Department of Computer Science, Mathematics & Physics

COMP2232 – Object-oriented Programming Concepts

Semester 1, 2021-22: Assignment #1

Roxia Restaurants

Read this entire document before starting your assignment!

This assignment is to be completed on an INDIVIDUAL basis. All work is to have been created by the student submitting the work. Any form of copying, cheating or plagiarism will result in the assessment being disqualified and awarded a grade of 0 (zero).

The purpose of this assignment is to provide exposure to arrays, classes, methods and fields and increase familiarization with their usage. It will also lay the foundation for building a framework for the **Roxia Training system**.

Introduction

Roxia Inc. are expanding their restaurant chain in the country of Ambroxia. The restaurants serve customers from Ambroxia as well as visitors from close neighbouring countries of Scorax and Zorax. The objective of this program is to train staff in correctly and efficiently serving diners in the restaurant. Proper service should result in maximum throughput and minimum deaths. Food service which is not carried out correctly will result in some messy situations and possible international tensions.

Below are a few things you will need to know before implementing your software for the restaurant.

The Restaurant

- The main dining area is divided into 2 sections: Zone A and Zone B.
- **Zone A:**
  - caters to Zoraxians and Scoraxians only. Ambroxians are advised not to enter (however, sometimes they ignore the signs and wander in like curious cats – it ends the same way for them as it does for the cats).
  - there are two long tables for seating: one table for Zoraxians and the other for Scoraxians. Each table has **15 seats**. More seats may be added to this restaurant at some time in the future.
- **Zone B:**
  - caters to Zoraxians and Ambroxians only. Scoraxians are not permitted in this area under any circumstances.
  - there are **30 tables** here, set up in a grid formation (5 rows of 6 tables each).
  - each table is identified by a number (ranging 1-30) and seats one diner, only.
- **Only** one diner may occupy a seat at any point in time.
- The restaurant serves **zoron**, **scoron**, **ambron** & **elixon**.

Ambroxians

- They have a maximum energy level of 25. Exceeding this results in 'death by greed'.
- They have a deep-seated fear of the number 6 and any number that is a multiple of 6. For example, 6, 12, 18, 24, etc … will send them into a panic and causes this normally passive species to go berserk, destroying everything around them. This behaviour becomes contagious to other

Ambroxians in the room. Therefore, seating allocations should be carefully thought out to avoid a disastrous incident.
- They eat zoron, scoron & ambron. Consumption of elixon increases their energy levels to full capacity.

Scoraxians

- They have a maximum energy level of 20. Exceeding this results in 'death by greed'.
- They are predators who find weak Zoraxians easy pickings.
- They eat scoron and ambron. They are highly allergic to zoron (consuming it results in 'death by allergic reaction'). Consumption of any amount of elixon depletes their energy level by 25%.

Zoraxians

- They have a maximum energy level of 15. Exceeding this results in 'death by greed'.
- They become weak and defenceless if their energy level falls below 3.
- They eat zoron and ambron; can also eat scoron but cannot digest it fully resulting in gaining only half of its energy value. Consumption of any amount of elixon increases their energy level by 25%.
- They are natural hunters and consider Ambroxian's a suitable meal.
- They hunt in **pairs**. Therefore, an Ambroxian should not be seated with a Zoraxian on BOTH its left and right or BOTH its front and back. For example, a seating arrangement of (Zoraxian, Ambroxian, Zoraxian) would be disastrous for the Ambroxian and filling for the Zoraxians.

How the program works

The user will be awarded service points for each fully fed diner; points will be deducted for deaths caused.

The Setup – Zone A

- To begin, the user will be required to enter their name.
- The tables are randomly populated with 8 diners (any number of Zoraxians and Scoraxians totalling 8.)
- Scoraxians are initialized with a random energy level between 5 and 10 inclusive.
- Zoraxians are initialized with a random energy level between 4 and 7 inclusive.
- Service Points, Deaths and Fed are initialized to 0 and displayed along with the user's name.

The Play

- There will be 7 rounds.
- At the start of each round, the tables are displayed indicating all seats. Filled seats indicate the energy level of the diner in the seat; empty seats are left blank.
- The chef will prepare a combination of 6 dishes (1 ambron, 1 scoron, 1 zoron, 3 random), each with a random energy value within a range of 2 to 5 inclusive. The list of 6 dishes (energy & type) will be displayed. (**Recall:** restaurant serves ambron, elixon, scoron & zoron)

- For each dish, the user will be asked to select the species of diner and seat number for the dish. User has the option to send unwanted dishes back to the kitchen for disposal.
- Any diners who have overeaten (exceed energy level) will throw a GreedyGutsException and be removed from the table. 3 Service Points will be deducted from the user along with the display of an appropriate Poor Service indicating incident details.
- Any diner who has been fully fed (reached max energy level) will be removed from the table. 2 Service Points will be awarded to the user along with the display of an appropriate message.
- Any Zoraxians who are weak (energy less than 3) will be consumed by the Scoraxian seated opposite. If this seat is empty the nearest Scoraxian (to the right) will consume it. 3 Service Points will be deducted from the user along with the display of an appropriate Poor Service message indicating incident details.
- Any Scoraxians who were served zoron will throw an AllergyException and be removed from the table. 3 Service Points will be deducted from the user along with the display of an appropriate Poor Service message indicating incident details.
- At the end of each second round (i.e. rounds 2, 4, 6, etc…), every diner who is remaining at their respective tables will become hungrier (deplete energy by 1).
- **Seating:** At the start of each round **except the first**:
  - the system will randomly add 2 more diners (Zoraxian and/or Scoraxian) to the restaurant. If the restaurant/table is full the user turns them away. Otherwise, the user places them in an appropriate seat.
  - the system will randomly determine if an Ambroxian will wander into the restaurant or not. If one does, the diner with the lowest energy will be allowed to eat the Ambroxian resulting in a 2-point energy increase for the lucky diner. An appropriate message indicating incident details will be displayed.

The End

At the end of the last round, a summary will be displayed with the following information:

- User's name
- Service Points
- Number of diners fully fed.
- Number of diners suffering death from Poor Service
- Number of Ambroxians who met an untimely demise through 'death by curiosity'.

At this point, Roxia Inc. requires a simple program to help them train staff in the management of **seating allocations** within Zone A and would like you to create a demo to show them what you can do.

Instructions for THIS Assignment

For **this assignment**, complete the following classes to create a framework for the **Restaurant Management System**. Each class must be placed in its own file, named using the format: ***<class-name>.java***

1. Create a class called **Diner** which will have the following properties.

   Fields
   - String name – name of the diner
   - char speciesCode - the speciesCode used to display the diner's species on the screen.
   - String species - holds the species of the diner.
   - int energyLevel – diner's current energy level

   Methods
   - Diner() - Constructor used to initialize species to "D", speciesCode to 'D' and name to "??".
   - Mutators & Accessors – An accessor method & mutator method for each data member.

2. Create a class called **Zoraxian** (*inherits from **Diner***), having the following properties.

   Methods
   - Zoraxian(String name) – Constructor which initializes data members: species to "zoraxian"; speciesCode to 'z'; name to value of parameter; and energyLevel to random value between 4 and 7 inclusive.
   - Mutators & Accessors – for any new data members.

3. Create a class called **Ambroxian** (*inherits from **Diner***), having the following properties.

   Fields
   - boolean isBerserk – holds current status

   Methods
   - Ambroxian(String name) – Constructor which initializes all data members: species to "ambroxian"; speciesCode to 'a'; name to value of parameter; energyLevel to 0; and isBerserk to false.
   - Mutators & Accessors – for any new data members.

4. Create a class called **Scoraxian** (*inherits from **Diner***), having the following properties.

   Methods
   - Scoraxian(String name) – Constructor which initializes all data members: species to "scoraxian"; speciesCode to 's'; name to value of parameter; and energyLevel to random value between 5 and 10 inclusive.
   - Mutators & Accessors – for any new data members.

5. Create a class called **Trainee** which will have **at least** the following properties.

   Fields
   - String traineeName
   - int servicePoints

- int numDeaths
- int numFed

Methods

- Trainee(String name) – Constructor which initializes name to value of parameter and hthe remaining data members to 0.
- Mutators & Accessors – for each data member.

6. Create a class called **Restaurant** which will have **at least** the following properties.

   Fields

   - int numAmbroxians
   - int numScoraxians
   - int numZoraxians
   - Zoraxian[] zoraxTableZoneA - Used for seating Zoraxians in Zone A
   - Scoraxian[] scoraxTableZoneA - Used for seating Scoraxians in Zone A
   - Diner[][] tablesZoneB – used for seating in Zone B

   Methods

   - Restaurant()
   - void setUpZoneA()

     This method will be used to randomly populate seats at the tables in Zone A. Each seat has one of three possible states: containing a Zoraxian, containing a Scoraxian or empty.

   - boolean fillSeat(int seatPos, Diner diner) *// do NOT change these parameters*

     This method is used to assign either a Scoraxian or Zoraxian to a seat given the seat position and diner object. If the action was successful the function returns `true`. Otherwise, `false`. **A diner may not be assigned to a seat which is already occupied.** *Note: Scoraxian/Zoraxian/Ambroxian objects may be used where a Diner object is used since they are subtypes of Diner.*

   - boolean clearSeat(int seatPos, char species)

     This method is used to clear (make empty) a seat given the seat position and species of diner. If the action was successful the function returns `true`. Otherwise, `false`. **A seat which is currently empty may not be cleared.**

   - void displayTables()

     Prints the content of the table arrays in the following format:

     ```
     Scoraxians:
     Seat #: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
     Energy:   | 4 | 3 |   |   | 5 |   | 10|   | 3

     Zoraxians:
     Seat #: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
     Energy: 4 |   | 3 |   | 12| 7 |   | 9 |   | 3
     ```

   - You may add other data members and methods to manage displaying and handling user menu, etc…

7. Create a class called **`RTSDemo`** which contain **`main`** and does the following:

    a. Welcomes the user to the management system.
    b. Creates an instance of **`Restaurant`**.
    c. Sets up the restaurant demo. (*See **The Setup – Zone A***)
    d. Displays the seats and allows the user to select actions to carry out: filling a seat/clearing a seat, displaying tables or exiting the demo. (*See section on **Seating** in **The Play***)
    e. **Note:** implementation of seating allocations is required. You will not be required to feed diners for **this** assignment.

DO NOT CHANGE THE FIELD OR METHOD NAMES OR DEFINITIONS. YOU ARE REQUIRED TO USE EACH OF THE FIELDS AND IMPLEMENT EACH OF THE METHODS AS GIVEN. YOU MAY CREATE ADDITIONAL FIELDS AND METHODS, IF NECESSARY, TO COMPLETE YOUR PROGRAM.

Java Random Class

To generate a random number, the following code snippet may be used for guidance.

```
import java.util.Random;            // import this library

Random randNum = new Random();      // instantiate random generator

val = randNum.nextInt(max); //gets a value between 0 and max-1 (inclusive)

val = randNum.nextInt(max) + 1; //gets a value between 1 and max(inclusive)
```

Marks

This assignment is worth **18%** of your final course mark. In addition to marks for code, overall marks will also be allocated for: Documentation/code formatting; User-friendly interface; Successful compilation; Correct execution

Marks will be deducted for:

- Violations of the Code Conventions. *Please ask if not sure.*
- Missing/inappropriate use of access specifiers

Deliverables

1. **Assignment 1 is due for submission on Sunday 10th October 2021 by 11:55pm via the Moodle/eLearning submission tool.**

2. **Only ZIP files will be accepted.** No other compression types should be used. Failure to comply with this instruction will incur a **penalty**.

3. You must submit a **Plagiarism Declaration Form** with this assignment, if you do not accept the online one.

PLEASE NOTE: The specifications for this assignment are subject to change. You will be notified if any such changes were to occur.