JSR 14

Adding Generics to Java

Principals of Programing Languages

Heather L. Dykstra Caitlin E. Hickey Michael D. Williams

What are Generics?

- Generics are a type or method to operate on objects of various types while providing compile-time type safety

Where are Generics found?

- They can be any of the following
  - Type Variables
  - Class
  - Interface
  - Method
  - Constructors

Motivations

- They are a common feature in many other languages like:
  - .Net
  - C#
  - C++
  - Visual Basic
  - F#
  - Scala

Collections

- You don't have to define a type
- Wonderful if you don't know what you are dealing with
- Great if you know you were dealing with multiple types

What was the aproximate timeline of their creation?

- May 11, 1999: Java Specification Request approved.
- September 30, 2004: Final Release

Review and Create

- May 18, 1999 until August 1, 2001 the public had the opportunity to review and create the implementation.

Type Safety

- The compiler ensures that all of your types are correct before run time

Code Before Generics

```
List v = new ArrayList();
v.add("test");
Integer i = (Integer)v.get(0);
```

Efficency from the Coder's Point of View

- You only have to write one set of code

Generics Example

```
template <typename T>
T max (T a, T b){
return (b < a) ? a : b;
}
```

Wild Cards

- The Wildcard types are the supertypes of all Lists.
- Written List<?>
- You can't add anything, except null, to List, but you can retrieve things and treat them as Objects.

Bounded Wild Cards

- Wildcards can have upper (List<? super T>) and lower (List<? extend T> bounds -You would use an upper bound wildcard for input and a lower bound wildcard for output

Wild Cards

```
class Bar<T> {
void buy(int howMany, List<? super T> fillDrink {... }
void vomit(List<? extends T> throwUpDrink) {...}
...

}

Bar<Drink> bartender = new Bar<Drink>();
List<Vodka> shot = ...;
List<Shot> tooMuchAlcohol = ...;

//Buy vodka from the Bar
bartender.buy(3, shot);
//You can puke up the shot the Bar gave you...
bartender.puke(tooMuchAlcohol);
```

Type Erasure

- The compiler erases all generics from code before runtime, so (List $<$ String $>$) becomes (List).

Inheritance and Subtypes

- Generics are not covariant- a subtype of a class cannot be substituted for the class.

Covariance Issues

```
Item[] chalk = new Item[3];
Object[] objs = chalk;
objs[0] = new crayon();
```

Over Complication

- Do the benefits of generics outweigh the detriment of added complexity?

Heap Pollution

- Variable of a certain type ends up pointing to a variable of a different type.

$$List\ ln = new\ ArrayList < Number > ( ) ;$$

$$List < String > ls = ln;\ //\ unchecked\ warning$$

$$String\ s = ls.get(0);\ //\ ClassCastException$$

Gilad Bracha

"Calling legacy code from generic code is inherently dangerous; once you mix generic code with non-generic legacy code, all the safety guarantees that the generic type system usually provides are void. However, you are still better off than you were without using generics at all. At least you know the code on your end is consistent."