Object Oriented System Design

Assignment #3

Game cafe: Part 3

- This assignment uses the code base you created in the first assignment.
- Buying coffee
- Threading
- Observer pattern
- Arraylist
- Inner class

Class GameCorner

Variables:

- private ArrayList<Game> gamesInCafe;
- private ArrayList<Game> rentedOutGames;

Methods:

- public GameCorner(ArrayList<Game> games);
- public ***double*** rentOutGame(String name); // Cost of renting is the game's price * 0.5
- public void returnGame(String name);
- public void buyGame(Game game);
- public void printCafeDetails(); // Prints all games in the cafe and rented out, and the money in the register
- public ***double*** repairGame(String name);
- private int getIndexGamesInCafe(String name); // Helper function to get the index of the given game in the array, -1 if game doesn't exist.
- private int getIndexRentedOutGames(String name); // Helper function to get the index of the given game in the array, -1 if game doesn't exist.

Class CoffeeCorner implements Oberver

Variables:

- private int coffeeIndex; // Set to 0 in constructor
- private HashMap<String, Double> coffeetypes; // HashMap of coffee types with prices
- private ArrayList<CoffeeMachine> machines; // Arraylist of all the available coffee machines

Methods:

- public CoffeeCorner(int machineAmount);
- public void listCoffeeTypes(); // Prints all types of coffee with the price
- public int makeCoffee(String type); // Start preparing the coffee of the given type
- public void update(int index); // Observer method, called when coffee is done from thread

Private class CoffeeMachine extends Thread

This class is an inner class of CoffeeCorner

Variables:

- private ArrayList<Observer> observers; // List of observers
- private int index; // Index of currently made coffee
- private string machineName; // Name of the machine

Methods:

- public CoffeeMachine(int index, String name);
- public void run(); //Thread's standard method
- public void subscribe(Observer observer); //Add observer method
- public void unsubscribe(Observer observer); //Remove observer method

Interface Observer

Methods:

- public void update(int index, String name)

  //this method gets called with the index of the coffee that is ready and the name of the machine that made the coffee

Interface Observable

Methods:

- public void subscribe(Observer observer) //adds an observer to this observable
- public void unsubscribe(Observer observer) //removes an observer from this observable

Main class

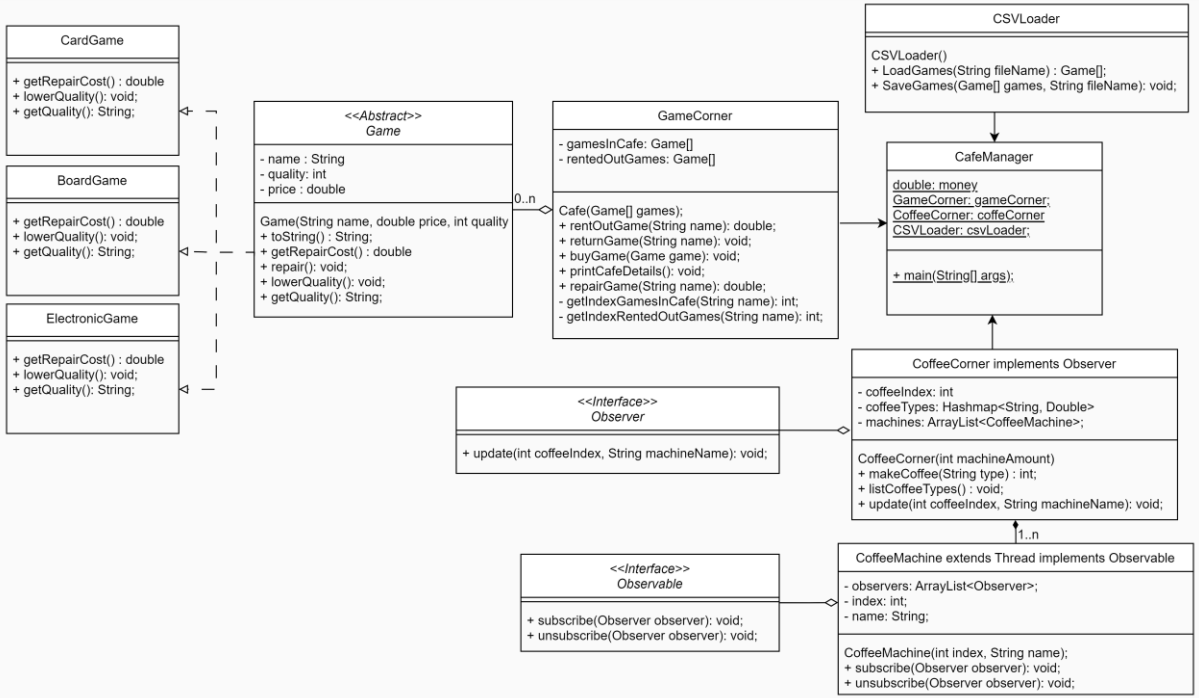
Variables:

- double money //Money moved from gameCorner class to here

method main:

- Initialize the CSVLoader
- Initialize the GameCorner
- Initialize CoffeeCorner with 2 coffee machines
- Buy a coffee
- Before the coffee is ready, rent a game to show you have implemented threading well
- Buy a new game.

UML


CardGame
+ getRepairCost() : double
+ lowerQuality(): void;
+ getQuality(): String;
BoardGame
+ getRepairCost() : double
+ lowerQuality(): void;
+ getQuality(): String;
ElectronicGame
+ getRepairCost() : double
+ lowerQuality(): void;
+ getQuality(): String;
<<Abstract>>
Game
- name : String
- quality: int
- price : double
Game(String name, double price, int quality
+ toString() : String;
+ getRepairCost() : double
+ repair(): void;
+ lowerQuality(): void;
+ getQuality(): String;
0..n
GameCorner
- gamesInCafe: Game[]
- rentedOutGames: Game[]
Cafe(Game[] games);
+ rentOutGame(String name): double;
+ returnGame(String name): void;
+ buyGame(Game game): void;
+ printCafeDetails(): void;
+ repairGame(String name): double;
- getIndexGamesInCafe(String name): int;
- getIndexRentedOutGames(String name): int;
CSVLoader
CSVLoader()
+ LoadGames(String fileName) : Game[];
+ SaveGames(Game[] games, String fileName): void;
CafeManager
double: money
GameCorner: gameCorner;
CoffeeCorner: coffeCorner
CSVLoader: csvLoader;
+ main(String[] args);
CoffeeCorner implements Observer
- coffeeIndex: int
- coffeeTypes: Hashmap<String, Double>
- machines: ArrayList<CoffeeMachine>;
CoffeeCorner(int machineAmount)
+ makeCoffee(String type) : int;
+ listCoffeeTypes() : void;
+ update(int coffeeIndex, String machineName): void;
<<Interface>>
Observer
+ update(int coffeeIndex, String machineName): void;
1..n
<<Interface>>
Observable
+ subscribe(Observer observer): void;
+ unsubscribe(Observer observer): void;
CoffeeMachine extends Thread implements Observable
- observers: ArrayList<Observer>;
- index: int;
- name: String;
CoffeeMachine(int index, String name);
+ subscribe(Observer observer): void;
+ unsubscribe(Observer observer): void;

Visualization

Buying coffee (1)

```
What would you like to do:
1: Rent a game, 2: Return a game, 3: Repair a game, 4: Buy a new game, 5: Save games, 6: Buy coffee
6
What kind of coffee do you want?
Capucino, Price: 3.75
Americano, Price: 2.5
Latte, Price: 3.25
Latte
The coffee is being prepared on Machine 1!
your number is: 1
```

```
Money: 8.25
Games in cafe:
Name: Uno, Quality: Good, Price: 8.0 , type: Card
Name: Battleship, Quality: Good, Price: 10.25 , type: Board
Name: Pokemon, Quality: Good, Price: 15.5 , type: Electronic
Games rented out:
What would you like to do:
1: Rent a game, 2: Return a game, 3: Repair a game, 4: Buy a new game, 5: Save games, 6: Buy coffee
6
What kind of coffee do you want?
Capucino, Price: 3.75
Americano, Price: 2.5
Latte, Price: 3.25
Latte
The coffee is being prepared on Machine 2!
your number is: 2
```

Visualization

Buying multiple cups coffee quickly after one another should make multiple threads run simulatiously

```
Money: 11.5
Games in cafe:
Name: Uno, Quality: Good, Price: 8.0 , type: Card
Name: Battleship, Quality: Good, Price: 10.25 , type: Board
Name: Pokemon, Quality: Good, Price: 15.5 , type: Electronic
Games rented out:
What would you like to do:
1: Rent a game, 2: Return a game, 3: Repair a game, 4: Buy a new game, 5: Save games, 6: Buy coffee
Coffee with number: 1 is 20% done.
Coffee with number: 2 is 20% done.
Coffee with number: 1 is 40% done.
Coffee with number: 2 is 40% done.
Coffee with number: 1 is 60% done.
Coffee with number: 2 is 60% done.
Coffee with number: 1 is 80% done.
Coffee with number: 1 is ready for pickup at Machine 1.
Coffee with number: 2 is 80% done.
Coffee with number: 2 is ready for pickup at Machine 2.
```

Rules

- ~~Only regular arrays [], no Lists or Maps.~~
- Use getters and setters when needed and only make them public if they are actually meant to be used from outside the class.
- Reading code is more difficult than writing code, place comments in complex methods.

Rules (2)

- You are not allowed to add extra arrays for the games, make use of the existing gamesInCafe and rentedOutGames arrays.
- You are not allowed to use external libraries to read the CSV file.
- The CSVLoader has to utilize exception handling
  - The system should not crash when the given filename is not found.
  - The system should not crash when the data in the file is not well formatted (ex: empty file, wrong data types).

Rules (3)

- You have to be able to keep using the cafe when coffee is being prepared
- Use the observer pattern for notifying the CoffeeCorner about the coffee
- Make use of interfaces when implementing the observer pattern
- The coffee machines give updates every 20% until the coffee is ready
- Making the coffee should take around 10 to 15 seconds for testing purposes
- The CoffeeCorner has at least 2 coffee machines
- Alternate the coffee machines every time you make a coffee

Hints

- Using a HashMap is one of the best ways to keep a key-value pair list, you will learn about this in the last week so here is a hint on how to use it.

  EN: https://www.w3schools.com/java/java_hashmap.asp

  KR: https://coding-factory.tistory.com/556

Submission

Submit your program code package(Assignment3_code_이름_학번) and design document (Assigment3_이름_학번) in LMS. (Korean/English)

The design document describes the main methods, flow of the code and overall design of the program. It should also mention and explain **every** method that you added that was not described in this powerpoint.

Deadline: 18th of june (6월 18일)23:59

Late penalties:

Up to 24 hours = -25%,

24-48 hours = -50%,

48-72 hours = -75%

after 72 Hours you get an automatic 0.