Practical Exercise 3 – Generics

Overall Objective

To understand and write programs that use generic classes and interfaces.

Background

You will need to know:

1. basic Java programming knowledge
2. classes and interfaces
3. ArrayList
4. **generics**

Description

Part 1: Discussion

1. Are there any compile errors in (a) and (b)?

```
ArrayList dates = new ArrayList();

dates.add(new Date());
dates.add(new String());
```

(a) Prior to JDK1.5

```
ArrayList <Date> dates =
     new ArrayList <Date>();
dates.add(new Date());
dates.add(new String());
```

(b) Since JDK 1.5

Answer:
(a) will compile fine, but (b) has a compilation error on Line 3, because dates is declared as a list of `Date` objects. You cannot assign a string to the list.

2. What is wrong in (a)? Is the code in (b) correct?

```
ArrayList dates = new ArrayList();

dates.add(new Date());
Date date = dates.get(0);
```

(a) Prior to JDK1.5

```
ArrayList <Date> dates =
     new ArrayList <Date>();
dates.add(new Date());
Date date = dates.get(0);
```

(b) Since JDK 1.5

Answer:
Casting is needed in (a), but no casting is necessary in (b) with the generic type `ArrayList<Date>`.

3. What are the benefits of using generic types?
Answer:
One important benefit is improving reliability and robustness. Potential errors can be detected by the compiler.

4. How do you declare a generic type in a class?
Answer:
To declare a generic type for a class, place the generic type after the class name, such as `GenericStack<E>`. To declare a generic type for a method, place the generic type for the method return type, such as `<E> void max(E o1, E o2)`.

5. How do you declare a generic method? How do you invoke a generic method?
   Answer:
   To declare a generic method, you place the generic type `<E>` immediately after the keyword `static` in the method. A generic method can be invoked just like a regular method. The compiler automatically discovers the actual type.

6. What is raw type? Why is a raw type unsafe?
   Answer:
   When you use generic type without specifying an actual parameter, it is called a raw type. A raw type is unsafe, because some errors cannot be detected by the compiler. The raw type is allowed in Java for backward compatibility.

7. Refer to the *WildCardNeedDemo.java*, *AnyWildCardDemo.java* and *SuperWildCardDemo.java* programs, identify the differences among an *unbounded wildcard*, a *bounded wildcard*, and a *lower-bound wildcard*.
   Answer: ? is unbounded wildcard
   ? extends T is bounded wildcard
   ? super T is lower bounded wildcard

8. What happens if lines 14-15 in *SuperWildCardDemo.java* program are changed to:
```
public static <T> void add(GenericStack<T> stack1,
      GenericStack<T> stack2)
```
   Answer:
   The program cannot be compiled, because the element type in `stack1` is `GenericStack<String>`, but the element type is `stack2` is `GenericStack<Object>`. `add(stack1, stack2)` cannot be matched.

9. What is erasure?
   Answer:
   Generic type information is used by the compiler to check whether the type is used safely. Afterwards the type information is erased. The type information is not available at runtime.

10. If your program uses `ArrayList<String>` and `ArrayList<Date>`, does the JVM load both of them?
    Answer: No. Only `ArrayList` is loaded.

11. What is the problem with this code: `E[] elements = new E[capacity]`?
    Answer:
    You cannot create an array using a generic type parameter because the type information is not available at runtime. Instead of it, you can circumvent this limitation by creating an array of the Object type and then casting it to `E[]`, as follows:
```
E[] elements = (E[]) new Object[capacity]
```

12. Can a method that uses a generic class parameter be static? Why?

Answer:

Since all instances of a generic class have the same runtime class, the static variables and methods of a generic class is shared by all its instances. Therefore, it is illegal to refer a generic type parameter for a class in a static method or initializer.

Part 2: Programming Exercise

1. ***Minimum element in an* `ArrayList`**

   Implement the following method that returns the minimum element in an `ArrayList`.

   ```
   public static <E extends Comparable<E>> E min(ArrayList<E> list)
   ```

2. ***Largest element in an* `ArrayList`**

   Implement the following method that returns the largest element in an `ArrayList`.

   ```
   public static <E extends Comparable<E>> E max(ArrayList<E> list)
   ```

Answer for Q1 and Q2:

```
import java.util.ArrayList;

public class FindMinAndMax {
  public static void main(String[] args) {
    ArrayList<Integer> list = new ArrayList<Integer>();
    list.add(14);
    list.add(24);
    list.add(4);
    list.add(42);
    list.add(5);

    System.out.print("Min = " + min(list));
    System.out.print("Max = " + max(list));
  }


  public static <E extends Comparable<E>> E min(ArrayList<E> list) {
    E currentMin = list.get(0);

    for (int i = 1; i < list.size(); i++)
      if (currentMin.compareTo(list.get(i)) > 0)
        currentMin = list.get(i);

    return currentMin;
  }

  public static <E extends Comparable<E>> E max(ArrayList<E> list) {
    E currentMax = list.get(0);

    for (int i = 1; i < list.size(); i++)
      if (currentMax.compareTo(list.get(i)) < 0)
        currentMax = list.get(i);

    return currentMax;
  }
}
```