CS20B: Homework 2 (80 points)

prof. dr. Irma Ravkic

Instructions: When you're creating your answer document please copy-paste the questions you're answering.

1 Questions (10 points)

1. (4 points) Explain why *false* is printed for this piece of code. How would you fix it to print *true*.

```java
    String s1 = new String("SMC");
    String s2 = new String("SMC");
    System.out.println(s1 == s2);
```

2. (6 points) True or False? Explain your answers.

   (a) (1 point) You can define constructors for a Java interface.

   (b) (1 point) Classes implement interfaces.

   (c) (1 point) Classes extend interfaces.

   (d) (1 point) A class that implements an interface can include methods that are not required by the interface.

   (e) (1 point) A class that implements an interface can leave out methods that are required by an interface

   (f) (1 point) You can instantiate objects of an interface.

2 Programming exercises (90 points)

For the programming exercises, please follow the provided instructions in the instructions.pdf. Each of the exercises below should be a separate zip of a Java project.

1. Project 1: (20 points) For this problem you must define a simple generic interface *PairInterface*, and two implementations of the interface, *BasicPair* and *ArrayPair*.

   (a) (4 points) Define a Java interface named *PairInterface*. A class that implements this interface allows creation of an object that holds a "pair" of objects of a specified type — these are referred to as the "first" object and the "second" object of the pair. We assume that classes implementing *PairInterface* provide constructors that accept as arguments the values of the pair of objects. The *PairInterface* interface should require both setters and getters for the first and second objects. The actual type of the objects in the pair is specified when the *PairInterface* object is instantiated. Therefore, both the *PairInterface* interface and the classes that implement it should be generic. Suppose a class named *BasicPair* implements the *PairInterface* interface. A simple sample application that uses *BasicPair* is shown here. Its output would be "apple orange."

```java
public class Sample {

public static void main (String[] args) {
    PairInterface myPair<String> =
    new BasicPair<String>("apple", "peach");
    System.out.print(myPair.getFirst() + " ");
    myPair.setSecond("orange");
    System.out.println(myPair.getSecond());
    }
}
```

(b) (3 points) Create a class called *BasicPair* that implements the *PairInterface* interface. This class should use two instance variables, *first* and *second*, to represent the two objects of the pair. Create a test driver application that demonstrates that the *BasicPair* class works correctly.

(c) (3 points) Create a class called *ArrayPair* that implements the *PairInterface* interface. This class should use an array of size 2 to represent the two objects of the pair. Create a test driver application that demonstrates that the *ArrayPair* class works correctly.

2. Project 2: (20 points) Add the following methods to the **ArrayBoundedStack** class, and create a test driver to show that they work correctly. In order to practice your array related coding skills, code each of these methods by accessing the internal variables of the ArrayBoundedStack, not by calling the previously defined public methods of the class.

    - (2.5 points) String toString()—creates and returns a string that correctly represents the current stack. Such a method could prove useful for testing and debugging the class and for testing and debugging applications that use the class. Assume each stacked element already provided its own reasonable toString method.
    - (2.5 points) int size()—returns a count of how many items are currently on the stack. Do not add any instance variables to the ArrayBoundedStack class in order to implement this method.
    - (5 points) void popSome(int count)—removes the top count elements from the stack; throws StackUnderflowException if there are less than count elements on the stack.
    - (10 points) boolean swapStart()—if there are less than two elements on the stack returns false; otherwise it reverses the order of the top two elements on the

stack and returns true.

Please put and group all the methods you implement from above in the end of a single ArrayBoundedStack class so I can find it easily in your .java file.

3. Project 3: (30 points) Using the ArrayBoundedStack or ArrayListStack class, create an application *EditString* that prompts the user for a string and then repeatedly prompts the user for changes to the string, until the user enters an X, indicating the end of changes. Legal change operations are:

   U—make all letters uppercase

   L—make all letters lowercase

   R—reverse the string

   C ch1 ch2—change all occurrences of ch1 to ch2

   Z—undo the most recent change

   You may assume a "friendly user," that is, the user will not enter anything illegal. When the user is finished the resultant string is printed. For example, if the user enters:

   All dogs go to heaven

   U

   R

   Z

   C O A

   C A t

   Z

   the output from the program will be "ALL DAGS GA TA HEAVEN"

   Make sure this driver test is possible to be compiled and run from command line. Include all the packages/dependencies in your submission!

4. Project 4: (20 points) Write code to **detect** if there are duplicates in a linked list stack or not. Make a test class that will test if the detection of the duplicates works. Test your code on the following stack of Strings: "top: A" → "B" → "B"→ "A", for which your method should return *true.* Also write a test for which the method returns *false.* Make sure to document your code!

   Bonus for the adventurous ones: write a code to **remove** the duplicates from the linked list stack. When fed with "top: A" → "B" → "B"→ "A", it should update the linked list stack to contain only "top: A" → "B".