

Homework 3 - Plan Composition & Object Equality

Due Tuesday by 11:59pm **Points** None

Available Nov 9 at 11am - Nov 19 at 11:59pm 11 days

Before you begin, please make sure you are familiar with the guidelines discussed on the [Expectations on Homework](#) page, the "Homework Assignments" section of the [CS 2102 FAQ Page](#), and the assignment FAQ at the bottom of this page.

Assignment Goals

- To think about object equality and how we can control it
- To be able to articulate the different subtasks for a programming problem
- To practice different ways of organizing problem solutions


Reminders

- **Please use the default package in your Java project.** There will be a penalty for using a named package.
 - Please include a Javadoc statement above each method and class that YOU write. (You do not need to comment anything we give you that you do not modify.) There will be a penalty for forgetting your Javadoc statements.
-

Part I: Overriding equals()

Cable providers maintain lists of TV shows and their airdates. A cable provider wants to organize its schedule into lists of daytime, primetime, and late night shows. Daytime shows have a start time at or after 6:00 am and before 5:00 pm. Primetime shows have a start time at or after 5:00 pm and before 10:00 pm. Late night shows have a start time at or after 10:00 pm and before 1:00 am. All other non-special shows are overnight and are not of interest to the cable provider at the moment. Specials are one-time shows that can be on any time of day.

A show's broadcast time is expressed in military (24-hour) time. For instance, if a show starts at 7:00 pm, then the value of broadcastTime will be 1900.

In the [set of starter classes](#)  (https://canvas.wpi.edu/courses/27993/files/4168988/download?download_frd=1) we give you, there is a `Show` class which represents a show and a `ShowSummary` class that will hold the organized lists of shows. Later (in Problem 1 of Part II), we'll write a method to generate this report, but to adequately test this method, we need to know what it means for two `ShowSummary` instances to be equal.

Assume that we have two `ShowSummary` instances. These two instances are equal if all of the following are true:

- Each pair of corresponding lists (ex. `daytime` in the first instance and `daytime` in the second instance) have the same size.
- Each pair of corresponding lists have the same shows in the same order.
- A show in one list is considered the same as a show in the other list if *only* the title and broadcast time of each show matches. It is NOT necessary that the two shows are the same object in memory or that the other fields of the two shows match.

In `ShowSummary`, override the `equals()` method to enforce this definition of equality.

In the `ShowExamples` class are four test cases that start `instructorTestShowSummary_*`. Right now, three of those test cases pass, and one fails. Once you have written the correct `equals()` method, all four tests should pass. You should develop additional test cases to further ensure that your new `equals()` method is working the way it should.

Part II: Plan Composition

For each of the following problems, write **two** solutions, where each solution solves the problem using a different approach. Approaches count as different if they cluster at least some subtasks of the problems differently (like we saw for the Rainfall solutions in lecture); mere syntactic differences, such as replacing an element-based for-loop with an index-based one, or moving code into a helper without changing the order of the underlying computation, don't count as different.

For each problem, you will be asked to identify (in a comment) the (sub)tasks that make up that problem. For Rainfall, we'd be looking for a sequence of phrases such as "summing elements, counting elements, ignoring negative elements, stopping at the -999". This wording doesn't need to be exact (it'll be graded by humans) - they just need to convey the core computational tasks within the problem.

Please use the set of starter classes from Part I. The starter files contain all the classes needed for this assignment, as well as the classes in which to put your answers (to aid us in grading). You may add whatever methods you wish to these classes, but please don't rename any classes, fields, or methods. The section below on "Using the Starter Files" explains the starter files and where you should edit them with your solutions.

Problem 1: ShowSummary Equality

Continuing from Part I, design a program called `organizeShows` that consumes a list of Shows and produces a report containing all of the daytime, primetime, and late night shows in the list that are not specials. You may assume that no two shows have the same name. Use the `ShowSummary` class in the starter files for the report.

The starter files provide a concrete test case for this method.

In a comment at the bottom of the `ShowExamples.java` file, identify the subtasks for Problem 1. Your comment will be graded.

NOTE: Please do not modify the Show file or further modify the ShowSummary file for this problem. All of your functionality should be in ShowManager1 and ShowManager2, and all of your examples should be in ShowExamples.

Problem 2: Earthquake Monitoring

Geologists want to monitor a local mountain for potential earthquake activity. They have installed a sensor to track seismic (vibration of the earth) activity. The sensor sends measurements one at a time over the network to a computer at a research lab. The sensor inserts markers among the measurements to indicate the date of the measurement. The sequence of values coming from the sensor looks as follows:

```
20151004 200 150 175 20151005 0.002 0.03 20151007 130 0.54 20151101 78 ...
```

The 8-digit numbers are dates (in year-month-day format). Numbers between 0 and 500 are vibration frequencies (in Hz). Frequencies below 0 sometimes occur, but they are physically impossible and thus ignored as erroneous data (usually due to equipment fault). This example shows readings of 200, 150, and 175 on October 4th, 2015 and readings of 0.002 and 0.03 on October 5th, 2015. There are no data for October 6th (sometimes there are problems with the network, so data go missing). Assume that the data are in order by dates (so a later date never appears before an earlier one in the sequence) and that all data are from the same year. The dates will always be 8-digit numbers in the format show above (and starting with a non-0 digit).

You may also assume that every date is followed by at least one frequency (in other words, every date has at least one measurement). Furthermore, you may assume that no date will be followed by only negative data.

Design a program `dailyMaxForMonth` that consumes a list of sensor data (doubles) and a month (represented by a number between 1 and 12) and produces a list of reports (`MaxHzReport`) indicating the highest frequency reading for each day in that month. Only include entries for dates that are part of the data provided (so don't report anything for October 6th in the example shown). Ignore data for months other than the given one, and ignore negative frequency values. Each entry in your report should be an instance of the `MaxHzReport` class in the starter files.

For example, given the sequence of values above and the month 10 (for October), the resulting list should be:

```
[MaxHzReport(20151004, 200),  
  MaxHzReport(20151005, 0.03),  
  MaxHzReport(20151007, 130)]
```

In a comment at the bottom of the `EarthquakeExamples.java` file, identify the subtasks for Problem 2. Your comment will be graded.

NOTE: Please do not modify the `MazHzReport` class.

Using the Starter Files

[Homework 3 Grading Rubric](https://canvas.wpi.edu/courses/27993/files/4168989/download?download_frd=1)  (https://canvas.wpi.edu/courses/27993/files/4168989/download?download_frd=1)

The starter files zip has a lot of files. Here's a guide to what you will find in there.

For Part I, you will mostly just be interested in the `Show.java`, `ShowSummary.java`, and `ShowExamples.java` files. These should be self-explanatory.

For each problem in Part II, there are at least three types of files:

- Files named `<Problem>1.java` and `<Problem>2.java`. Put one of your solutions in each of these files. The headers are already there, but the methods currently return null so that things will compile. Replace the method bodies with your methods. The `Earthquake` files also provide two helper functions for breaking down dates.
- A file named `<Problem>Examples.java`. Your test cases and data for the problem go in here, as usual. We have provided one simple test for each problem, to make sure you are computing what we expect you to. You need to add your own tests atop these.
- Auxiliary files with the classes for reports, Shows, etc. You do not need to touch these.

Please post to the forum if you need help understanding the files. We set these up in advance to (a) help autograding, and (b) save you from having to do this setup yourselves.

Submission Guidelines

Edit the starter files, then zip them up and submit that zip to InstructAssist. The name of the project in InstructAssist is **Homework 3**. For Part II, there is a separate Examples class for each problem, which will assist us in autograding. Each test in your Examples classes should test only the single method for that problem. (`ShowExamples.java` should also contain any equality tests you wrote for Part I.)

Grading and Testing Expectations

For Part II, to aid us in grading, **submit tests written against the version 1 class (`ShowManager1`, `Earthquake1`)**. Follow the setup of tests shown in the starter files. If you want to test your version 2, simply change which version's class you create at the top of the Examples class. But please reset these to version 1 before submitting.

Your Examples classes do NOT need to have separate copies of the tests for both versions. We will assume you ran the same tests against both versions.

In grading this assignment, we will check for

- (Part I) Whether your equals() method passes all of our test and appears to be functionally correct
- Whether your methods produce the right answers
- (Part II) Whether your two solutions to the same problem differ in how they organize the computation
- Whether you are producing clean (well organized and documented) code
- Whether you have a good set of tests for these problems, where good means that your tests catch routine errors that a solution might make.
- (Part II) Whether you can accurately identify the (sub)tasks involved in each problem.

FAQs

Q: My `assertTrue()` statements for my `equals()` method work, but then my `assertEquals()` statements fail.

A: Check that the method signature EXACTLY matches the one for the default method signature in object, i.e.

```
public boolean equals(Object obj)
```

Everything has to be the same, including the data type of the parameter. You will then need to cast the data type to the object you want. For an example of this, please see the code from the days in class that we talked about overriding `equals()`.

Q: For Part II, do I need to include my subtasks for both versions of `organizeShows()` and `dailyMaxForMonth()`?

A: The list of subtasks for both versions of your method should be the same. It's only when you get to step 3 (the outline) of the planning process that you will see differences. Remember that the subtasks is essentially just a "to-do list" for your method, and you're not worried about implementation just yet. (This means that your subtasks should NOT be a line-by-line description of your code.)

Q: Do I need to include my input/output and outline?

A: No, but we STRONGLY encourage you to do them anyway for your own benefit. You should take some time to go through all three steps (for both Part I and Part II) to help you think through possible solutions before you start writing code.

Q: When tracing through my code, I notice that a line like `if(show.broadcastTime < 0100)` is evaluating to false when it should be evaluating to true (or vice versa).

A: Do not put leading zeros before any numbers in Java. This makes Java think the number is base 8 instead of base 10. Instead, just write the line like this:

```
if(show.broadcastTime < 100)
```

Q: I'm getting an out of bounds exception in my index-based for loop, which looks like this:

```
for(int i = 0; i < list.size(); i++) {  
    ...  
    i++  
    ...  
}
```

A: It's generally bad practice to increment the index counter inside the body of an index-based for loop, as it's easy to lose track of where you are in the list and end up skipping over items or going out of bounds.

This situation occurs most commonly in the Earthquake problem because it's tempting to "look ahead" at the readings in the list that follow a date. Instead, think about how to handle the data values in the list one at a time. You don't need to look ahead.

Q: I'm not sure if my two solutions are different enough.

A: Compare the outlines of your two programs. Are they nearly identical line for line? If so, they're probably too similar. If you want to consider a second solution that may better fit the assignment criteria, please see the next question.

Q: I'm having trouble coming up with a second solution.

A: Think about the cleaning and parsing approaches we looked at in class. How could you use one (or both) of them in your method?

Q: I'm not sure why I'm getting this error in my code.

A: You are welcome to use office hours/Slack/the IA forum for help, but before you do, please try using the debugging tools that I demonstrated in class. That video is posted to Canvas. In addition, here are some helpful tutorials:

[Debugging in Eclipse \(https://www.vogella.com/tutorials/EclipseDebugging/article.html\)](https://www.vogella.com/tutorials/EclipseDebugging/article.html)

[Debugging in Eclipse \(Video\) \(https://www.youtube.com/watch?v=jybuG6QrLQg\)](https://www.youtube.com/watch?v=jybuG6QrLQg)

[Debugging in IntelliJ \(https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html\)](https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html)

[Debugging in IntelliJ \(Video\) \(https://www.youtube.com/watch?v=IAWnIP1S6UA\)](https://www.youtube.com/watch?v=IAWnIP1S6UA)

Q: Why do I get errors when I try to remove an object from a LinkedList (using the `remove()` method)?

A: Are you trying to call `remove()` from within a loop? Removing items from a list (or adding items to a list) while you're iterating over it is generally a bad idea, as it can cause problems with your iteration. This most commonly occurs when cleaning data. Instead of removing the data you don't want, consider adding the data you want to keep to a new list. The Library and Rainfall code from class provides excellent examples of this.

