Supermarket Simulation

Project

Copyright © 2022 Colin Depradine.
All rights reserved.

Table of Contents

Supermarket Simulator

Introduction

You work for a local software development company, *Mango Simulations Inc*, as a programmer. The company creates custom made simulators for a variety of industries. You have been given the job of creating a simulator for a local supermarket, *Sweetest Fruits and Vegetables*. They want to use the simulator to improve their customer service and overall business model.

Your team has decided that an object-oriented programming language is the most suitable type for this project. The team chooses the Java programming language since the customer wishes to run the simulation on a variety of platforms such as Windows and Mac. The customer also wishes to reduce the maintenance costs of the simulation.

Objectives

This is a group project of 2-3 persons per group.

The objectives are:

- Developing your object-oriented programming skills
- Develop your Java coding skills.
- Develop team building skills.

Terms

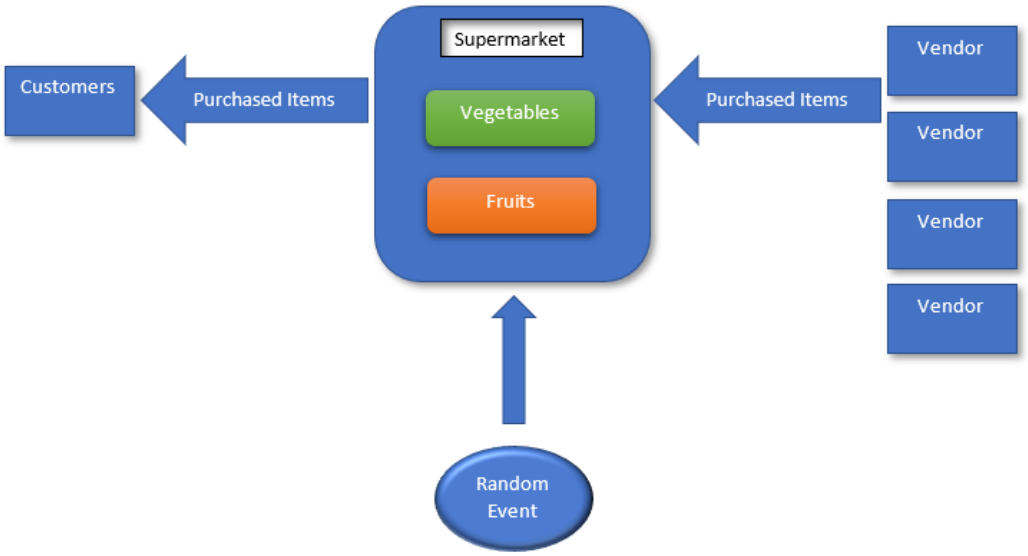
The following terms are used in this documentation.

Item	Description
Supermarket	Represents the supermarket being simulated.
Vendor	Represents a company that sells vegetables, fruits or both to the supermarket.
Item	Generic term that refers to both fruits (all five types) and vegetables (all five types).
Vegetable	Represents a vegetable that is bought or sold. There are five types of vegetables: carrots, lettuce, cucumbers, parsley and onions
Fruit	Represents a fruit that is bought or sold. There are five types of fruit: mangoes, avocados, limes, bananas and watermelons.

Specifications

In this project you are creating a simple supermarket simulator which simulates activities that take place during the day at the supermarket. On each day several transactions take place, such as customers purchasing items or vendors delivering goods. The available items are fruit (mangoes,

avocados, limes, bananas and watermelons) and vegetables (carrots, lettuce, cucumbers, parsley and onions). Occasional, serious event occurs such as the fridge breaking down.



Each cycle of the simulator represents a single day. The default number of cycles the simulation will run is 50.

The simulation process is as follows.

Initialise all objects as necessary. The minimum initialisation is as follows.

1. Process any command line arguments.

2. Randomly generate values for the total vegetables and fruit that each vendor currently has in stock. This value must fall in the range 10 – 100 (inclusive of both numbers).

3. Display the menu. The menu will have three mandatory options; run the simulation, display the log to the screen and exit the program. The menu will be text-based. You can add more options as deemed necessary.

After the initialisation, the total number of cycles is executed. During each cycle, a set of activities is performed, as described below.

1. Determine and execute a random event. Descriptions of random events are given later in the document.

2. Each item has a spoilt value which is the total number of cycles that must occur before the item is spoilt. This value is randomly generated when the items are first created for the vendor and lies between 5 and 10.

   At the start of each cycle, the spoilt value is decremented by 1. When it reaches 0, the item is considered spoilt and is removed from the list of items available for purchase.

3. Customers purchase items. The type and number of items purchased is randomly generated from the set of allowed fruits and vegetables as stated previously. The minimum number of items that can be purchased is 1 and the maximum is 20.

   If the number of items desired is less than what is available, then all of the remaining items are sold to the customer and the inventory for that item becomes zero.

   This is performed 10 times since it is assumed that 10 customers per day will purchase items.

   The supermarket will then restock the item by purchasing from the available vendors (see step 5).

4. The supermarket maintains a maximum number of each item per day. At the end of the day, the number of items that have spoiled is determined and they are removed. The total number of remaining items is calculated, and this determines how many items need to be purchased at the end of the day.

   The maximums are:
   - Vegetables: 100 of each type of vegetable
   - Fruit: 200 of each type of fruit

5. Each vendors restocks their goods. This is done on a random basis where the minimum number of items is 10 and the maximum is 100 for the random number generation.

6. Vendors deliver new items based on orders made by the supermarket. On arrival, the supermarket pays for the items.

   The supermarket cannot purchase goods from the vendor if there is not enough money. You are to determine what happens in this case.

   There are three available vendors:
   a. Fruits Are Here (sells only fruits)
   b. Tasty Vegetables (sells only vegetables)
   c. All You Can Eat (sells both fruits and vegetables)

   A vendor may not be able to supply all of the items needed and so the supermarket will purchase all that is available and then seek the remainder from other vendors. The number of items that a vendor has per day is randomly generated as described in step 5.

7. The total amount of money made by each vendor, for the entire simulation is tracked. Therefore, the money made in the previous step is added to this running total per vendor.

8. The profit made from sales of items, purchasing items from a vendor and lost from spoilage is calculated.

   profit = total sales – total purchases – total cost of spoilt items

   – losses caused by a random event

The profit is added to the supermarket's bank account. It is possible that a negative profit can occur which is referred to as a loss.

9. If a terminating condition is reached, terminate the program, otherwise return to step 1. Terminating conditions are discussed in the next section.

Terminating Conditions

The following conditions will cause the simulation to terminate.

- The simulation has executed the total number of iterations.
- The supermarket runs out of money.
- An exception not predefined for this assignment has occurred.

Random Number Generation

Throughout the simulation several items need to be randomly generated. Those items are listed below.

Item	Values to be Generated
Number of cycles it takes an item to spoil.	This value is randomly generated and lies between 5 and 10.
Customers purchase items.	Randomly determine which item to purchase. The minimum number of the selected items that can be purchased is 1 and the maximum is 20.
Vendor initially stocks and restocks goods.	This is done on a random basis where the minimum number of items is 10 and the maximum is 100 for the random number generation. After each cycle, a new value is generated per vendor.
Fridge breaks down.	The supermarket must pay the cost. The cost is randomly generated from the range 0 to 100.
Electricity goes off.	The length of time is randomly generated.
Items spoil faster than expected.	The percentage of items that spoil is randomly generated.
Vendor does not deliver the goods.	Randomly select one of the three vendors.

The code below is a simple method for random number generation, but it is not suitable for complex calculations such as cryptography.

```
Random rand = new Random();

// Get a number between 0 and 10 (excluding 10).

int r = rand.nextInt( 10 );
```

Menu

The simulator will display a text-based menu with the following three mandatory options.

```
Supermarket Simulator

====================

A. Start the Simulation

B. Display the Log

C. Exit

Enter your selection:
```

The user will select the appropriate letter. The following Java code can be used for reading the input from the console.

```
import java.util.Scanner;

class DemoInput
{
	public static void main(String[] args) {
		Scanner scan = new Scanner(System.in);
		String selection;

		// Enter the menu option and press Enter
		System.out.println("Enter your selection: ");
		selection = scan.nextLine();
	}
}
```

Log

Your program must keep track of each action that takes place during each cycle of the simulation. This is referred to as the log. The information stored in the log can be displayed on the screen or stored to a file.

The minimum actions that are to be tracked are:

Item purchased from a vendor:

- Name of the item
- Type of item
- Cost of the item
- Name of the vendor
- Total purchased

Total number of items sold in one day:

- Name of the item
- Type of item

- Total sold
- Total money received

Exceptions that are thrown:

- Name of the exception
- Details of the exception such as the name of the fruit

Random event:

- Name of the event

To write to a text file, the following Java code can be used.

```
import java.io.FileWriter;
import java.io.IOException;

public class DemoWrite
{
	public static void main(String[] args) {
		try {
			FileWriter writer = new FileWriter("log.txt");
			writer.write("log information.");
			writer.close();
		}
		catch (IOException e) {
			System.out.println("Could not create log file.");
		}
	}
}
```

Random Events

At the start of each day, a random event may occur. The possible events are:

1. **Fridge breaks down:** The supermarket must pay the cost. The cost is randomly generated from the range 0 to 100.
2. **Electricity goes off:** The number of hours off is randomly generated between 1 to 5 hours. For each hour off, 1% of all of our items will spoil,
3. **Vendor does not deliver the goods**: Randomly select the vendor.
4. **Items spoil faster than expected:** The percentage of items that spoil is randomly generated but lies between 1% and 10%..
5. **No purchases are made:** Customers do not buy anything from the supermarket.

A random event occurs on every 10 cycles (cycle 10, 20, 30 etc). A random event does not occur on cycle 1.

Commands

The simulation is a console driven program and so does not use a graphical user interface. As with any console-based program, command line arguments can be used to pass specific commands to the program.

To read the command line arguments passed to the program use the code below (the code has also been provided in the SimulationMain.java class included with this assignment).

```
public static void main( String args[] ){
	for(int i = 0; i < args.length; i++){
		if(args[i] == "iter") {
			if(i < args.length - 1){
				i++;
				try{
					int iter = Integer.parseInt(args[i]);
					// set total simulation iterations
				}
				catch(NumberFormatException e) {
					System.out. println("Invalid iter command,
total number of iterations is missing");
				}

			}
		}
	}
}
```

It should be noted that there are more advanced mechanisms for checking a string to see if it has a number. The one chosen for this assignment is one of the simplest and is suitable for this assignment since the test is only performed once at the start.

To pass command line arguments to the program, you include them when using the Java interpreter as follows.

```
java SimulationMain iter 100 verbose
```

The modes that are to be provided for this assignment are:

Mode	Command	Description

Total number of iterations.	`iter <total number of iterations>`
	
	For example:
	
	`iter 10`
	
	Simulation will run10 iterations or cycles.
		This command changes the number of iterations that the simulator will run from its default value of 50 iterations.

Display detailed system messages during each cycle.	`verbose`	This command indicates that detailed messages are to be displayed on the screen for each action performed. The required messages are described in the next section. The default behaviour is to not display any detailed messages.

Save logged information to a text file.	`log <filename>`
	
	For example:
	
	`log log.txt`
	`log c:\logs\log.txt`
		This command states where the information collected in the log (discussed later in the document) is to be stored.
		
		This is an ordinary text file that can be read by a basic text editor.

System Messages

As discussed in the previous section, the verbose command generates detailed system messages for each cycle. The default behaviour is to not display any detailed messages. In this mode the following is displayed when the program runs.

```
Simulation start
Commands passed are:
<list commands>
Total number of cycles performed: #
Total supermarket profit: $#
Total vendor profit: $#
Simulation has ended
```

The structure and details of the messages for the verbose mode are as follows.

```
Simulation start
Commands passed are:
<list commands>
    Cycle #
    <actions performed for cycle #>
    End of cycle #
Total number of cycles performed: #
Total supermarket profit: $#
Total vendor profit: $#
Simulation has ended
```

Repeat for each cycle

Each action listed has the following format.

Action	Text
Spoilt items	Spoilt: Fruit (watermelon): 10 *This means 10 watermelons have spoiled.*

Customer purchase	Total customer purchases: Fruit (10), Vegetables (20) *This means 10 fruit and 20 vegetables were purchased in total.*
Items purchased from vendor	Fruits Are Here: Fruit (10) Tasty Vegetables: Vegetables (30) All You Can Eat: Fruit (50), Vegetables (60) *List all of the items the supermarket purchased from each vendor.*
Random event	Vendor does not deliver the goods: Fruits Are Here *For this event, the vendor was Fruits Are Here.*

Fundamental Classes

The fundamental classes to be used in this assignment are as follows

Class	Description
`SimulationMain`	Contains the main function and processes the command line arguments.
`Supermarket`	The main class that manages the simulation.

The `SimulationMain` and `Supermarket` classes have been provided with basic code. You are required to use these fundamental classes. The code can be extended as necessary.

Exceptions

Exceptions are generated for the following events.

Exception	Description
`NotEnoughFunds`	The supermarket does not have enough money to purchase the specified items from the vendor.
`VegetableNotAvailable`	Is generated when a vegetable is not available. Contains the name of the vegetable.
`FruitNotAvailable`	Is generated when a fruit is not available. Contains the name of the fruit.
`UnknownCommand`	An unknown command was passed to the program.

Your program must generate and handle these exceptions without terminating the simulation.

Grading Components

This section describes a few key areas that are being considered during the grading process. The grading rubric contains all the areas that are being graded.

Design

This project's primary design approach is object-oriented programming using Java. This means that the focus is the creation of the appropriate classes.

It is tempting to cram everything in a few classes but is important to note that it is better to maintain the principle that each class should focus on one area only. For example, a class called Bicycle focuses on bicycles and not vans.

Testing

The following minimum tests and checks will be performed. Other tests will be performed based on the nature of the code written.

Test	Expected Outcome
All commands at the command line.	Each command is correctly interpreted. Incorrectly formatted commands are detected.
Console display	The menus and information displayed during each cycle is correct for both verbose and non-verbose modes.
Save log information to a file.	Information is correctly outputted and saved to a text file that can be read by a basic text editor.
On each cycle, each of the detailed actions listed under specifications.	Each has been implemented correctly.
Random events	All of the random events are correctly implemented.
Random number generation.	Numbers are randomly generated in the correct range.

Design Documentation

The design document template is provided on eLearning and should be used for the documentation requirement of this project.

In-Code Documentation

In-code documentation refers to the commenting of code to ensure that the functionality and purpose of the code is understood. The minimum comments that should be included are:

- At the top of each .java file, there should be a description of what the class does as well as the name of programmer(s).

- At the top of each function/method in the .java file there should be a description of the purpose of the function, the values passed to the function and what is returned by the function.

- Next to each field there should be a description of its purpose.

- Utilise sensible names for classes, fields and functions/methods that reflect their purpose.

Grading Rubric

This project is worth 17% of the total course mark. The grading scheme is given below.

Area	Excellent	Good	Average	Unsatisfactory
Design Documentation (10)	All functionality has been described. All sections have been completed. Descriptions are fully detailed and understandable. The design choices have been fully justified. **(10 marks)**	Most of the functionality has been described. Most of the sections have been completed. Descriptions are generally detailed and understandable. Most of the design choices have been fully justified. **(7-9 marks)**	About half of the functionality has been described. About half of the sections have been completed. Descriptions are partially detailed and understandable. About half of the design choices have been fully justified. **(4-6 marks)**	Very little of the functionality has been described. The sections have not been completed. Descriptions are not detailed or understandable. The design choices have not been fully justified. **(0-3 marks)**
Class Decomposition (30)	The responsibilities have been well subdivided between the classes. Each class has a single purpose. The methods and fields are appropriate for each class.	Good subdivision of responsibilities between the classes. Most of the classes have a single purpose. The methods and fields are appropriate for most of the classes.	Average subdivision of responsibilities between the classes (about half). About half of the classes, have a single purpose. The methods and fields are appropriate for half of the classes.	Very little subdivision of responsibilities between the classes. A few of the classes have a single purpose. The methods and fields are appropriate for a few of the classes.

	(24-30 marks)	**(16-23 marks)**	**(8-15 marks)**	**(0-7 marks)**
Data structures and algorithms chosen (30)	Excellent use of the appropriate data structures and algorithms. Excellent coding practices have been used with no redundant or unnecessary code. Algorithms are efficient. No unnecessary data is used. **(24-30 marks)**	Good use of the appropriate data structures and algorithms. Good coding practices have been used with little redundant or unnecessary code. Algorithms are mostly efficient. Small amounts of unnecessary data is used. **(16-23 marks)**	Average use of the appropriate data structures and algorithms. Considerable redundant or unnecessary code is present. Algorithms are generally not efficient. Significant amounts of unnecessary data is used. **(8-15 marks)**	Poor use of the appropriate data structures and algorithms. Redundant and unnecessary code is largely present. Algorithms are not efficient. Most of the data that is used is unnecessary. **(0-7 marks)**
Implementation and Execution (30)	Over 80% of the required functionality has been implemented and executes correctly. **(16-20 marks)**	Between 50% - 80% of the required functionality has been implemented and executes correctly. **(11-15 marks)**	Between 25%-49% of the required functionality has been implemented and executes correctly. **(6-10 marks)**	Less than 25% of the required functionality has been implemented and executes correctly. **(0-5 marks)**
Compiling (10)	The software compiles with no errors and less than 3 warnings. **(10 marks)**	The software compiles with 1-6 errors and 4-6 warnings. **(7-9 marks)**	The software compiles with 7-12 errors and 7-10 warnings. **(4-6 marks)**	The software compiles with greater than 12 errors and greater than 10 warnings. **(0-3 marks)**
Naming Conventions (10)	Over 80% of the code uses the correct naming conventions. **(10 marks)**	Between 51%-80% of the code uses the correct naming conventions. **(7-9 marks)**	Between 26%-50% of the code uses the correct naming conventions. **(4-6 marks)**	Between 0%-25% of the code uses the correct naming conventions. **(0-3 marks)**
In-Code Documentation (10)	Over 80% of the code has been fully documented with comments laid out in a	Between 50%-80% of the code has been fully documented with comments	Between 25%-49% of the code has been fully documented with some of the	Less than 25% of the code has been fully documented with very few of

	readable and understandable form. **(10 marks)**	mostly laid out in a readable and understandable form. **(7-9 marks)**	comments laid out in a readable and understandable form. **(4-6 marks)**	the comments laid out in a readable and understandable form. **(0-3 marks)**
Total (130)				

Deliverables

The final deadline for this assignment is 11th November 2022 at midnight. It should be submitted on eLearning.

The deliverables are:

- The Java code in the .java files. Do not include the .class files.
- The completed design document which can be found on eLearning, must be completed and included with the submission.