CS 2112 A1
Introduction to Java and OOP

January 30, 2014

Instructions

This assignment is intended to be an introduction to the Java language and basic object-oriented programming concepts. It should help you become familiar with Java syntax and some important language constructs.

Grading

Solutions will be graded on both correctness and style. A correct program compiles without errors or warnings, and behaves according the 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.

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.
There are 10 questions grouped into 3 parts. Stub files are provided on CMS for each one. Questions may ask you to have a written solution answering questions about the code, modify the code, or fill in empty functions. All sections in the code that need to be filled out are marked with a `TODO` tag.

Updates

- (Jan 30) Template code has been updated with more helpful specs in code.
- (Jan 27) Part 1, question 2 has been updated with slightly different code.

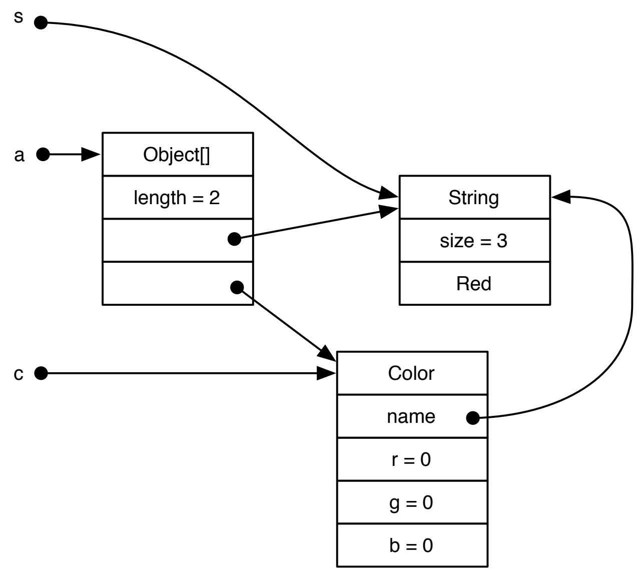
1 Semantics and object diagrams

Semantics is the study of meaning. Programming language semantics is about understanding what programs mean. A useful way to understand what Java programs do is to draw object diagrams in which objects are represented by boxes and references between objects are represented represented as arrows. Every object, including arrays, is tagged with its run-time class.
For example, consider the following code that might be found in some method:

```
class Color {
   String name;
   int r = 0, g = 0, b = 0;
}

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

At the end of executing the code, we end up with the following object diagram in the style we will use in this class. Notice that arrays (and strings) keep track of how many elements (and characters) they contain.



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

1 Suppose the line reading "`a[1] = c;`" were changed to read "`a[1] = a;`". Draw the resulting object diagram. Submit your answer as a PDF file (scans are accepted) named `p1q1_solution.pdf` in the root directory.

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

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

Now draw the resulting object diagram. Submit it as `p1q2_solution.pdf` in the root directory.

2 Coding & Syntax

Implement the following short programs by modifying the given stub files. No points will be deducted for inefficient code (within reason).
No library imports may be used other than the ones already provided. 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 arguments and the type of the return value must remain unchanged. You may add new methods to the classes, however.

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

2 Implement a method `flatten(String... list)` that returns a String of all the items of `list` concatenated together. EX. `flatten("abc","de","fg")` returns `"abcdefg"`. (Par: 5 lines)

3 Implement a method `copyNTimes(String s, int n)` that returns the empty string concatenated with the String `s` a total of `n` times. (Par: 5 lines)

4 Implement a program that reads in a file from the file system and outputs its lines in reverse order. (Par: 20 lines)

5 Write function `findUnionIntersection(int[] a, int[] b)` that takes two integer arrays and returns a new object containing both the union of the arrays and their intersection. (Par: 25 lines)

3 Data Structures

1 Run the code in `part3.Q1`. Draw an object diagram showing arrays `a` and `b` before and after the `for` loop executes. Submit your answer as a pdf (scans are accepted) named `p3q1_solution.pdf` in the root directory.

2 Run the code in `part3.Q2`. The function `isPalindrome(String s)` fails some tests; make corrections so that it meets the specification and passes all tests.

3 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 2 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.Q3` to convert the arguments provided to the program on the command line into a linked list that is printed out.

Karma

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

1 Make function `getConsecutiveSums(int n)` that returns an array of all arrays of consecutive positive numbers that sum to `n`. Because all numbers are consecutive, only the first and last are needed to be given (to represent the sum of `[1,2,3,4,5]` only `[1,5]` is needed.

Karma: ●●○○○

2 Write function `solvePath(bool[][] maze, Point start, Point end)` that returns true if a path exists between the start and end points, and false otherwise. Note that `Point` is a class containing two fields `x` and `y`. A path is defined by traveling between the four neighbors. True (denoted as 1) allows for a path and False (denoted as 0) marks a wall.

```
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
```

Karma: ● ● ● ○ ○

3 Make function `getWatershedSizes(int[][] topo)` where `topo` is a topological (height) map of a piece of terrain. The function returns a list of sizes of the all watersheds in reverse sorted order.

A watershed is an area of land that groups where rainfall collects. Any drop of rain that falls into a given watershed will end up at the same place. Rain always flows to the lowest of the nearest four neighbors. In the event of a tie, it can flow to any of the lowest, and that square is considered to be a part of multiple watersheds.

EXAMPLES:

```
 1  2  3  4  1
returns [3,2]
```

```
 1  2  3  4  3
returns [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
returns [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]
```

Karma: ꩜ ꩜ ꩜ ꩜ ꩜

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/Q1.java`
`part2/Q2.java`
`part2/Q3.java`
`part2/Q4.java`
`part2/Q5.java`

`p3q1_solution.pdf`
`part3/Q2.java`
`part3/Q3.java`

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.