Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Laboratory 3

Title of the Laboratory Exercise: Introductory exercises in Java Classes, Interfaces and Constructors

1. Introduction and Purpose of Experiment

   Students apply object oriented programming to solve simple problems and get familiar with Java language

2. Aim and Objectives

   Aim

   To apply object oriented programming to solve simple

   Objectives

   At the end of this lab, the student will be able to

   - Apply object oriented programming for solving simple problems
   - Express solutions in Java language
   - Use Greenfoot IDE

3. Experimental Procedure

For the problems listed below, design the algorithm(s) and write the program(s). Tabulate the output for various inputs and verify against expected values. Compare the programming method in Java with C and Haskell programming languages. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

a. Write a program to develop a game for the scenario posed using Greenfoot IDE

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Documentation:	
a. Procedure and Algorithm(s):	**Procedure:** 1. Generating the Greenfoot Java Scenario Open Greenfoot and create a New Project by clicking on Scenario -> Java Scenario, Type the name of the Scenario and specify the location of the same. This generates the default Java Scenatio Template form Greenfoot and provides with Basic Classes such as the World and the Actor along with helper class Greenfoot. 2. Designing the Class Diagram Analyze what you need to do, and think of the possible Classes and Objects and do the Object decomposition, this can also be done on the fly while writing code but the basic structure needs to be thought of before typing anything, if possible make a UML diagram of the same. 3. Creating classes and Writing code Create subclass of World by clicking on World on the right, and selecting New subclass, type the filename and the class name and select the image if you want, click OK, this will create a Class with specified name and extends World. Now write the code for the Class, create more classes and extend the ones as required. 4. Execute and Debug Click on compile on the BlueJ editor, this will compile the files, create a new World by clicking on your World and select new World(), Cick on Run, to debug, click on Tools -> Show Debugger. 5. Generating the JAR file Click on Scenatio -> Share, select JAR and click on Export, this will give an executable jar file, which is portable for all systems.

Algorithms:

Algorithm _move for Player

Step 1: Start

Step 2: If KeyPressed is "right" then turn by 5°

Step 3: If KeyPressed is "left" then turn by −5°

Step 4: If KeyPressed is "up" then turn by −90°, move 5 pixels, turn by 90°.

Step 5: If KeyPressed is "down" then turn by 90°, move 5 pixels, turn by −90°.

Algorithm _move for Enemies

Step 1: Start

Step 2: move by speed pixels, here speed is a property of enemy.

Step 3: If getRandomNumber(100) less than 10 then turn by 8°, (there's 10% chance that roation will be done)

Step 4: Else if getRandomNumber(100) < 10 then turn by −8°.

Step 5: Stop

Algorithm eat

Step 1: Start

Step 2: Get oneIntersectingObject of Food and store it in toEat

Step 3: if toEat is not null then go to Step 4, else go to Step 7

Step 4: set the Score to currentScore + Food Points of toEat

Step 5: Add a new Fish to the World

Step 6: Increase the Difficulty of the Game.

Step 7: Stop

Algorithm checkBounds

Step 1: Start

	Step 2: If current X Position + Actor height/2 is greater than World width then set the location to (Actor Height/2, current Y Position) Step 3: Else If current X Position is less than equal to 0 then set the location to (World Width, current Y Position) Step 4: If current Y Position + Actor width/2 is greater than World height then set the location to (current X Position, Actor Width/2) Step 5: Else if current Y less than 0 equal to 0 then set the location to (current X Position, World Height); Step 6: Stop **Algorithm kill for Enemies** Step 1: Start Step 2: Get oneIntersectingObject of type Player and save it in player. Step 3: If player is not null Go to Step, else go to Step 6 Step 4: remove the player form the World. Step 5: run GameOver routine Step 6: Stop
b. Conclusions :	I have created a simple Java Sceneraio Game using Greenfoot IDE, that comes with an Actor-Model interfcce. I've learnt to use Greenfoot IDE, to create the Game. I've used Object Oriented Programming, using Inheritance for Code Reuse, this decreases the number of lines of code and also makes the development more intuitive. The Solution is expressed in Java, where I've used OOPs, and also some other Java features such as Higher Order Function and Lambda Expressions. One recommendation that I would give to developers is to not use Java for Game Development, because it's slow, it wasn't intended to make games, it lags, whereas C++ is

	really good for game development and most Game Companies use C++ for the same. Greenfoor IDE itself is rather very slow and very laggy and get's stuck at times, *simply put it's a pain to work on*. Apart from that the Inheritance Diagram can be improved at little with correct access specifiers, so that the further development becomes easier for the same.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Results and Discussions:

Screenshot:

```
public class SeaActors extends Actor{
    public void checkBounds() {
        if (getX()+getImage().getHeight()/2 >= getWorld().getWidth())
            setLocation(getImage().getHeight()/2, getY());
        else if (getX() <= 0)
            setLocation(getWorld().getWidth() - getImage().getHeight()/2, getY());

        if (getY()+getImage().getWidth()/2 >= getWorld().getHeight())
            setLocation(getX(), getImage().getWidth()/2);
        else if (getY() <= 0)
            setLocation(getX(), getWorld().getHeight() - getImage().getWidth()/2);
    }
}
```

Discussion:

This method applies for every SeaActor, since I need to check if the actor is coming close to the border of the world, and change the Actor's position accordingly. The methods that we use such as getX(), getImage() and getWorld() are methods defined in Actor and since our SeaActors are inheriting Actor they also inherit those methods.

Screenshot:

```
public void _move() {
    move(speed.intValue());
    if (Greenfoot.getRandomNumber(100) < 10) turn(8);
    else if (Greenfoot.getRandomNumber(100) < 10) turn(-8);
}
```

Discussion:

This method is for Enemies, they need to move, so we call this method every frame, it moves the Enemy by some units, the property speed is for Enemy, and as our difficulty increases, we increase this value, so the logic here is simple we get a RandomNumber(LIMIT), which generates random Integer number from 0(inclusive) to LIMIT (exclusive). And if it is less than 10 then we turn, Which means 10% of the time we are turning. This randomNumber method is a static method, hence is preceded by the class name.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Screenshot:

```
public void kill() {
    Actor player = getOneIntersectingObject(Player.class);
    if (player != null) {
        getWorld().removeObject(player);
        ((SeaWorld)getWorld()).gameOver();
    }
}
```

Discussion:

Every enemy has the ability to kill the player, hence any Enemy that inherits Enemy Class gets this method, getOneInteresectingObject return one object that intersect this object. This takes the graphical extent of objects into consideration. The Player.class is passed so the method will only look for Players, If null is passed then it will look for any kind of object. If the returned value is not null then we get the world and remove the instance of the object from the world, also we then call the gameOver method of the World.

Screenshot:

```
public class Player extends SeaActors {

    public void _move() {
        if (Greenfoot.isKeyDown("right")) turn(5);

        if (Greenfoot.isKeyDown("left")) turn(-5);

        if (Greenfoot.isKeyDown("up")) {
            turn(-90);
            move(5);
            turn(90);
        }

        if (Greenfoot.isKeyDown("down")) {
            turn(90);
            move(5);
            turn(-90);
        }
    }
```

Discussion:

Player is a class that Extends SeaActors, _move() is a method for Player to move the actor based on some keyboard keys, isKeyPressed is a static method that checks whether a given key is currently pressed down and returns a Boolean, either true or false. turn() and move() are methods defined in Actor Class and are inherited

into SeaActors and then into Player and anyone who inherits Player also inherits those, this is a multi-level inheritance.

Screenshot:

```
private void eat() {
    Actor toEat = getOneIntersectingObject(Food.class);
    if (toEat != null) {
        getWorld().removeObject(toEat);
        Score currWorldScore = ((SeaWorld)getWorld()).getWorldScore();
        // Increase the score
        currWorldScore.setScore(currWorldScore.getScore()+((Food)toEat).getFoodPoints());
        currWorldScore.incrementFishEaten();
        Food toAdd = null;
        if (toEat instanceof ClownFish)
            toAdd = new ClownFish();
        if (toEat instanceof PufferFish)
            toAdd = new PufferFish();

        getWorld().addObject(toAdd,Greenfoot.getRandomNumber(getWorld().getWidth()), Greenfoot.getRandomNumber(getWorld().getHeight()));

        currWorldScore.updateDifficulty();
    }
}
```

Discussion:

eat() is a method for the Player, here also we use the getOneInteresectingObject method of Actor to get any Intersecting Object of Food Type, since the Player can eat Foods or other Fishes, we remove the object is there is any intersecting Food, and every food has a score and a method getFoodPoints, Since we know that the Object returned is of type Food we TypeCast it explicitly to Food and we get the points and add it to our score board. Now that there is a food eaten we need to add another food item. We check if the eaten food is `instanceof` (keyword in java that return is the object is of that type) which type of food and adds the respective Food to the world at a random location. Also the difficulty is now updated/increased.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

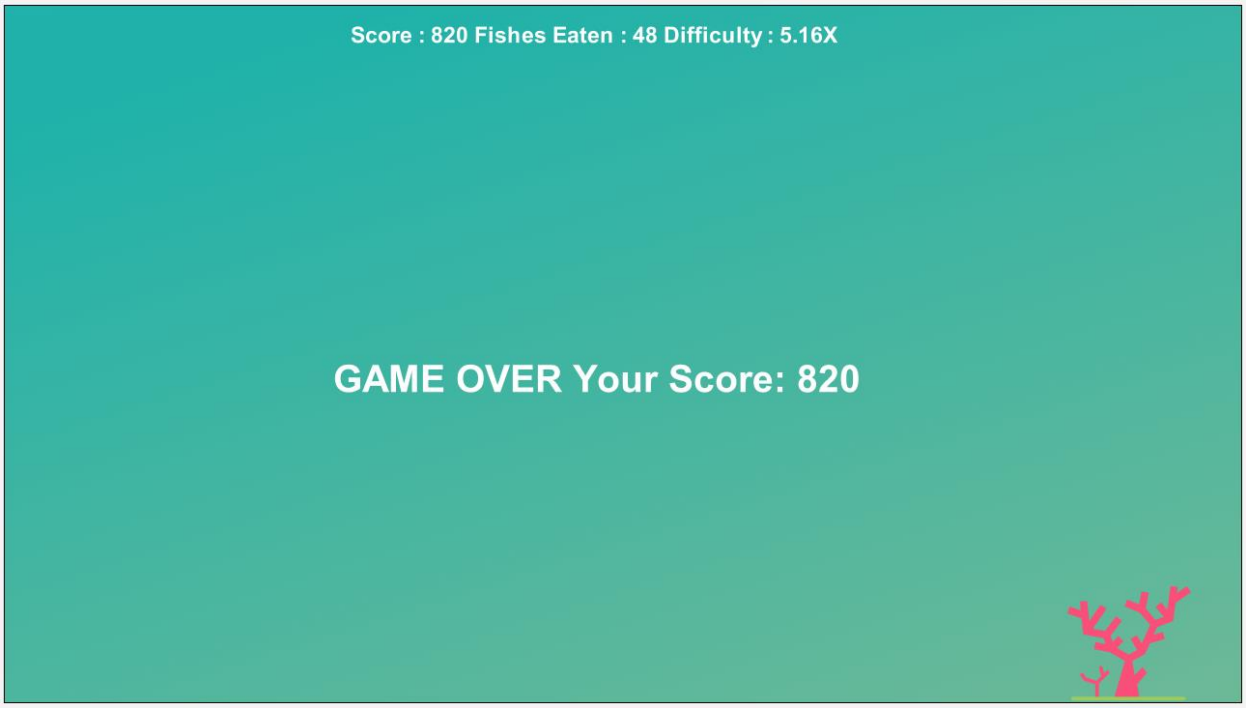
Screenshot:

Discussion:

The game when initially Run is presented with a fixed number of Enemies and Food, which are internally stores in an ArrayList. All the Objects in the Scene above are Actors, except the Coral at the bottom right, since they are actors the act() method is called every frame, and the definition written in this method of each of the actors is executes every frame, this is how the objects move in the Scene/World. The Score is updated dynamically as the Food is eaten and the Stats are displayed at the top. The player is moved by using the Arrow-Keys from the keyboard and as it makes collision the food that it made collision with is returned and the Fish that it has eaten is determined and accordingly the getPoints is called, which is the score related to that Food, hence the score is increased, also the Number of Fish eaten is incremented, the Difficulty is also increased, internally everytime there is a fish eaten updateDifficulty is called, which checks if the number of fishes eaten at current time is a multiple of 5, if it is then the difficulty is increased by 20%. This keeps on happening.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Screenshot:



Discussion:

The GameOver Screen, at this stage all the Objects are deleted from the World and the final score is displayed as a text. The text was made using Greenfoot's image creater, which takes a String and two Colors as parameters and then the image is set to the center of the Screen. The Game I made is an endless type and the goal is to get a maximum score, the fishes are generated as they are eaten and the difficulty is increased as the number of fishes are eaten increases.