

Java 1.5 : principales nouveautés

- classes paramétrées : generics
- encapsulation des valeurs de types primitifs : auto[un]boxing
- itération sur les boucles
- types énumérés
- nombre d'arguments variable des méthodes : varargs
- <http://java.sun.com/developer/technicalArticles/releases/j2se15/>
- java.sun.com/developer/technicalArticles/releases/j2se15langfeat/

Génériques en Java 1.5

- But : manipuler des classes paramétrées
 - pour un code plus sûr,
 - plus lisible
 - et potentiellement plus efficace
- Contraintes :
 - utiliser la même machine virtuelle
 - être compatible ascendant (programmes 1.4 compilables)

Motivations

- typage statique générique pour
 - diminuer les tests dynamiques de types
 - écrire des structures de données génériques classiques et effectuer des calculs dessus
 - faciliter la lecture des programmes
- répondre aux critiques d'autres langages :
 - C++, Ada95, O'Caml, Haskell ...
- tenir compte de propositions d'extension :
 - Pizza, GJ, ...
- répondre à l'avance à C#

Contraintes

- compatible avec les versions antérieures :
 - du langage
 - des bibliothèques
 - de la machine abstraite
- cohabitation possible entre codes/bibliothèques antérieurs
- ne pas être coûteux si on ne s'en sert pas

Influences

- polymorphisme paramétrique (ML,)
- propositions Pizza et GJ :
 - Pizza : <http://pizzacompiler.sourceforge.net/>
 - GJ : : <http://homepages.inf.ed.ac.uk/wadler/pizza/gj/>
- Génériques pour C# et .NET :
<http://research.microsoft.com/projects/clrgen/>

Java 1.4 : API

vecteurs extensibles:

java.util

Class ArrayList

hiérarchie de classes:

java.lang.Object

 java.util.AbstractCollection

 java.util.AbstractList

 java.util.ArrayList

principales méthodes:

ArrayList(int initialCapacity)

void add(int index, Object element)

Object get(int index)

Object set(int index, Object element)

Java 1.4 : utilisation (UD.java)

```
import java.util.ArrayList;

class UD {
    public static void main(String[] a) {
        ArrayList all=new ArrayList(3);
        all.add(0,new Integer(3));
        all.add(1,"salut");
        Integer x = (Integer)(all.get(0));
        Integer y = (Integer)(all.get(1));        //
        int res = x.intValue() + y.intValue();
    }
}
```

- **compilation** : `javac -source 1.4 UD.java`
- **exécution** : **exception**

```
$ java UD
Exception in thread "main"
    java.lang.ClassCastException: java.lang.String
        at UD.main(UD.java:8)
```

Java 1.5 : API

vecteurs extensibles:

```
java.util  
  Class ArrayList<E>
```

hiérarchie de classes:

```
java.lang.Object  
  java.util.AbstractCollection<E>  
    java.util.AbstractList<E>  
      java.util.ArrayList<E>
```

principales méthodes:

```
ArrayList(int initialCapacity)  
void add(int index, E element)  
E get(int index)  
E set(int index, E element)
```


Java 1.5 : utilisation (UD.java)

⇒ warnings à la compilation

```
$ javac -source 1.5 UD.java
```

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

```
Note: Recompile with -Xlint:unchecked for details.
```

```
-bash-3.00$ javac -source 1.5 -Xlint:unchecked UD.java
```

```
UD.java:5: warning: [unchecked] unchecked call to add(int,E)
           as a member of the raw type java.util.ArrayList
    all.add(0,new Integer(3));
           ^
```

```
UD.java:6: warning: [unchecked] unchecked call to add(int,E)
           as a member of the raw type java.util.ArrayList
    all.add(1,"salut");
           ^
```

```
2 warnings
```

⇒ exception à l'exécution

```
$ java UD
```

```
Exception in thread "main"
```

```
java.lang.ClassCastException: java.lang.String
    at UD.main(UD.java:8)
```

Java 1.5 : classe paramétrée (us.java)

```
import java.util.ArrayList;
class US {
    public static void main(String[] a) {
        ArrayList<Integer> all=
            new ArrayList<Integer>(3);
        all.add(0,new Integer(3));
        all.add(1,"salut");
        Integer x = (Integer)(all.get(0));
        Integer y = (Integer)(all.get(1));
        int res = x.intValue() + y.intValue();
    } }
```

⇒ erreur à la compilation

```
US.java:6: cannot find symbol
symbol   : method add(int,java.lang.String)
location: class java.util.ArrayList<java.lang.Integer>
    all.add(1,"salut");
           ^
1 error
```

Limitations

- paramètre de type instancié par une classe ou une interface
pas par un type primitif
⇒ auto-boxing
- pas de cast avec type paramétré (warning)
- ni d'instanceof (erreur)
- pas de type paramétré comme type des éléments d'un tableau :
cela vient du sous-typage entre tableaux.

Typage et compatibilité

- *raw type* : type paramétré sans paramètre (compatibilité)
 - `Type<A>` vers `RawType`
 - `RawType` vers `Type<A>` : warning
- pas de sous-typage sur les paramètres de types :

```
ArrayList<String> as =  
    new ArrayList<String>(3);  
ArrayList<Object> ao = as;
```

- pas de création de tableaux paramétrés :

```
A[] aa = new A[10];
```

Warning à la compilation

```
javac -Xlint:unchecked UD.java
```

Danger : voir de 2 manières une même structure

```
ArrayList<String> als = new ArrayList<String>(10);
```

```
ArrayList al = als;
```

```
als.add(1, "Salut");
```

```
als.add(2, new Integer(4));
```

pas de *unchecked warning*

⇒ pas d'exception *ClassCastException*

Exemple : QueueD.java (1)

```
import java.util.ArrayList;
```

```
class Vide extends Exception {}  
class Pleine extends Exception {}
```

```
class QueueD {  
    int taille, longueur;  
    ArrayList q;  
    int tete, fin;  
  
    QueueD(int n) {taille = n; q = new ArrayList(n);}  
  
    void entrer(Object x) throws Pleine {  
        if (longueur < taille) { q.add(fin++ % taille,x); longueur++;}  
        else throw new Pleine();  
    }  
  
    Object partir() throws Vide {  
        if (longueur > 0) { longueur--; return q.get(tete++ % taille);}  
        else throw new Vide();  
    }  
}
```

Exemple : Queue.java (2)

```
class Vide extends Exception {}
class Pleine extends Exception {}

class Queue<A> {
    int taille, longueur;
    A[] q;
    int tete, fin;

    Queue(int n) {taille = n; q = new A[n];}

    void entrer(A x) throws Pleine {
        if (longueur < taille) { q[fin++ % taille] = x; longueur++;}
        else throw new Pleine();
    }

    A partir() {
        if (longueur > 0) { longueur--; return q[tete++ % taille];}
        else throw new Vide();
    }
}
```

Exemple : QueueS.java (3)

```
import java.util.ArrayList;
```

```
class Vide extends Exception {}
```

```
class Pleine extends Exception {}
```

```
class QueueS<A> {
```

```
    int taille, longueur;
```

```
    ArrayList<A> q;
```

```
    int tete, fin;
```

```
    QueueS(int n) {taille = n; q = new ArrayList<A>(n);}
```

```
    void entrer(A x) throws Pleine {
```

```
        if (longueur < taille) { q.add(fin++ % taille, x); longueur++;}
```

```
        else throw new Pleine();
```

```
    }
```

```
    A partir() throws Vide {
```

```
        if (longueur > 0) { longueur--; return q.get(tete++ % taille);}
```

```
        else throw new Vide();
```

```
    }
```

```
}
```

Exemple : compilation (4)

● QueueD :

```
$ javac -Xlint:unchecked QueueD.java
```

```
QueueD.java:14: warning: [unchecked] unchecked call to add(int,E) as a  
    if (longueur < taille) { q.add(fin++ % taille,x); longueur++;}  
                                ^
```

```
1 warning
```

● Queue :

```
$ javac -Xlint:unchecked Queue.java
```

```
Queue.java:10: generic array creation
```

```
    Queue(int n) {taille = n; q = new A[n];}  
                                ^
```

```
1 error
```

● QueueS :

```
$ javac -Xlint:unchecked QueueS.java
```

Exemple : exécution (5)

● UQS.java

```
import java.util.ArrayList;
class UQS {
    public static void main(String[] a) {
        QueueS<Integer> q = new QueueS<Integer>(3);
        try {
            q.entrer(new Integer(3));
            q.entrer(new Integer(4));
            Integer x = q.partir();
            Integer y = q.partir();
            int res = x.intValue() + y.intValue();
            System.out.println(res);
        }
        catch (Exception e) {System.out.println(e.toString());}
    }
}
```

● compilation :

```
$ javac -Xlint:unchecked UQS.java
```

● exécution

```
$ java UQS
```

Méthodes paramétrées

- introduction d'une variable de type au niveau de la méthode

```
import java.util.*;
interface Comparator<T> {
    public int compare(T x, T y); }
class ByteComparator implements Comparator<Byte> {
    public int compare (Byte x, Byte y) { return (x - y); } }

class Collections {
    public static <T> T max(Collection<T> col, Comparator<T> cmp) {
        Iterator<T> it = col.iterator();
        T elt = it.next();
        while (it.hasNext()) {
            T elt2 = it.next();
            if (cmp.compare(elt,elt2) < 0 ) elt = elt2;
        }
        return elt;
    }
}
```

Méthodes paramétrées (2)

En O'Caml <http://caml.inria.fr/pub/docs/manual-ocaml/manual005.html> :

```
#class intlist (l : int list) =
  object
    method empty = (l = [])
    method fold : 'a. ('a -> int -> 'a) -> 'a -> 'a =
      fun f accu -> List.fold_left f accu l
  end;;
```

```
#let l = new intlist [1; 2; 3];;
val l : intlist = <obj>
```

```
#l#fold (fun x y -> x+y) 0;;
- : int = 6
```

```
#l#fold (fun s x -> s ^ string_of_int x ^ " ") "";;
- : string = "1 2 3 "
```

Typage et sous-typage

- un type paramétré $\tau_2 \langle T_2 \rangle \leq \tau_1 \langle T_1 \rangle$ ssi :
 - $\tau_2 \leq \tau_1$
 - et $T_2 = T_1$
 - un type paramétré $\tau \langle T \rangle \leq \tau$
 - un type paramétré $\tau \langle T \rangle \leq \text{Object}$
-
- $\tau_2 \langle T_2 \rangle$ n'est pas sous-type de $\tau_1 \langle T_1 \rangle$ si $T_2 \neq T_1$

Redéfinition et surcharge (1)

```
class QueueSR {
    Object m1(QueueS<String> q) throws Vide {
        System.out.println("QueueSR m1 QueueS<String>");
        return q.partir();
    }
    Object m1(QueueS<Integer> q) throws Vide {
        System.out.println("QueueSR m1 QueueS<Integer>");
        return q.partir();
    }
    /* Object m1(QueueS q) {
        System.out.println("QueueSR m1 QueueS");
        return q.partir();
    }
*/
    public static void main(String[] args) {
        QueueSR sr = new QueueSR();
        QueueS<String> q = new QueueS<String>(3);
        try {
            q.entrer("salut");
            sr.m1(q);
        }
        catch (Exception e) { }
```

Redéfinition et surcharge (2)

```
javac QueueSR.java
```

```
QueueSR.java:4: name clash: m1(QueueS<java.lang.String>)  
                and m1(QueueS<java.lang.Integer>)
```

```
    have the same erasure
```

```
    Object m1(QueueS<String> q) throws Vide {  
        ^
```

```
QueueSR.java:9: name clash: m1(QueueS<java.lang.Integer>)  
                and m1(QueueS<java.lang.String>)
```

```
    have the same erasure
```

```
    Object m1(QueueS<Integer> q) throws Vide {  
        ^
```

```
2 errors
```

Polymorphisme borné (1)

sous-typage sur des types paramétrés :

- introduction d'inconnues de types `<?>`
 - `?` (wildcard) hérite d'`Object`
- 2 `?` sont différents (variables fraîches).
- 2 bornes :
 - `<? extends A>` : inconnue sous type de `A`
 - `<? super A>` : inconnue sur(per) type de `A`
 - `<?> ≡ <? extends Object>`
- inconnue non instanciable
pas de `new A<?>()`

Polymorphisme borné (2)

API :

```
ArrayList(Collection<? extends E> c)
```

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

inférence de types pour la détermination des inconnues.

Example

```
import java.util.*;

class A<T> {
    T x;
    void set_x(T x) {this.x = x;}
    T get_x() {return x;} }

class B<T> extends A<T>{
    T y;
    void set_y(T y) {this.y = y;}
    T get_y() {return y;} }

class G {
    public static void main(String[] a) {
        A<? extends B> v1 = new B<B>();
        A<? extends A> v2 = v1;
        A<? extends Object> v3 = v2;

        A<? super A> v4 = new B<Object>();
        A<? super B> v5 = v4;
        A<?> v6 = v5;
    }
}
```

Compilation (1)

inference de types: sur les inconnues ?

- capture (liaison) d'un ? avec un paramètre de type ($\langle T \rangle$):
 - liaison unique (en dehors du type de retour)
 - et le type paramétré n'est pas argument d'un autre type paramétré
- aide à l'inférence en indiquant explicitement les paramètres de type de retour

Compilation (2)

- code compatible 1.4
- pas de changement de machine virtuelle
- remplace les types paramétrés par les types sans paramètres :
 - pas d'information du paramètre de type à l'exécution
 - ajoute des tests de typage dynamiques (warning)
 - ne va pas plus vite
- limite les possibilités de *debug*

Compilation (3)

- papiers sur Pizza pour les techniques de compilation :
 - monomorphisation : code spécialisé pour chaque paramètre de type instancié
 - tests de typage dynamiques : code plus compact mais plus lent
- papiers sur le CLR de .NET modifié pour intégrer les *generics* de C#

⇒ la compatibilité Java coûte cher.

Autres lectures

- **cours de Forax (Mlv) :**

<http://www-igm.univ-mlv.fr/~forax/ens/java-avance/cours/pdf/>

- **cours de M-V. Aponte et F. Barthélémy (Cnam) :**

<http://deptinfo.cnam.fr/~barthe/00/typage-genericite.pdf>

<http://deptinfo.cnam.fr/~barthe/00/>

- **tutorial GJ :**

<http://www.cis.unisa.edu.au:80/~pizza/gj/Documents/gj-tutorial.pdf>

<http://www.cis.unisa.edu.au:80/~pizza/gj/Documents/>

- **tutorial Java 1.5 :**

<http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf>

- **tutorial Pizza :**

<http://pizzacompiler.sourceforge.net/doc/tutorial.html>

- **sur .NET et les generics C#:**

<http://research.microsoft.com/projects/clrgen/>