Object-Oriented Programming Final Project Report

“Virtual Pet Game”

Lecturer:
Jude Joseph Lamug Martinez, MCS

Report done by:
Ostein Vittorio Vellim
2602206783

Binus School of Computer Science Undergraduate Program
Universitas Bina Nusantara
Jakarta
2023

Table of Contents

I. Project Overview

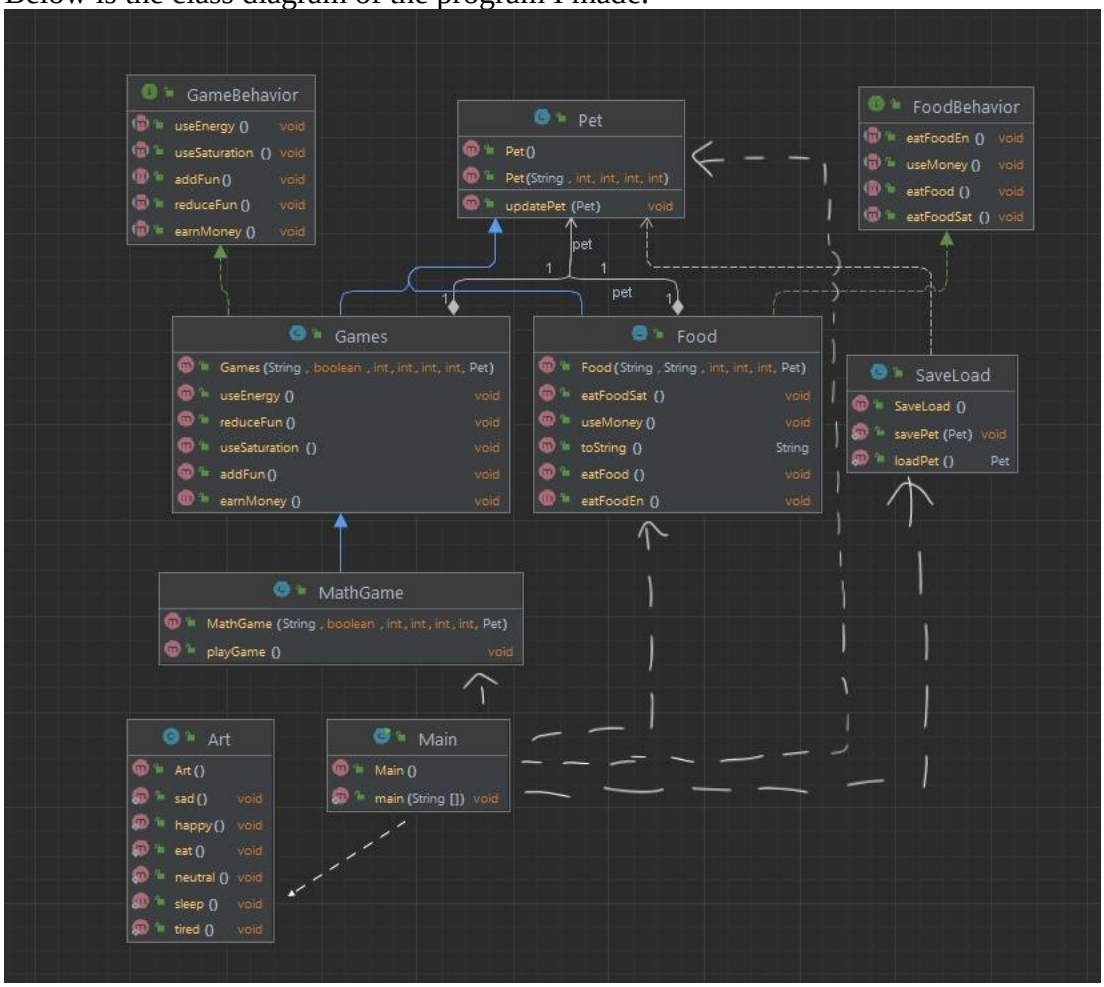
Humans are social creatures, but sometimes we get lonely. That's why sometimes we seek companions, be it friends, family, lovers, or even pets. However, it is common for people to struggle with seeking a companion, that's why there are alternative ways. We can connect with people online through games, find a lover through dating apps, or even start up a virtual pet game and care for them. But what is a virtual pet?

A virtual pet is a computer program designed to accompany humans by simulating a companion. In this program, the user will be able to interact with the virtual pet as if it were a real pet. Typical functions in virtual pet programs include but are not limited to activities like feeding, playing, or cleaning the pet. While an overarching goal is optional, for instance keeping your pet alive for as long as possible, most users play the game with the goal of building a deeper bond between them and the pet and to make the pet happy. One upside of a virtual pet program is that due to its artificial nature, the options are not inherently necessary limited to animals people traditionally would keep as pets. There are options to keep normal animals, but also fictional creatures, for example the animals from the *Tamagotchi* line of products.

For my object-oriented programming final project, I created a virtual pet program featuring activities like feeding your pet, playing with them, and sleeping with them as well as a saving and loading feature.

II. Solution Design

Below is the class diagram of the program I made.



These are the classes I used and a brief explanation on their role to the overall program:

Main

The main class is used to run the main program. This class calls all the functions from the different classes in order to simulate the virtual pet. This class is also where new objects from the MathGame and Food classes, as well as to create a Pet object for the program.

Art

The art class houses all the different sprites used for the virtual pet's different expressions and reactions.

Pet

The pet class is used to initialize and construct all of the virtual pet's stats. This includes the pet's name, saturation, fun, energy, and the amount of money at hand.

Food

The food class serves as the template for all food items that are consumable for the pet. It implements from the FoodBehavior interface and contains several functions that modify the pet's stats depending on the food it eats. The food object itself is created in the main class.

Games

The games class serves a similar purpose to the food class, serving as a template for all games that the virtual pet can play. It implements the GameBehavior interface and it also contains functions that modify the pet's stats.

MathGame

The mathGame class contains a simple math game the player can play with the pet to increase its fun and money. This class extends the games class in order to inherit its stats-modifying attributes, and the main class itself creates an object from this class.

SaveLoad

The saveLoad class is used to house the saving and loading function of the program. It uses the BufferedWriter and BufferedReader library in order to be able to write data into a text file to save the data and parse through the file and reading it to load the game.

III. Implementations

There are several algorithms I made in order to give my virtual pet program its basic functionalities. In this section, I will explain some of my algorithms in detail.

Pet Class

The pet class is the first class I created. It is the class that contains all the pet's attributes that will be affected by the features. This class is also where I set the default stats for the pet. It also contained the updatePet() method which I will explain in a later section.

```
public String petName;
public int petSaturation;
public int petFun;
public int petEnergy;
public int petMoney;

public Pet(){
    this.petName = "Steve";
    this.petSaturation = 50;
    this.petFun = 60;
    this.petEnergy = 40;
    this.petMoney = 500;
}

public Pet(String petName, int petSaturation, int petFun, int petEnergy, int
petMoney) {
    this.petName = petName;
    this.petSaturation = petSaturation;
    this.petFun = petFun;
    this.petEnergy = petEnergy;
    this.petMoney = petMoney;
}
```

Food and Saturation

One of the features I added in my program is the ability to feed your pet. This is done by having a certain food object interact with the pet through a method called in the Pet class. The food object itself is created based on the Food class.

```
public String foodName;
public String foodDescription;
public int foodPrice;
public int foodSaturation;
public int foodEnergy;
private Pet pet;

public Food(String foodName, String foodDescription, int foodPrice, int
foodSaturation, int foodEnergy, Pet pet) {
    this.foodName = foodName;
    this.foodDescription = foodDescription;
    this.foodPrice = foodPrice;
    this.foodSaturation = foodSaturation;
    this.foodEnergy = foodEnergy;
    this.pet = pet;
}
```

As seen in the code snippet, the food object contains five distinct attributes: its name, description, price, saturation, and energy. The name of the food serves as its main point of distinction from the other food objects. The description itself is used in as an adjective for when the food is fed to the pet. The price, saturation, and energy is what affects the pet's own attributes. The foodPrice refers to the cost of the food, which will automatically reduce its value from the amount of money the pet currently has. The attributes foodSaturation and foodEnergy refers to the amount of saturation and energy the food will add to the pet's stats respectively.

```
public void eatFoodSat(){
    pet.petSaturation += foodSaturation;
}

public void eatFoodEn(){
    pet.petEnergy += foodEnergy;
}

public void useMoney(){
    pet.petMoney -= foodPrice;
}

public void eatFood(){
    if (pet.petMoney>=foodPrice){
        eatFoodSat();
        eatFoodEn();
        useMoney();
        Art.eat();
        System.out.println("You fed " + pet.petName + " some " +
foodDescription + " " + foodName + "." );
    }
    else{
        System.out.println("Not enough money!");
    }
}
```

In this code snippet, you can see the functions I created to affect the pet's stats. All those functions will be called in the eatFood() method, which is the main action done to feed the

pet. I also added a conditional gate, so that the pet won't be able to eat the food item if they are not able to afford it. You may also noticed an additional method that was not included in the class called Art.eat(). This is a static method I called from the Art class to show the pet's reactions.

The food objects I created in the Main class. For this program, I made four objects:

```
Food steak = new Food("Steak", "Yummy", 50, 50, 60, steve);
Food fries = new Food("Fries", "Tasty", 20, 30, 50, steve);
Food water = new Food("Water", "Splashy", 10, 20, 25, steve);
Food candy = new Food("Candy", "Sweet", 10, 10, 10, steve);
```

Games and MathGame

The other main feature I added is the ability to play games with your pet. Similar to the food class, the games would also be able to affect the stats of the pet.

```
public String gameName;
public boolean gameWin;
public int gameSaturation;
public int gameEnergy;
public int gameFun;
public int gameMoney;
private Pet pet;

public Games(String gameName, boolean gameWin, int gameSaturation, int
gameEnergy, int gameFun, int gameMoney, Pet pet) {
    this.gameName = gameName;
    this.gameWin = gameWin;
    this.gameSaturation = gameSaturation;
    this.gameEnergy = gameEnergy;
    this.gameFun = gameFun;
    this.gameMoney = gameMoney;
    this.pet = pet;
}
```

Here we can see some attributes different from the Food class, but they are also mostly used to modify the pet's stats. The attribute gameName is self-explanatory, being the name of the game. The attributes gameSaturation, gameEnergy, gameFun, and gameMoney are useld to determine the values at which the pet's respective stats will be modified. An attribute exclusive to the Games objects is gameWin, which serves as the game's loop in order to keep playing. You may also notice the pet attribute, which is also present in the Food class. This pet attribute is necessary in order for the objects derived from both Food and Games class to be able to modify the value of the pet's stats. Essentially, while it doesn't directly create a new pet object, it makes a sort of temporary object so that the attributes will be able to modify the stats. Then the stats will be updated within the Pet class.

```
public void useSaturation(){
    pet.petSaturation -= gameSaturation;
}
public void useEnergy(){
    pet.petEnergy -= gameEnergy;
}
public void addFun(){
    pet.petFun += gameFun;
}
public void reduceFun(){
    pet.petFun -= gameFun;
}

public void earnMoney(){
    pet.petMoney += gameMoney;
}
```

These are the main functions used by the games to modify the pet's stats. However the main method that calls these functions aren't located within the Games class, but rather the MathGame class.

I created the Games class with the idea that more games can be added. The only game currently available is a simple math game, where two numbers between one to ten are randomly generated as well as three random operator: addition, subtraction, or multiplication. The user will be tasked to answer correctly. If the paly answered correctly, they will be awarded money and fun will be added to the pet's stats. If the player answered incorrectly, they will lose fun, but the amount of money would stay the same. However every time the player finishes a game, the pet will use a certain amount of energy and saturation regardless of if the player wins or loses. I also added a conditional to play the game in the Main class where the pet's energy and saturation must be above zero in order to play. You can see the how the functions which affect the pet's stats are called after the player completed each game.

```
public void playGame(){
    gameWin = false;
    while (gameWin == false) {
        Random rand = new Random();
        int random_num_upperbound = 11;
        num1 = rand.nextInt(random_num_upperbound);
        num2 = rand.nextInt(random_num_upperbound);

        int operator_chooser = 3;
        opr = rand.nextInt(operator_chooser);
        if (opr == 0) {
            System.out.println(num1 + "+" + num2);
            ans = num1 + num2;
        } else if (opr == 1) {
            System.out.println(num1 + "-" + +num2);
            ans = num1 - num2;
        } else {
            System.out.println(num1 + "*" + num2);
            ans = num1 * num2;
        }

        Scanner math = new Scanner(System.in);
        System.out.println("What is the answer?");
        Integer answer = math.nextInt();
        if (answer.equals(ans)) {
            System.out.println("Correct!");
            useSaturation();
            useEnergy();
            addFun();
            earnMoney();
            gameWin = true;
            Art.happy();
        } else {
            System.out.println("Wrong!");
            useSaturation();
            useEnergy();
            reduceFun();
            gameWin = true;
            Art.sad();
        }
    }
}
```

Like the food class, I created the MathGame object in the Main class:

```
MathGame math = new MathGame("Math Game", false,5,5, 10, 10, steve);
```

Saving and Loading Data

For the virtual pet, I find that playing the game for hours at a time to be repetitive, so I implemented a saving and loading feature so that the users can reload their pet and play another time. For this function, I used BufferedWriter to write the save data and BufferedReader to read it.

```
public class SaveLoad {

    private static final String FILE_PATH = "petdata.txt";

    public static void savePet(Pet pet) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(FILE_PATH))) {
            writer.write(pet.petName + "," + pet.petSaturation + "," +
pet.petFun + "," + pet.petEnergy + "," + pet.petMoney);
            System.out.println("Pet saved successfully!");
        } catch (IOException e) {
            System.out.println("Error saving pet: " + e.getMessage());
        }
    }

    public static Pet loadPet() {
        Pet petNew = null;
        try (BufferedReader reader = new BufferedReader(new
FileReader(FILE_PATH))) {
            String line = reader.readLine();
            if (line != null) {
                String[] petData = line.split(",");
                if (petData.length == 5) {
                    petNew = new Pet(petData[0],
Integer.parseInt(petData[1]), Integer.parseInt(petData[2]),
                            Integer.parseInt(petData[3]),
Integer.parseInt(petData[4]));
                    System.out.println("Pet loaded successfully!");
                }
            }
        } catch (IOException e) {
            System.out.println("Error loading pet: " + e.getMessage());
        }
        return petNew;
    }
```

I used BufferedWriter because of its simplicity, and considering the scale of my project I feel that it is appropriate for my program. The BufferedWriter will save the five main attributes of the pet: its name, saturation, fun, energy, and money. It writes the data into a text file called “petdata.txt”. What I find convenient is that if you do not have the file path initially, it will create the text file for you, but once you have the file it will just replace the contents.

To load I used BufferedReader for the same reason, and the idea for the loading system is that the reader will parse through the text file for each of the data, each separated by a comma, before replacing the current pet's data with another pet that is created containing attributes from the saved data. However initially I encountered a problem with the feature where the system would say that it successfully loaded, but the pet's current attributes would remain the same.

Eventually I solved the issue by adding an update function within the Pet class, shown below:

Pet Class:

```
public void updatePet(Pet petNew) {
    this.petName = petNew.petName;
    this.petSaturation = petNew.petSaturation;
    this.petFun = petNew.petFun;
    this.petEnergy = petNew.petEnergy;
    this.petMoney = petNew.petMoney;
}
```

Main Class:

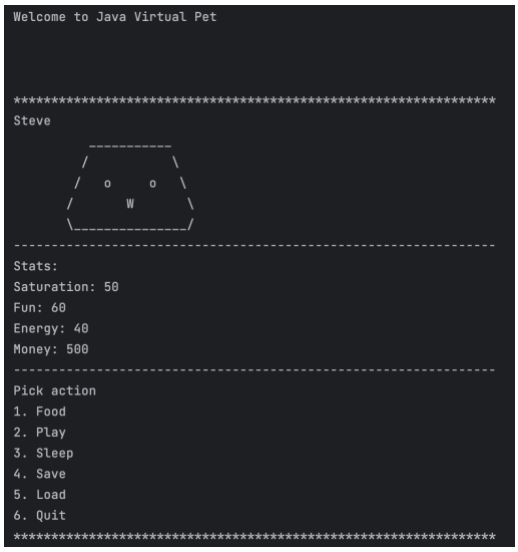
```
steveTemp = SaveLoad.loadPet();
if (steveTemp != null) {
    steve.updatePet(steveTemp);
}
```

The saved data will be loaded and created into a temporary pet object. Said pet object will then be used to update the current pet's data. With this function, it solved the initial issue as well as another issues that popped up where the save data would successfully load but wouldn't be modified after any action.

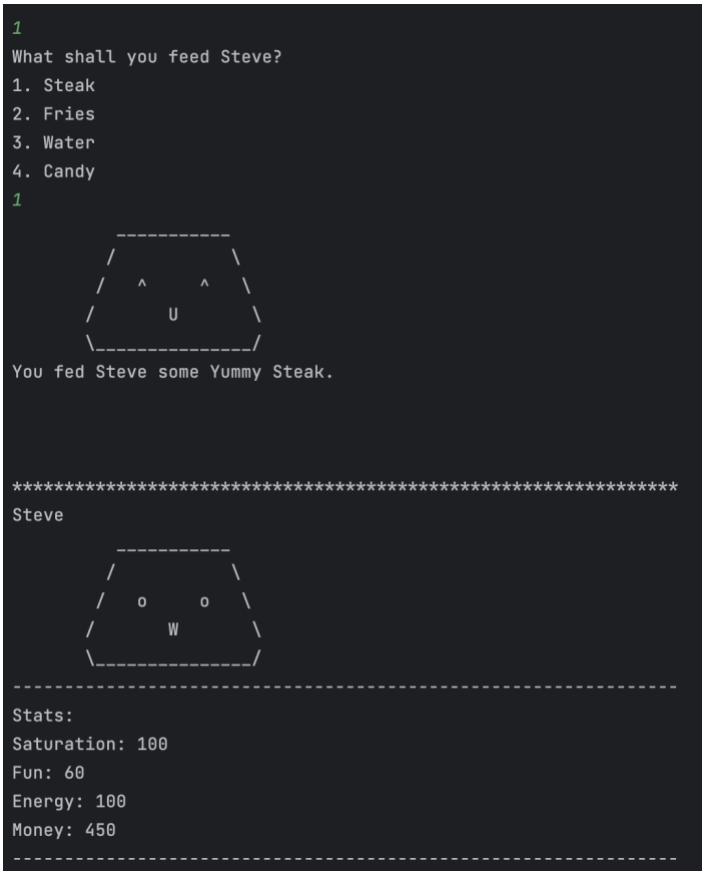
Art class

The art class contains the sprite used for the virtual pet program. There are six distinct sprites in total, and each of them will display when different conditions are met. For instance, the Eat sprite shows when the user feeds the pet and the Sleep sprite shows when the user prompts the pet to sleep. You can see some of the sprites in the next section.

IV. Program Screenshots



This is the main screen. It loads after you entered your pet's name. For this case, I named the pet Steve.



Feeding the pet. Note how the stats for saturation, energy, and money changed from the previous screen.

Playing a game with the pet. These pictures are taken right after feeding the pet. Notice how answering correctly (left) increases the fun and answering incorrectly (right) decreases the fun. Also note how the saturation and energy level decreases regardless of the answer.

Sleeping restores the pet's energy back to its maximum.

```
************************************************************
Steve
          ___________
         /           \
        /   o     o   \
       /       W       \
       \_________________/
------------------------------------------------------------
Stats:
Saturation: 90
Fun: 60
Energy: 100
Money: 460
------------------------------------------------------------
Pick action
1. Food
2. Play
3. Sleep
4. Save
5. Load
6. Quit
************************************************************
4
Pet saved successfully!
```

petdata.txt

```
Steve,90,60,100,460
```

Saving the data (left). Inside the save file (right).

```
************************************************************
Brian
          ___________
         /           \
        /   o     o   \
       /       W       \
       \_________________/
------------------------------------------------------------
Stats:
Saturation: 50
Fun: 60
Energy: 40
Money: 500
------------------------------------------------------------
Pick action
1. Food
2. Play
3. Sleep
4. Save
5. Load
6. Quit
************************************************************
```

```
------------------------------------------------------------
Stats:
Saturation: 50
Fun: 60
Energy: 40
Money: 500
------------------------------------------------------------
Pick action
1. Food
2. Play
3. Sleep
4. Save
5. Load
6. Quit
************************************************************
5
Pet loaded successfully!



************************************************************
Steve
          ___________
         /           \
        /   o     o   \
       /       W       \
       \_________________/
------------------------------------------------------------
Stats:
Saturation: 90
Fun: 60
Energy: 100
Money: 460
```

To showcase the loading function, I initialized another instance of the pet. This time I named him Brian (left). After initializing the pet, I loaded my save file and current pet's stats will now be updated to match the save file, bringing back Steve (right).

V. Reflection and Closing Words

While I am creating my project, I am reminded by my previous work, the text-based adventure game. I find that my final project for this semester greatly resembled my previous semester's final project, particularly in terms of style and simplicity. However, I wasn't very sure if I outdid myself this semester.

When coming up with the idea of the project, I had lots of ideas for a lot of features. A scrapped feature, for example, is a shop feature where the user can purchase items like food and cosmetics. I also initially had the idea to implement more games, hence the games class acting more like a template rather than an actual game.

I looked back at my ambitions and I felt as though I wasn't able to fulfil them fully, but I looked at what I am able to accomplish and I felt somewhat proud of it. To be able to learn a language from scratch and then being able to create a functional game out of it six months later brings me a great sense of accomplishment. From creating a flag-based interactive story to a virtual companion, all in all, I'm proud of both of my creations equally.

Though I am proud of what I have made, I know that I am still underexperienced and I am capable of creating greater things. I hope for the next years to come I will be able to learn more and accomplish greater things.