Paris JUG

www.parisjug.org

09/06/2009

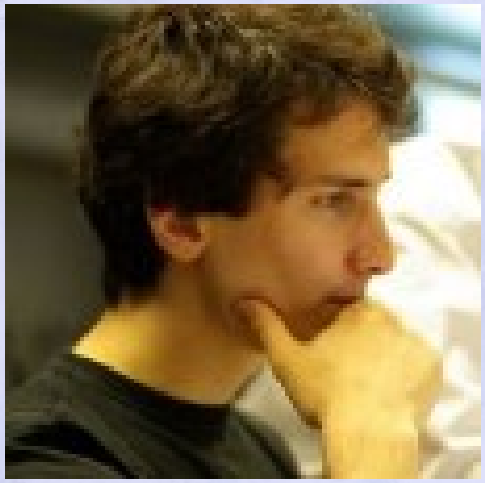
09/06/2009

Typage et généricité

Alexandre Bertails

Atos Origin

Open Source Center



www.parisjug.org

Here is a puzzle (1/2)

Here is a puzzle (2/2)

Java generics mystery

Here is a puzzle for Generics gurus.

Can somebody explain why

```
Set<Address> addresses = new HashSet<Address>();
Set<?> things = addresses;
```

compiles but

```
Set<ConstraintValidator<Address>> validatedAddresses =
    new HashSet<ConstraintValidator<Address>>();
Set<ConstraintValidator<?>> validatedThings =
    validatedAddresses;
```

does not compile?

More specifically, the assignment on the second line breaks.

Zoologie du typage

Typage statique vs typage dynamique

Typage fort vs typage faible

Héritage vs sous-typage

Polymorphisme, abstraction

Sous-typage et variance

Principe de substitution de Liskov

T est un **sous-type** de U si on peut **substituer** une valeur de type T là où un type U est attendu

Variance : ordre de variation des types

- Invariance
- Covariance
- Contravariance

Le typage dans Java

bytecode JVM : typage dynamique

Java : typage statique

Jusqu'à 1.4 : en cohérence

Depuis 5 : arrivée des types génériques (et des ennuis)

Covariance des types de retour

```
class Object {
   public Object clone() { ... }
}

class CyclingGod /* extends Object */ {
   @Override
   public CyclingGod clone() { ... }
}
```

Et les types de paramètres ?

```
class CyclingGod extends God { ... }

class Person {
  public void invoke(God god) { ... }
}

class Cyclist extends Person {
  @Override
  public void invoke(CyclingGod god) { ... }
}
```

Et les types de paramètres ?

```
Cyclist antonio = new Cyclist() ;
CyclingGod cochonou = new CyclingGod() ;
antonio.invoke(cochonou);
```

Mais...

```
class JavaGod extends God { ... }
Person antonio = (Person) new Cyclist();
JavaGod duke = new JavaGod() ;
antonio.invoke(duke);
```

Histoire des types génériques

JSR 14: Add Generic Types To The JavaTM Programming Language

http://jcp.org/en/jsr/detail?id=14

Motivations :

- généricité
- expressivité

Objectif : compatibilités **ascendante** et **descendante**

Tout le monde sait ça :)

```
List myIntList =
  new LinkedList();

myIntList.add(new Integer(42));

Integer x =(Integer)
  myIntList.iterator().next();
```

Tout le monde sait ça :)

```
List<Integer> myIntList =
  new LinkedList<Integer>();

myIntList.add(new Integer(42));

Integer x =(Integer)
  myIntList.iterator().next();
```

Invariance : intuition

```
List<Integer> ints =
  Arrays.asList(1,2,3);

List<Number> nums = ints;

nums.put(2, 3.14);
```

Invariance : exemple (1/2)

```
public abstract class Currency {}
public class Euro extends Currency {}

public interface God<T> {
   public T invoke(T t);
}

public interface Person<T> {
   public T hold();
}
```

Invariance : exemple (2/2)

```
Person<Euro> antonio = ... ;
God<Euro> cochonou = ... ;
Euro euro = antonio.hold();
cochonou.invoke(euro); // returns Euro
```

Covariance : intuition

```
List<Integer> ints =
  Arrays.asList(1,2,3);

List<? extends Number> nums = ints;

nums.put(2, 3.14);
```

Covariance : exemple (1/2)

```
public abstract class Currency {}
public class Euro extends Currency {}

public interface God<T extends Currency> {
   public T invoke(T t);
}

public interface Person<T> {
   public T hold();
}
```

Covariance : exemple (2/2)

```
Person<Euro> antonio = ... ;
God<Currency> cochonou = ... ;
Currency euro = antonio.hold();
cochonou.invoke(euro); // returns Currency
```

Contravariance (1/3)

```
public static <T> T doInvocation(
  Person<T> person, God<T> god) {
    T t = person.hold();
    return god.invoke(t);
  }


Person<Euro> antonio = ... ;

God<Currency> cochonou = ... ;

doInvocation(antonio, cochonou);
```

Contravariance (2/3)

```
public static <T> T doInvocation(
  Person<? extends T> person, God<T> god) {
    T t = person.hold();
    return god.invoke(t);
  }


Person<Euro> antonio = ... ;

God<Currency> cochonou = ... ;

doInvocation(antonio, cochonou); // returns
                                 // Currency
```

Contravariance (3/3)

```
public static <T> T doInvocation(
  Person<T> person, God<? super T> god) {
    T t = person.hold();
    god.invoke(t);
    return t;
  }


Person<Euro> antonio = ... ;

God<Currency> cochonou = ... ;

doInvocation(antonio, cochonou); // returns
                                 // Euro
```

List != List<?>

Raw type != wildcard type

<?> → type **inconnu**

`God<?>` est le super-type de tous les types `God`

Question : quelles valeurs possibles en paramètres et retour de fonctions ?

Raw type → type **oublié**

Le typage statique ne fonctionne plus pour ces types

Question : quelles valeurs possibles en paramètres et retour de fonctions ?

@emmanuelbernard

Can somebody explain why

```
Set<Address> addresses = new HashSet<Address>();
Set<?> things = addresses;
```

compiles but

```
Set<ConstraintValidator<Address>> validatedAddresses =
    new HashSet<ConstraintValidator<Address>>();
Set<ConstraintValidator<?>> validatedThings =
    validatedAddresses;
```

does not compile?

Tableaux suck

En java, les tableaux sont covariants...

```
Object[] objects = new String[]{"1","2","3"};
objects[0] = new Integer(1); // sans warning !
```

« We wanted to have a simple means to treat arrays generically »,
James Gosling

```
void sort(Object[] a, Comparator cmp) { ... }
```

Type erasure

```
public class Test {
  public static void test(List<String> input){
    String s = input.get(0);
  }
}

$ javac Test.java

$ javap -c Test
```

Type erasure

```
public class Test extends java.lang.Object{

public Test();

  Code:

   0:  aload_0

   1:  invokespecial  #1; //Method java/lang/Object."<init>":()V

   4:  return

public static void test(java.util.List);

  Code:

   0:  aload_0

   1:  iconst_0

   2:  invokeinterface   #2,  2; //InterfaceMethod java/util/List.get:
   (I)Ljava/lang/Object;

   7:  checkcast  #3; //class java/lang/String

   10: astore_1

   11: return

}
```

Compilation source-à-source

```
public static void test(List<String> input){
   String s = input.get(0);
}

public static void test(List input){
   String s = (String) input.get(0);
}
```

Où sont passés les generics ?

```
$ javap -c Test -verbose


Constant pool:
const #13 = Asciz
  (Ljava/util/List<Ljava/lang/String;>;)V;
   ...
public static void test(java.util.List);
   Code:
   ...
   Signature: length = 0x2
    00 0D
}
```

Les types dans Java 7

Type Annotations (JSR 308) and the Checker Framework

http://groups.csail.mit.edu/pag/jsr308/

`@NonNull`, `@Immutable`, `@Interned`, ...

Là encore, on assure la compatibilité du bytecode

Plugins compilateur : vérification statique ou code runtime

```
void marshal(@Readonly Object jaxbElement,
  @Mutable Writer writer) @Readonly { ... }

@NotEmpty List<@NonNull String> strings =
  new ArrayList<@NonNull String>();
```

Conclusion

Le système de types de Java, c'est...

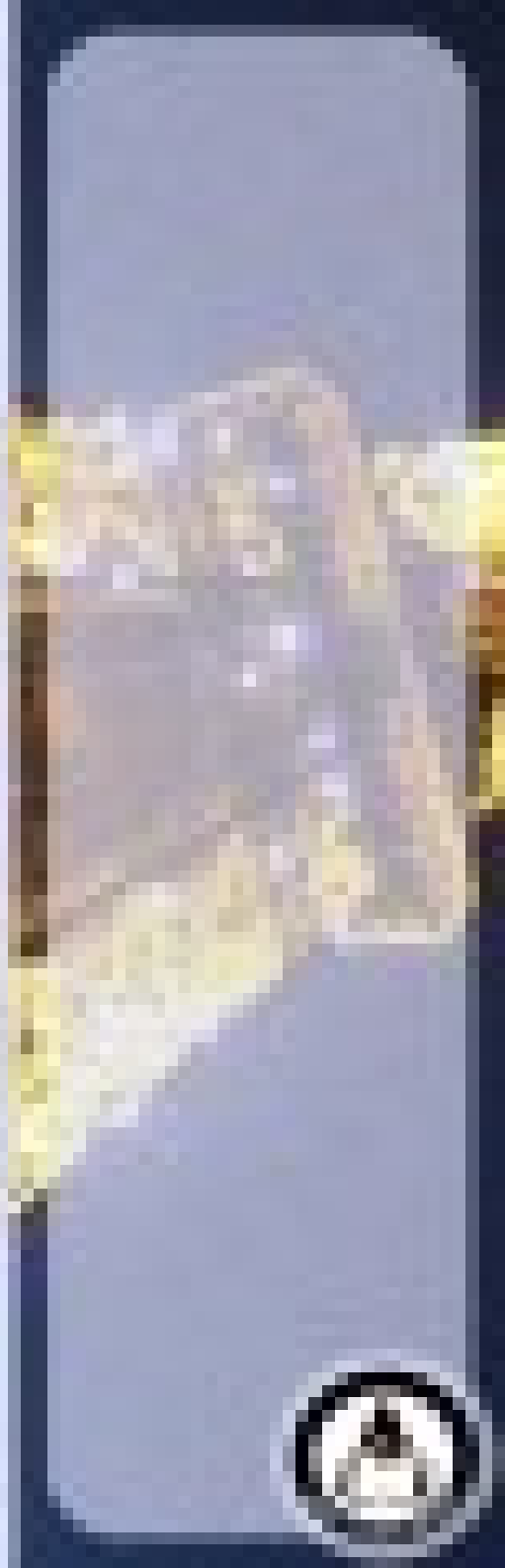
- du bon sens mais plutôt difficile d'accès
- une syntaxe assez indigeste
- plus de robustesse pour un typage pas si *fort* que ça
- cher payé pour ce qui est gratuit ailleurs

Du coup :

- langages dynamiques en force
- typage statique pas à sa juste valeur

Prêt pour le grand saut ?

Questions / Réponses



www.parisjug.org

Sponsors

Merci de votre attention!

www.parisjug.org

Licence

- http://creativecommons.org/licenses/by-nc-sa/2.0/fr/