CSCI 2110 Data Structures and Algorithms
Fall 2023
Laboratory No. 2
Week of September 18-September 22

Due: Sunday, September 24, 2023, 11.59 PM

All the data structures that we will be implementing in this course will be with Generics. The objective of this lab is to familiarize you with Generics in Java. Please be sure to read all the instructions and do the exercises in a step-by-step manner.

Marking Scheme: Throughout this course, we will be focusing on the complexity of algorithms and consequently, the efficiency of programs. While in your first year, your main objective was getting the code to work, in this course you must not only get the code to work, but also make it efficient. This will involvereducing unnecessary coding and designing, choosing good algorithms, and coming up edge cases to test your program.

Each exercise will list the test cases that you need to test the code. Ensure that the output that you generate covers all the test cases.
Each exercise carries 10 points.
Your final score will be scaled down to a value out of 10. For example, if there are three exercises and you score 9/10, 10/10 and 8/10 on the three exercises, your total is 27/30 or 9/10.

Error checking: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input

Submission Requirements:

- No submission other than a single ZIP file will be accepted.
- You MUST SUBMIT .java files that are readable by the TA. If you submit files that are unreadable such as .class file, the lab will be marked 0.
- Please additionally comment out package specifiers.

What to submit:

- One ZIP file containing all source code (files with .java suffixes) and a text file with sample inputs and outputs.

Late Submission Penalty: The lab is due on Sunday at 11.59 PM. Late submissions up to 5 hours (4.59 AM on Monday) will be accepted without penalty. After that, there will be a 10% late penalty per day on the mark obtained. For example, if you submit the lab on Monday at 12 noon and your score is 8/10, it will be reduced to 7.2/10. Submissions past two days (48 hours) after the grace submission time, that is, submission past 4.59 AM Wednesday will not be accepted.

Exercise 0 (Review – no marks)

This is a simple exercise that was discussed in the lectures. Test the code provided, change the parameters and try to make sure you understand how generics work in Java.

```
public class Grade<T>{

  private T value;

  public Grade(T entry){
    value = entry;
    }

  public T getValue(){
    return value;
  }

  public void setValue(T entry){
    value = entry;
  }

  public String toString(){
    return ""+value;
  }
}

//Demo class
public class Exercise0{
  public static void main(String[] args){

    Grade<String> m1 = new Grade<String>("A+");
    Grade<Integer> m2 = new Grade<Integer>(91);

    System.out.println(m1);
    System.out.println(m2);

    m1.setValue("B+");
    m2.setValue(78);

    System.out.println(m1);
    System.out.println(m2);
  }
}
```

Exercise1 (Point Class)

For this exercise, you will write

- a Point class that has xpos and ypos as its fields,
- get and set methods,
- a toString method.

It must be written as a generic class, which means that it should work for any type of Object. The `xpos` and `ypos` could be either Integer Objects, or Double Objects, or String Objects.

A sample test program is given below. **Modify the test program to accept input data from a user and try it for different input values. Name the program PointDemo.java.**

```
public class PointTester{
  public static void main(String[] args){
    Point<Integer> point1 = new Point<Integer>(10, 20);
    Point<Double> point2 = new Point<Double>(14.5, 15.6);
    Point<String> point3 = new Point<String>("topleftx", "toplefty");

    System.out.println(point1);
    System.out.println(point2);
    System.out.println(point3);
  }
}

Output:
XPOS: 10    YPOS: 20
XPOS: 14.5  YPOS: 15.6
XPOS: topleftx  YPOS: toplefty
```

Your program should accept input on three lines, one pair of integers, one pair of doubles, and one pair of Strings, as shown in the sample run below. The program should then construct the three Point objects and display their x and y coordinates as shown below. Do not hard code inputs.

A sample run of the program would look as follows:

```
Enter integer coordinates x and y: 23 400
Enter double coordinates x and y: 2.45 0.01
Enter String coordinates x and y: somewhere elsewhere

XPOS: 23 YPOS: 400
XPOS: 2.45 YPOS: 0.01
XPOS: somewhere YPOS:  elsewhere
```

Test your program with a variety of inputs to ensure its proper operation. **Save data from one test run in a text file for submission.**

In Exercises 2 and 3, you will design a **Generic Stack data structure using an ArrayList**.

Exercise2 (Generic Stack)

A Stack is a Last-In-First-Out data structure that has push, pop, and peek as its basic operations. For this exercise, you will write a GenericStack class that is implemented with an ArrayList to store the Stack elements. The class has the following specifications:

Instance Variables

stack: A generic ArrayList<T>

size: integer

Methods

Constructor: Creates an empty Stack object

int size(): returns the size of the Stack object

T peek(): returns the item at the top of the Stack object; Stack is not altered

void push(T item): pushes an item to the top of the Stack object

T pop(): pops the top item from the Stack object and returns it

boolean isEmpty(): returns true if the Stack object is empty, false otherwise

The initial part of the code is given below. Fill in the TODO lines of code.

```
import java.util.ArrayList;

public class GenericStack<T>{
     private ArrayList<T> stack;
     private int size;

     //TODO – continue your code here

}
```

Next write the GenericStackDemo class. The program should do the following:

a) Create two Stack objects, one for storing Strings and another for storing Integer objects.
b) Accept 2 lines of input from a user. The first line of input is an arbitrary number of Strings separated by whitespace to be pushed into a String Stack object, with the final String 'quit' indicating the end of input.

   The second line of input will be an arbitrary number of integers (Positive integers only) separated by whitespace to be pushed into an Integer Stack object, with the final int '-1' indicating the end of input.

c) Push the input values into the appropriate Stack.
d) Pop each Stack and display the results.
e) In your display, the first output line lists the contents of the String Stack, each element on its own line. The second output lists the contents of the Integer Stack, each element on its own line.

A sample screen dialog for GenericStackDemo is given below.

```
Halifax Montreal Toronto Calgary Vancouver quit

10 534 678 90008 52 3 20 34 –1

String Stack Contents:
Vancouver
Calgary
Toronto
Montreal
Halifax

Integer Stack Contents:
34
20
3
52
90008
678
534
10
```

The template for GenericStackDemo.java is given below. Fill in the TODO lines of code.

```
import java.util.*;
public class GenericStackDemo{
    public static void main(String[] args){

     GenericStack<String> stack1 = new GenericStack<String>();
     GenericStack<Integer> stack2 = new GenericStack<Integer>();

    //TODO- continue your code here


}
```

Test your program for the given test case and one more test case of your own to ensure its proper operation. Do not hard code inputs. Save data from one test run into the output text file for submission.

Exercise3 (Student Records)

This exercise will reuse the Generic Stack you created in the previous one. You will store a new Object, called a StudentRecord on your Stack. Write a StudentRecord class with three fields: `String firstName`, `String lastName`, `int bannerID`. Add an appropriate constructor, get, set and `toString` methods.

Next, write a test program. Your program should accept input from a user in the form of a String representing a file name, then read a file that contains student records, one on each line. The file will look like this:

```
Ichabod Crane 123456
Brom Bones 456321
Emboar Pokemon 111222
Rayquazza Pokemon 333111
Cool Dude 101010
Trend Chaser 654321
Chuck Norris 112233
Drum Dude 111222
```

The above file (called StudentList.txt) has been provided to you alongside the Lab document.

Your program should accept input in the following format:

a) Your program should accept 1 line of input from a user, representing the name of an input file.
b) It should then read an input file of an arbitrary number of lines.
c) Each line of the input file will consist of a first name, last name, and ID number separated by whitespace.

Note: A **StringTokenizer** could be used to read a file like the one described in this exercise, splitting lines into first name, last name and ID number. If you implement this solution, you will likely read the ID number as a String and then have to convert it to an Integer object. Some of the code needed to get started working with a StringTokenizer is given in the partial solution that follows.

A sample test program is given below. Your test program should perform the following tasks:

1. Create a Stack (`stack1`) to hold StudentRecord Objects.
2. Read a line from the input file, create a StudentRecord object, and push it onto `stack1`.
3. Repeat until all lines in the input file are read.

4. Create a Stack (`stack2`) to hold String Objects.
5. Pop `stack1` item by item and push the last name associated with each StudentRecord onto `stack2`.
6. Pop items from `stack2` and print.

With the example set of records given above, this would display the last names in the same order:

```
Crane
Bones
Pokemon
Pokemon
Dude
Chaser
Norris
Dude
```

A sample test program is given below. (It is assumed that you have a separate StudentRecord class.) Fill in the TODO lines.

```
import java.util.*;
import java.io.*;

public class Exercise3{
  public static void main(String[] args)throws IOException{

    //TODO: Create stack1 to hold StudentRecord Objects

    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the filename to read from: ");
    String filename = keyboard.nextLine();

    File file = new File(filename);
    Scanner inputFile = new Scanner(file);
    StringTokenizer token;
    while (inputFile.hasNext()){
      String line = inputFile.nextLine();
      token = new StringTokenizer(line, " ");
      String firstName = token.nextToken();
      String lastName = token.nextToken();
      String IDString = token.nextToken();
      //convert String IDString to an Integer Object IDNum
      Integer IDNum = Integer.valueOf(IDString);

      //TODO: Create a StudentRecord Object with the first name, last name
      //and ID number, push it into stack1

    }
    inputFile.close();

    //TODO: Continue with remaining steps

  }
}
```

A sample test run (for the input file StudentList.txt) is given below:

```
Enter the name of the input file:  StudentList.txt

Crane
Bones
Pokemon
Pokemon
Dude
Chaser
Norris
Dude
```

Test your program with different input files to ensure its proper operation. Do not hard code inputs. **Save data from the test run for the input file StudentList.txt (as given above) for submission.**

NOTE REGARDING STATIC GENERIC METHODS: If you need to include **static generic methods** in your test program or other classes, the header for the method should be written as follows:

```
public static <T> returnType methodName(input parameters)
```

Here's a simple generics program with a static method:

```
public class GenericMethodDemo{
  public static void main(String[] args){
    Integer[] integers = {4, 5, 6, 7};
    String[] strings = {"A", "B", "C", "D"};
    GenericMethodDemo<Integer> print(integers);
    GenericMethodDemo<String> print(strings);
  }

  public static<T> void print(T[] list){
    for(int i=0; i<list.length; i++)
      System.out.print(list[i] + " ");
    System.out.println();
  }
}
```

Submission Summary

Document Type	File name	Exercise
Java Source Codes	Point.java	Exercise 1
	PointDemo.java	
	GenericStack.java	Exercise 2
	GenericStackDemo.java	
	StudentRecord.java	Exercise 3
	Exercise3.java	
txt	Output.txt	Exercises 1- 3

What to submit: One zip file containing the above documents.