Tutorial/Lab 4 - Generics

Aim

This tutorial/lab aims to further the understanding of Week 4's lecture on the topic of Generics.

Exercise 4.1 Generic ArrayList Shuffling

Consider again the Exercise 3.1 of last week, in which you are asked to write a method that shuffles an ArrayList of Numbers, using a data structure **ArrayList<Number>**. Now we have learned generic type. You task is to revise the data structure to store generic objects. The shuffle method should start with an identifier of the following format:

public static <E> void shuffle(ArrayList<E> list)

Guideline:
(a) Review the code you developed for Exercise 3.1 last week.
(b) Make necessary changes to the codes, to ensure that the ArrayList contains generic type of items, and the shuffle method can handle ArrayList of different types.
(c) Test your shuffle method with an ArrayList for **Integer** objects.

Exercise 4.2 Comparable & Generic

1. Based on your answer in Exercise 4.1 above, write a method that returns the largest element in the ArrayList. The method should start with the following identifier:

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

2. Use plain English to explain the meaning of the above line.
The method "max" takes an arraylist of _____, and returns _____.

Guideline to task 1:
(a) Finding the largest would require comparisons among the Arraylist's elements. Java has a build-in compareTo() method that you can use for this purpose.
(b) Test your max method with an ArrayList for **Integer** (not int).

Exercise 4.3 Multiple Generics

Read and implement the code below. It has a data structure **Pair** composed of two generic elements,**first** and **second**. These two elements have to be the same type **E**. A **print()** method is developed to output these two elements in a single line.

```
1  public class Pair<E> {
2      public E first; // First element
3      public E second; // Second element
4      /** Constructor: a pair e, f */
5      public Pair(E e, E f) {
6          first  = e;
7          second = f;
8      }
9      public static void main(String[] args) {
10         Pair<Integer> p1 = new Pair<>(1, 85);
11         Pair<Integer> p2 = new Pair<>(2, 63);
12         print(p1);
13         print(p2);
14     }
15     public static void print(Pair p) {
16         System.out.println(p.first +" "+ p.second);
17     }
18 }
```

After examining the code, we feel that **first** and **second** to be the same type is a too strong requirement. Instead, we may want to store different types, such as Integer, Double, or Strings (or other objects) in **first** and **second** in a single **Pair**. The code segment below shows an example. You task is to revise the above code, the **Pair** data structure specifically, so that it may be used to store different types of objects, such as the example shown below.

```
9   public static void main(String[] args) {
10      Pair<Integer, Double> p1 = new Pair<>(1, 85.5);
11      Pair<Integer, String> p2 = new Pair<>(2, "good");
12      print(p1);
13      print(p2);
14  }
```

Exercise 4.4 Wildcard

The code below has a "print" method designed to print out all elements of an ArrayList. The type of the elements is <Object>; the argument passed to is <Integer>. According to the knowledge of last lecture (page 47 of lecture 3), Integer is a subclass of Number which is subclass of Object. It seems that the print method should be able to process Integer arguments. However, the code below would not compile successfully. Why? Give an explanation and make ONE correction to the code so that it can compile and run successfully.

```
1  import java.util.*;
2
3  public class week4test1{
4      public static void main(String[] args) {
5          ArrayList<Integer> c = new ArrayList<>();
6          c.add(3);
7          c.add(4);
8          c.add(12);
9          print(c);
10     }
11     public static void print(ArrayList<Object> o)
12     {
13         for (Object e : o)
14             System.out.println(e);
15     }
16 }
```