CSC172 LAB 1

GENERICS

1 Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Labs will typically have an even number of components. The two partners in a pair programming environment take turns at the keyboard. This paradigm facilitates code improvement through collaborative efforts, and exercises the programmers cognitive ability to understand and discuss concepts fundamental to computer programming. The use of pair programming is optional in CSC172. It is not a requirement. You can learn more about the pair programming paradigm, its history, methods, practical benefits, philosophical underpinnings, and scientific validation at http://en.wikipedia.org/wiki/Pair_Programming

Every student must hand in their own work, but every student must list the name of their lab partner if any on all labs. The textbook and lectures present examples of the code necessary to complete this lab. Collaboration is allowed. You and your lab partner may discuss the lab with other pairs in the lab. It is acceptable to write code on the white board for the benefit of other lab pairs, but you are not allowed to electronically copy and/or transfer files between groups.

This lab has six parts. You and your partner(s) should switch off typing each part, as explained by your lab TA. As one person types the lab, the other should be watching over the code and offering suggestions. Each part should be in addition to the previous parts, so do not erase any previous work when you switch.

2 Generics

Consider the following code, where the printarray() method is used to print out arrays of different types:

```
Integer [] intArry = {1, 2, 3, 4, 5 };
Double [] doubArry = {1.1, 2.2, 3.3, 4.4};
Character [] charArray = {'H','E','L', 'L', 'O' };
String [] strArray = {“once”, “upon”, “a”, “time” };
printarray(intArry);
printarray(doubArry);
printarray(charArray);
printarray(strArray);
```

1. Write a Java program with a main method that includes the example code above. Implement the static printarray() method using an array of Objects for the parameter. Compile and run your code.

2. Comment out, but do not delete, the previous implementation of the printarray() method and implement the printarray() method using method overloading. Write four versions of printarray(), one for each appropriate array type. Compile and run your code.

3. Comment out, but do not delete, the previous implementations of printarray(). Now write a single method that uses the Generic programming technique so that you have a single method supporting all the types and maintain type safety.

4. Using non-generic techniques, write a method that takes an array of type Comparable and returns the maximum element in the array [i.e. `"public static Comparable getMax(Comparable [] a)"`]. Add the following code to your main routine:

```
System.out.println("max Integer is: " + getMax(intArry);
System.out.println("max Double is: " + getMax(doubArry);
System.out.println("max Character is: " + getMax(charArry);
System.out.println("max String is: " + getMax(strArry);
```

   Compile your code. Copy and paste any compiler warnings you get into a comment block at the head of your implementation of getMax(). Run your code.

5. Comment out, but do not delete, your previous implementation of getMax(). Using the generic techniques to specify super-class relationships, implement a type safe version of getMax().

6. Consider the code example of the Function Object in Figures 1.18 and 1.19 of your textbook (3rd Edition). Copy the code from Figure 1.18 into a new source code file. Modify the code so that it can also find the maximum Character in a Character array. You must use the Function Object technique. Be sure to create a Character array and print the result of calling findMax() on it in your main method. The comments in your code should contain a citation of the source.

3 Hand In

Hand in the source code from this lab at the appropriate location on the Blackboard system at my.rochester.edu. You should hand in a single zip file (compressed archive) containing your source code, README, and OUTPUT files, as described below.

1. A plain text file named README that includes your contact information, your partner's name, a brief explanation of the lab (a one paragraph synopsis. Include information identifying what class and lab number your files represent.), and one sentence explaining the contents of any other files you hand in.
2. A single Java source code file representing the work accomplished for sections 1-5 of this lab. All source code files should contain author and partner identification in the comments at the top of the file.
3. Java source code files (you may decide how many you need) representing the work accomplished in section 6 of this lab. All source code files should contain author and partner identification in the comments at the top of the file.
4. A plain text file named OUTPUT that includes author information at the beginning and shows the compile and run steps of your code. The best way to generate this file is to cut and paste from the command line.

4 Grading

Each section (1-6) accounts for 15% of the lab grade (total 90%)

README file counts for the remaining 10%