Small App-Display & Review of Concepts PROG1400

Author: Daniel Donovan
Date: 2/10/2022

Class Exercise

Table of Contents

List of Tables

List of Figures

Class Exercise

1. Project Overview

Describe the overview of the project.
Create a 2D projectile game that launches projectiles from one side of your screen towards the other side. With it displaying on a Jpanel,

2. Project Requirements

Small Application – With a 2D Display and prints its location.

[30%] Take the previous projectile application and put it on the display (JPanel).
Use simple shapes for your projectiles
Use the JPanel example discussed in class
continue to print x, y positions on the console for debugging
These code mechanisms each should have their own paragraph in your document:

[10%] Use of the Abstract method (and corresponding derived method)
[10%] The relationship between object.start() <=> extends Thread <=>Thread Run() method
[10%] The relationship between the timer <=> ActionListener <=> ActionPerformed() <=> repaint() <=>PaintComponent
[10%] The use of bufferedImage inside DrawHere.paintComponent()
[10%] Create (update) an Activity diagram for a projectile

3. Design Plans

This was to provide print out program that will show you the object position as it is fired.

3.1. Design Sub-Paragraphs

Simulating gravity for objects and hitting its target(ground) and print out its coordinates.

3.1.1. Main Projectile Design

Snip of my main class. Which will make the full canvas for my projectile to be drawn onto.

Table 1: Main Line Program

```
public class Main {

    static Timer timer;
    static DrawHere d = new DrawHere();

    /**This is the run class which will execute the application */
    public static void main(String[] args) {

        /**This is setting up the JFrame which will be the window which
will hold/build my project within */
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 700);
        frame.setVisible(true);
        frame.setContentPane(d);
        frame.getRootPane().setBackground(Color.LIGHT_GRAY);

        /**This is a delay timer, so it will have time to load all the
math */
        timer = new Timer(5, d);
        timer.setInitialDelay(1);
        timer.setCoalesce(true);
        timer.start();
    }
}
```

The following table shows the sample output of the running program. Not the display part, will put this into later parts of this document.

Table 2: Sample Output

```
Main Program is starting...

position x = 2
position y = 312
A Target has hit its Target
Orange Goes BANG!!!!
```

3.1.2. Design of 2D Display

This part of the Application will show where all my objects get instantiated and get all their parameters.

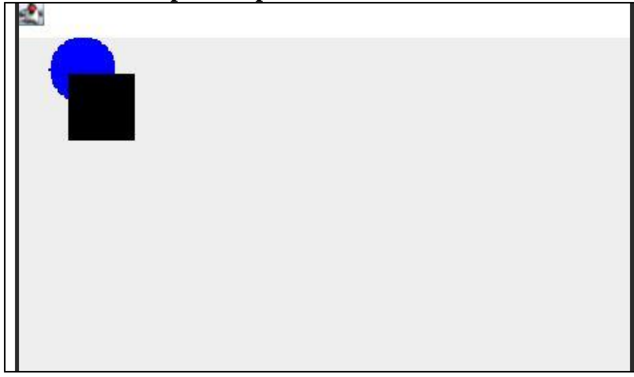
The following table shows the sample output of the running program.

Table 3: Snippet of the referred class

```
public DrawHere() {
    Random r = new Random();

    for (int i = 0; i <1; i++){
        list.add(new Orange(r.nextInt(5), r.nextInt(50), r.nextInt(5), r.nextInt(25)));
        list.add(new WaterBottle(r.nextInt(5), r.nextInt(50), r.nextInt(5), r.nextInt(25)));
        list.add(new WaterBomb(r.nextInt(5), r.nextInt(50), r.nextInt(5), r.nextInt(25)));
    }

    for (Projectile p : list){
        p.start();
     }
```

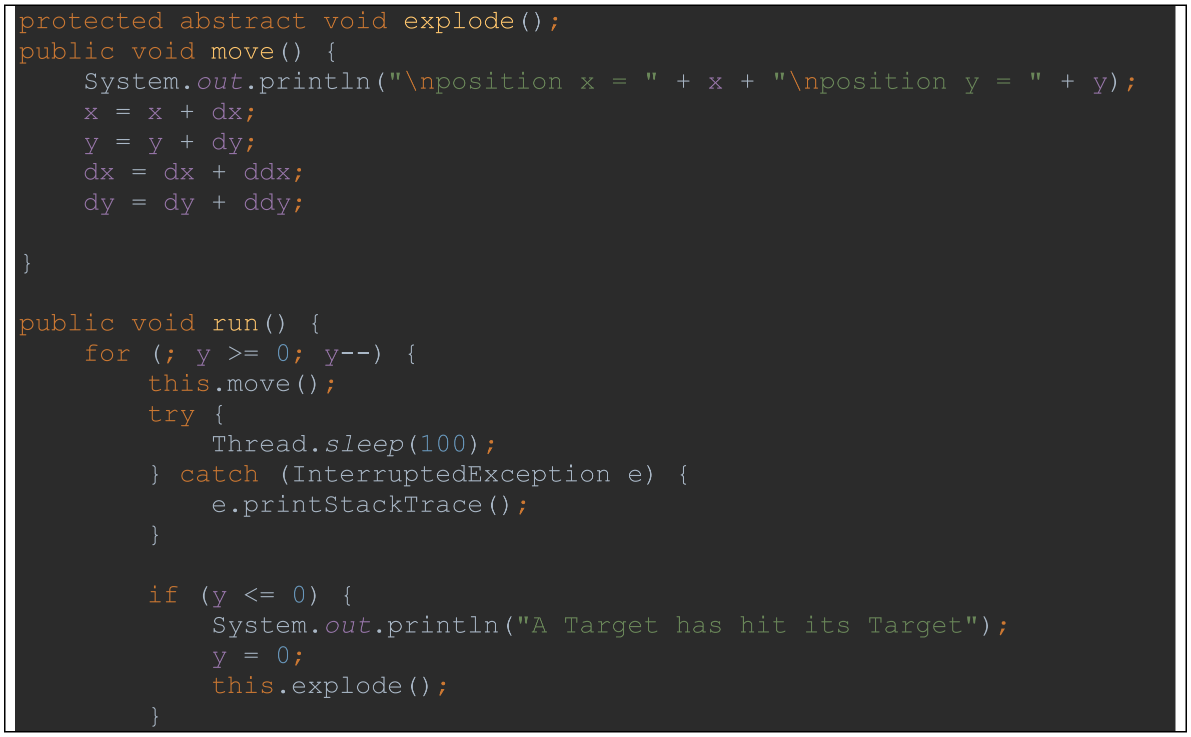
The following table shows the sample the objects being displayed.

Table 4: Sample Output



3.1.3. Design of Threads

Here I will show how I am implementing threads. This will help with being able to show movement of my objects and check to see if the condition of explode is met.

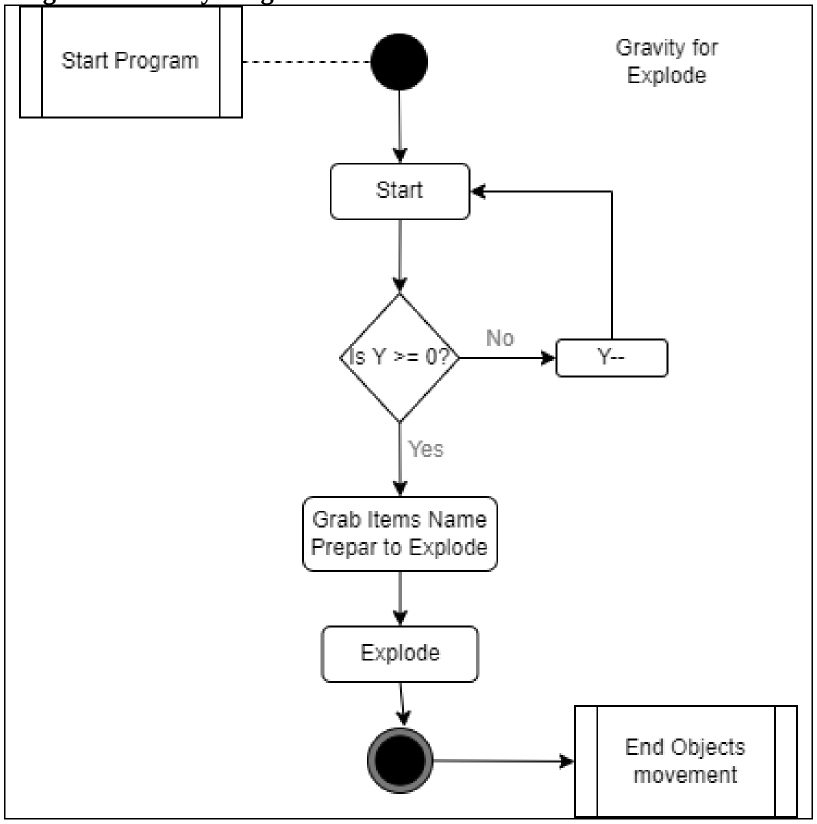


```java
protected abstract void explode();
public void move() {
    System.out.println("\nposition x = " + x + "\nposition y = " + y);
    x = x + dx;
    y = y + dy;
    dx = dx + ddx;
    dy = dy + ddy;

}

public void run() {
    for (; y >= 0; y--) {
        this.move();
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        if (y <= 0) {
            System.out.println("A Target has hit its Target");
            y = 0;
            this.explode();
        }
```

3.1.4. Activity Diagram for Gravity

This is the way I designed my program to run and call upon what it needs and what it does.

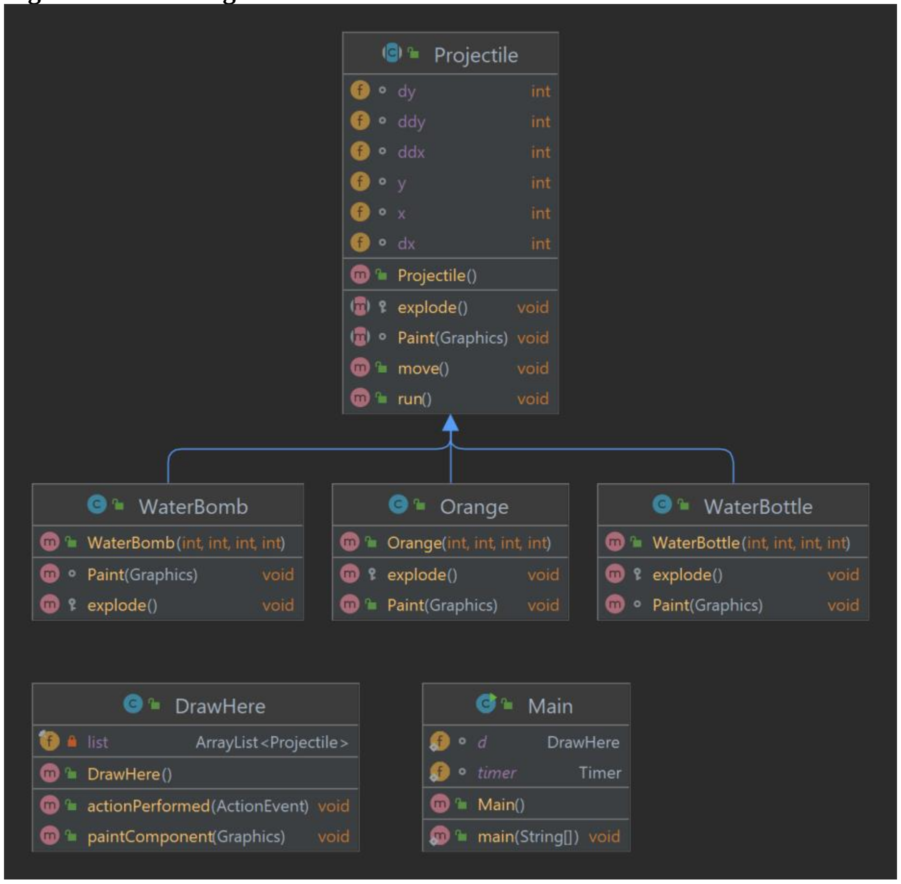
Figure 1:Activity Diagram



3.2. Design of my Classes

This here is my class diagram that is showing its dependences and all the other useful information. From this you can see how I have set up my project. Refer to Figure 2: Class Diagram

Figure 2: Class Diagram



Class Exercise

4.1 Review of Concepts

During this part of the document, I will be providing explanation of a list of object-oriented related terms with supplying code snippets of example of where I applied this.

<u>List</u>

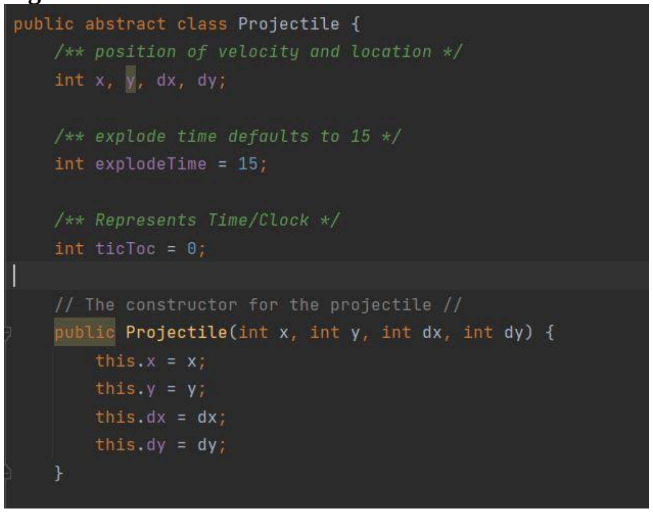
Abstract Classes:
Derived classes:
Constructor:
Inheritance:
Polymorphism:
Overriding methods:
Java Keyword abstract:
Java Keywords private, public, and protected:
~~Enumerations:~~

Exceptions (try-catch-finally) and throw: throws
Arrays or array list: show several display objects using an array.
[50%] You must have an example of each of the above...in your own code. If you don't have such examples, update your code. [Optionally, you may have an array list of Projectiles and put one of each projectile subclass...orange, water bottle, waterbomb, ...]

4.2 List of terms with examples.

<u>Abstract Classes</u>: *A method which is declared as abstract and does not have implementation is known as an abstract method.*

Figure 3: Abstract Classes

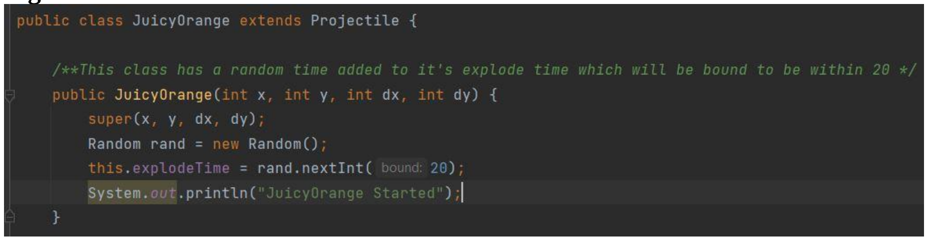


```
public abstract class Projectile {
    /** position of velocity and location */
    int x, y, dx, dy;

    /** explode time defaults to 15 */
    int explodeTime = 15;

    /** Represents Time/Clock */
    int ticToc = 0;

    // The constructor for the projectile //
    public Projectile(int x, int y, int dx, int dy) {
        this.x = x;
        this.y = y;
        this.dx = dx;
        this.dy = dy;
    }
```

Class Exercise

<u>Derived Classes</u>: *A class can be derived from the base class in Java by using a keyword. This keyword is basically used to indicate that a new class is derived from a class using inheritance.*

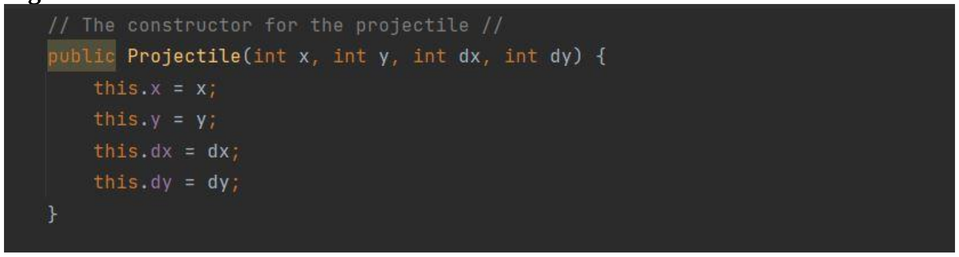
Figure 4: Derived classes



```java
public class JuicyOrange extends Projectile {

    /**This class has a random time added to it's explode time which will be bound to be within 20 */
    public JuicyOrange(int x, int y, int dx, int dy) {
        super(x, y, dx, dy);
        Random rand = new Random();
        this.explodeTime = rand.nextInt( bound: 20);
        System.out.println("JuicyOrange Started");
    }
```

<u>Constructor</u>: *A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.*

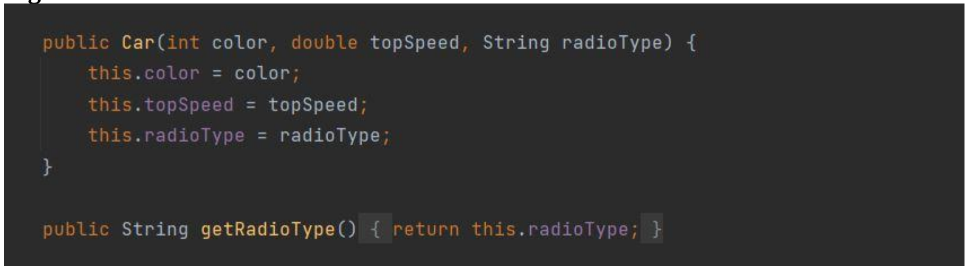
Figure 5: Constructor



```java
// The constructor for the projectile //
public Projectile(int x, int y, int dx, int dy) {
    this.x = x;
    this.y = y;
    this.dx = dx;
    this.dy = dy;
}
```

<u>Inheritance</u>: *In Java, it is possible to inherit attributes and methods from one class to another. Typical used in reference is parent and child classes.*

Figure 6: Inheritance

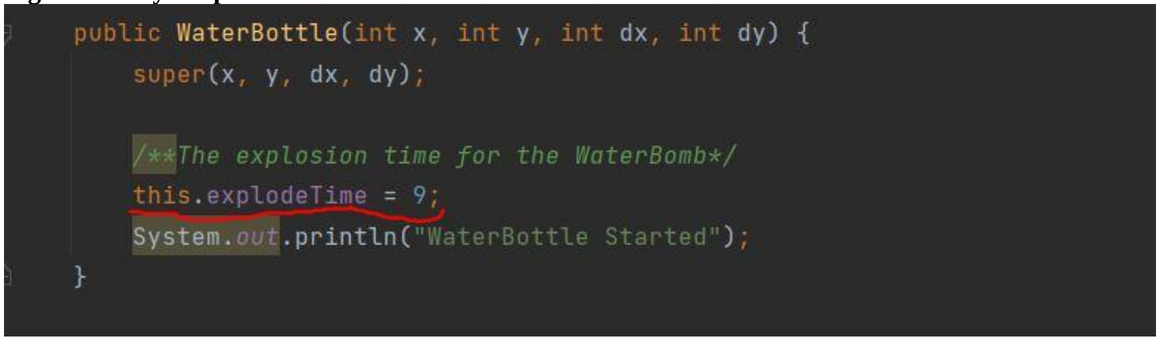


```java
public Car(int color, double topSpeed, String radioType) {
    this.color = color;
    this.topSpeed = topSpeed;
    this.radioType = radioType;
}

public String getRadioType() { return this.radioType; }
```

Class Exercise

Polymorphism: *Is the process of diving deeper into inheritance but calling the different methods which can be seen as subclass functions that use the parents name with the new method attached to the name.*

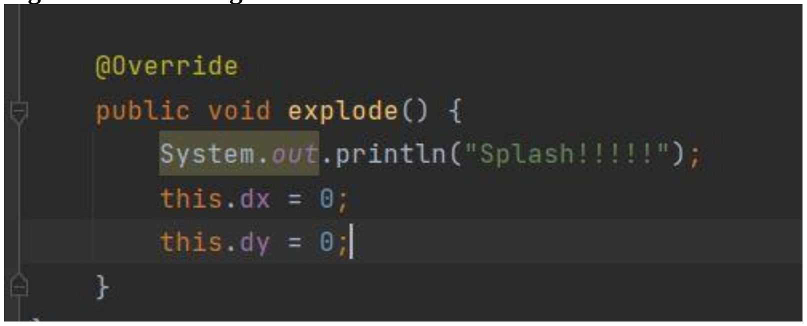
Figure 7: Polymorphism



```
public WaterBottle(int x, int y, int dx, int dy) {
    super(x, y, dx, dy);

    /**The explosion time for the WaterBomb*/
    this.explodeTime = 9;
    System.out.println("WaterBottle Started");
}
```

Overriding methods: *This is mostly used to override the inherited method from the parent class but to do something different like return different outputs/information.*

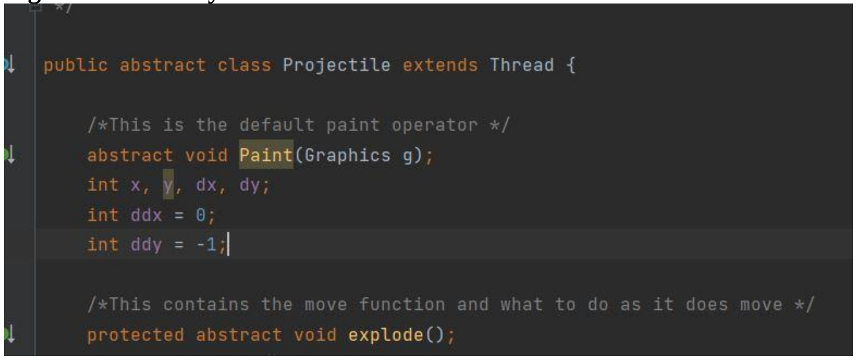
Figure 8: Overriding methods



```
@Override
public void explode() {
    System.out.println("Splash!!!!!");
    this.dx = 0;
    this.dy = 0;
}
```

Java Keyword abstract: *The abstract keyword is used to achieve abstraction in Java. It is a non-access modifier which is used to create abstract class and method. Which can get confused with using s0ome of these keywords are final, private, static.*

Figure 9: Java Keyword abstract



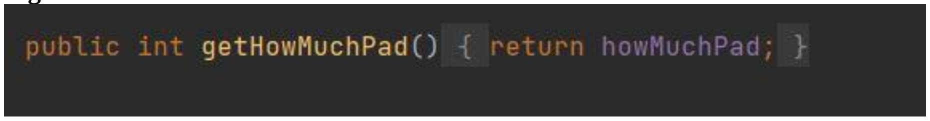
```
public abstract class Projectile extends Thread {

    /*This is the default paint operator */
    abstract void Paint(Graphics g);
    int x, y, dx, dy;
    int ddx = 0;
    int ddy = -1;

    /*This contains the move function and what to do as it does move */
    protected abstract void explode();
```

Class Exercise

Java Keywords private, public, and protected:

Public- The code is accessible for all classes.

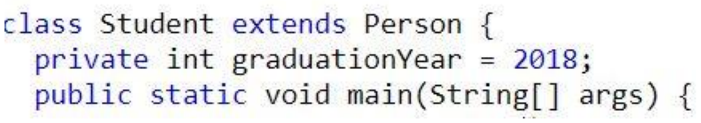
Figure 10: Public



```java
public int getHowMuchPad() { return howMuchPad; }
```

Private- The code is only accessible within the declared class.

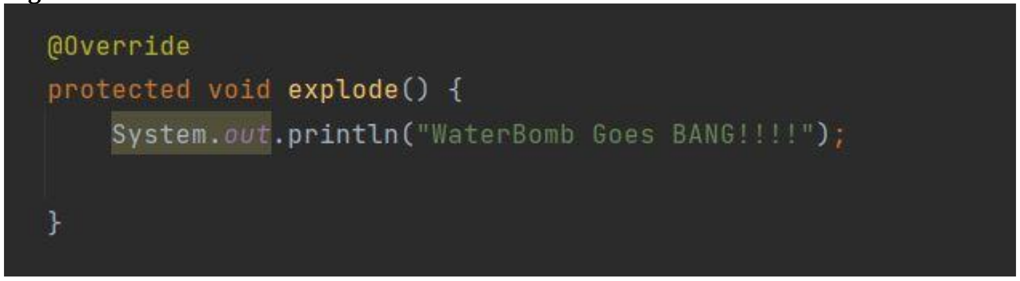
Figure 11: Private



```java
class Student extends Person {
  private int graduationYear = 2018;
  public static void main(String[] args) {
```

Protected- The code is accessible in the same package and subclasses. You will learn more about subclasses and super classes in the Inheritance chapter

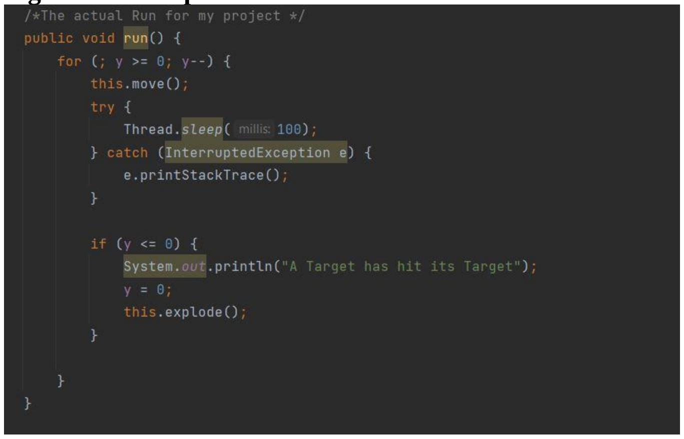
Figure 12: Protected



```java
@Override
protected void explode() {
    System.out.println("WaterBomb Goes BANG!!!!");

}
```

Exceptions: An exception is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted, and the application will do if it is meet and do something else.

Figure 13: Exceptions

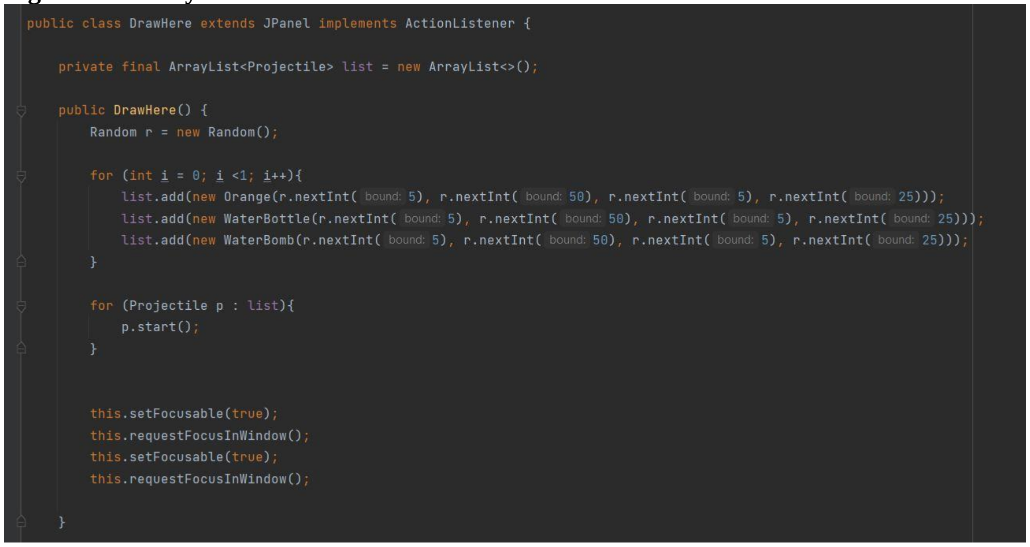


```java
/*The actual Run for my project */
public void run() {
    for (; y >= 0; y--) {
        this.move();
        try {
            Thread.sleep( millis: 100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        if (y <= 0) {
            System.out.println("A Target has hit its Target");
            y = 0;
            this.explode();
        }

    }
}
```

Class Exercise

<u>Arrays-lists</u>: *Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.*

Figure 14: Array-lists



```java
public class DrawHere extends JPanel implements ActionListener {

    private final ArrayList<Projectile> list = new ArrayList<>();

    public DrawHere() {
        Random r = new Random();

        for (int i = 0; i <1; i++){
            list.add(new Orange(r.nextInt( bound: 5), r.nextInt( bound: 50), r.nextInt( bound: 5), r.nextInt( bound: 25)));
            list.add(new WaterBottle(r.nextInt( bound: 5), r.nextInt( bound: 50), r.nextInt( bound: 5), r.nextInt( bound: 25)));
            list.add(new WaterBomb(r.nextInt( bound: 5), r.nextInt( bound: 50), r.nextInt( bound: 5), r.nextInt( bound: 25)));
        }

        for (Projectile p : list){
            p.start();
        }


        this.setFocusable(true);
        this.requestFocusInWindow();
        this.setFocusable(true);
        this.requestFocusInWindow();

    }
```

Requirements Traceability

Introduce the appendix and what it contains.

Paragraph Defined	Requirements Text	Paragraph Implemented	Paragraph Tested
4.2	List of terms	4.2	
4.2	Examples figures 3-14	4.2	