CS601: Principles of Software Development

Generics.

Olga A. Karpenko

This presentation is based on the lecture notes of Anupam Chanda.

Java Generics

- Allow a type or method to operate on objects of various types
  - compile-time type safety

Example of using parameterized class:

```
List<Employee> list = new ArrayList<Employee>();
```

Motivation

```
public class Box {
        private Object data;
        public Box(Object data) {
                this.data = data;
        }
        public Object getData() {
                return data;
        }
}
```

```
Box intBox = new Box(42);
Integer i = (Integer)intBox.getData();

Box stringBox = new Box( “IamAString” );
String s = (String) stringBox.getData();
```

Motivation

```
 public class Box {
       Object data;
       public Box(Object data) {
             this.data = data;
       }
       public Object getData() {
             return data;
       }
 }

Box intBox = new Box(42);
Integer i = (Integer)intBox.getData();

Box stringBox = new Box( “IamAString” );
String s = (String) stringBox.getData();

Integer n = (Integer)stringBox.getData();
```

Motivation

```
 public class Box {
        Object data;
        public Box(Object data) {
               this.data = data;
        }
        public Object getData() {
               return data;
        }
 }

Box intBox = new Box(42);
Integer i = (Integer)intBox.getData();

Box stringBox = new Box( “IamAString” );
String s = (String) stringBox.getData();

Integer n = (Integer)stringBox.getData(); // Compiles
// ClassCast exception at runtime
```

Motivation

```
 public class Box {
        Object data;
        public Box(Object data) {
               this.data = data;
        }
        public Object getData() {
               return data;
        }
 }

Box intBox = new Box(42);
Integer i = (Integer)intBox.getData();

Box stringBox = new Box( “IamAString” );
String s = (String) stringBox.getData();

intBox = stringBox; // Compiles and runs
Integer s1 = (Integer)intBox.getData(); // Compiles, but
RunTime Errors
```

Solution #1

- IntBox for Integers

```
public class IntBox {
    Integer data;
    public IntBox(Integer data){
        this.data = data;
    }
    public Integer getData() {
        return data;
    }
}
```

- StringBox for Strings

```
public class StringBox {
    String data;
    public StringBox(String data){
        this.data = data;
    }
    public String getData() {
        return data;
    }
}
```

Solution #1

```
IntBox intBox = new IntBox(15);
int x = intBox.getData();

StringBox strBox = new StringBox("Alice");
String s = strBox.getData();

Integer n = strBox.getData(); // Compiler error
intBox = strBox; // Compiler error
```

- Errors are now caught early
- What's wrong with this solution?

Solution #1

```
IntBox intBox = new IntBox(15);
int x = intBox.getData();

StringBox strBox = new StringBox("Alice");
String s = strBox.getData();

Integer n = strBox.getData(); // Compiler error
intBox = strBox; // Compiler error
```

- Errors are now caught early
- What's wrong with this solution?
  - Maybe infinitely many classes

Java Generics

- Parameterize type definitions
  - Parameterized classes and methods
- Provide type safety
  - Compiler performs type checking
  - Prevent runtime cast errors

Solution #2: Parameterized Class

```
public class Box<T> {

  private T data;
  public Box(T data) {
    this.data = data;
  }
  public T getData() {
    return data;
  }
}
```

- T is a type, a parameter to the class

Type Parameter Naming Conventions

- A single uppercase letter
- Commonly Used Names
  - T
  - N - number
  - E - element
  - K - key
  - V - value
  - S, U - second, third types

Solution #2: Parameterized Class

```
public class Box<T> {

  private T data;
  public Box(T data) {
    this.data = data;
  }
  public T getData() {
    return data;
  }
}
```

- To use this class, T must be replaced with a specific class

Solution #2: Parameterized Class

```
public class Box<T> {

  private T data;
  public Box(T data) {
    this.data = data;
  }
  public T getData() {
    return data;
  }
}

Box<Integer> intBox = new Box<Integer>(15);
Integer n = intBox.getData(); //no casting needed
```

Solution #2: Parameterized Class

```
public class Box<T> {

  private T data;
  public Box(T data) {
    this.data = data;
  }
  public T getData() {
    return data;
  }
}

Box<Integer> intBox = new Box<Integer>(15);
Integer n = intBox.getData();//no casting needed

Box<String> strBox = new Box<String>("Alice");
String s = strBox.getData(); //no casting needed
```

Using Parameterized Classes

- Will these errors be caught by Compiler or at Runtime?

```
String s = (String)intBox.getData();
int y = (Integer)strBox.getData();
intBox = strBox;
```

Using Parameterized Classes

- Will these errors be caught by Compiler or at Runtime? Compile time!

```
String s = (String)intBox.getData();
int y = (Integer)strBox.getData();
intBox = strBox;
```

Parameterized Classes

- Particularly useful for “container” classes
  - Containers hold but do not process data
  - Collection framework classes are defined using generics

Syntax

- Multiple parameters:

```
public class GenericClass<T, S, U> {

    //  …

}
```

Example

- Implementing a generic Stack class
  - `Stack.java`
  - `StackExample.java`
- From Deitel & Deitel, “ Java: How to program”

Methods

A Regular Method inside a Parameterized Class

```
public class Bar<T> {  // Bar is parameterized
  public T myMethod(T x) { // regular method
     return x;
  }

  public static void main(String[] args) {

   Bar<Integer> bar = new Bar<Integer>();
   int k = bar.myMethod(5);
   String s = bar.myMethod("abc"); //Compiler error
  }
}
```

- Created Bar<Integer>, so are locked to a specific T

Parameterized Methods

- Class Foo is **not** parameterized
- myMethod **is** parameterized

```
public class Foo {

   public <T> T myMethod(T x) { // parameterized method
        // Note: will not compile without <T>
        return x;
   }

  public static void main(String[] args) {
   Foo foo = new Foo();
   int k = foo.myMethod(5);
   String s = foo.myMethod("abc");

  }
}
```

Use of Parameterized Methods

- Adding type safety to methods that operate on different types
  - Return type dependent on input type

Examples

- StackUtil.java
- MapUtil.java