Reified Type Parameters Using Java Annotations

Prodromos Gerakios Aggelos Biboudis Yannis Smaragdakis

Department of Informatics
University of Athens

12th International Conference on Generative Programming: Concepts & Experiences (GPCE'13)
October 27, 2013
Indianapolis, IN, USA
Java Generics 101

```java
class ArrayList<X> {
    X [] arr ...
}

class ArrayList {
    Object [] arr ...
}
```

```java
l = new ArrayList<String>();
l.add("bar");
String s = l.get(0);
```

```java
l = new ArrayList<String>();
l.add("bar");
String s = (String)l.get(0);
```
Erasure examples
(Technicalities.FAQ108 of Generics, Angelika Langer)

<table>
<thead>
<tr>
<th>Type Parameter</th>
<th>Erasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;T&gt;</td>
<td>Object</td>
</tr>
<tr>
<td>&lt;T extends Number&gt;</td>
<td>Number</td>
</tr>
<tr>
<td>&lt;T extends Comparable&lt;T&gt;&gt;</td>
<td>Comparable</td>
</tr>
<tr>
<td>&lt;T extends Cloneable &amp; Comparable&lt;T&gt;&gt;</td>
<td>Cloneable</td>
</tr>
<tr>
<td>&lt;T extends Object &amp; Comparable&lt;T&gt;&gt;</td>
<td>Object</td>
</tr>
<tr>
<td>&lt;S, T extends S&gt;</td>
<td>Object, Object</td>
</tr>
</tbody>
</table>
What type erasure implies (the good)

- Code sharing in the bytecode level
- Compatibility with the “unaware of genericity” JVM
- Fewer source code type casts
- Type safety preservation
What type erasure implies (the bad)

✗ Type casts required
✗ By default autoboxing occurs
✗ No reflection support for generic parameters
✗ Expressiveness limitations (for X type param)
  ✓ new X
  ✓ X.class
  ✓ class C<X> extends X
    - “mixins”
We introduce a new annotation

class C<@reify X,Y> { ... }
Contributions

• Add **reified** generics for Java **without** a custom compiler!

• Translate by expansion while sharing generated code

• Control expansion or erasure of generics

• Support new patterns in Java `new T()` and `extends T`, where T type parameter

• Design as pluggable type checker
@reify will enable selective reification

```java
class ReifiedGeneric <@reify X, Y> {
    Class classOfX = X.class;
    Y id(Y y) { return y; }
    X newInstance() { return new X(); }
}
```
@reify will introduce mixin support

```java
class Serial <$reify T$> extends T {
  public long getSerialNumber() { ... }
}

class TimeStamped <$reify T$> extends T {
  public long getTimestamp() { ... }
}

TimeStamped <Serial <Customer>> customer = new TimeStamped <Serial <Customer>>();

sn    = customer.getSerialNumber();
timeS = customer.getTimestamp();
```
Is this a well-formed mixed-in definition?

No.
Two overloaded methods with distinct return types. X in ObjectFactory needs constraint.
Code Generation: Sharing Code

```java
class Foo<@reify X> {
    void meth() { X local = new X(); }
}

class Foo$Shared<X> implements iface$Foo<X> {
    X new$X() { return null; }
    void meth() { X local = new$X(); }
}

interface iface$Foo<X> { … }

Foo<Integer> foo = new Foo<Integer>();

class Foo$Integer extends Foo$Shared<Integer> {
    Integer new$X() { return new Integer(); }
}
```

- **One-time generation**
- **Generating an interface and a class**
- **new class w/o reified generics**
- **subclassing Foo$Shared<X>**
- **overriding new$X**
JSR 308 & Checker

• JSR 308 introduced annotations on types
• JSR 308 will be part of Java 8
• Can be processed as in JSR 269 (annotation processing) but also ...
• Can be processed with Checker Framework to write a checker plugin for type checking and generation
Conclusions

• Java generics with selective reification
• Translation scheme by-expansion
• Mixins and allocation expressions with generic parameters
• A solution that complies with Java 8
Questions?

Contact
biboudis @ di.uoa.gr
www.di.uoa.gr/~biboudis/
@biboudis

See you tomorrow at 6:00-9:00 pm in Splash Poster Reception - Cosmopolitan Foyer