CS 2112 Fall 2020
Assignment 1
Introduction to Java
Due: Tuesday, Sept 15, 11:59PM

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

Updates

- Object diagram fixed (9/8)

1   Instructions

Read all these instructions carefully and make sure you understand them before you begin. If there is anything you are not sure about, read carefully and think about the handout, and if necessary, ask for a clarification on Piazza. If there is any real ambiguity in the assignment, you are free to identify that ambiguity and resolve it in a reasonable way as long as you can justify it.

This first assignment is very explicit about what you need to do. In the future, more of the design will be left up to you. Nevertheless, you should get an early start, because there will be one-time startup costs, such as getting the JDK and Eclipse installed on your machine, figuring out how to navigate the Java class library, and learning where to go for help. Get started early to make sure you have plenty of time to surmount unexpected difficulties that arise.

1.1   Grading

Solutions will be graded on both correctness and style. A correct program compiles without errors or warnings and behaves according to 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. Use brief but mnemonic variable names. Spacing and indentation should be consistent. Your code should include comments as necessary to explain the **programmer's intent** without belaboring the obvious.

You should follow the common Java conventions regarding naming and code structure. The Java style guide is a useful reference.

1.2 Partners

You must work alone for this assignment. The course staff are happy to help with any difficulties that might arise. Use Piazza for questions and attend office hours for help.

1.3 Assignment structure

This assignment consists of eight written questions and six coding questions. Materials can be found in the archive A1release.zip on CMS. Download and extract the contents. The materials for the written questions (§2) are in the folders written and polynomial, and those for the coding questions (§3) are in coding.

2 Written Questions – Semantics and Object Diagrams

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

    Programming language semantics is about the meaning of programs. In object-oriented languages like Java, the meaning of programs depends heavily on the meaning of **objects**. An **object** is a collection of related data. A useful way to understand objects is to draw **object diagrams**. In an object diagram, each object is represented by a box tagged with its run-time class and associated data.

    The data associated with an object is stored in its **fields**, also called **instance variables** because a distinct variable exists for each object instance. Every field has a **name**, which is like an ordinary variable name, and a **type**, which determines what kind of data it represents. The type of a field can be either a **reference type** if it references another object that is created by the program or a **primitive type** such as int or boolean, for which the possible values in some sense exist even before the program starts running. A field that references another object is represented in the object diagram by an arrow from the field to the object, whereas field of primitive type is conventionally represented by writing the value of the field directly into the field.

    Figure 1 shows the 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  }
```

The object has four fields. The field name is of type String, which is a reference type. The other three fields, r, g, and b, are ints a primitive type that can represent any integer between 2^{-31} and $2^{31} - 1$. There are eight primitive types: boolean, byte, char, short, int, long, float, and double. All other types are reference types.

    The initial value of name is null, meaning the field does not have an assigned value yet. In the object diagram, this is represented by the "ground" symbol $\doteq$ shown in Figure 1. (We could also represent it by writing name=null in the field.)
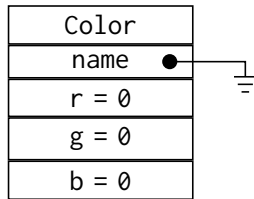
Figure 1: An object diagram for a newly created object of type `Color`
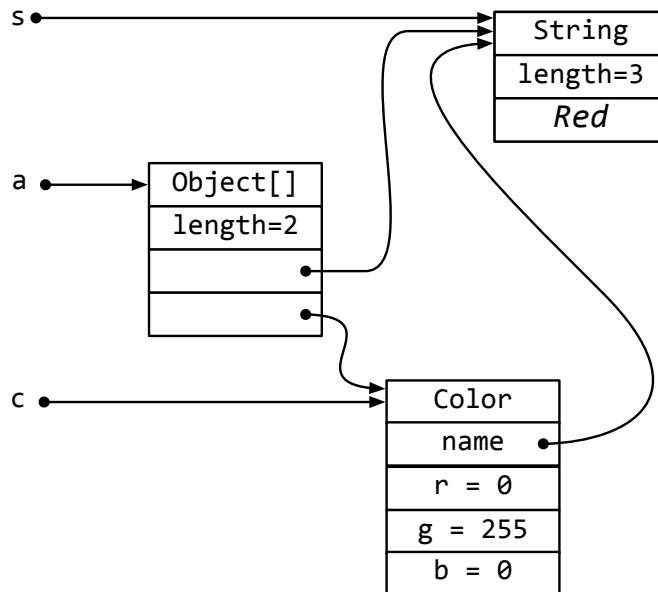

Figure 2: An object diagram after executing the code snippet

Now suppose we execute the following code snippet:

```
1  String s = "Red";
2  Color c = new Color();
3  c.name = s;
4  c.g = 255;
5  Object[] a = new Object[2];
6  a[0] = s;
7  a[1] = c;
```

The resulting object diagram is shown in Figure 2. The object that the variable a points to is an array object. Array objects have an instance field `length` to keep track of how many elements they contain. The drawing of the `String` is a white lie because it shows the characters of the string inside the string object itself. The actual internal representation of `String` objects is not accessible and has changed from one release of Java to the next. In the current Java release, `String` objects reference a char array containing their characters, but this is not important and we do not expect you to show this.

For the following problems, you are welcome to experiment by running the code pro-

vided to understand what it does. All the code for these questions can be found in the `written` folder.

1. Suppose we change line 5 in the code snippet above from `a[0] = s;` to `a[0] = a;`. Draw the resulting object diagram.

   The code for this question can be found in the file `Q1.java`. Submit your answer as a `.pdf` file `Q1.pdf`. Scans of handwritten diagrams are acceptable.

2. Suppose that we do **not** make the previous change, but instead add the following lines at the end of the code snippet:

```
7  Object[] b = a;
8  b[1] = new Color[2];
9  b = (Color[])a[1];
10 b[0] = c;
11 b[1] = b[0];
```

   Draw the resulting object diagram.

   The code for this question can be found in the file `Q2.java`. Submit your answer as a `.pdf` file `Q2.pdf`. Scans of handwritten diagrams are acceptable.

3. Run the code in `Q3.java`. Draw an object diagram showing the values of the field `total` and the arrays a and b before and after the `for` loop executes.

   The code for this question can be found in the file `Q3.java`. Submit your answer as a `.pdf` file `Q3.pdf`. Scans of handwritten diagrams are acceptable.

4. In the handout on Java I/O, there is a piece of code in the examples of §5.4 and §5.5 that asks whether the user wants to overwrite an existing file.

```
1  if (outFile.exists()) {
2    System.out.print("Output file exists; overwrite [yes/no]? ");
3    if (!sysin.nextLine().equals("yes")) return;
4  }
```

   The test in line 3 returns from the method immediately without performing the write if the user does **not** respond with yes. You might think that the following alternative test would be just as good:

```
3    if (sysin.nextLine().equals("no")) return;
```

   However, there is a subtle but important reason why the former test is preferable. Can you say what it is? Submit your answer in a text file `Q4.txt`.

   In the released code (in the `polynomial` directory) you will find files `Polynomial.java` and `Main.java` that respectively implement a polynomial abstraction and lightly test it. We want you to understand this code and make some improvements. The next four questions are about this code.

5. Draw object diagrams showing the final state of the objects referenced by variables p, q, and z in Main.main(). Submit your answer as a .pdf file Q5.pdf. Scans of handwritten diagrams are acceptable.

6. Explain briefly why the implementation of degree() uses Math.max(), and why the implementation of create() doesn't simply assign result.coefficients = coeffs. Submit your answer in a text file Q6.txt.

7. The correctness of the implementation of one of the Polynomial methods relies on the caller satisfying a currently unspecified precondition (other than the implicit precondition that arguments are non-null unless otherwise allowed). Identify this method, show an example of a call that would break the implementation, and state an appropriate precondition to prevent such calls. Submit your answer in a text file Q7.txt.

8. The correctness of multiple Polynomial methods relies on its representation satisfying a class invariant. Give this class invariant. Submit your answer in a text file Q8.txt.

3   Coding Questions

Implement the following short programs by modifying the given stub files (find these in the coding directory). 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.

9. 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".

10. Write a program that reads in a file from the file system containing a string of text on each line. Then, read in an integer n from the console (i.e., entered by the user from keyboard), skip past n lines in the file, and print the remaining lines **in reverse order** to the console.

    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.

11. 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.

12. Implement method `cntUniquePts(int[][] points)` that returns the number of unique ordered pairs in the input array. If there are five copies of the same ordered pair in `points[][]`, then only one of them should be counted towards the total.

    Example: The follow function call should return 3:

    ```
    1  cntUniquePts(new int[][] { {1, 2}, {1, 4}, {1, 2}, {2, 4}, {1, 2} });
    ```

    The three unique points are (1, 2), (1, 4), and (2, 4).

13. Run the code in `coding.Q13`. The method `isPalindrome(String s)` fails some tests. Make corrections so that it meets the specification and passes all tests.

14. 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 `coding.Q14` to convert the arguments provided to the program on the command line into a linked list that is printed out.

4 Submission

Some of the problems ask you to turn in diagrams. You may turn in scans of hand-written pages. However, high-resolution scans of paper may result in files that are too large to turn in. The problem with scanning paper is that the scan preserves a huge amount of subtle and unnecessary detail about the grain of the paper. By adjusting the contrast and brightness of the image to wash out that detail, you can usually substantially reduce the amount of space needed. It may also helpful to reduce image resolution as long as it does not impair readability. Various tools may be used for these transformations, such as Preview (Mac), GIMP (Linux, Mac), and Photos (Windows). To decrease the file sizes, please also run your files through an online PDF compressor. Some good ones are PDF Compressor and Small PDF. Then, compress exactly these files into a `zip` file and submit the `zip` file to CMS. Do not include any other files. In particular, do not include any `.class` files.[1] Make sure you submit your solution code, not our release code.

- `README.txt`: This file should contain your name, your Cornell NetId, all known issues with your submitted code, and the names of anyone else you have discussed the homework with (excluding course staff).
- `written/Q1.pdf`
- `written/Q2.pdf`
- `written/Q3.pdf`
- `written/Q4.txt`
- `written/Q5.pdf`

---

[1] Note that on many systems, files beginning with . are not visible. Search the web for "Show hidden files" to find out how to view them on your platform.

- `written/Q6.txt`
- `written/Q7.txt`
- `written/Q8.txt`
- `coding/Q9.java`
- `coding/Q10.java`
- `coding/Q11.java`
- `coding/Q12.java`
- `coding/Q13.java`
- `coding/Q14.java`

Note: The / is the file separator (\ on Windows). It is not a character in the file name.

All `.java` files should compile and conform to the prototypes we gave you. In particular, do not change the package structure or any method or field declarations. This is important for compatibility with our testing software. You may add your own private methods and fields if you wish.

To reiterate the important points:

- Make sure your `README.txt` file has all the requested information.
- Make sure your code compiles.
- Do not change the package structure or any method or field declarations.
- Do not include any `.class` files or any other files in your submission except those listed above.

Violation of any of these will result in a point deduction, so check carefully before submitting.