Homework 5: Fun with Scala
Due: 5:00 PM, Mar 16, 2018

Contents

1 Shopping Cart 2
2 Vote Counting 3
3 Earthquake 4
4 Arrays 5

Objectives

By the end of this homework, you will be able to:

- Use the functional aspects of Scala
- Compare functional vs imperative solutions to problems
- Pattern match in Scala

How to Hand In

For this (and all) homework assignments, you should hand in answers for all the non-practice questions. For this homework specifically, this entails answering the Vote Counting, Earthquake, and Shopping Cart questions.

In order to hand in your solutions to these problems, they must be stored in appropriately-named files with the appropriate package header in an appropriately-named directory. The source code files should comprise the hw05.src package, and your solution code files, the hw05.sol package.

Begin by running the command ‘cs018_hw05setup’ to create your sol folders in both the javaproject and scalaproject directories. There is no source code for this homework.

After completing this assignment, the following solution files should be in your ~/course/cs0180/workspace/scalaproject/sol/hw05/sol directory:

- Vote Counting
  - Vote.scala, containing class Vote.
  - VoteCounting.scala, containing class VoteCounting.
  - VoteCountingTest.scala, containing class VoteTest, which tests all methods you wrote for this problem.
- Earthquake
  - MaxHzReport.scala, containing class MaxHzReport.
  - Earthquake.scala, containing class Earthquake.
  - EarthquakeTest.java, containing class EarthquakeTest, which tests all methods you wrote for this problem.

- Shopping Cart
  - CartItem.scala, containing class CartItem.
  - Checkout.scala, containing class Checkout.
  - CheckoutTest.scala, containing class CheckoutTest, which tests all the methods you wrote for this problem.

The following solution files should be in your
~/course/cs0180/workspace/javaproject/sol/hw05/sol directory:

- Shopping Cart
  - CartItem.java, containing public class CartItem.
  - Checkout.java, containing public class Checkout.
  - CheckoutTest.java, containing public class CheckoutTest, which tests all the methods you wrote for this problem.

To hand in your files, navigate to the ~/course/cs0180/workspace/ directory, and run the command 'cs018_handin hw05'. This will automatically hand in all of the above files. Once you have handed in your homework, you should receive an email, more or less immediately, confirming that fact. If you don’t receive this email, try handing in again, or ask the TAs what went wrong.

**Please Note:** For this homework only, you are handing in from the *workspace* as opposed to the javaproject or scalaproject directory. Make sure your Java solutions are in the correct javaproject folder, and your Scala solutions are in the correct scalaproject folder.

**Problems**

1 **Shopping Cart**

An online hardware store applies discounts during checkout. A shopping cart is a list of the items being purchased. Each item has a name (a String like 100-count nails) and a price (a Double like 6.95), which is greater than 0. Design a class Checkout that has a method, totalCost, which consumes a shopping cart and produces the total cost of the cart after applying the following two discounts:

- If the cart contains at least $50 worth of nails, take 15% off the cost of JUST the nails (nails are any item with a name that is exactly “nails”)
• If the cart contains at least two smoke alarms, take $10 off the total of the cart (a smoke alarm is any item whose name is exactly “smoke alarm”). You can assume that the total cost of 2 smoke alarms will always be at least $10.

The input shopping cart is represented as an `ArrayList` (in Java) or `List` (in Scala) of objects of a class named `CartItem`. The constructor for `CartItem` takes a `String` name and a `Double` cost. For example, assuming we have the `CartItem` and `Checkout` classes defined:

```java
CartItem apple = new CartItem("apple", 0.75);
CartItem cape = new CartItem("cape", 50.01);
ArrayList<CartItem> cart = new ArrayList<CartItem>();
cart.add(apple);
cart.add(cape);
Checkout check = new Checkout();
check.totalCost(cart); // returns 50.76
```

**Task:** In Java, create the `CartItem` class, which takes in a `String` name and a `Double` cost in its constructor. You do not have to error check for invalid input.

**Task:** In Java, create the `Checkout` class, which has the `totalCost` method. This method should take in an `ArrayList` of `CartItem` objects, and return the total `Double` cost of all the objects, applying the 2 discounts above as necessary. You can assume that an empty cart (i.e., an empty `ArrayList`) has cost 0.0.

**Task:** Test your `Checkout` class exhaustively in `CheckoutTest.java`.

**Task:** In Scala, create the `CartItem` class, which has the same specifications as before.

**Task:** In Scala, create the `Checkout` class, which has the same specifications as before (replacing the `ArrayList` with `List`).

**Note:** You MUST use the functional aspects of Scala to get full credit for this problem. This means that you should strictly *never* mutate a variable. We advise that you try to use higher-order-functions (filter, foldLeft etc.), helper-functions, pattern matching, recursion, and think about whether passing functions as parameters would help, although these are not strictly necessary. Additionally, while you can not mutate any variables, you can make local variables that are the result of performing various operations. You are also welcome to use basic object properties like `length` and `isEmpty` (as these are things which can be done easily in a functional way).

**Task:** Test your `Checkout` class exhaustively in `CheckoutTest.scala`. See Lecture 19 for details on how to use the tester in Scala if you are unsure how to do so.

## 2 Vote Counting

You’ve decided to find out once and for all who is the best superhero of all time by asking all of your friends who is their favorite hero. Now it’s time to count up the votes, so you will to write a Scala program to help figure out who won!

Your voting system receives as input a `List` of `Votes`, represented as a class in Scala. Your task is to create a class which can tally a list of votes, and return percentages of votes for each hero.
Task: In *Scala*, create a class `Vote`, which takes two `Strings`: a hero’s name, and the universe which the hero is from (e.g. Marvel, DC, Harry Potter, etc.) in its constructor.

Task: In Scala, create a class `VoteCounting`, which takes in its constructor a `List[Votes]`.

Now that you have your class, you want to be able to count the votes for a particular hero, and return a double representing the percent of votes that this candidate received. However, you are worried that the polarizing recent film “Star Wars: The Last Jedi” may skew the results, so you decide that all characters from the “Star Wars” universe are disqualified from the vote.

Task: Create a `candPercentage` method in your `VoteCounting` class, which takes in a `String` for a candidate’s name, and returns a `Double` for the percent of votes (eg. 0.45) received. This percent calculation should not count any votes from the “Star Wars” universe, and should not exceed 1.0. An empty list should return 0.0 for any input candidate, as should a list containing only characters from “Star Wars”. Additionally, an input for a candidate that is not in the list should output 0.0.

Note: You MUST use the functional aspects of Scala to get full credit for this problem. See the first problem for clarification on what this means.

Task: In a class `VoteCountingTest` test your `VoteCounting` class exhaustively.

3 Earthquake

Geologists want to monitor a local mountain for potential earthquake activity and have installed a sensor to track seismic (vibration of the earth) activity. 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 inclusive are vibration frequencies (in Hz). 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 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).

Task: In *Scala*, create the `MaxHzReport` class, which takes in its constructor a date, which is an 8-digit `Double` in the same format as above, and a measurement, which is a `Double` representing the highest frequency recorded for that particular date. You may assume that the input is exactly as specified as above, and don’t have to error check this.

Task: In *Scala*, create the `Earthquake` class, which has a method `dailyMaxForMonth`. This method consumes a `List[Double]` of sensor data and a month (an `Int` between 1 and 12) and produces a `List[MaxHzReport]` of reports 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 above). Ignore data for months other than the given one. Each item in the `List` should be an instance of `MaxHzReport`.

For example, given the sequence of values above and the month 10 (for October), the resulting `List` should contain:
- MaxHzReport(20151004, 200)
- MaxHzReport(20151005, 0.03)
- MaxHzReport(20151007, 130)

**Hint:** You might find the functions `takeWhile` and `dropWhile` to be helpful. In particular, you can use these to strategically generate a `List[List[Double]]`. Start here on your attempts to solve this problem. The syntax for these methods looks like the following:

```scala
def takeWhile(data: List[Double], condition: (Double) => Boolean): List[Double] = 
  data.takeWhile(condition)
def dropWhile(data: List[Double], condition: (Double) => Boolean): List[Double] = 
  data.dropWhile(condition)
```

**Note:** You *MUST* use functional aspects of Scala to get full credit for this problem. See the first problem for clarification on what this means.

**Hint:** Stuck on trying to get the month out of an 8 digit double? Try the following (though make sure that you understand each part of what’s going on!):

```scala
val date: Double = 19950331
val month: Int = (date.toInt/100) % 100 \ Returns 3
```

**Task:** Exhaustively test the Scala version of the Earthquake class in `EarthquakeTest.scala`.

## 4 Arrays

The last portion of this assignment is a short online assessment of your level of understanding of arrays. This is an important diagnostic tool for us, and should help you figure out what you may still be having trouble with. This is graded for completeness, not correctness, but please do try your best!

**Task:** Go to the site linked [here](http://example.com) and follow the instructions.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/cs018/feedback](http://cs.brown.edu/courses/cs018/feedback)