CS203 Programming with Data Structures

Generics

Chengyu Sun
California State University, Los Angeles

Example: Integers in Stack

```
ArrayStack stack = new ArrayStack();
stack.push( 10 );
stack.push( 20 );

int sum = 0;
while( ! stack.isEmpty() )
{
   Object value = stack.pop();
   sum += value; // error!!
}

System.out.println(sum);
```

How can we make it work??

Example: Integers in Stack

```
ArrayStack stack = new ArrayStack();
stack.push( 10 );
stack.push( "20" );

int sum = 0;
while( ! stack.isEmpty() )
{
   Object value = stack.pop();
   sum += (Integer) value; // runtime error!!
}

System.out.println(sum);
```

Motivation

- Classes should be designed as general as possible for better *reusability*
- Runtime errors are bad
- Compilation errors are better

- Can the compiler find errors like `stack.push( "20" )` during compilation time?

Simple Generic Example: Non-Generic Version

```
public class Comparison {

   private Comparable op1, op2;
   …

   public Comparable getOp1() { return op1; }
   public void setOp1( Comparable op1 ) { this.op1 = op1; }

   public Comparable max()
   {
      return op1.compareTo(op2) > 0 ? op1 : op2;
   }
}
```

Simple Generic Example: Generic Version

```
public class Comparison<T> {

   private T op1, op2;
   …

   public T getOp1() { return op1; }
   public void setOp1( T op1 ) { this.op1 = op1; }

   public T max()
   {
      ??
   }
}
```

Some Observations

- Converting from Non-generic version to generic version
  - Specify a type `T`
  - Replace `Comparable` with `T`
  - T is assumed to be a class type
  - T can be specified as a subclass/subtype of some class/interface
- Syntax of defining and using the class

Syntax

Class definitions:

public class Comparison**<T>**

public class Comparison**<T extends Comparable>**

pubic class Foo**<T,S>**

Declarations and initialization:

Comparison**<Integer>** c1 = new Comparison**<Integer>**();

Comparison**<String>** c2 = new Comparison**<String>**();

Terminology …

- Generic, generic type
  - A.K.A. *parameterized type*, *parameterized class*
  - A.K.A. *Template* (in C++)
  - E.g. `Comparison<T>`
- Type parameter, e.g. `T`
- Concrete type
  - E.g. `Comparison<Integer>`, `Comparison<String>`
- Raw type
  - E.g. `ArrayList` (instead of `ArrayList<T>`)

… Terminology

- Static vs. Dynamic
  - Static: things happened during compilation time
  - Dynamic: things happened during runtime
- Static/dynamic type checking

Another Example: Generic ArrayStack

- Convert `ArrayStack` to a generic type
  - What works??
  - What doesn't??

Type Erasure

- Compiler removes the information about the type parameters so such information is not available at runtime
  - Maintain binary compatibility with older Java applications and library
- Causes problem for some code, e.g.

```
if (item instanceof T) // error!!

T item = new T(); // error!!

T items[] = new T[10]; // error!!
```

Generic Method

- Suppose we want to add a static method `T max(T a, T b)` to Comparison class

```
public static T max( T a, T b )  // error!!
{
      return a.compareTo(b) > 0 ? a : b;
}
```

Why do we get an error when the non-static `max()` method is fine??

About Generic Methods

Syntax:

```
public static <T extends Comparable> T max( T a, T b )
{
      return a.compareTo(b) > 0 ? a : b;
}
```

Type inference:

```
Comparison.max( 10, 20 );
Comparison.max( "cs203", "monday" );
Comparison.max( 10, "20"); // error!!
```

Comparable vs. Comparable<T>

```
static <T extends Comparable> T max( T a, T b )

static <T extends Comparable<T>> T max( T a, T b )
```

Explain the different outcome we got when we try to do `Comparison.max(10, "20")`

Wildcard Type

- Suppose we want to write a method that can accept a list of any type, e.g. `List<Integer>`, or `List<String>`…
- For example:

```
public void reverse( List<?> list )
```

Wildcard type

More About Generics

- Generics in the Java Tutorial - http://java.sun.com/docs/books/tutorial/java/javaOO/generics.html
- More in-depth discussion of Generics - http://java.sun.com/docs/books/tutorial/extra/generics/index.html
- Java Generics FAQ - http://www.angelikalanger.com/GenericsFAQ/JavaGenericsFAQ.html