CS2112—Fall 2014
Assignment 1
Introduction to Java and Object-Oriented Languages
Due: Wednesday, September 3, 11:59PM

This assignment is an introduction to the Java language and basic object-oriented programming concepts, to help you become familiar with Java syntax and certain important language constructs.

0 Updates

- None yet; watch this space.

1 Instructions

1.1 Grading

Solutions will be graded on both correctness and style. A correct program compiles without errors or warnings, and behaves according the requirements given here. A program with good style is clear, concise, and easy to read.

A few suggestions regarding good style may be helpful. You should use brief but mnemonic variables names and proper indentation. Your code should include comments as necessary to explain how it works, but without explaining things that are obvious.

1.2 Partners

You *must* work alone for this assignment. But remember that the course staff is happy to help with problems you run into. Use Piazza for questions, attend office hours, or set up meetings with any course staff member for help.

1.3 Assignment structure

There are 10 questions grouped into 3 parts. Stub files are provided on CMS for each one. Questions may ask you to provide a handwritten response about the code, to modify the code, or to complete an implementation. All sections in the code that need to be filled out are marked with a `TODO` tag.

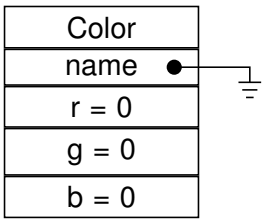

Figure 1: An object diagram for a newly created object of type `Color`

2 Semantics and object diagrams

> **semantics**:
> the study of the meanings of words and phrases in language
> — Merriam-Webster Dictionary

Programming language semantics is about understanding what programs mean. A useful way to understand what Java programs do is to draw object diagrams. In an object diagram, each object is represented by a box tagged with its run-time class. Each variable that references an object is represented by an arrow from the variable to the object.

For example, an object diagram for a newly created instance of the following class:

```
1 class Color {
2     String name;
3     int r = 0, g = 0, b = 0;
4 }
```

is shown in Figure 1. The initial value of instance variable `name` is `null`, meaning the variable does not have a value assigned yet. As a result, in the object diagram, the arrow from variable `name` does not point to anywhere.

Now, suppose we execute the following code snippet:

```
1 String s = "Red";
2 Color c = new Color();
3 c.name = s;
4 Object[] a = new Object[2];
5 a[0] = s;
6 a[1] = c;
```

The object diagram that results from the execution is shown in Figure 2. Notice that array objects use instance variable `length` to keep track of how many elements they contain, and each object of type `String` uses a `char` array named `value` to record the characters in the strings.

For the following problems, you are welcome to experiment by running the code to understand what it is doing.

1 Suppose Line 6 in the code snippet were changed from `a[1] = c;` to `a[1] = a;`. Draw the resulting object diagram.

Submit your answer as a PDF file named `p1q1_solution.pdf` in the root directory. Scans of handwritten diagrams are acceptable.

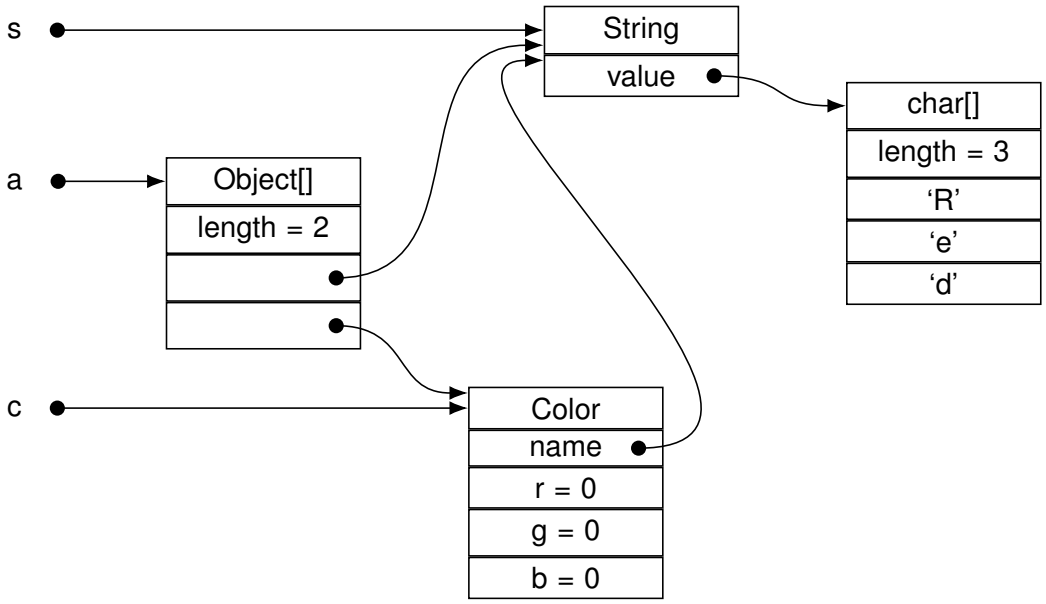

Figure 2: An object diagram after executing the code snippet

2 Suppose that instead of making the previous change, we added the following lines at the end of the above code snippet:

```
1 Object[] b = a;
2 b[1] = new Color[1];
3 b = (Color[]) a[1];
4 b[0] = c;
```

Draw the resulting object diagram.

Submit your answer as a PDF file named `p1q2_solution.pdf` in the root directory. Scans of handwritten diagrams are acceptable.

3 Coding and syntax

Implement the following short programs by modifying the given stub files. No points will be deducted for (reasonably) inefficient code.

You need not use all the library imports provided, but no additional library imports may be added. You may not change the declarations of the methods you are implementing without permission from the course staff. For example, the number and types of the parameters and the return type must remain unchanged. You may add new methods to the classes, however.

3 (Par: 3 lines) Implement method `print(int n)` that prints all of the numbers in range $[0, n-1]$ to the console in ascending order.

4 (Par: 7 lines) Implement method `stutteredFlatten(int n, String... list)` that returns a `String` of the concatenation of `n` copies of each item of `list` in the order as given.

Example: `stutteredFlatten(3, "abc", "de", "f")` returns `"abcabcabcdededefff"`.

5 (Par: 20 lines) Write a program that reads in a file from the file system and outputs its lines to the console in reverse order. The filename is given from the command line; that is, as an argument to the `main` method of the program.

6 (Par: 22 lines) Implement method `interact()` that reads the following inputs from the console, i.e., entered by the user from keyboard:

- The first line is an integer n.
- The next n lines are strings.
- The last line is an integer k.

With these inputs, call the method `stutteredFlatten` in Problem 4 that concatenates k copies of the given strings *in reverse order*.

If the user enters a value in an incorrect format, e.g., entering a string instead of an integer, keep asking the user to reenter the value.

Example: Suppose the inputs are

```
3
indian
little
one
2
```

The method should return "`oneonelittlelittleindianindian`" as the result.

7 (Par: 27 lines) Implement method `findUnionIntersection(int[] a, int[] b)` that takes two integer arrays and returns a new object containing both the union and the intersection of the arrays. Each of the two arrays can be assumed not to contain duplicate elements; that is, each array represents a set. The order of the elements in the resulting array does not matter, but the array should also represent a set.

4 Data structures

8 Run the code in `part3.Q8`. Draw an object diagram showing arrays `a` and `b` before and after the `for` loop executes.

Submit your answer as a PDF file named `p3q8_solution.pdf` in the root directory. Scans of handwritten diagrams are acceptable.

9 Run the code in `part3.Q9`. The method `isPalindrome(String s)` fails some tests. Make corrections so that it meets the specification and passes all tests.

10 A *linked list* is a data structure composed of nodes containing a value and a pointer pointing to the next node in the list. Linked lists can be implemented with a class having two fields: `value` holding some data in the node, and `next` holding a pointer to the next node in the list. The empty list is represented by the value `null`.

Complete the program `part3.Q10` to convert the arguments provided to the program on the command line into a linked list that is printed out.

5 HARMA

HARMA questions do not affect your raw score for any assignment. They are given as interesting problems that present a challenge.

11 Implement method `getConsecutiveSums(int n)` that returns an array of all arrays of consecutive positive numbers that sum to `n`. Because the numbers are consecutive, each array can be represented by only the first and last numbers. For example, to represent the array `[1, 2, 3, 4, 5]`, only `[1, 5]` is needed. The order of the arrays should be ascending in their actual sizes.

Example: `getConsecutiveSums(15)` should return array `[[15, 15], [7, 8], [4, 6], [1, 5]]` as the result.

HARMA: ♣ ♣ ♣ ♣ ♣

12 Implement method `solvePath(boolean[][] maze, Point start, Point end)` that returns `true` if a path exists between the start and end points, and `false` otherwise. A `true` in the maze indicates that traveling through that position is permitted, while a `false` indicates a wall. `Point` is a class containing two fields `x` and `y` representing a coordinate in the maze such that the upper-right position of the maze is at coordinate `(maze[0].length - 1, 0)`. A path is defined as a sequence of maze coordinates where each coordinate refers to a `true` position, and the position immediately before or after any given position in the sequence must be one of the up-to-four neighbors of the given position.

Examples: In the following, `1` represents `true` and `0` represents `false` in the maze, while `X` indicates one of the two end points. The maze values are always `true` at these `X`'s.

```
0 0 0
X X 0
0 0 0
```
Returns `true`.

```
0 0 0
X 0 X
0 0 0
```
Returns `false`.

```
0 0 0
X 1 X
0 0 0
```
Returns `true`.

```
0 X 0
X 0 0
0 0 0
```
Returns `false`.

```
0 X 1
X 0 1
1 1 1
```
Returns `true`.

```
X 0 1
0 1 0
1 0 X
```
Returns `false`.

HARMA: ♣ ♣ ♣ ♣ ♣

13 A topological map is a map representing the height of various locations on a terrain. Such a map can be represented by a two-dimensional array of integers where each number denotes the height of a location of the terrain.

Each topological map defines a *watershed*, an area of land where surface water from rain and melting snow or ice converges to a single point at a lower elevation[1]. In our setting, surface water at any given location always flows to the lowest of its up-to-four neighbors that is also lower than it. When two or more of these neighbors have the same lowest height, water can flow to any of them, and the given location could be part of multiple watersheds.

The size of a watershed is defined as the number of locations of the terrain for which surface water flows to the same contiguous region of the terrain. (See examples below.)

Implement method `getWatershedSizes(int[][] topo)` that takes a topological map of a piece of terrain and returns an array of sizes of all its watersheds in descending order.

Examples:

```
1 2 3 4 1
```

This terrain contains two watersheds of size 3 and 2. The larger watershed consists of the three leftmost locations. The smaller watershed consists of the two rightmost ones. Surface water at elevation 4 flows to the east because elevation 1 is lower than elevation 3. The method should return array `[3, 2]`.

```
1 2 3 4 3
```

This terrain contains two watersheds of size 4 and 2. The larger watershed now includes the location at elevation 4, where the surface water can flow either to the east or to the west. The method should return array `[4, 2]`.

```
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
```

In this flat terrain, surface water does not flow anywhere, so we have a contiguous region of 25 locations that water collects. Hence, the size of this watershed is 25.

```
1 1 1 1 1
1 1 1 1 1
1 1 9 1 1
1 1 1 1 1
1 1 1 1 1
```

In this terrain, surface water at elevation 9 can flow to any of its four neighbors, but the water still ends up in the same watershed. Hence, the method should return array `[25]`.

```
3 2 2 2 5
1 4 2 5 1
1 1 5 1 1
1 5 1 4 1
5 1 1 1 3
```

Returns `[9, 9, 9, 4]`.

```
4 3 8 3 4
9 5 1 5 9
2 7 6 7 2
9 5 1 5 9
4 3 8 3 4
```

Returns `[5, 5, 4, 4, 2, 2, 2, 2]`.

HARMA: ♧ ♧ ♧ ♧ ♧

[1] Definition taken from Wikipedia. For your reference, Ithaca belongs to the Great Lakes watershed, and is approximately 11 miles from the St. Lawrence Continental Divide.

6 Submission

You should compress exactly these files into a `zip` file that you will then submit on CMS:

- `README.txt`: This file should contain your name, your NetID, all known issues you have with your submitted code, and the names of anyone you have discussed the homework with.

- `p1q1_solution.pdf`
- `p1q2_solution.pdf`

- `part2/Q3.java`
- `part2/Q4.java`
- `part2/Q5.java`
- `part2/Q6.java`
- `part2/Q7.java`

- `p3q8_solution.pdf`
- `part3/Q9.java`
- `part3/Q10.java`

If you do attempt a **HARMA** question, also include your implementation in the `zip` file. The filenames should indicate clearly which questions you attempted.

Do not include any files ending in `.class`.

All `.java` files should compile and conform to the prototypes we gave you. We write our own classes that use your classes' public methods to test your code.