COMP 401 Midterm

Tuesday, Oct 17, 2017 11am-12:15pm

Instructions

1. Please spread out and try and sit in alternate seats.
2. This is a closed book exam.
3. You will not be penalized for errors in Java syntax.
4. Write on the exam itself. Write on a blank page (in particular, the last page) if there is not enough space to solve a problem and indicate you have done so.
5. There are:
   - 8 numbered pages including this one and any marked blank pages.
   - 3 questions.
   - 80 possible points. Point values appear in brackets next to each question.
6. You are not required to comment or annotate any code you write, but may get partial credit if you write appropriate comments/annotations but incorrect code.
7. If you need to make any assumptions to clarify a problem, *write your assumptions down.* Only reasonable assumptions get full credit.
8. Please inform the proctor of anything in the exam that you think is a mistake.
9. Your code will be evaluated not only for correctness, but also for time and space efficiency and *style*.
10. You cannot use any Java capabilities not covered in class.
11. To answer questions about some piece of code given here, you can mark the code directly.
12. If you do not understand some English word, do not hesitate to ask the proctor. Naturally, you are expected to know the computer science terms defined in class.
13. **Write clearly using a pen/pencil with a dark color** – we will be scanning your exams.
14. Please try and write in the allocated space, as that will reduce mistakes in grading of scanned exams.
15. Put your initials/onyen in the header of each page, in case the pages are separated. **Points have been allocated for writing the onyen and name clearly in capitals (Q1).**

NAME , ONYEN , and EMAIL if it is not onyen@live.unc.edu (ALL CAPITALS PLEASE)

(NAME) [2 pts]_____________________________________(ONYEN) [3pts]________________ EMAIL___________________

Pledge: I have neither given nor received unauthorized aid on this exam.

(signed)______________________________________

For survey purposes, please indicate the time at which you turned in the exam.

____

Please do not write below

1._____/5 2. _____/31 3. _____/34 4. _____/10

Total: _____/ 80

2. [31 pts.] TRIMMING A STRING

Implement a class, Trimmer, that implements a public **static** method, called trim(), that takes a single String argument and returns a *copy* of the string with leading and trailing spaces (these are prefix and suffix substrings containing only the space (' ') character) removed. The following main method and its output illustrates the use of this class.

Main method:

```
public static void main (String[] anArgs) {
      System.out.println ("Empty string:<" + Trimmer.trim("") + ">");
      System.out.println ("Space-only:<" + Trimmer.trim("  ") + ">");
      System.out.println ("Spaceless:<" + Trimmer.trim("hello:123")+ ">");
      System.out.println ("Leading one-space:<" + Trimmer.trim(" hello:123" + ">"));
      System.out.println ("Trailing one-space:<" + Trimmer.trim("hello:123 ") + ">");
      System.out.println ("Variable spaces:<" + Trimmer.trim("    hello:123    ")+">");
      System.out.println ("Interleaved spaces:<" + Trimmer.trim(" h e l l o ") + ">");
}
```

Output:

```
Empty string:<>
Space-only:<>
Spaceless:<hello:123>
Leading one-space:<hello:123>
Trailing one-space:<hello:123>
Variable spaces:<hello:123>
Interleaved spaces:<h e l l o>
```

Carefully understand what this class is supposed to do, as it is used as a running example for the remaining two questions also, which you can do in any order. You can get full credit for the other questions, without implementing any code in this question.

You can and are encouraged to not implement a single "monolithic" trim() method. To receive full credit, you should follow all style principles with which you are familiar to create code that is easy to understand, change and reuse, and does not have duplication. This means you are encouraged to implement additional meaningful (public and/or non-public) methods in the class that are called by trim(). You need not put these methods in another class.

To get full credit you must cover all cases and follow style rules. You will get the vast majority of points even if you do not follow the style rules. You will receive partial credit based on how many of the cases illustrated in the output are supported by your implementation. However, if you do not cover all cases, you must tell us what these cases are. For example if you cover only the first two cases, write down, on this page or in comments on the next page, the following:

CASES COVERED: *Empty String, Space-only string*.

This information will explain your code, and lack of such information can result in erroneous solutions getting no credit.

The String instance methods you can use are:

```
public char charAt(int index) //returns the char value at the specified index.
public int length()//returns the length of this string.
public String substring(int beginIndex, int endIndex)// Returns a new string that is a substring of this string.
// The substring begins at the specified beginIndex and extends to the character at index endIndex - 1.
// Thus the length of the substring is endIndex-beginIndex.
```

You can, of course, use the String "+" operator, which returns a concatenation of its two operands. You cannot use any other String method (such as the built-in String trim() method, which was not covered in class.)

[Initials/Onyen:]

3. [34 pts.] UNDERSTANDING CODE

Consider the following code, which is repeated in the following question:

```
public interface DuplicatedString {
	public String getString1();
	public String getString2();
}
```

```
public class ADuplicatedString implements DuplicatedString {
	String string;
	public ADuplicatedString(String aString) {
		string = aString;
	}
	public ADuplicatedString() {
	}
	public String getString1() {
		return string;
	}
	public void setString1(String newVal) {
		string = newVal;
	}
	public String getString2() {
		return getString1();
	}
}
```

```
public class ADuplicatedTrimmedString extends ADuplicatedString {
	public ADuplicatedTrimmedString(String aString) {
		super (aString);
	}
	public String getOriginalString() {
		return string;
	}
	public String getString1() {
		return Trimmer.trim(super.getString1());
	}
}
```

```
public class DuplicatedTrimmedStringDriver {
	public static String format(String aString) {// called by print
		return "<" + aString + ">";
	}
	public static void print(ADuplicatedString aDuplicatedString) {// called by main
		System.out.println("getString1():" + format(aDuplicatedString.getString1()));
		System.out.println("getString2():" + format(aDuplicatedString.getString2()));
	}
	public static void main (String[] args) {
		ADuplicatedString anUninitializedDuplicate =
				new ADuplicatedString ();
		print(anUninitializedDuplicate);
		ADuplicatedString anInitializedDuplicate =
				new ADuplicatedString (" Hello "); //single-space prefix and suffix
		print(anInitializedDuplicate);
		ADuplicatedTrimmedString anInitializedTrimmedDuplicate =
				new ADuplicatedTrimmedString(" Hello ");//single-space prefix and suffix
		print(anInitializedTrimmedDuplicate);
	}
}
```

[Initials/Onyen:]

(a)[2 pts] Name an instance variable in the given code.

(b)[2 pts] Name a property in the given code.

(c)[2 pts] Give headers of two overloaded methods/constructors in the given code.

(d)[2 pts] Name a superclass in the given code.

(e)[2 pts] Name an overridden method in the given code.

(f) [12 pts] Draw the logical structures of the objects assigned to `anInitializedDuplicate` and `anInitializedTrimmedDuplicate`. This is the structure derived from properties. **Make sure each distinct object is drawn only once; it is ok to show multiple edges (lines) ending in the same object.** Make sure to also label each edge with the appropriate property name. A string object can be depicted by showing its value. The drawings should show the differences between the logical structures of the two objects. You can also explain in words the differences to ensure there is no ambiguity.

(a) `anInitializedDuplicate` (b) `anInitializedTrimmedDuplicate`

(g) [12 pts] Give the output of the program.

4. [10 pts.] Improving Code Style

This is a repetition of the code of the previous question:

```
public interface DuplicatedString {
        public String getString1();
        public String getString2();
}
```

```
public class ADuplicatedString implements DuplicatedString {
        String string;;
        public ADuplicatedString(String aString) {
                string = aString;
        }
        public ADuplicatedString() {
        }
        public String getString1() {
                return string;
        }
        public void setString1(String newVal) {
                string = newVal;
        }
        public String getString2() {
                return getString1();
        }
}
```

```
public class ADuplicatedTrimmedString extends ADuplicatedString {
        public ADuplicatedTrimmedString(String aString) {
                super (aString);
        }
        public String getOriginalString() {
                return string;
        }
        public String getString1() {
                return Trimmer.trim(super.getString1());
        }
}
```

```
public class DuplicatedTrimmedStringDriver {
        public static String format(String aString) { // called by print
                return "<" + aString + ">";
        }
        public static void print(ADuplicatedString aDuplicatedString) {// called by main
                System.out.println("getString1():" + format(aDuplicatedString.getString1()));
                System.out.println("getString2():" + format(aDuplicatedString.getString2()));
        }
        public static void main (String[] args) {
                ADuplicatedString anUninitializedDuplicate =
                                new ADuplicatedString ();
                print(anUninitializedDuplicate);
                ADuplicatedString anInitializedDuplicate =
                                new ADuplicatedString (" Hello ");//single-space prefix and suffix
                print(anInitializedDuplicate);
                ADuplicatedTrimmedString anInitializedTrimmedDuplicate =
                                new ADuplicatedTrimmedString(" Hello ");//single-space prefix and suffix
                print(anInitializedTrimmedDuplicate);
        }
}
```

[Initials/Onyen:]

Explain how this code can be improved to better follow specific relevant stylistic rules, covered in class, not explicitly stated here. In other words, you must adhere to the same specific requirements that you would if this was a class assignment (with a perfect human grader!). Your answer should be as specific as possible, and ideally create Java code that shows all changes to the given code. You can mark on the code on the left and/or reproduce the changes. You do not have to explicitly state or justify the rules, but may get partial credit for doing so if you do not implement them correctly. You can assume all types you (re-) define are in the same package (the default package), and do not need to give any package declaration.