Homework 5

1 Introduction

This week's homework will be about using Object Oriented Programming concepts to model a business's financial transactions. Please read the entire document and **do not forget to put the collaboration statement at the top of your submission!**

2 Problem Description

In order to test systems and new features, businesses will often run simulations to verify that they will work as expected. For this simulation you will model people buying items from a business. After simulating these transactions you will generate a well formatted report to view the results of the simulation.

Your program should take in 3 arguments from the command line corresponding to the number of people in the simulation, how many days(iterations) will be simulated, and the random number generator seed respectively. If the proper number of arguments is not provided, a message similar to the one shown in the example should be printed and the program should exit without error.

Each person should start the simulation with $5,000 and the same people should be used for the duration of the entire simulation. Every person should purchase a single item (one of any particular item available) on each day of the simulation if possible. The item to be purchased should be selected randomly by generating a random integer corresponding to the number of available items, disregarding the amount left of each item. For example if there are 10 items being sold, the number generated should be between 0 and 9 inclusive even if some of those 10 items are out of stock. This functionality should be provided using the `Random` class seeded with the third argument passed in from the command line. For any day, if the person does not have enough money or the selected item is out of stock then that transaction should not happen and the simulation should continue (i.e. the same person should **not** re-attempt a transaction if it fails the first time that day).

When the simulation is complete, your program should print a summary of the simulation as shown in the example below. The summary should include the following information:

- Name of the business
- Execution time in milliseconds
- Number of days simulated
- Total number of transactions completed
- Total sales in dollars
- Number of items being sold

- Name of best selling item

The execution time is the time for only the simulation to complete in milliseconds. The number of items in stock is the number of distinct items (not the sum of item quantities). For the data provided this number would be 10. The best selling item is determined by the highest quantity sold of any particular item. If there are multiple items with the same maximum amount sold, the name of first item found with this amount should be displayed in the summary.

- Running the program with command line arguments. The first argument is the number of people, the second argument is the number of days, and the third is the seed for the random number generator.

```
$ java MarketSim 5 10 1331
Running simulation with 5 people over 10 days...
Simulation Report: Amazon
Execution time: 0.08ms
============================================================
Days of simulation:                     10
Total Transactions:                     50
Total Sales:                            $3823.58
Number of Items being sold:             10
Best selling Item:                      "Jaybird X2"
============================================================
```

- Running the program without correct number of arguments

```
$ java MarketSim
MarketSim requires 3 to arguments to be run. Usage: java MarketSim <people> <days> <seed>
```

3 Solution Description

You will create classes to represent a person (`Person.java`), business (`Business.java`), and item (`Item.java`) in order to simulate financial transactions for a business. You will also create a fourth class that contains the main method and will run the simulation called `MarketSim`. This class will read in the command line arguments and run the simulation with the number of people given in the first argument for the number of days specified in the second argument using the provided seed in the third argument. After the simulation has completed, `MarketSim` will print a summary containing all of the information shown in the above example. You must properly encapsulate all of the classes you create.

We have provided the `SimData` class which has a field called `businessData`. This field contains a business name and item information in a 2D `String` array for testing your code. The first row of the array will have a single element containing the name of the business. All following rows contain the item information in the order of the item's name, price, and initial quantity in stock. Feel free modify the data to make sure your code works, but do not change the organization of the data.

3.1 Item Class

The `Item` class represents an item that a business can sell. Each item must have a name and a price. There must be a way to keep track of the quantity of this item remaining in stock. Items may only be purchased for their given price so there must also be a way to update the amount in stock and prevent purchases from being made when the amount left is 0.

3.2 Person Class

The `Person` class represents people that can purchase items. People should have an amount of money to be used for purchases stored as a floating point number.

3.3 Business Class

The `Business` class contains methods that will allow people to interact with a business. The business must have a name and be able to store any amount of `Items` provided in `SimData`. `Person` objects should be able to purchase items from the business at the item's defined price. People must not be able to purchase items if they do not have enough money to do so.

3.4 MarketSim Class

The `MarketSim` class is the entry point of the simulation and will use the classes described above to implement the simulation and print out a summary as detailed in the Problem Description.

4 Important Notes, Tips, and Tricks

- You should use `System.nanoTime()` to find the execution time of the simulation.
- You are NOT permitted to used any other data structure in the place of arrays. Additionally, you are NOT permitted to use any class that trivializes this assignment. This includes but is not limited to `Arrays.java`, `ArrayList.java`, etc.
- If your program crashes with correct input, it will be **-25 points**.
- Enhanced for loops can be very useful when working with arrays when you do not need direct access to the the array.
- You will be using `java.util.Random` and some of its various methods. Whenever you want to use the `nextInt()` method, use the method by the same name which takes in a bound: `nextInt(int bound)`. Read the API as to what numbers are generated and think about what number you might want to use as your bound.
- Your `Random` object will be seeded with a command line argument so each run with the same arguments and data should produce the same output. If not this may be an indication that something is going wrong.
- Feel free to consult outside sources for help! The TAs are always here to assist, and a quick Google search for some topic or concept can be beneficial as well.
- For those of you that like to procrastinate, don't. This homework might be a little tricky even though it doesn't sound like it. I'd hate for you to realize that on the day it is due.

5 Javadocs

For this assignment you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online documentation for them is very detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to have are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 13.31
 */
public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */
    public Dog(){
        ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */
    public int add(int a, int b){
        ...
    }
}
```

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

5.1 Javadoc and Checkstyle

You can use the Checkstyle jar mentioned in the following section to test your javadocs for completeness. Simply add -j to the checkstyle command, like this:

```
$ java -jar checkstyle-6.2.1.jar -j *.java
Audit done. Errors (potential points off):
0
```

6 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **20** points.
Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar -a *.java
Audit done. Errors (potential points off):
0
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off.

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **20** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

7 Turn-in Procedure

Non-compiling or missing submissions will receive a zero. NO exceptions

Submit `Item.java`, `Person.java`, `Business.java` and `MarketSim.java` to T-Square. **Do not submit** `SimData.java`, any compiled bytecode (`.class` files), or the Checkstyle jar file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code through Checkstyle!

7.1 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.

2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.

3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.

4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps insure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!