CS211 Spring 2007
Assignment 1 — Basic Java and Object-Oriented Programming
Due 7 February 2007, 11:59:59pm

This assignment will help you refresh your knowledge of Java and the Java API. The material is roughly equivalent to a comprehensive CS100 assignment. This assignment will also help you practice writing complete applications, including command-line arguments. For help with using Java's command-line environment, refer to the Java Bootcamp notes on the course website and/or the Java summary in the textbook.

In this first assignment, especially in §5, we have been rather explicit in the design of the programs to help you get started. We have tried to use good object-oriented design principles so that you have a model of what we will expect from you on future assignments. In the future we will leave more of the design aspects to you.

Good luck with the assignment. We hope you have fun.

0 General Instructions

Here are some general instructions that apply to this assignment and future ones. We will not necessarily repeat Section 0 with every assignment.

0.1 Grading Criteria

Solutions will be graded on both correctness and style. For correctness, at the very least your program must compile without errors or warnings. For style, we like concise, clear, easy-to-read code. Use mnemonic variable names. Use proper indentation. Comment where necessary for clarification, but do not overcomment. Be concise; a two-page solution to a problem that can be done in a few lines will lose points, even if correct.

Occasionally, bonus points will be awarded for implementing optional features. Bonus points do not count in your score, but are counted separately.

0.2 Partners

In subsequent assignments, you will be allowed to work with a partner, but for this first assignment you must work alone. This is because we need to assess your basic Java and programming skills.

0.3 Academic Integrity

You must abide by the Code of Academic Integrity at all times. There is a general Code for all of Cornell University, a code specific to Computer Science, and a code specific to this course. Links to all of these are provided on the course website under Course Info. You must be familiar with all of them. If there is any doubt, *ask*.

0.4 Submission

Follow the submission format requirements as given on the course website. They will be enforced strictly. The last section of the assignment (here, §6) typically summarizes the files that you need to submit to CMS.

0.5 Time Estimates

We will generally provide rough estimates of the time it should take you to do these exercises to help you with your time management. Of course, these estimates are only approximate; your mileage may vary, depending on your level of preparation. For this first assignment, note that we have not included startup time for getting an IDE installed and running and for learning how to use it, for getting the right version of Java installed on your machine, and for recalling all those little details about Java that you have forgotten since CS100. So please do not wait until the eleventh hour to get started.

1 Communication

Purpose: To get connected
Estimated time: 10 minutes
Points: 1

Review the information about Usenet on the course website. In your favorite newsreader, subscribe to `cornell.class.cs211` and `cornell.class.cs211.talk`. Post a message in `cornell.class.cs211.talk`. In the subject line, enter only `A1: <your Cornell NetId>`. In the message say hi to everyone.

The newsgroup `cornell.class.cs211.talk` is meant for test posts and casual conversation on topics related to 211. The newsgroup `cornell.class.cs211` is the actual course newsgroup that we reserve for technical questions regarding course material.

2 Course Website and Policies

Purpose: To gain familiarity with the course website that we have so painstakingly assembled for your benefit
Estimated time: 1 hour
Points: 9

Review the entire course website to answer the following questions.

1. How long do think it will take to do this assignment? How long did it actually take?
2. What is ASCII text?
3. What are the dates and times of the CS211 prelims?
4. Suppose a student writes some of the solutions for a particular assignment with a partner. Then, the two students "divorce" because of irreconcilable differences and wish to work alone or perhaps with other partners for the remainder of the current assignment. According to this course's rules of academic integrity, may they "divorce"? May they divorce for the remaining assignments?
5. Suppose that you work with a new partner for an assignment because your previous partner wasn't keeping up with you. About two days before the deadline, your previous partner pleads with you to show your solution, though he/she swears they won't copy it. After all, he/she didn't know you worked with someone else and you've been partners all semester. What should you do according to the rules of academic integrity?
6. What is the procedure to submit a regrade for an assignment?
7. What is the procedure to submit a regrade for a prelim?
8. Where is the CS211 consulting office?
9. According to the course grading policy, if you receive an average score of 72% for assignments 1-4, 81% on assignment 5, 65% for prelim 1, 58% for prelim 2, a 74% on the final exam, do all the quizzes, and complete the course evaluation, what is the lowest guaranteed letter grade that you will receive based on the current grading weights?

Submit your answers in a plain text file `Exercise2.txt`. If we cannot open the file with a text editor, you will receive no credit. If you do not know what plain text is, refer to the submission format requirements on the website.

3 The End of Time

Purpose: Data representation, two's complement arithmetic, exploring the Java API
Estimated time: 30 minutes
Points: 10

In Java, time is reckoned in milliseconds (1/1000 of a second) since 00:00:00, January 1, 1970, Greenwich Mean Time. The count is represented as a `long` value. This is the value that is returned on a call to `System.currentTimeMillis()`. Write a Java program that calculates the date and time, to the nearest second, when time runs out. Your program should print out the answer as a date and time in Eastern Standard Time. Any reasonable output format is ok.

Hint: There is a one-line solution. It may be helpful to know something about how `int` and `long` values are represented and the `Date`, `Calendar`, and `Long` classes in the Java API.

Submit your solution in the file `EndOfTime.java`.

4 Turning Back the Clock

Purpose: Command-line input, `String` and `Character` classes, working with the Java API, exceptions and error handling
Estimated time: 8 hours
Points: 25

Suppose you lost lots of points on your last homework assignment for late submission. You really had it done in time, but you just forgot to submit it. Unfortunately, somehow the time of last modification on your file indicates that it has been modified since the deadline. To convince your professor that you really had it done in time, you would like to reset the time of last modification on the file. Write a Java program `SetDate` to do this.

Your program should accept a `String` of the form MM/DD/YYYY representing the new date, a `String` of the form HH:MM:SS representing 24-hour time in Eastern Standard Time, and a `String` representing the name of a file in the current directory. For example, 1/27/2007 13:05:55 represents 1:05:55pm on January 27, 2007. These values will be entered on the command line when the user runs the program. For example, the user might type

```
java -cp . SetDate 1/27/2007 13:05:55 myfile.java
```

in a directory containing `SetDate.class` and `myfile.java`, the file whose date should be changed. (You don't need the `-cp .` if you have your `CLASSPATH` set. See the Java Bootcamp notes on the webpage for instructions.)

Your program should consist of a single public class `SetDate` with a method

```
public static void setDate(String date, String time, String fileName)
   throws IllegalArgumentException {
      //your code here
}
```

as well as a `main` method. The `main` method should accept three command-line arguments—recall that they are provided in the `String[]` array parameter—and call the `setDate` method with them. It should fail gracefully with an informative error message if there are fewer than three arguments. If there are more than three, it should print a warning and continue, ignoring the extras.

The call to `setDate` in the `main` method should be in the `try` block of a `try-catch` statement to catch any `IllegalArgumentException` thrown by `setDate`. See the text p. 907 for the exact syntax.

The `setDate` method should extract integers representing the month, day, year, hour, minute, and second from the given date and time strings, create a `Date` object from them, then change the date and time of last

modification of the specified file to the specified date and time. The indexOf, lastIndexOf, and substring methods of the String class will be useful here. You can convert a String representation of an integer to an int using Integer.parseInt. All these things are well documented in the Java API.

Your setDate method should enforce proper formatting on the date and time strings. It should also check whether the date and time strings represent a valid date and time and whether the file exists. In case of error, it should not print out an error message itself, but should throw an IllegalArgumentException, and that is the only exception that it should ever throw. In particular, converting a String to an int using Integer.parseInt can throw a NumberFormatException, so you should put these calls in a try-catch statement that catches the NumberFormatException and throws an IllegalArgumentException instead. We will check your program automatically on lots of inputs, both legal and illegal, so you better had too.

When you throw an IllegalArgumentException in setDate, you can supply an informative error message; for example,

```
throw new IllegalArgumentException("File does not exist: " + fileName);
```

The catch block in the main method that catches the exception can then print out the message as shown on p. 907.

To check whether the given date and time are valid, you will need to check whether the values are out of bounds. For example, 13/32/2007 25:61:61 is clearly invalid. But how about 2/29/5000 11:59:59? This depends on whether 5000 is a leap year (it turns out that it is not). However, you do not have to do this calculation yourself; the getTime method of the Calendar class will do it for you, provided you have previously called setLenient(false) on the Calendar object. Whatever you do, be sure to test your code thoroughly on known valid and invalid inputs to make sure errors are being caught.

The File, Date, and Calendar classes will be useful in this exercise. The name of the current directory can be obtained by a call to System.getProperty("user.dir").

Submit a file SetDate.java with a single public class SetDate that implements this program.

5 Krzmrgystan General Hospital

Purpose: Classes and objects, functional abstraction, static and instance fields, linked lists
Estimated time: 15 hours
Points: 50

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. Download it, open a command window, navigate to the directory containing Hospital.jar, 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 file A1.zip contains the skeleton of the source code for this game with several missing parts, which you will have to supply. Extract the contents. You will get 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.

5.1 The Doctor class

During gameplay, there is just one instance of this class, representing the doctor who goes 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 we have given it to you in its entirety. The method useMedicine decrements the medicine field by DOSAGE, provided there is at least that much medicine left. The method medicineLeft 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 in the future.

5.2 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 that determine the parameters of the game. You should not change these values in the code you submit.

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 in a constructor. 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.

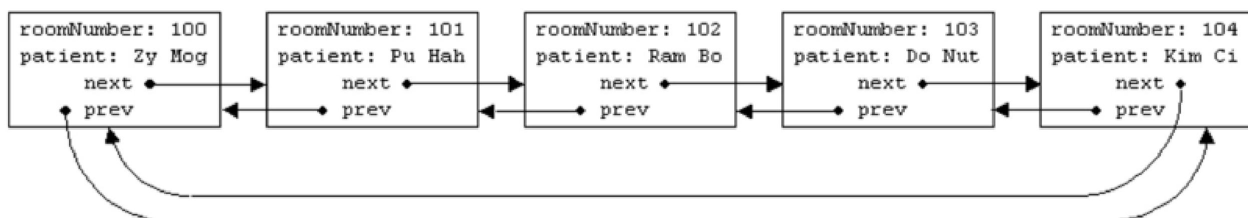
Keep your code general and use the named constants.

5.3 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:



Thus we can move the doctor to an adjacent room by assigning doctor.location = doctor.location.next or doctor.location = doctor.location.prev.

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. 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.

5.4 The `Hospital` class

This is the main driver of the program. It contains the `main` method, a constructor, 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 create the rooms and the doctor and set 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. It 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.

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. We have provided the `cured` method as an example of how to do this.

You are free to use any other private methods, classes, or data structures if you find them helpful.

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

6 Submitting Your Work

Points: 5

Submit the following files in CMS:

- Exercise 2: `Exercise2.txt`
- Exercise 3: `EndOfTime.java`
- Exercise 4: `SetDate.java`
- Exercise 5: `Doctor.java`, `Patient.java`, `Room.java`, `Hospital.java`

Do not submit any other files. In particular, do not submit any `.class` files. As a reminder, refer to the submission format requirements on the course website before submitting any work. Yes, it's long, but we do not want you to lose points for easily preventable mistakes, like forgetting headers, submitting the wrong files, or forgetting to submit files.