DM8XX Proposal Talk
Generic Types in Java

IMADA, SDU
April 27, 2009

Martin Kamp Jensen
<mkjens04@imada.sdu.dk>

Contents

Why introduce generic types

- Detect type errors at compile-time, i.e., avoid ClassCastException at runtime

```
// Non-parameterized (raw) types
List list = new ArrayList();
list.add("text");
Number number = (Number) list.get(0); // Runtime error

// Parameterized types
List<String> strList = new ArrayList<String>();
strList.add("text");
Number number = strList.get(0); // Compile-time error
```

Generic types are not covariant

```java
String string = "text";
Object object = string; // OK

String[] strArray = new String[] { "text", "more text" };
Object[] objArray = strArray; // OK

List<String> strList = new ArrayList<String>();
List<Object> objList = strList; // Compile-time error because
objList.add(new Object()); // this should not be allowed
```

Introduction to wildcards 1/3

```
    (…)
    List<String> strList = new ArrayList<String>();
    strList.add("text"); strList.add("more text");
    print1(strList); // Compile-time error
    print2(strList); // OK
  }

  public static void print1(Collection<Object> collection) {
    for(Object object : collection) {
      System.out.println(object.toString());
    }
  }

  public static void print2(Collection<?> collection) {
    for(Object object : collection) {
      System.out.println(object.toString());
    }
  }
}
```

Introduction to wildcards 2/3

```java
public static void print3(Collection<Number> collection) {
  for(Number number : collection) {
    System.out.println(number.toString());
  }
}

public static <N extends Number> void print4(
    Collection<N> collection) {
  for(N number : collection) {
    System.out.println(number.toString());
  }
}
```

Introduction to wildcards 3/3

```
List<Number> numList = new ArrayList<Number>();
numList.add(1); numList.add(3.14f);

List<Float> fltList = new ArrayList<Float>();
fltList.add(3.14f);

print3(numList); // OK
print3(fltList); // Compile-time error

print4(numList); // OK
print4(fltList); // OK
```

How are generic types implemented

- Extension to the front-end of the compiler
- Type erasure removes type information
- Type safety ensured at compile-time

Design choices and problems

- Type erasure enables Java 1.5 to be backward compatible with earlier versions and vice versa
- Type information is not available at runtime
- Generic types have no performance impact

My project

- Performance analysis between e.g. Java's ArrayList<E> and “EArrayList” (and/or others)

  and/or

- Performance analysis between Java's PriorityQueue<E> and an implementation of a generic ternary heap

More information

- http://java.sun.com/j2se/1.5.0/docs/guide/language/generics.html
- http://java.sun.com/docs/books/tutorial/java/generics/
- http://www.ibm.com/developerworks/java/library/j-jtp01255.html
- http://www.ibm.com/developerworks/java/library/j-jtp04298.html