Java Generics

Lab #5

COMP3021 2022 Spring

ChengPeng Wang(cwangch@cse.ust.hk)
Yiyuan Guo(yguoaz@cse.ust.hk)
Bowen Zhang(bzhangbr@cse.ust.hk)
Heqing Huang(hhuangaz@cse.ust.hk)

- **Objectives of this lab**

  Learn how to programming with Java Generic types and methods

  Learn how to use bounds and wildcards

- **The world without Generics**
  - Use non-Generic version
    - List nums = new ArrayList();
      nums.add(1);
      nums.add("a string");
  - Can we assure that it returns integer?
    - nums.get(0);
  - Need to define a new List for every contained Java type?
    - IntList
    - FloatList
    - DoubleList
    - ListofList
    - …
  - Then our code could be very ugly…

- **Java Generics**
  - Generics enables **types** to be parameterized when defining classes, interfaces, and methods.
  - It enables us to re-use code on different types.
    - E.g. sort on integer, float, double, …
  - We can do compile-time type checking based on it

- **Generic Class and Interface**
  - Define a generic class

```
class Test<T> {
    public void show(T obj) {System.out.println(obj);}
}
```

  - Define a generic interface

```
interface Comparable<T> {
    public int compareTo(T other);
}
```

  - Use instanceof on generic instances
    - A generic class is shared by all its instances regardless of its actual type

```
Test<Integer> t1 = new Test<Integer>();
Test<String> t2 = new Test<String>();

s1 instanceof Test; // true
s2 instanceof Test; // true
```

- **Generic method**
  - Generic method is written with single method definition, but can be called with arguments of different types.
    - The type parameter should be placed before the return type

```
public static <E> void printArray(E[] lst) {
    for(E e: lst) {
        System.out.println(e + "");
    }
}
```

```
public static void main(String args[]) {
    Integer[] intArray = new Integer[] {1,2,3};
    Stringp[] stringArray = new String[] {"hello", "world"};

    printArray(intArray);
    printArray(stringArray);
}
```

  - Generic methods can have more than 1 type parameter, separated by commas in method signature

- **Generics in static context**
  - Static context should have its own type parameters.
    - The first one is not allowed because X belongs to the instance of this class

```
public class Test<X> {
    public static void func1(X arg) {} // not allowed
    public static <Y> func2(Y arg) {} // allowed
}
```

- **Type Erasing**
  - Type information is not available at runtime and all the generics stuffs are process as java.lang.Object.

  - Primitive types are not allowed to be type parameters
  - Can not make any use of type parameter at runtime
  - Exception types can not be generic

- **Bounded Generics**
  - We can restrict the types that can be accepted by a method
    - For example, we can specify that we accept the type and all its subclasses.

```
public <T extends Number> List<T> fromArrayToList(T[] a) {...}
```

- **Wildcards and Inheritance**
  - Is ArrayList<Integer> a subtype of ArrayList<Number>?
    - No
  - Wildcards comes to solve this problem
    - You can read “?” as “anything”
  - Some facts:
    - ArrayList<Object> is a subtype of ArrayList<?>
    - ArrayList<Integer> is a subtype of ArrayList<? Extends Number>
    - ArrayList<Number> is a subtype of ArrayList<? Extends Number>
    - ArrayList<?> is a subtype of ArrayList<? Extends Object>
    - List<? Extends Integer> is a subtype of List<? Extends Number>

- **LAB #5 Keypoints**

1. Download template code Heap.java, and implement the TODOs.
   - In Lab#5, we use package **lab5**, because it's not in series with Lab 2 3 4
2. Download HeapTest.java, and test your code.
   - All the testcase should output "**Success**".
3. Submission
   - Push code to Github
   - Upload screenshots of your:
     - Main function code
     - execution result of main

- **LAB #5 Goal**
  - We are implementing a generic min heap for comparable types.
    - https://en.wikipedia.org/wiki/Heap_(data_structure)

END OF LAB #5

Don’t forget to commit and push your code.