



La généricité

Marie-Laure Nivet,
Université de Corse

nivet@univ-corse.fr

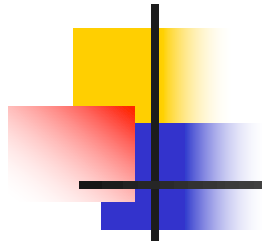
Bat 018, 2ième





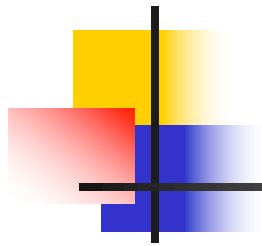
Objectifs

- Donner un aperçu rapide de ce qu'est la généricité
- Fournir les premiers éléments de compréhension afin de pouvoir comprendre la manipulation des collections
- Attention : vous n'aurez pas compris toutes les subtilités de la généricité à la fin de ce cours (cf biblio)



Historique et apparition

- Concept introduit avec le J2SE 5.0



Plan du cours

- Définition, Principe, utilisation
- Classes génériques et sous-typage
 - Propriétés et limitation
- Types jokers et types jokers liés
- Compatibilité avec les anciennes versions, interopérabilité



La généricité

Qu'est-ce que c'est ?
Comment la définir ?
Comment l'utiliser ?
Pourquoi l'utiliser ?



Cas d'école...

- Une classe conteneur de n'importe quoi, BoiteNonGenerique...
 - attribut de type « n'importe quoi »
 - constructeur à un paramètre type « n'importe quoi »
 - « n'importe quoi » getValue()
 - setValue(« n'importe quoi » valeur)
 - toString

Quelle sera votre solution 10mn max ?



Deux solutions

- Boite d'Object : utilisation du polymorphisme
 - Intérêt : on peut mettre tout ce qu'on veut dedans
 - Limitations : une fois la boite fermée, si on ne s'en rappelle plus, on ne peut pas savoir ce qu'il y a dedans... si on se trompe le compilateur ne le sait pas...
- Boite<T> : utilisation de la généricité
 - Intérêt : on contraindre ce qu'on peut mettre dedans et même si la boite est fermée on sait encore ce qu'il y a dedans ; le compilateur peut vérifier
 - Limitations, on perd en souplesse

```

package l3.poo.cours.genericite;

public class BoiteNonGenerique {
    private Object contenu;

    public BoiteNonGenerique(Object o){
        contenu = o;
    }

    public Object getContenu() {
        return contenu;
    }

    public void setContenu(Object contenu) {
        this.contenu = contenu;
    }

    @Override
    public String toString() {
        return "BoiteNonGenerique{" +
            "contenu=" + contenu +
            '}';
    }
}

```

```

package l3.poo.cours.genericite;

public class BoiteGenerique<E>{
    private E contenu;

    public BoiteGenerique(E contenu) {
        this.contenu = contenu;
    }

    public E getContenu() {
        return contenu;
    }

    public void setContenu(E contenu) {
        this.contenu = contenu;
    }

    @Override
    public String toString() {
        return "BoiteGenerique{" +
            "contenu=" + contenu +
            '}';
    }
}

```



```

import l3.poo.cours.genericite.BoiteGenerique;
import l3.poo.cours.genericite.BoiteNonGenerique;

import java.awt.*;

public class Main {

    public static void main(String[] args) {
        //Dans nos boîtes on veut mettre des Point, mais dans la deuxième on se trompe :-(
        BoiteNonGenerique b1= new BoiteNonGenerique(new Point(x:10, y:15));
        BoiteNonGenerique b2= new BoiteNonGenerique(o:2);
        //equivalent à new BoiteNonGenerique(new Integer(2));
        //Le contenu de la boîte 2 est mauvais, mais le compilateur ne le sait pas !
        BoiteNonGenerique b3= new BoiteNonGenerique(new Point());
        //Pb si on ne se rappelle plus ce qu'il y a dedans difficile de récupérer l'objet
        //...
        Point contenub1 =(Point)b1.getContenu();
        Point contenub2 =(Point)b2.getContenu();//Ici il va y avoir une ClassCastException...
        Point contenub3 =(Point)b3.getContenu();
        //Version avec Genericité
        BoiteGenerique<Point> b5 = new BoiteGenerique<>(new Point(x:20, y:10));
        BoiteGenerique<Point> b6 = new BoiteGenerique<>(contenu:2);
        BoiteGenerique<Point> b7 = new BoiteGenerique<Point>(contenu:3);
        Point contenub5 = b5.getContenu();
    }
}

```



Qu'est ce que la généricité ?

- Paramétrer les classes ou interfaces par un type
 - abstraction sur les types lors de la conception des classes
 - les comportements communs sont définis quel que soit le type des objets
- Un type générique, ou type paramétré peut prendre différentes valeurs
 - L'instanciation se fait suivant un ou plusieurs types de données
- Pour utiliser un type générique
 - spécifier un type pour chacun des paramètres de type demandé,
 - le type générique est alors contraint à ne prendre en compte que les objets des types que vous avez spécifié



Utilisation d'une classe générique

`LinkedList<Integer>`

- Remplacer le paramètre de type `<E>` par un type concret, comme `<Integer>` ou `<String>` ou `<VotreType>`
- Une `LinkedList<Integer>` ne peut stocker que des instances d'`Integer` ou instances de filles de `Integer` (sous-type)

```
LinkedList<Integer> li = new LinkedList<Integer>();  
li.add(new Integer(0));  
Integer i = li.iterator().next();
```



Raisons à l'introduction de la généricité

- Liées à la gestion des collections d'objet
- Avant la version 1.5 les collections manipulent des types `Object`
 - On peut donc avoir des collections hétérogènes
 - Il est impossible de contraindre la collection à ne manipuler que des objets d'un type prédéfini
 - Grande souplesse, mais limitation du contrôle des éventuelles erreurs



Exemple

- Supposez que vous souhaitez stocker des chaînes de caractères dans un `Vector`
- Par erreur vous ajoutez un `Integer`...
- Le compilateur ne peut détecter l'erreur

```
Vector v = new Vector();  
v.add(new String("valid string")); // voulu  
v.add(new Integer(4)); // erreur  
// ClassCastException pendant l'exécution  
String s = (String)v.get(1);
```

Cast obligatoire



Raisons à l'introduction de la généricité

■ Problème

- Le compilateur ne peut vérifier dans les collections d'objets la cohérence des types
- Utilisation systématique du forçage de type (Cast)
- Levée d'exceptions du type `ClassCastException` lors de l'exécution

■ Solution offerte par la généricité

- Vérification de la compatibilité des types à la compilation
- Le cast est effectué par le compilateur
- L'erreur précédente est évitée...



Exemple : Cast automatique

Avant la généricité

```
List myIntList = new LinkedList();  
myIntList.add(new Integer(0));  
Integer x = (Integer) myIntList.iterator().next();
```

```
List<Integer> myIntList = new LinkedList<Integer>();  
myIntList.add(new Integer(0));  
Integer x = myIntList.iterator().next();
```

Après la généricité



Classes ou interfaces génériques

Classe ou interface paramétrée
par une section de paramètre de
type formels



Définir une classe générique

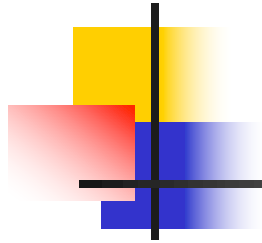
- Faire suivre le nom du type (classe ou interface) d'un ou plusieurs paramètres de type (par convention une lettre majuscule)
- Si T est un paramètre qui désigne un type inconnu au moment de la compilation, il peut intervenir dans les déclarations de variables, méthodes, classes, interfaces, collections
- T est un paramètre de type formel

```
public class ClasseGenerique<T, E, ...>{...}
```



Conventions de nommage des paramètres de type

- E – *Element*
 - Élément (utilisé en particulier dans le cadre des collections)
- K - *Key*
 - Clé (table de hachage)
- N - *Number*
- T - *Type*
- V - *Value*
- S,U,V etc. - 2nd, 3rd, 4th types



Exercice : Stack générique

- Reprenez le code de l'interface pile et rendez la générique
- Reprenez ensuite le code de la classe `ConcreteStackArray` pour qu'elle respecte cette nouvelle interface.



Exemple

```
public class Stack{  
    private ArrayList items;  
    ...  
    public void push(Object item){}  
    public Object pop(){}  
    public boolean isEmpty(){}  
}
```

Avant la généricité

Après la généricité

```
public class StackG<T>{  
    private ArrayList<T> items;  
    ...  
    public void push(T item){}  
    public T pop(){}  
    public boolean isEmpty(){}  
}
```



Utilisation d'une classe générique

- On doit spécifier le paramètre de type
 - Ce type peut être une classe concrète, abstraite, ou une interface (pas primitif)

```
Stack s = new Stack();  
s.push("hi");  
String greeting = (String)s.pop(); //cast obligatoire
```

Avant la généricité

```
StackG<String> s = new StackG<String>();  
s.push("hi");  
String greeting = s.pop(); //pas de cast
```

Avec la généricité

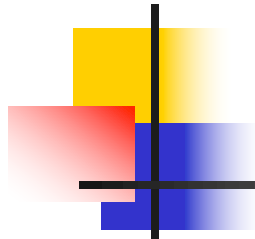


Lien avec les Templates C++ ?

- Idée globalement équivalente
- Traitement différent
 - A la compilation en C++, le code de classe est dédoublé en autant d'incarnation nécessaire
 - Il y a autant de classes compilées que de valeur différentes pour T
 - Avec java 5, une seule classe compilée.
 - Lors de la création des objets instances de cette classe, le paramètre T est affecté et remplacé par la valeur donné à T
 - A chaque valeur de T correspond alors un type paramétré



Méthodes génériques



Méthodes générique

- Les méthodes peuvent, comme les classes et les interfaces être paramétrées par un type
 - Méthodes d'instances, de classe et constructeurs
- Une méthode générique peut être incluse dans une classe générique (si elle utilise un autre paramètre que les paramètres de type formels de la classe) ou dans une classe non générique



Syntaxe

- Une liste de paramètres apparaît dans l'en-tête de la méthode

```
[modificateur]<T,E,...>... nomMethode (...) {...}
```

```
public interface Collection<E> extends Iterable<E>{  
    ...  
    public abstract <T> T[] toArray(T[] a);  
}
```

↑
Indique que la méthode est générique, cad prend un type en paramètre

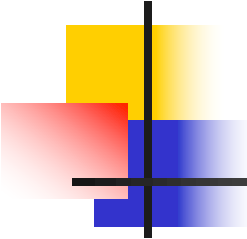


Utilisation, invocation

- Les méthodes génériques s'invoquent comme les méthodes "classiques"
- Le compilateur déduit du contexte le type qui doit être utilisé

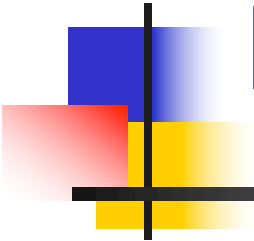
```
public interface Collection<E> extends Iterable<E>{...  
    public abstract <T> T[] toArray(T[] a);}
```

```
ArrayList<String> liste;  
...  
String[] result = liste.toArray(new String[0]);  
//Implicitement liste.<String>toArray(...)
```



Exemple d'illustration, méthodes génériques dans une classe non générique

```
public class Util {  
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {  
        return p1.getKey().equals(p2.getKey()) &&  
            p1.getValue().equals(p2.getValue());  
    }  
}  
  
public class Pair<K, V> {  
    private K key;  
    private V value;  
    public Pair(K key, V value) {  
        this.key = key;  
        this.value = value; }  
    public void setKey(K key) {  
        this.key = key; }  
    public void setValue(V value) {  
        this.value = value; }  
    public K getKey() { return key; }  
    public V getValue() { return value; }  
}
```



Propriétés de la généricité, Utilisation des types joker Problème du sous-typage

Attention tout n'est pas si simple !



Limitation au niveau sous-typage

- Ces instructions sont elles légales ?

```
List<String> ls = new ArrayList<String>();  
List<Object> lo = ls;
```

- Pour en juger, regardez les instructions suivantes

```
lo.add(new Object());  
String s = ls.get(0);
```

Attention on essaie d'assigner un Object à une String
OK à la compilation, mais ClassCastException à l'exécution

Limitation au niveau sous-typage

- Si F hérite de M
 - F est un sous-type de M
 - les classes `Generic<F>` et `Generic<M>` n'ont aucun lien de sous-typage

Exemple raisons : Si `ArrayList<F>` était un sous-type de `ArrayList<M>`

```
ArrayList<M> lm = new ArrayList<F>();  
lm.add(new M());
```

lm contiendrait des M qui ne sont pas des F...

```
ArrayList<Object> ao = new ArrayList<Integer>();
```

- Une méthode `m(List<M>)` ne pourra pas être utilisée avec une liste de F...



Généricité et sous-typage

- A l'intérieur de la collection les relations de sous-typage sont toujours effectives

```
ArrayList<Number> an = new ArrayList<Number>();  
an.add(new Integer(5)); //OK  
an.add(new Long(1000L)); //OK  
an.add(new String("Hello")); //Erreur de compilation
```



Problème

- Comment écrire une méthode permettant d'afficher tous les éléments d'une collection quels qu'ils soient

```
static void printCollection(Collection c){  
    Iterator i = c.iterator();  
    for (k = 0; k < c.size(); k++){  
        System.out.println(i.next());  
    }  
}
```

Bonne version <1.5

```
static void printCollection(Collection<Object> c){  
    for (Object e : c) System.out.println(e);  
}  
  
public static void main(String[] args){  
    Collection<String> cs = new Vector<String>();  
    printCollection(cs); //Erreur de Compilation  
    List<Integer> li = new ArrayList<Integer>(10);  
    printCollection(li); //Erreur de Compilation
```

Essai malheureux
avec la version 1.5



Problème,

- `Collection<Object>` n'est pas le super type de toutes les collections !!!
- La solution : utiliser les *wildcard type* ou type joker... Prononcer collection d'inconnus

```
static void printCollection(Collection<?> c) {  
    for (Object o : c) System.out.println(o);  
}  
public static void main(String[] args) {  
    Collection<String> cs = new Vector<String>();  
    printCollection(cs); //OK  
    List<Integer> li = new ArrayList<Integer>(10);  
    printCollection(li); //OK
```

Bonne version 1.5



Wildcards type, ou types joker

- `<?>` type fixé mais inconnu
- `<? extends A>` désigne un type fixé mais inconnu qui est `A` ou un sous type de `A`, une classe fille
- `<? super A>` désigne un type fixé mais inconnu qui est `A` ou un sur type de `A`, une classe mère
- `A` peut être une classe, une interface, ou même un paramètre de type formel



Utilisation des types Joker (i)

- Pour accéder aux items d'une collection d'éléments de type inconnu vous devez passer par une référence au type `Object`

```
static void printCollection(Collection<?> c) {  
    for (StringBuffer o : c) //Erreur de compilation  
        System.out.println(o);  
}
```



Utilisation des types Joker (ii)

- Il est interdit d'ajouter n'importe quel type d'objet à une collection de type inconnu
- Raison : On ne sait pas à quoi sera lié le type joker au moment de l'exécution

```
static void printCollection(Collection<?> c) {  
    c.add(new Object()); // Compile time error  
    c.add(new String()); // Compile time error  
}
```



Types jokers constraints, *Bounded Wildcard*

- `<? extends A>` désigne un type fixé mais inconnu qui est A ou un sous type de A, une classe fille
- `<? super A>` désigne un type fixé mais inconnu qui est A ou un sur type de A, une classe mère

```
static void printCollection(Collection<? extends Number> c) {  
    for (Number o : c) System.out.println(o);  
}  
public static void main(String[] args) {  
    Collection<String> cs = new Vector<String>();  
    printCollection(cs); //Erreur de compilation  
    List<Integer> li = new ArrayList<Integer>(10);  
    printCollection(li); //OK
```



Compatibilité avec les anciennes versions (legacy code), *Raw type*

- La compatibilité est assurée
 - Possibilité d'utiliser les types génériques en les instanciant sans type argument on parle alors de « types bruts » ou *raw type*

```
List lraw = new LinkedList(); //raw type
```

- Les codes Pré J2SE 5.0 continuent à fonctionner
 - Des warnings sont générés par le compilateur pour souligner les éventuelles erreurs

```
Note: MaClasse.java uses unchecked or unsafe operations.
```



Interopérabilité

- Mélange des versions, comment ça se passe ?
- Que va-t-il se passer à la compilation ?
À l'exécution ?

```
public class GenericsInteroperability {  
    public static void main(String[] args) {  
        List<String> ls = new LinkedList<String>();  
        List lraw = ls;  
        lraw.add(new Integer(4));  
        String s = ls.iterator().next();  
    }  
}
```



Interopérabilité

- A la compilation : Warning
- A l'exécution : `ClassCastException`

```
public class GenericsInteroperability {  
    public static void main(String[] args) {  
        List<String> ls = new LinkedList<String>();  
        List lraw = ls;  
        lraw.add(new Integer(4));  
        String s = ls.iterator().next();  
    }  
}
```




Bibliographie

- Le cours de Sang Shin sur le site JavaPassion
 - <http://www.javapassion.com>
- Le toujours et plus que jamais excellent tutorial Sun
 - <http://java.sun.com/docs/books/tutorial/extra/generics/index.html>
- Le tutorial spécial Généricité, pour ceux qui veulent creuser le sujet
 - <http://java.sun.com/docs/books/tutorial/extra/generics/index.html>
- Cours de Richard Grin
 - <http://deptinfo.unice.fr/~grin/messupports/java/>