Assignment 4

Due date

- April 20th, 11:59PM EST.

Github link

The git links are the following:

- https://classroom.github.com/a/qgic7Zc9

Submit your code as per the provided instructions.

Assignment Goal

- Implement the state pattern to capture the changes in spending potential of a person. This is an individual assignment.

Programming Language

You are required to program using Java.

Compilation Method

- You are required to use ANT for compilation of code written in Java.
- Your code should compile and run on *remote machines* or the *debian-pods* in the Computer Science lab in the Engineering Building.

Policy on sharing of code

- EVERY line of code that you submit in this assignment should be written by you or be part of the code template provided for this assignment. Do NOT show your code to any other student. Our code-comparison software can very easily detect similarities.

Project Description

- Let us say that a person's spending potential and priorities change based on the running average of the money earned. *Note that for this assignment, currency does not matter. Therefore, money is just represented as a number.*

- The running average of the money earned determines the state the person is in. The window size for calculating running average will be provided via commandline.
- The following are the categories of items.
    - **basic**
    - **moderatelyExpensive**
    - **superExpensive**
- The person can be in one of the following states. The conditions to be satisfied to be in each of these states are also provided.
    - **BASIC** - Default starting state. Signifies that the person only spends money on basic items.
      Condition → $0 \leq$ running average of money earned < 10000.
    - **LUXURIOUS** - Signifies that apart from the basic necessities, the person also spends money on moderatelyExpensive items.
      Condition → $10000 \leq$ running average of money earned < 50000.
    - **EXTRAVAGANT** - Signifies that apart from basic and moderatelyExpensive items, the person also spends money on superExpensive items.
      Condition → $50000 \leq$ running average of money earned.
- Depending on the state, the person either agrees/disagrees to purchase an item.

Input

- The available items filename is provided via commandline option. Below is the format of the file. Note that words are camelcased starting with lowercase letter.

    basic:<item name>
    basic:<item name>
    ...
    moderatelyExpensive:<item name>
    moderatelyExpensive:<item name>
    ...
    superExpensive:<item name>
    superExpensive:<item name>
    ...

  Example available items file is shown below.

    basic:householdItem
    basic:medicine
    basic:food
    moderatelyExpensive:cinema
    moderatelyExpensive:automobile
    moderatelyExpensive:vacationTrip
    superExpensive:yacht

superExpensive:privateJet
superExpensive:island

- The money earned and and items that can be purchased are provided in the input file, the name of which is also provided via commandline. Below is the format of the file. Note that any number in the file should be an integer.

  money:<amount>
  money:<amount>
  ...
  item:<item to purchase or not>
  item:<item to purchase or not>
  ...
  money:<amount>
  money:<amount>
  ...
  item:<item to purchase or not>
  item:<item to purchase or not>
  ...

  Example input file is shown below.

  money:10000
  money:4000
  item:cinema
  item:medicine
  item:householdItem
  money:500
  item:medicine
  item:householdItem
  money:25000
  item:yacht
  item:vacationTrip
  money:60000
  item:medicine
  item:householdItem
  money:60000
  item:yacht

- The following commandline options are to be accepted.
  - *-DinputFile* - Input filename.
  - *-DavailableItemsFile* - Available items file.
  - *-DrunningAverageWindowSize* - Window size for running average calculation.
  - *-DoutputFile* - Output filename.

Output

- The name of the output file will be provided via commandline.
- When processing the input file, do the following.
    - If the line represents money earned, then DO NOT write anything to the output file.
    - If the line is an item, then write the following to the output file.
        1. Name of the state.
        2. Item.
        3. Yes/No indicating whether the person agrees to purchase the item or not.

        The format of the above information in the output file would be **<state name>::<item name>--<YES/NO>**.

    - Instead of directly writing to the output file, use the Results class as a mediator.
- The output file will have the following format.

    <state name>::<item name>--<YES/NO>
    <state name>::<item name>--<YES/NO>
    ...

    Assuming that the running average window size = 2, the example output for the above input is shown below.

    BASIC::cinema--NO
    BASIC::medicine--YES
    BASIC::householdItem--YES
    BASIC::medicine--YES
    BASIC::householdItem--YES
    LUXURIOUS::yacht--NO
    LUXURIOUS::vacationTrip--YES
    LUXURIOUS::medicine--YES
    LUXURIOUS::householdItem--YES
    EXTRAVAGANT::yacht--YES

Code Organization

- Follow a similar directory structure as that of the previous assignments for the driver code and util package. However, for the other classes and/or interfaces, make sure that they are in the appropriate package.

Submission

- Same as before

General Requirements

- Start early and avoid panic during the last couple of days.
- Class participation points will be given to students who submit 5 interesting input files along with corresponding output files. Note that the input/output should we well formatted and valid.
- Submit a README.md file (placed at the top level).
- Apply all design principles (wherever applicable).
- Separate out code appropriately into methods, one for each purpose.
- You should document your code. The comments should not exceed 72 coloums in width. Use javadoc style comments if you are coding in Java.
- Do not use "import XYZ.*" in your code. Instead, import each required type individually.
- All objects, in Java, that may be needed for debugging purposes should have the "toString()" method defined. By default, just place a toString() in every class.
- Every class that has data members, should have corresponding accessors and mutators (unless the data member(s) is/are for use just within the method.).
- If a class does not implement an interface, you should have a good justification for it. For example, it is ok to have an abstract base class and a derived class, wherein both do not implement interfaces. Note that the Driver code is written by end-users, and so the Results class must implement the interface, or else the source code for Results will have to be exposed to the end-user.
- Include javadoc style documentation for at least 3 classes. It is acceptable for this assignment to just have one parameter of a method described for each method's documentation.
- For the classes provided in the code template, add interfaces as appropriate

Design Requirements

- Programming to an interface for states and context.
- Context is not aware of which is the current state and only holds an interface reference.
- Input file should be processed one line at a time.

Late Submissions

- The policy for late submissions is that you will lose 10% of the grade for each day that your submission is delayed.

Grading Guidelines

Grading guidelines have been posted here.