SUNY, Korea Computer Science

CSE 114 Handout 7: Problem Set 7

Due: Nov 12, 2019
Points: 30

Read This Right Away

This problem set is due at **11:59 pm on Nov 12, 2019, KST**. Don't go by the due date that you see on Blackboard because it is in EST. Go by the one given in this handout.

• To solve each problem below, you will be implementing a Java class.
• Please read carefully and follow the directions exactly for each problem. Files and classes should be named exactly as directed in the problem (including capitalization!) as this will help with grading.
• You should create your programs using emacs.
• Your programs should be formatted in a way that is readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.
• This problem set assumes you have installed Java and emacs on your computer. Please see the course web for installation instructions.
• You must use the command-line interface to run your programs. That is, you must use the `javac` and `java` commands to do this problem set. Do not use Eclipse yet.
• **Important!** You must use only 1 Scanner object. Assignments may require a Scanner object to be passed into other methods but do not create a new Scanner inside those methods! Use the one passed to the method. ***Creating multiple scanners causes a problem with the automated grading procedure and doing so will result in a loss of points on the assignment!***
• **Be sure to include** a comment at the top of each file submitted that gives **your name** and **email address**.
• **Remember:** We are now also assessing coding style. Be sure to use consistent indentation, good and informative variable names and write comments in your code to describe what it is doing.

What Java Features to Use

For this assignment, you are *not* allowed to use more advanced features in Java than what we have studied in class so far.

Problem 1 (30 Points)

The goal of this problem is to have you familiarize yourself with the mechanics of creating a class and some objects using the class. Then, do some more interesting things with the objects that you create. This is more of a tutorial than a problem. Follow along as I guide you through the process.

We are designing a message building system for a remote weather station. The weather station periodically reads data from various weather sensors. It then generates a formatted text message with the data and transmits the data over a dedicated communication link to a collection station in a large data center. We will first design a class that will hold 1 set of readings and be used to generate the data message to be transmitted. Later, we will extend the class to allow it to report hardware anomalies and failures to the data center.

1. Name the class `DataMessage`
2. The class will hold the following data:
   - There will be a String that holds a *name* or identifier for the particular remote station (i.e., Manitoba_1_7). This will be used to create a 'source' address for the transmission later on.
   - There will be a *date* that is represented as a string (e.g., "Fri, Nov 8, 2019 at 1128 PM").
   - There will be a fields for sensor readings as follows:
     - *airTemp* to hold a double value that is the latest air temperature
     - *groundTemp* to hold a double value that is the latest ground temperature
     - *windSpeed* to hold a double value with the current windspeed in km/hour.
     - *humidity* to hold an integer value between 0 and 100 indicating the percentage humidity.
   - There will be a Boolean flag called *urgent*, which is true if the message is notifying the center of a possible hardware failure.

Note that each attribute will be represented as a private field in your class. Do not use `public` fields in the classes that you design. Use `private` fields and provide getters and setters so that accesses to the fields from outside the class can only be made through getters and setters. More on this next.

3. For each field in your class, add a method that can be used to read the value of the field. That is, add a *getter* method for each field if it makes sense to have a getter.
4. For each field in your class, add a method that can be used to change the current value of the field. That is, add a *setter* method. Given a field of a class, a value can be assigned through one of several different ways: (i) It may be assigned by passing a value to a constructor as an argument at the time an object is created. This is how you initialize a field. (ii) It may be assigned by calling a setter after an object has already been created. Or, (iii) it can be assigned with a value generated inside a constructor at the time the object is being created rather than receiving one through a parameter. It all depends on what sort of field we are dealing with. So, think about each field and decide which way would make sense for each. For now you will want to use one of the first two ways I described above.
5. Since your getters and setters are added to be used by other classes outside the `DataMessage` class, they should be declared as `public`. Since the fields will be accessed (for read and write accesses) via getters and setters respectively, there is no reason to set the fields themselves to be `public`. In fact, the fields *must* be declared to be `private` so that the methods within `DataMessage` can access them directly, but no other methods in any other class in your system can access them directly without going through the getters and setters. This is how you control the visibility of the state information in an object. The values of all the fields collectively represent the state of the object at any point in time. Since the values can be changed using the setters, the state information can change. So, the state information in an object is time-dependent.
6. The fields, getters, and setters must be declared to be non-static, i.e., do not add the `static` keyword in their declarations. Since we are planning on creating many message objects based on the definition of this `DataMessage` class (the *blueprint*), they should be declared to be non-static. In fact, so far in the class `DataMessage`, *nothing* should be declared as `static`.
7. Introduce a constructor that does not take any argument. In this case, the fields should be initialized with some reasonable default values within the constructor: for the `name` and `date` fields, use an empty String (""). For *airTemp*, *groundTemp*, *windSpeed* and *humidity* use 0.0 or 0 as appropriate for a default value. Finally, for *urgent*, use *false*.

8. Let us also add another constructor that takes six arguments: one for each of the six fields. You know what to do in the body of this constructor, right? Note: This constructor only needs six parameters, not seven since you should still set *urgent* to *false.* That member will only be turned on later by a check routine you will write in the next section.

Now that we have a class representing data messages, let us build a class that we can use to test the implementation of the `DataMessage` class. We will call the new class `UseDataMessage` in a file named `UseDataMessage.java`. (We could have chosen the name of this class to be anything we wanted, but `UseDataMessage` would be a logical choice since we will be using that class to test the implementation of the `DataMessage` class.) `UseDataMessage` class includes only one method (a `static`) method in it named `main` that does the following in the given order:

1. Create one object (instance) of the `DataMessage` class and name it `msg1`. Be sure to use the names exactly as I specify them. Think about what the type of this variable `msg1` should be. This message object should be created with the constructor that does not take any argument.
2. Now print the value of each field in `msg1` to the standard output device (i.e., screen) using the values returned by the getters. As you print out the values for the fields, properly annotate the values being printed out so they will be meaningful. Use something that indicates what physical thing (air temp, etc) the value represents. NOT the data type!
3. This time change the value of each field in `msg1` using the setters if available. You may use any reasonable values for the fields as you modify them.
4. Now print the value of each field in `msg1` again using the getters. Be sure that your getters are now seeing the new values now that we have changed them using the setters. Again as you print out the values for the fields, properly annotate the values being printed out with the physical property the value represents.
5. Create another object of type `DataMessage` and name it `msg2` this time. This data message object should be created with the constructor that takes six arguments this time. You can use any reasonable values as arguments in the call to the constructor.
6. Now, print the value of each field in `msg2` using the getters. Be sure that your getters are seeing the correct values. Again, as you print out the values for the fields, properly annotate the values being printed out.

Now, run the `main` in `UseDataMessage`.

Assuming that everything works as expected so far, let us add some more as follows in the given order:

1. Add a String field to the class called *notes.* Update both of the constructors to set this field to an empty string (“”). Note: do NOT add new parameters to the constructors since you are hardcoding the empty string when setting the value of *notes*.
2. Add a public non-static method called *serialize*() which takes no arguments and returns a String. The String return value should contain a ‘compact’ form of all the data in the object. The compact form should include all the weather measurements with preceding labels. Use ‘:’ to separate the labels from the values and use ‘;’ to separate the name/value pairs. For instance, with air temperature of 20.5, ground temperature of 22.2, wind speed of 25, and humidity of 15 and nothing in notes, the output of serialize() should be:
airTemp:20.5;groundTemp:22.2;windspeed:25.0;humidity:15;notes:””
3. Add a public non-static method called *checkHardware*(). This method does not return a value! Since we do not have real hardware, you will simulate this by using a random object to determine if there is a failure and, if so, on which sensor the failure occurred. Create a random object. Use random.nextInt(10). If the return value is greater than or equal to 6, then there is a hardware problem. If it is less than 6, the hardware is fine and you need do nothing further. If there is a hardware problem, do the following:
    a. Use random again (random.nextInt(4)). Use the return value plus 1 to indicate the sensor number of the bad sensor (1=airTemp, 2=groundTemp, 3=windspeed, 4=humidity). [Do you see why you need to use the returned random number + 1? (returned random value will be 0-3)]
    b. Set the *urgent* member to *true.*
    c. Set the *notes* field to “**!!! Hw failure: sensor: <sensor name>**” (without the quotes ””) translating the number returned by the second random call into the matching string (see ‘a’ above) to replace **<sensor name>** in the string).

4. Add a public, non-static method called *transmit*. Again, we do not have a 'communication' line to a data collection server, so rather than sending the data over a network, we will just print it on the console. Do the following:
    a. Print the line: **Source: <name>**
       Where **<name>** is replaced by the *name* field in the object
    b. Print the line: **Date: <datestamp>**
       Where **<datestamp>** is the *date* field from the object.
    c. Call *serialize()* and print a line: **Data: <serializeOutput>**
       Where **<serializeOutput>** is the string returned from *serialize()*

5. Add to the class `DataMessage` a non-static method named `toString` that takes no parameter and returns a string representation of the object. For the exact signature of the method, refer to the documentation for the `toString` method in `java.lang.Object` class. The `toString` that you added can simply use the *serialize()* method and return the String it generates.
6. Add a piece of code in the `main` of `UseDataMessage` that prints the state information of `msg1` using the `toString` method that you just added to `DataMessage`.
7. Add a piece of code in the `main` of `UseDataMessage` that creates an array of data messages of length 5. Name that array `dataMessages`. Add `msg1` as the first element of the array and `msg2` as the second element of the array. Create three more `DataMessage` objects of your choice and add them to the array too.
8. In the `main`, write a `for` loop that loops through the message objects in the `dataMessages` array and calls *checkHardware()* for each one.
9. In the `main,` add a `for` loop that loops through the data message objects in the `dataMessages` array and prints each message in the array using the *transmit()* method that you added to `DataMessage`.
10. Add another `for` loop that loops through the data message objects in the `dataMessages` array once more and prints only *urgent* messages in the array using *transmit()* this time.

Hand in both `DataMessage.java` and `UseDataMessage.java`

Submission Instructions

Please follow this procedure for submission:

1. Place the deliverable files (`DataMessage.java` and `UseDataMessage.java`) into a folder by themselves. The folder's name should be CSE114_PS7_<yourname>_<yourid>. So if your name is Joe Cool and your id is 12345678, the folder should be named 'CSE114_PS7_JoeCool_12345678.

2. Compress the folder and submit the zip file.
    a. On windows, do this by pressing the right-mouse button while hovering over the folder. Select 'Send to -> Compressed (zipped) folder'. The compressed folder will have the same name with a .zip extension. You will upload that file to the Blackboard dropbox for PS7.
    b. On mac, move the mouse over the folder then right-click (or for single button mouse, use Control-click) and select **Compress**. There should now be a file with the same name and a .zip extension. You will upload that file to the Blackboard dropbox for PS7.
3. Navigate to the course blackboard site. Click **Assignments** in the left column menu. Click **PS7** in the content area. Under **ASSIGNMENT SUBMISSION**, click **Browse My Computer** next to **Attach Files.** Find the zip file and click it to upload it to the web site.

4. ***Important!!*** Click the **Submit** button! If you fail to click **Submit**, the assignment will not be downloadable and will not be graded.