CS 2112 Fall 2019
Assignment 1
Introduction to Java

Due: Tuesday, Sept 10, 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

- Added some suggestions about how to avoid submitting large image files and to reflect the correct CMS link. (9/1)
- This document was updated to mention how to turn in Q5–Q8.
- The code for `Polynomial.java` was updated to fix the implementation of the `evaluate()` method. (8/31)

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. You are encouraged to download and install the Eclipse style template that the course staff use.

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 four written questions and a programming assignment. Materials can be found in the archive `A1release.zip` on CMS. Download and extract the contents. You should find a file `Hospital.jar` and two folders `written` and `hospital` with materials for the written questions (§2) and programming assignment (§3), respectively.

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 are called its **instance fields**. 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 or a **primitive type** such as `int` or `boolean`. A field that references another object is represented in the object diagram by an arrow from the field to the object.

Figure 1 shows the object diagram for a newly created instance of the following class:

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

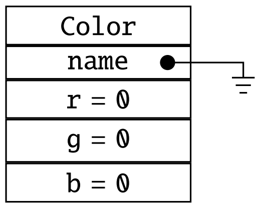


Figure 1: An object diagram for a newly created object of type `Color`

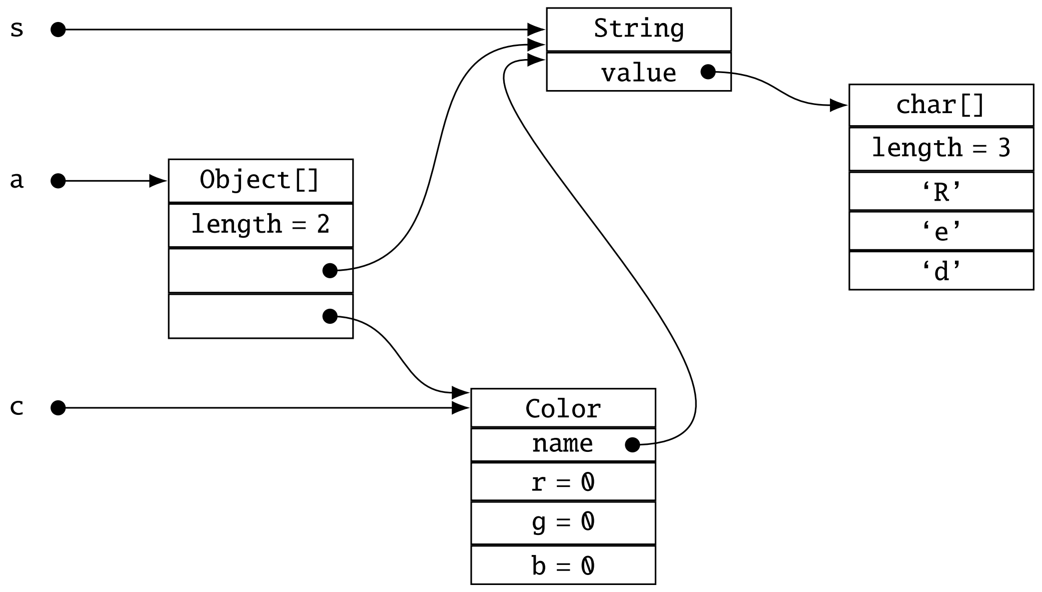


Figure 2: An object diagram after executing the code snippet

The object has four instance fields. The field `name` is of type `String`, which is a reference type. The other three fields `r`, `g`, `b` are `int`s (integers), a primitive type. 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 symbol ⏚ shown in Figure 1.

Now suppose we execute the following code snippet:

```
String s = "Red";
Color c = new Color();
c.name = s;
Object[] a = new Object[2];
a[0] = s;
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. `String` objects use a `char` array named `value` to record the characters in the string.

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 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:

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

```java
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:

```java
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 (inthe `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()`.

6. Explain briefly why the implementation of `degree()` uses `Math.max()`, and why the implementation of `create()` doesn't simply assign `result.coefficients = coeffs`.

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.

8. The correctness of multiple `Polynomial` methods relies on its representation satisfying a class invariant. Give this class invariant.

3 Programming Assignment – Krzmrgystan General Hospital

There has been an outbreak of a deadly form of influenza in Krzmrgystan. The hospital is overrun with patients needing medical attention. There is an effective treatment for this devastating illness, but unfortunately, there is only a limited supply of medicine available. Moreover, the patients are deteriorating rapidly, so time is of the essence. You must save as many of them as you can with the limited time and resources.

The hospital is circular in shape, with the rooms laid out in a ring. You, the physician administering the treatment, must go from room to room. For every unit of medicine you give the patient, their condition improves markedly. But it takes time to administer the treatment, and it takes time to move from room to room.

The program `Hospital.jar` contains a simple game based on this scenario. Open a command window, navigate to the directory containing the program, and type `java -jar Hospital.jar`. Type `h` at the prompt for instructions. Play with it a bit to get a feel for the game.

The `hospital` folder contains the skeleton of the source code for this game with several missing parts, which you will have to supply. There are four files: `Doctor.java`, `Patient.java`, `Room.java`, and `Hospital.java`, each containing a single public class of the same name. We describe the contents of each of these below.

Create a project with a package named `hospital` containing the four files. In Eclipse, it should look similar to the following in the package explorer window.

```
A1
   src
      hospital
         Doctor.java
         Hospital.java
         Patient.java
         Room.java
```

Important: Do not change the package structure or the declarations of any methods. This includes the name, number and type of parameters, return type, declared exceptions, or access modifiers. This restriction is to maintain compatibility with our testing software.

The sections in the code that need to be filled in are marked with a `TODO` tag.

3.1 The `Doctor` class

During gameplay, there is just one instance of this class, representing the doctor who moves around and treats patients. There are two instance fields: `int medicine`, representing the amount of medicine remaining; and `Room location`, representing the current location of the doctor.

This class is very simple and needs only a few lines filled in. The method `useMedicine` decrements the `medicine` field by `DOSAGE`, provided there is at least that much medicine left. The method `medicineLeft` should return `true` if there are at least `DOSAGE` units of `medicine` left. Your code should not depend on the assumption that the value of `DOSAGE` is 1, because we may change it when testing your code.

3.2 The `Patient` class

During gameplay, there is one instance of this class for each patient. There are several constants (`INITIAL_HEALTH`, `CURED`, `DEAD`, `TREATED_GAIN`, and `UNTREATED_LOSS`), which determine the parameters of the game. You should not change these values in your code.

Each `Patient` has a `name`, a `health`, and an `age` that are initialized when the `Patient` object is created. You need to supply this initialization code. Initialize `age` to a random integer between 10 and 79, inclusive. The initialization of the other fields should be obvious.

We have provided a `getName` method, which constructs a random Krzmrgystani name. It uses two submethods `consonant` and `vowel` that select a randomly chosen consonant and vowel from the arrays `CONSONANT` and `VOWEL`, respectively. You need to supply this code. These are easy one-liners, but make sure you do not bake in any assumptions about the length or contents of the arrays. It should be possible to change the `CONSONANT` and `VOWEL` arrays later without changing your code.

There are three other methods for which you have to supply code. The `boolean`-valued method `treatable` should return `true` if the patient is neither dead nor cured. The method `treat` treats the patient. If the patient is treatable (that is, if the patient is neither dead nor cured), the patient's health should be incremented by `TREATED_GAIN`, up to a maximum of `CURED`. Similarly, the method `untreated` causes deterioration of health. If the patient is treatable, the patient's health should be decremented by `UNTREATED_LOSS`, down to a minimum of `DEAD`.

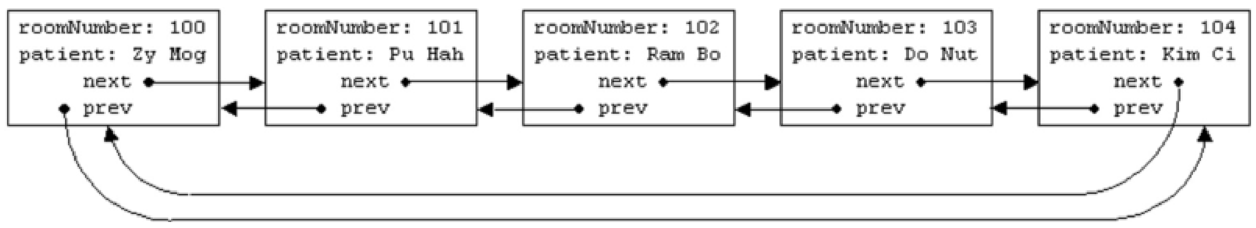
Keep your code general and use the named constants.

3.3 The `Room` class

There is one instance of this class for each room in the hospital. You need to supply all the code for this class. Each `Room` instance has four fields: an integer `roomNumber`, two `Room` fields `prev` and `next`, and a `Patient patient`.

The room numbers are assigned sequentially, starting at 100, when a `Room` instance is created. The best way to do this is to use a private static field `nextRoomNumber`, which is incremented after each access.

The fields `next` and `prev` are references to the `Room` objects with the next higher and next lower room number, respectively, except that the `next` field of the highest-numbered room points to the lowest-numbered room and the `prev` field of the lowest-numbered room points to the highest-numbered room. That is, the rooms are arranged in a circular doubly-linked list. If there were five rooms, the structure would look like this:



We can move the doctor to an adjacent room by assigning to `doctor.location` the value of `doctor.location.next` to move forward or `doctor.location.prev` to move backward.

The `Room` constructor should assign a room number and populate the room with a new `Patient`.

The class should also provide a static method `createRooms` to create the circular doubly-linked list of rooms and return a reference to the lowest-numbered room. It should start by initializing `nextRoomNumber` to 100. The integer constant `ROOMS` indicates the number of rooms to create. Create the first instance and remember it in a local variable; this will be returned at the end. For each instance created after the first, link it to the previous one as shown in the diagram by setting the appropriate `next` and `prev` fields. When you are done, link up the last and first instances in the same way.

Your code should work for any positive value of `ROOMS`.

3.4 The `Hospital` class

This is the main driver of the program. It contains the `main` method, a `play` method to play the game, and various other command handlers and utility methods. We have provided the methods `displayHelp` and `displayStatus`, but you have to supply everything else.

The `main` method should just construct a new instance of `Hospital` and call its `play` method. The constructor of `Hospital` should initialize the game by creating the rooms and the doctor and setting the `firstRoom` field to the lowest-numbered room as returned by the `createRooms` method. This should be the doctor's initial location. The `play` method starts by displaying a welcome message, which we have provided, and creates an instance of `Scanner` for reading from the keyboard using `new Scanner(System.in)`. It then enters a read-eval-print loop, which repeatedly

- prints out the current status of the game by calling `displayStatus`,
- prompts for the user's input,
- reads a sequence of single-character commands from the keyboard (use the `nextLine` method of `Scanner`), and
- processes the command sequence by calling `processCommand` with the user's input string,

continuing until a termination condition is met, as determined by a `boolean`-valued method `done`. Note that the read-eval-print loop itself is fairly simple, calling helper methods to do all the hard work, including checking the termination condition. See the handout on Java I/O for examples of reading from the command line and a read-eval-print loop.

After the read-eval-print loop exits, `play` should print a final status and report the number of patients cured and the amount of medicine remaining.

The `boolean`-valued method `done` should return `true` if there is not enough medicine left for a treatment or if no patient is treatable (that is, all patients are either dead or recovered).

The `processCommand` method takes a `String` parameter `cmd` that represents a sequence of single-character commands. It contains another loop that processes each command individually in order from left to right. You can use `cmd.charAt(i)` to extract the `i`th character, or convert to a character array. Based on the character, it calls one of the following methods:

`treat`	treat the patient in the room that the doctor is currently visiting
`move(direction)`	move to an adjacent room
`displayHelp`	display the help screen
`quit`	exit the program

If the character is not a valid command, print a message and continue to the next character. The termination condition should be checked by calling `done` with each command and quitting the loop if so.

The `treat` method should call the `treat` method of the patient occupying the room the doctor is currently visiting, which will cause that patient's health to improve. For all other patients, it should call the `untreated` method so that their health deteriorates.

The `move` method should take a parameter indicating which direction to move and should move the doctor in that direction. In addition, it should make all the patients' health deteriorate by calling `untreated` on each patient.

In some of these methods, you (not the doctor!) must walk around the ring of rooms and do something in each room. The `cured` method is an example.

Submit your solution in the four files `Doctor.java`, `Patient.java`, `Room.java`, and `Hospital.java` in a folder `hospital`.

Do not change the name, number and types of parameters, return type, declared exceptions, or access modifiers of any provided method or field. However, you are free to use any other methods, classes, or data structures if you find them helpful.

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). If you wish to participate in the Krzmrgystan Hospital contest (see §5), also include your submission here.
- `written/Q1.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/Q2.pdf`
- `written/Q3.pdf`
- `written/Q4.txt`
- `written/Q5.pdf`
- `written/Q6.txt`
- `written/Q7.txt`
- `written/Q8.txt`
- `hospital/Doctor.java`
- `hospital/Hospital.java`
- `hospital/Patient.java`
- `hospital/Room.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.

5 Krzmrgystan Hospital Contest

In the Krzmrgystan Hospital game, it is possible to save all the patients. Just for fun, send us your best result in the form of a string of commands (`t`, `n`, `p`) in the `README.txt` file. A prize will be awarded for the most patients cured. Ties will be broken by the amount of medicine left over. Further ties will be broken by the time of the earliest final submission to CMS.