CMPT 270 – 2017-2018 Term 01

Assignment 2:

Getting Started with Java

Submission Deadline:
Part A: 29 September 2017 at 11:55pm – No late submissions accepted for Part A
Part B: 6 October 2017 at 11:55pm – Submit via Moodle!

Description

The objective of this assignment is to get familiar with the basics of the Java language, and the Java system. To do this, you will write the classes to model three entities of part of a hospital application, and test them.

In particular, you must create the three classes: a `Person` class, a `BasicDoctor` class, and a `Ward` class. Each class should have a constructor, and the fields and methods specified below. For each of the classes, all the fields should be declared as private. Be sure to follow the specification given below, since the classes will be used in subsequent assignments in order to build a simple hospital management system. In addition, each class is to have a static main method that should be used to test the class following the principles of regression testing as discussed in class.

You are not expected to develop a driver class to run the application (this will be part of assignment #3).

Deliverables

***** Upload .java files, .class files and .txt files for documentation (no Word .docs) *****

Part A:

The `Person` class is the simplest class, and it does not dependent upon the others. Therefore, it will be done first for Part A of the assignment. We will get it marked as quickly as possible, so that you can correct any faults in it and eliminate any similar faults in the other classes before handling them all in Part B.

For Part A, upload a .java file of your `Person` class and the output obtained from the execution of your main method. If your output is a blank page, then be sure to clearly state that the output is a blank page. External documentation is not required for Part A, but proper internal documentation is required (including proper javadoc comments).

Part B:

For Part B, upload .java and .class files for all three classes, with the `Person` class revised in any ways that seem appropriate after its initial version has been marked, and it has been integrated with the other classes. Each class should be well written as described above. Also, you are expected to do a good job of testing all routines, including any additional tests for the `Person` class that you think should be added.

We may run your classes to verify that they run as you state. You should also include .txt files for:

(i) the external documentation for the assignment

(ii) a listing of all the classes,

(iii) a listing of the output from running your classes

The markers will annotate the .java files to provide written feedback on your work.

Hospital Management Application

Each of the classes is now discussed in more detail:

`Person` class:

This class models a person. For our purposes, a person will have a name and a health number, where the name might change but the health number is not allowed to change. Thus the following features are required:

- a constructor with parameters for the person's name and health number
- a String field to store the person's name
- an int field to store the person's health number
- an accessor to return the person's name
- an accessor to return the person's health number
- a mutator to change the person's name
- the toString() method to return a string representation of all the information about the person in a form suitable for printing
- a main method to test the above features

`BasicDoctor` class:

This class is the representation for a doctor. While it is anticipated that a doctor will have a number of attributes that could be stored, we will keep this basic version very simple by only having a name (that is not to be changed). The features for a doctor are:

- a constructor with the name for the doctor as a parameter
- a String field to store the name
- an accessor method to access the name
- the toString() method to return a string representation of the information related to the doctor
- a main method to test the above features

`Ward` class:

This class represents a ward in a hospital. The ward class should store the people that are in the beds of the ward. Obviously, a container is needed to store these people. To keep it simple, we will assume that each bed of the ward has a unique integer label, and that the integers are consecutive indices. In order to be able to access the person in a specific bed, an array is the appropriate data structure to use to implement the container of people in the beds of the ward. One complication is that a hospital will have many wards, and each bed in the hospital will have a unique integer. Hence, the integer labels for a ward will not likely start at 0, while in Java the indices of an array must start at 0. The features are:

- a String field to store the name of the ward
- an int field to store the integer label of the first bed
- a field for the array to store the people in the beds of the ward

- a constructor with parameters for the name, the first integer label (say int min) for a bed in the ward, and the last integer label (say int max) for a bed in the ward; note that from the external perspective (everywhere other than this class), the beds are labeled min, min+1, min+2, ..., max
- accessor to access the name of the ward
- accessor to access the smallest integer for a bed label
- accessor to access the largest integer for a bed label
- a function to convert a valid external bed label to the corresponding array index
- a function to convert a valid array index to the corresponding external bed label
- an accessor to test whether there is a person in the bed with a specified integer label
- an accessor to obtain the person in the bed with a specified integer label
- a mutator to assign a person to a specific bed, where the bed is specified by its integer label
- the toString() method to return a string representation (suitable for printing) of the information related to the ward; this includes:

  1) the name of the ward

  2) for each bed:

     - if the bed is empty: the integer label for the bed
     - if the bed is non-empty: the integer label for the bed and the name of the person in the bed
- a main method to test the above features

Additional Guidelines

Bed labels

As further information about the integer labels of the beds, suppose that the first bed has label 200 and the last one has label 250. Recall that in Java, the indices of an array always start at 0. Thus, the array indices will be 0, 1, 2, ..., 50. In addition, integer bed label 200 will need to map to array index 0, 201 map to 1, etc. All classes outside the `Ward` class will specify a bed by its integer label (say 200 ... 250), while the Ward class must also deal with array indices. All the complications in dealing with array indices should be within the `Ward` class. Hence, if later a more complex scheme is used to label beds, all the changes to convert the new labels to array indices will be in the `Ward` class.

Narrow scope of this assignment

The above classes do not do much – they are simply entity classes for use in a larger system. A hospital application would have a whole other part (perhaps quite large)

- to obtain information and requests from users,
- to determine what operations are to be done,
- to carry out those operations by invoking methods of the entity classes,
- to return appropriate information back to the user.

Later assignments will fill in these parts. You do not need them for this assignment.

Internal documentation

Proper internal documentation includes:

- A comment just before the class header that gives an overview of the class. For example, if the class models an entity, the comment might state what entity the class models and specify the key features. Alternatively, if the class serves as a container, state what type of items the container stores, and how these items are accessed. Inside the body of the class, a comment might state what type of container it is (stack, queue, list, keyed or non-keyed dictionary, etc), and what representation is used to store the items. If the class is an interface, state for what it provides an interface and is there anything special about the interface. Finally, if it has a control function, what is it doing and controlling? Recall that comments for a class appear before the class and begin with /**

- A comment just before each field stating what is stored in the field, again beginning with /**

- A comment, beginning with /**, just before each constructor and each method stating what it does. Note that the comment only gives what the routine does, not how it does it. If it isn't obvious from the code how it accomplishes its goal, comments on how it is done belong in the body of the routine.

- Each class should be easily read. This includes good use of white space, especially reasonable indentation. The code should be easy to read in printed form as well as on a screen. Hence, be sure that you do not have any lines cutoff and few lines (if any) wrapped in a printout. It is suggested that you use an indent of 3 or 4 spaces, and tabs are printed as this same number of spaces. On Windows machines in the lab, there is a utility called PrintFile, with a shortcut on the desktop, to nicely print listings, including the conversion of tabs to a specified number of spaces. On Linux machines in the lab, the similar utility is called NicePrint, and it is a script used on the command line. If you use tabs for indentation, you are encouraged to use these utilities to print your classes. Note that you should not use a small font to print your classes, as the code must be easy to read.

- Local variables that are simply used as temporary variables, and parameters that are just assigned to fields can be short, but other variables should have good descriptive names. The name should take into account of its location. For example, in the class `Person` it does not make sense to call a field personsName, since the field is in the `Person` class so the name would obvious be the person's name. Hence, simply call the field name. Be sure that if a variable is used as a local variable, it is declared as a local variable, not as a field. The only data stored in fields should be data that is intrinsically part of the object that needs to be stored from one use of the object (method invocation) until the next use.

- Although it isn't relevant for this assignment, no routine should be very long. An exception is a test routine. Test routines are often simply a long sequence of simple tests, so they might be longer.

- Every class should have a main routine to test the class. There should not be input statements. Instead, all data should be hard coded, so that you know and can check the results. Be sure to invoke every method to ensure that it is behaving correctly. When testing one class, assume that other classes that it uses are correct. Therefore, you can concentrate on testing one class at a time. The testing should have the properties of regression testing as discussed in class. In particular, the tests for a class should be in its main routine, and the output should mostly only report errors. If there is any non-error output, the output should be checkable by reading the output without referring to the test routine. This is best done by having the output should state what is being output. For example,

  The printout for Pete with number 123456 is

  ```
  Name:  Pete Health number:  123456
  ```

External documentation

For external documentation, include the following:

- A description of how to execute your tests of the classes. What is necessary to compile your classes? What program or programs need to be run to show the execution of your program? For example, it might be necessary to compile and execute all three classes individually. On the other hand, you might have written a driver program (another class with a static main method that invokes the main methods of the three required classes; the static main method of class A can be invoked by A.main(null);). In the latter situation, we need to know the name of the driver class. This description should be very short.

- The status of your assignment. What is working and what is not working? What is tested and what is not tested? If it is only partially working, the previous point should have described how to run that part or parts that work. For the part or parts not working, describe how close they are to be working. For example, some of the alternatives for how close to working are (i) nothing done; (ii) designed but no code; (iii) designed and part of the code; (iv) designed and all the code but anticipate many faults; or (v) designed and all the code but with a few faults; (vi) working perfectly and thoroughly tested.

- Maintenance manual. This is information for the person or persons who must keep your system running, fix any faults, and do any upgrades. The reader of the maintenance manual is expected to have the same background and experience as yourself, but of course not having developed your system. This part of the documentation will vary from assignment to assignment. For this assignment, it is sufficient to include a UML class diagram showing all the features of each class, and the relationships (inheritance, uses and aggregation) amongst the classes.

- UML diagrams should be generated with a program like UMLet or an online tool, such as `https://www.gliffy.com/uses/uml-software/`. Make sure that you use the correct arrows and arrowheads. Incorrect arrowheads will lose several marks. Part A does not require any external documentation, while the external documentation for part B should include all the parts.

Marking (total 40)

Part A.1 (3) `Person` class

Part B.2 (3) `BasicDoctor` class

Part B.3 (11) `Ward` class

Part B.4 (8) Testing code, proper regression testing

Part B.4 (3) Correctly functioning of all classes

Part B.5 (7) Clarity and quality of code style, internal documentation, submission organization.

Part B.6 (5) Clarity and quality of external documentation.

This is an individual assignment. You are encouraged to discuss the general concepts of classes, types, containers, etc. with you classmates, but the specific details of the Hospital Management System in this assignment should be done completely individually. Students that copy / share work will be penalized.