CS 2112 Fall 2024
Assignment 1
Introduction to Java
Due: Thursday, Sept 5, 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

- 8/30/2024 - Remove Q14 from list of submitted files
- 8/31/2024 - Clarify Figure 2

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. If necessary, ask for a clarification on Ed. If there is any genuine ambiguity in the assignment, you are free to resolve it in a reasonable way, provided you clearly identify the ambiguity and can justify your approach.

This first CS 2112 assignment is very explicit about what you need to do. In the future, much 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 IntelliJ installed on your machine, figuring out how to navigate the Java class library, and learning where to go for help. Getting started early ensures you have plenty of time to surmount unexpected difficulties that arise.

1.1 Grading

Submissions will be graded for 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. Use consistent spacing and indentation. Include comments as necessary to explain the **programmer's intent**, but don't belabor the obvious.

In particular, you should follow common Java conventions regarding naming and code structure. In CS 2112, we use: Kernighan & Ritchie style (Java variant). The 2110 Java style guide is a useful reference, as is the Google style guide, though we do not follow all the prescriptions of either guide, such as mandatory braces. IntelliJ's default code style scheme is a good starting place, and we prefer spaces to tabs for indentations.

1.2 Partners

You must work alone for this assignment. The course staff are happy to help with any difficulties that might arise. Use Ed for questions and don't be shy about coming to office hours if you need help.

1.3 Generative AI

Generative AI, such as ChatGPT, is not permitted in any capacity for this assignment. Basic code completion in IntelliJ is permitted. Consult with the course staff if you have specific questions.

1.4 Assignment structure

This assignment consists of eight written questions and six coding questions. Materials can be found in the archive `A1release.zip` on CMSX. 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, as well as operations on that 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 variable has a **name**, which is like an ordinary variable name, and a **type**, which determines what kind of data it can refer to. 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 variable that references another object is represented in the object diagram by an arrow from the variable to the object, whereas of primitive type is conventionally represented by writing the value of the variable directly into the box representing the variable.

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 }
```

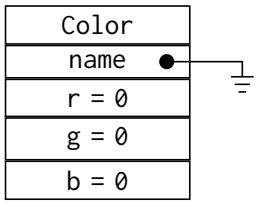

Figure 1: An object diagram for a newly created object of type `Color`

The object has four instance variables: `name` is of type `String`, which is a reference type. The other three fields, `r`, `g`, and `b`, are of type `int`, 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.

Unlike local variables, instance variables are assigned a default value when an object is created. The default value of value of all reference types is `null`, meaning the field does not have an assigned value yet. In the object diagram, this is represented by the "ground" symbol ⏚ shown in Figure 1. (We could equally well represent it by writing `name=null` in the field.)

Now suppose we execute the following code snippet:

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

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.

For the following problems, you are welcome to experiment by running the code provided to understand what it does. All the code for these questions can be found in the `written` folder.

1. Suppose we change line 7 in the code snippet above from `a[1] = s;` to `a[1] = 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[0] = new Color[2];
```

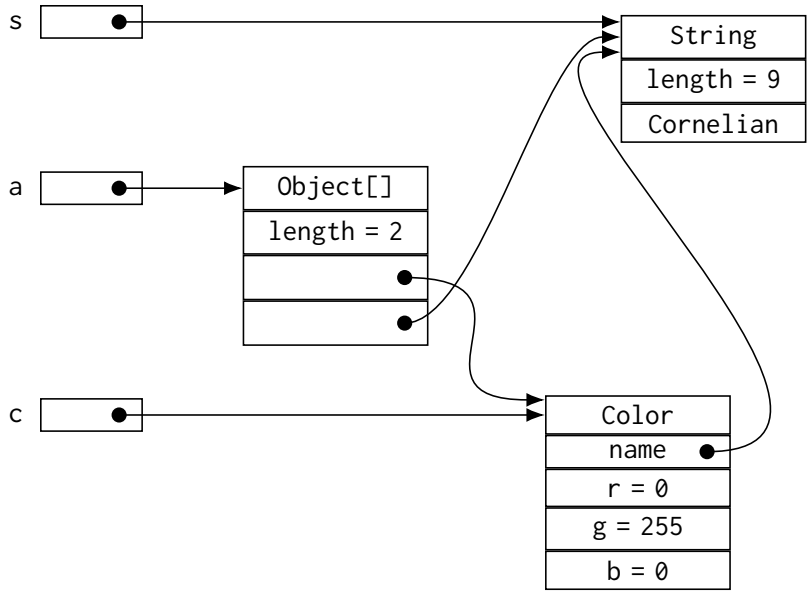

Figure 2: An object diagram after executing the code snippet

```
9  b = (Color[])a[0];
10 b[1] = c;
11 b[0] = b[1];
```

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`, `z`, `f`, and `g` in `Main.main()`. Submit your answer as a PDF file `Q5.pdf`. Scans of handwritten diagrams are acceptable.

6. 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 `Q6.txt`.

7. 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 `Q7.txt`. Hint: imagine that there is an adversary who can put whatever you want into the fields of the object. What invariant would prevent this adversary from causing any of the methods to work incorrectly?

8. In the `Polynomial` method `scaleBy`, suppose we delete the statement on line 110: `if (c == 0) return theZero`. Does this change affect the correctness of the method `scaleBy`? Explain why or why not in a one or two sentences. 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 writting inefficient code, provided it isn't excessively inefficient.

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, however, add new methods to the classes.

9. Implement method `filter` from `coding.Q9` according to its specification. Be careful to return an array of the correct length. Below is an example usage case.

Example: `filter({"Hello", "and", "Goodbye", "world!"}, {'e', 'f', 'g', '!'})` returns `{"Hello", "world!"}`.

10. Write a program in `coding.Q10` 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 the first `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 re-enter a value until they enter an integer.

11. Implement method `findSymDifference(int[] a1, int[] a2)` in `coding.Q11` that takes two integer arrays and returns a new array containing their symmetric difference. The *symmetric difference* of `a1` and `a2` consists of those elements which belong to exactly **one** of `a1` and `a2`. That is, any elements that belong to both arrays should be excluded from the result. The resulting array does not need to be in any particular order.

12. Implement method `findMaxZeros(int[][] points)` in `coding.Q12`, which returns the array from `points` that contains the most zeros. If there is a tie for the number of zeros, the array appearing earlier in `points` should be returned.

Example: The following function call should return $\{0, 4, 0\}$:

```
1 findMaxOnes(new int[][] { {0, 2}, {0, 4, 0}, {0, 0}, {2, 4}, {3, 0} });
```

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

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.

A hint about scanning: often a scan will preserve 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 CMSX. Do not include any other files. In particular, do not include any `.class` files. Also, pay attention to hidden files, as some operating systems and zip utilities will include additional files without your knowledge. Search the web for "Show hidden files" to find out how to view them on your platform. You may want to consider using `zip` on the command line to ensure your submission includes only the files you want. Please 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`
- `written/Q6.txt`
- `written/Q7.txt`
- `written/Q8.txt`
- `coding/Q9.java`
- `coding/Q10.java`
- `coding/Q11.java`
- `coding/Q12.java`
- `coding/Q13.java`

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

All Java files must compile and should 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. Make sure to check for hidden and platform dependent files!

Violation of any of these will result in a point deduction, so check carefully before submitting. It's also a good idea to download your submission from CMSX after you have uploaded it to verify that it has the exact files you want graded.