Practice questions

COMP-202A, Fall 2011, All Sections

These questions are not necessarily the same as a final as they aren't necessarily exactly representative of the degree of difficulty, length, or material of the exam. That is, you should not assume that the questions on the exam will be exactly like these. However, doing these ON PAPER will help aide you in studying.

Part 1 (0 points): Warm-up

Warm-up Question 1 (0 points)
Define a new type `Product` . A `Product` will represent something that you can sell. It should have as properties a `String name`, a `double weight`, and a `String manufacturer`. These should all be private properties. You should add a constructor with 3 variables as input, as well as getters and setters for each of these three properties.

Warm-up Question 2 (0 points)
Define a new type `Merchant`. A `Merchant` will represent a particular store. It should have as private properties a `String name`, a `String location`, and a `boolean isOnline`. You should add getters, setters, and a constructor which takes 3 variables as input.

Warm-up Question 3 (0 points)
Define a new type `Offer`. An `Offer` will store as a private property a `Merchant`, a `Product`, a `double currentPrice`, a `double normalPrice`, and a `boolean limitedSupply`. You should add a constructor as well as geters for these. You do not need to add setters.

Warm-up Question 4 (0 points)
Define a new type `Store`. A `Store` will store as a private property an `ArrayList<Offer>`. You should add as methods the following:

- A method `getAllOnSale()` This method should return an `ArrayList<Offer>` which represents every `Offer` for which the current price is less than the normal price.
- A method `getMostDollarSavings()` This method should return the `Offer` which represents the `Offer` which saves you the most dollars by calculating the `difference` (subtraction) between the current price and normal price.
- A method `getMostPercentSavings()` This method will be similar to `getMostDollarSavings` but it should return the best deal based on calculating the Offer with the largest savings in *percentage* discounted. You can calculate the percentage by taking the smallest of the current price divided by normal price.
- A method `getProduct` which takes as input a `String name` and returns all products that match the same name as the input String.
- A method `getAllLessThan` which takes as input a `double price` and returns an `ArrayList<Offer>` that have price less than the input price.
- A method `getAllOffers` which returns a *duplicate* of the `ArrayList<Offer>` stored as a property. You may use the constructor for `ArrayList` that generates the duplicate, but you may find it good to practice by doing this "manually" with a loop.

Warm-up Question 5 (0 points)
Define a type `Shopper`.

- Define a *static* method `getAllOffers` which takes as input an `ArrayList<Store>` and returns an `ArrayList<Offer>` representing *all* the `Offer`s across all the stores.
- Write a method `getValue` that takes as input an `ArrayList<Offer>` and returns a double representing the summed value of each item. It should do so by calling an (imaginary) method called `getUtility` which is part of the `PracticeLongAnswer` utilities and has the following method header:

```
public static double getUtility(Product product)
```

- Define a *static* method `getMostUseful` that takes as input an `ArrayList<Store>`, and a double `availableMoney`. This method should return an `ArrayList<Offer>` representing the best offers one can use with `availableMoney`. It should do this by doing the following:

1. First write a private helper method `getMostUseful` that takes as input an `ArrayList<Offer> offers`, a `double availableMoney`, and an `ArrayList<Offer> chosen`.
2. The helper method should use recursion to calculate the best `ArrayList<Offer>`.
3. One base case is if there are no items in `offers`, then the method should return `chosen`.
4. A second base case is if `availableMoney` is less than 0, then the method should return `null`.
5. If there are items, then your method should recursively call the method `getMostUseful` twice. To do this, you should create duplicate `ArrayList<Offer` for both `offers` and `chosen`.

   To make the calls, you need to fill 3 parameters. In both calls, the `offer` parameter should have the first item removed. In the first cases you should subtract from `availableMoney` the cost of the first item and add the first item to the `ArrayList<Offer> chosen`. In the second case you should *not* do this and should call the method with the same input as before.

   Your method should then call the method `getValue` on the returned `ArrayList<Offer>` and return the `ArrayList<Offer>` with higher value.
6. Once you have written the above private helper method, you can make your public method a one line method that simple calls the private method with the same `offers`, the same `availableMoney` and an empty `ArrayList<Offer>` for the variable chosen. The public method should return the value returned by the private method.