Lecture 14

Generics

References:

1. Herbert Schildt, Chapter 14, The Complete Reference Java 10th edition, McGraw Hill

1

Topics covered

- What are Generics?
- Generic Class with Two Type Parameters
- Bounded Types
- Creating a Generic Method
- Generic Interfaces
- Generic Class Hierarchies

2

Generics

- Define an algorithm once
  - Independently of any specific type of data
- Collection is a group of objects
  - Collection: list, set, queue,…
  - Able to work with any type of object
- Generic classes, interfaces, and methods created in a type-safe manner
  - Compared to generalized Object
- C++
  - Templates

3

What are Generics? (1/2)

- Generics
  - Generics mean parameterized types
  - Class, interface, or method operating on generics, namely parameterized types
  - Referred to as generic class, interface, or method

4

What are Generics? (2/2)

- Prior to Generics
  - Object reference can refer to any type object
  - Generalize classes, interfaces, and methods using Object references to operate on various types of objects
  - However, they are not type-safe
- Need Generics
  - For type safety
    - Streamlined the process without explicit casts between Object and a type of data

5

Simple Generics Examples (1/2)

- E.g., GenDemo.java
  - T is a type parameter
  - Use an angle bracket < > to define a type parameter
  - Gen is a generic class

6

Simple Generics Examples (2/2)

- Generics work only with Reference Types
  - Cannot use primitive type
  - Gen<int> intOb = new Gen<int>(53) //wrong!
- Generic Types differ based on Type Arguments

  Gen<Integer> iOb = new Gen<Integer> (88);

  Gen<String> strOb = new Gen<String> (“Generics Test”);

  ….

  iOb = strOb // Wrong!

Generalized Object

- E.g., NonGenDemo.java
  - Replace T with Object
  - Java compiler does not have any knowledge about the type of data stored in NonGen
    - Explicit casts needed to retrieve the stored data
    - Type mismatch errors cannot be found until run time
- Through generics
  - Runtime errors converted into compile-time errors
- Benefit of generics
  - the type safety ensured for all operations
    - No need type casts

Generic Class with Two Type Parameters

- More than one type parameter declared in generic classes
  - Use a comma-separated list
  - E.g., SimpleGen.java
- General Form of a Generic Class

  Class class-name<type-param-list> { // … }

  Class-name<type-arg-list> var-name = new class-name<type-arg-list>(cons-arg-list);

9

Bounded Types (1/3)

- Parameter types bounded using class hierarchy
- E.g., Stats<T> program

```
// Stats attempts (unsuccessfully) to create a generic class that can compute
// the average of an array of numbers of any given type.  The class contains an error!
class Stats<T> {
  T[] nums; // nums is an array of type T
  // Pass the constructor a reference to an array of type T.
  Stats(T[] o) {
    nums = o;
  }
  // Return type double in all cases.
  double average() {
    double sum = 0.0;
    for(int i=0; i < nums.length; i++)
      sum += nums[i].doubleValue(); // Error!!!
    return sum / nums.length;
  }
}
```


10

Bounded Types (2/3)

- Stats<T> program
  - All numeric classes are subclasses of Number
  - Number defines doubleValue()
    - Available to all numeric wrapper classes
  - Compiler has no way to know
    - You intend to use only numeric types

Bounded Types (3/3)

- Java provides bounded types
  - An upper bound declared with the superclass
  - Use of an *extends* clause
    - <T extends superclass>
    - T replaced by superclass or subclass of superclass
- E.g., BoundsDemo.java

Creating a Generic Method

- A generic method using one or more type parameters
  - Defines constraints for parameter types
  - E.g., GenMethDemo.java
    - <T extends Comparable <T>, ….>
    - T requiring an upper bound of **Comparable**
    - Comparable <T> is an interface
      - Instantiates objects to compare
- Generic Constructors
  - Possible for constructors to be generic

Generic Interfaces

- Possible to have generic interfaces
- If a class implements a generic interface
  - That class must also be generic
  - E.g., MyClass.java

Generic Class Hierarchies

- A generic class acting as a superclass or a subclass
- Using a Generic Superclass
  - At least one subclass needed to define a generic class type
  - Subclass also can have its own a generic class type parameter
  - E.g., HierDemo.java