de type générique
 - `Pair<?> p = new Pair<MyClass>`
 - Après pour l'affectation c'est plus dur (il faut caster, faire attention)
 - Mais c'est parfait pour le paramètre d'une méthode

? extends super

- ? : n'importe quoi
- ? extends MyClass : classes dérivées de
- ? super MyClass : super classes de

Exercice

- Changer Pair pour avoir deux type différents
 - Syntaxe class name<T, S>