Project Documentation For Projectile Simulator

Author: Ka Tin (June) Man

Date: Feb 17, 2022

Project document for Projectile Simulator

Table of Contents

List of Tables

List of Figures

Project document for Projectile Simulator

1. Project Overview

This Java program will demonstrate classes, inheritance, and threads. Two objects class were used to simulate the motion of projectile with physical logic.

2. Project Requirements

Animation – Projectile Simulator

This project need to fulfill below requirements

1. The projectile class include x-y position with movement
2. The movement of projectile should simulate the gravity.
3. The Y position of the projectile is decremented to simulate gravity
4. The projectiles are shooting from one side of the screen to the other.
5. Need to have an abstract projectile class with abstract explode method
6. Launch a derived type from projectile.
7. Print the explode position when the projectile hit the ground (y=0). Print the projectile's x-y coordinates until it lands/explodes.
8. Make the Threads and let the projectile move independently of the mainline.
9. Projectile as a 2D graphic display on the screen.
10. A module displays and instantiates serval projectiles.

3. Design Plans

3.1 No Display Version

3.1.1. Main Projectile Design

Use "courier new" font (or some other fixed-spaced font) inside a 1x1 table to display short segments of logs or scripts.

Table 1: Main Line Program

```
public static void main(String[] args) {

      // Setup a random function for power up
      Random r = new Random();

      // Thread -0 waterboom run
      waterboom waterboom = new waterboom(1, 1, r.nextInt(10), 
r.nextInt(10));

      waterboom.setName("Water Monster!!");
      System.out.println("Thread 0 name: " + waterboom.getName());
      waterboom.start();

      // Thread -1 Orange run
      orangeboom orangeboom = new orangeboom(1, 1, r.nextInt(10), 
r.nextInt(10));

      orangeboom.setName("Orange Monster!!");
      System.out.println("Thread 1 name: " + orangeboom.getName());
      orangeboom.start();

      // loop 10 count to print out each thread
      for (int i = 0; i < 10; i++) {

            try {
                  Thread.sleep(500);}
            catch (InterruptedException e) {
                  e.printStackTrace();

                  }

      System.out.println(waterboom.getName() + waterboom.toString());
      System.out.println(orangeboom.getName() + orangeboom.toString());
      System.out.println("\n");

}

      }
```

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

Table 2: Sample Output - main

```
Thread 0 name: Water Monster!!
Thread start
Thread 1 name: Orange Monster!!
Thread start
Water Monster!!  x= 10 | y= 9 | dx= 9 | dy= 7
Orange Monster!!  x= 10 | y= 3 | dx= 9 | dy= 1


Water Monster!!  x= 19 | y= 16 | dx= 9 | dy= 6
Orange Monster!!  x= 19 | y= 4 | dx= 9 | dy= 0


Water Monster!!  x= 28 | y= 22 | dx= 9 | dy= 5
Orange Monster!!  x= 28 | y= 4 | dx= 9 | dy= -1


Water Monster!!  x= 37 | y= 27 | dx= 9 | dy= 4
Orange Monster!!  x= 37 | y= 3 | dx= 9 | dy= -2


Water Monster!!  x= 46 | y= 31 | dx= 9 | dy= 3
Orange Monster!!  x= 46 | y= 1 | dx= 9 | dy= -3


ORANGE MONSTER EXPLODE!!!!!!!!!

Water Monster!!  x= 55 | y= 34 | dx= 9 | dy= 2
Orange Monster!!  x= 55 | y= -2 | dx= 9 | dy= -4


Water Monster!!  x= 64 | y= 36 | dx= 9 | dy= 1
Orange Monster!!  x= 55 | y= -2 | dx= 9 | dy= -4


Water Monster!!  x= 73 | y= 37 | dx= 9 | dy= 0
Orange Monster!!  x= 55 | y= -2 | dx= 9 | dy= -4


Water Monster!!  x= 82 | y= 37 | dx= 9 | dy= -1
Orange Monster!!  x= 55 | y= -2 | dx= 9 | dy= -4


Water Monster!!  x= 91 | y= 36 | dx= 9 | dy= -2
Orange Monster!!  x= 55 | y= -2 | dx= 9 | dy= -4


WATER MONSTER EXPLODE!!!!!!!!!
```

3.1.2. Design of Abstract Class

The following table shows the abstract class and other public class

Table 3: Abstract Class – projectileNew

```
package projectile_testing;

public abstract class projectileNew extends Thread {

      int x, y;
      static int dx;
      int dy = 0;
      int gravity = -1;

      String print;

      public projectileNew(int x, int y, int dx, int dy) {
            this.x = x;
            this.y = y;
            this.dx = dx;
            this.dy = dy;

      }

      public void run() {

            System.out.println("Thread start");

            while (y >= 0) {

                  try {
                        Thread.sleep(500);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }

                  this.x = this.x + this.dx;
                  this.y = this.y + this.dy;

                  this.dy = this.dy - 1;

            }

            explode();
      }

      abstract void explode();

      public String toString() {
            return "  x= " + this.x + " | " + "y= " + this.y + " | " + "dx= "
+   this.dx + " | " + "dy= " + this.dy;
```

3.1.3. Design of Public Class

Table 4: Public class – waterboom

```
package projectile_testing;

public class waterboom extends projectileNew {

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

      }

      @Override
      public void explode() {
            System.out.println("WATER MONSTER EXPLODE!!!!!!!!! \n");
```

Table 5: Public class – orangeboom

```
package projectile_testing;

public class orangeboom extends projectileNew{

      //int explodeTime = 4;
      //int tic = 0;

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

      }

      @Override
      public void explode() {

            //if (this.y <= explodeTime) {

            System.out.println("ORANGE MONSTER EXPLODE!!!!!!!!! \n");

            }
      }
```

3.1.4. Design of Thread

Table 6: Thread

```
public void run() {

            System.out.println("Thread start");

            while (y >= 0) {

                  try {
                        Thread.sleep(500);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }

                  this.x = this.x + this.dx;
                  this.y = this.y + this.dy;

                  this.dy = this.dy - 1;

            }

            explode();
      }

      abstract void explode();
```

3.2 With Display Version

3.2.1 Design of Abstract Class

Table 7: Abstract Class - projectile

```
package projectile_testing;

import java.awt.Color;
import java.awt.Graphics2D;

public abstract class projectile extends Thread {
      /**
       * projectile will start writing on the page (g) using the text 
in the given
       * buffer. projectile set as a abstract class
       */

      int x = 0;
      int y = 0;
      int dx = 0;
      int dy = 0;
// Acceleration, ddy is gravity
      int ddx = 0;
      int ddy = -1;

      String projectileName;
      Color projectileColor;

      public projectile(String theText, int x, int y, int dx, int dy, 
Color projectileColor) {
            this.projectileName = theText;
            this.x = x;
            this.y = y;
            this.dx = dx;
            this.dy = dy;
            this.projectileColor = projectileColor;

      }

      public void run() {

            System.out.println("Thread start");

            while (y >= 0) {

                  try {
                        Thread.sleep(100);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }

                  this.x = this.x + this.dx;
                  this.y = this.y + this.dy;
                  //simulation of the gravity
                  this.dy = this.dy - 1;

                  System.out.println("x= " + x + " | " + "y= " + y + "
| " + "dx= " + dx + " | " + "dy= " + dy);

            }

      }

      // Draw the projectile
      public void Paint(Graphics2D g2d) {
            g2d.setColor(projectileColor);
            g2d.fillOval(x, y, 30, 30);

      }
}
```

3.2.2 Design of Thread Relationship

Table 8: Relationship - object.start( ) <=> extends Thread <=>Thread Run() method

```
1. In JPanel object.start() the thread

public class DrawPanel extends JPanel implements ActionListener {

      orangeboom o1 = new orangeboom("orange 1", 0, 50, 5, -8,
Color.ORANGE);
      waterboom w1 = new waterboom("water 2", 0, 20, 2, -8,
Color.BLUE);


      public DrawPanel() {
            o1.start();
            w1.start();

      <snip>

      }
}


=======================================================================
```

2. object (o1,w1) extend abstract class (projectile)> extend Thread

```
public abstract class projectile extends Thread {

     <snip>
     <snip>
```

3. Thread run() method inside the abstract class

```
      public void run() {

            while (y <= 450) {

                  try {
                        Thread.sleep(100);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  } finally {
                        System.out.println("Ready!!");
                  }


                  this.x = this.x + this.dx;
                  this.y = this.y + this.dy;
                  //simulation of the gravity
                  this.dy = this.dy + 1;

                  System.out.println("Projectile emission==>   " +
projectiles[0] + "  |  " + projectiles[1]);
                  System.out.println("x= " + x + " | " + "y= " + y + "
| " + "dx= " + dx + " | " + "dy= " + dy);

            }

            explode();

      }
```

4. The cycle return to object.start()

3.2.3 Design of Action

Table 9: Relationship - timer <=> ActionListener <=> ActionPerformed() <=> repaint() <=>PaintComponent

1. The timer in mainline, setup a timer as trigger for drawing the frames, Timers are constructed by specifying both a **delay parameter** and an **ActionListener**. Use the **start()** method to start the timer,

```
public class mainpanel_projectile {

      static Timer timer;

      static DrawPanel d = new DrawPanel();

      public static void main(String[] args) {

            <snip>

            // Listener
            timer = new Timer(10, d);
            timer.setInitialDelay(10);
            timer.setCoalesce(true);
            timer.start();

      }
}
```

==

2. **ActionListener** is used to receive action event and receive the message from the **ActionPeformed.**

```
public class DrawPanel extends JPanel implements ActionListener {

      <snip>

public void paintComponent(Graphics g) {

            BufferedImage bufferedImage = new 
BufferedImage(this.getWidth(),
                        this.getHeight(), BufferedImage.TYPE_INT_BGR);
            //Graphic display
            Graphics2D g2d = bufferedImage.createGraphics();
            g2d.setColor(getBackground());
            g2d.fillRect(0, 0, this.getWidth(), this.getHeight());

            o1.Paint(g2d);
            w1.Paint(g2d);
            Graphics2D g2dComponent = (Graphics2D) g;
            g2dComponent.drawImage(bufferedImage, null, 0, 0);

      }
====================================================================
```

3. **ActionPerformed** is used to perform the this.repaint() to re-paint the **paintcomponent** and send the message to ActionListener, then ActionListener connect with the **timer** inside the mainline to run the loop and show the graphic on JFrame.

```
@Override
      public void actionPerformed(ActionEvent e) {
            this.repaint();
      }
```

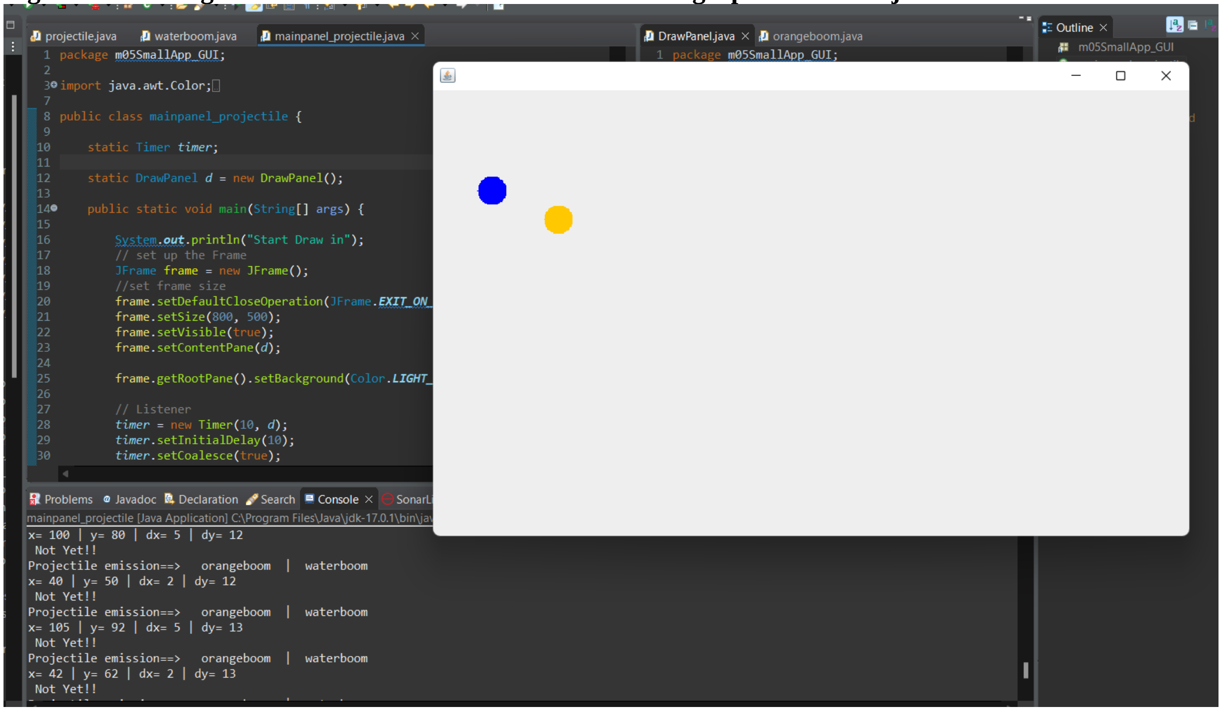
Table 10: The use of bufferedImage inside DrawPanel.paintComponent()

BufferedImage is used to describes an Image that has an image data buffer that can be accessed. Upper left corner coordinate of all BufferedImage objects is (0, 0). As a result, each Raster used to create a BufferedImage must have minX=0 and minY=0.

```
public class DrawPanel extends JPanel implements ActionListener {

      <snip>

public void paintComponent(Graphics g) {

            BufferedImage bufferedImage = new
BufferedImage(this.getWidth(),
                        this.getHeight(), BufferedImage.TYPE_INT_BGR);
            //Graphic display
            Graphics2D g2d = bufferedImage.createGraphics();
            g2d.setColor(getBackground());
            g2d.fillRect(0, 0, this.getWidth(), this.getHeight());

            o1.Paint(g2d);
            w1.Paint(g2d);


            Graphics2D g2dComponent = (Graphics2D) g;
            g2dComponent.drawImage(bufferedImage, null, 0, 0);

      }
```

3.2.4. Running Output

Figure 1: Running the main class than the JFrame draw the graphic of two object class



```
package m05SmallApp_GUI;

import java.awt.Color;

public class mainpanel_projectile {

    static Timer timer;

    static DrawPanel d = new DrawPanel();

    public static void main(String[] args) {

        System.out.println("Start Draw in");
        // set up the Frame
        JFrame frame = new JFrame();
        //set frame size
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_
        frame.setSize(800, 500);
        frame.setVisible(true);
        frame.setContentPane(d);

        frame.getRootPane().setBackground(Color.LIGHT_

        // Listener
        timer = new Timer(10, d);
        timer.setInitialDelay(10);
        timer.setCoalesce(true);
```

```
package m05SmallApp_GUI;
```

mainpanel_projectile [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\jav

```
x= 100 | y= 80 | dx= 5 | dy= 12
 Not Yet!!
Projectile emission==>   orangeboom  |  waterboom
x= 40 | y= 50 | dx= 2 | dy= 12
 Not Yet!!
Projectile emission==>   orangeboom  |  waterboom
x= 105 | y= 92 | dx= 5 | dy= 13
 Not Yet!!
Projectile emission==>   orangeboom  |  waterboom
x= 42 | y= 62 | dx= 2 | dy= 13
 Not Yet!!
```

4. Diagram

4.1. Activity Diagram

This session is including the active diagram with no display version and the display interface version

Figure 2:Activity Diagram – no display version (with explode)

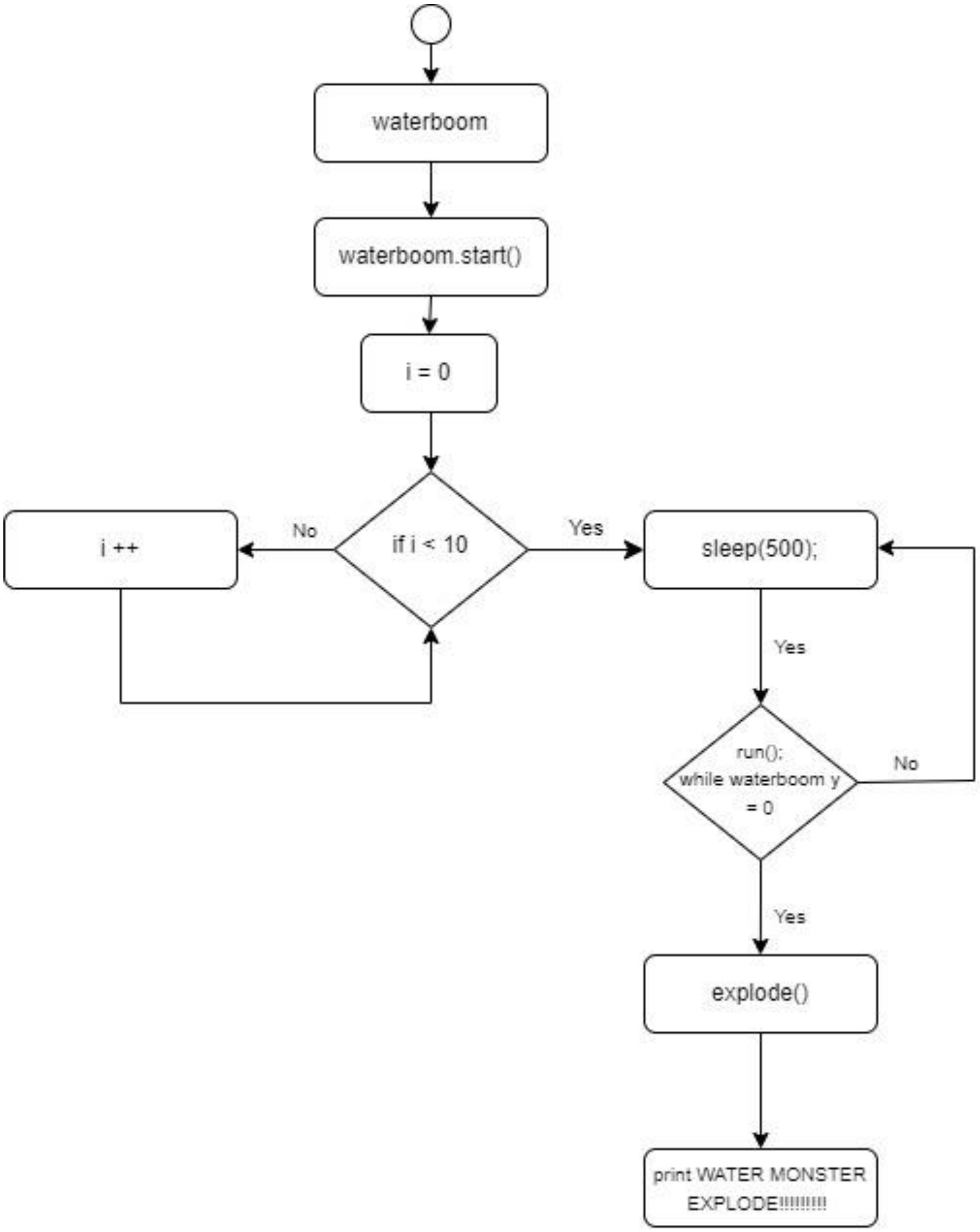
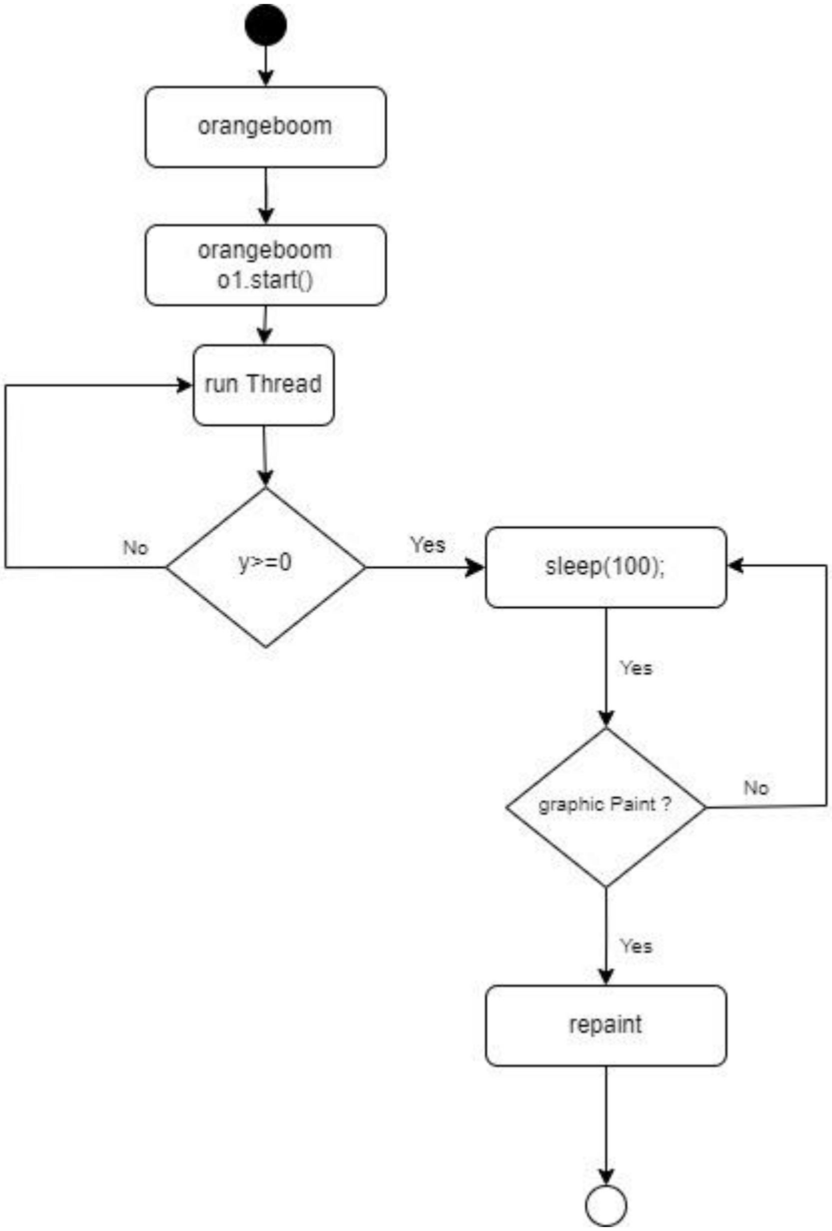
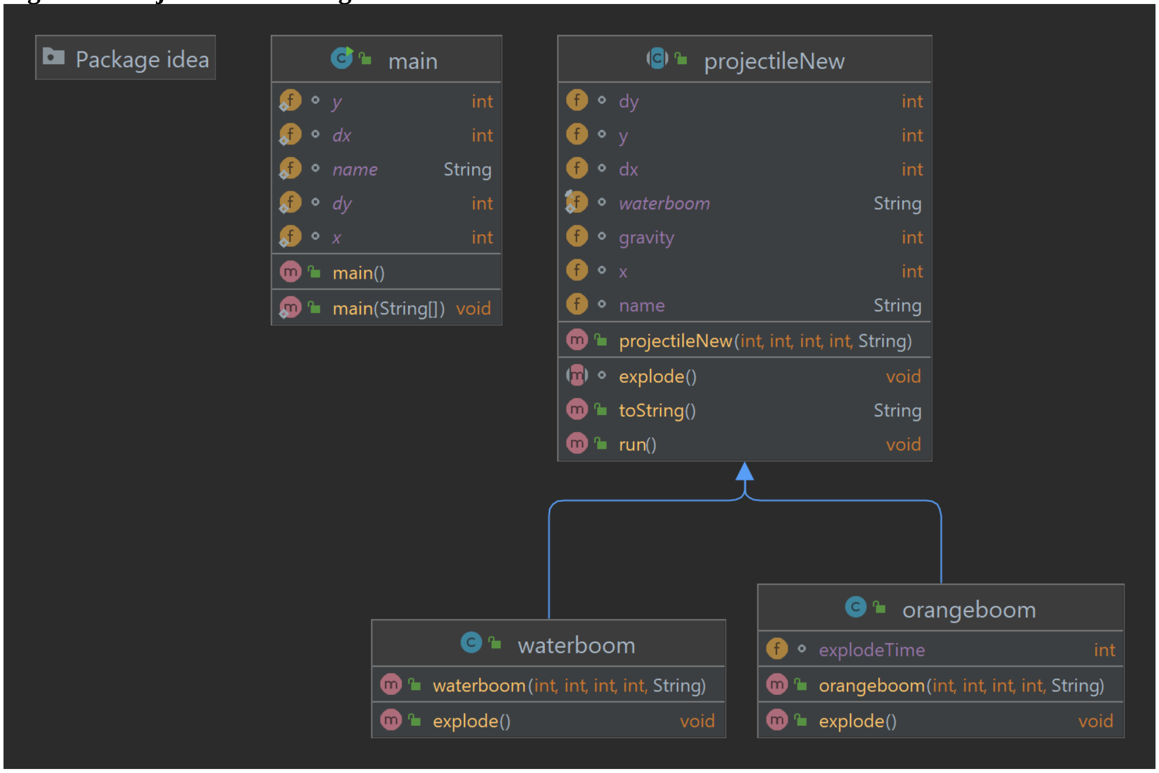


Figure 3: Activity Diagram – Display version, draw the graphic on JFrame



4.1.2. Class Diagram

Figure 4: Projectile Class Diagram



Appendix A. Requirements Traceability

Paragraph Defined	Requirements Text	Paragraph Implemented	Paragraph Tested
Paragraph 2	The projectile class include x-y position with movement	N/A	N/A
Paragraph 2	The movement of projectile should simulate the gravity.	N/A	N/A
Paragraph 2	The Y position of the projectile is decremented to simulate gravity	N/A	N/A
Paragraph 2	The projectiles are shooting from one side of the screen to the other.	N/A	N/A
Paragraph 2	Need to have an abstract projectile class with abstract explode method	N/A	N/A
Paragraph 2	Launch a derived type from projectile.	N/A	N/A
Paragraph 2	Print the explode position when the projectile hit the ground (y=0). Print the projectile's x-y coordinates until it lands/explodes.	N/A	N/A
Paragraph 2	Make the Threads and let the projectile move independently of the mainline.	N/A	N/A
Paragraph 2	Projectile as a 2D graphic display on the screen.	N/A	N/A
Paragraph 2	A module displays and instantiates serval projectiles.	N/A	N/A

Appendix B. Review of Concept

This appendix is a review of concept which are used in this project

1. Concept: Abstract Classes
   Used in: projectile class
   Description: This concept was used to achieve the security of projectile. The projectile needs to be inherited from other class. With this concept, it will be easier to discover and fix the error without affecting other classes.

```
public abstract class projectile extends Thread {

      private int x = 0;
      private int y = 0;
      private int dx = 0;
      private int dy = 0;
// Acceleration, ddy is gravity
      private int ddx = 0;
      private int ddy = -1;

      protected int explodeTime = 400;
      protected String projectileName;
      protected Color projectileColor;


      public projectile(String theText, int x, int y, int dx, int dy,
Color projectileColor) {
            this.projectileName = theText;
            this.x = x;
            this.y = y;
            this.dx = dx;
            this.dy = dy;
            this.projectileColor = projectileColor;

      }
}
```

2. Concept: Derived Classes
   Used in: waterboom class / orangeboom class
   Description: This concept was used to make the subclass inherit from the super class with the *extends* keyword.
   e.g., public class waterboom (subclass) extends projectile (super class)

```
public class waterboom extends projectile {

      public waterboom(String theText, int x, int y, int dx, int dy, 
Color projectileColor) {
            super(theText,x, y, dx, dy, projectileColor);

            this.explodeTime = 400;

      }
```

```
public class orangeboom extends projectile{

            public orangeboom (String theText, int x, int y, int dx, 
int dy, Color projectileColor) {
            super (theText,x,y,dx,dy,projectileColor);

            this.explodeTime = 400;

      }
```

3. Concept: Constructor
   Used in: `mainpanel_projectile class`
   Description: This concept was used to initialize objects and the constructor is called when an object of the mainpanel_projectile class is created.

```
public class mainpanel_projectile {

      static Timer timer;

      static DrawPanel d = new DrawPanel();

      String DrawPanelName;

      public mainpanel_projectile() {
            DrawPanelName = "Girl with an orange boom and a water boom 
wants to play with you!";
      }

      public static void main(String[] args) {

            // Create an object of class mainpanel_projectilem, this 
will  call the constructor
            mainpanel_projectile dname = new mainpanel_projectile();

            System.out.println(dname.DrawPanelName);

            <snip>
}
```

4. Concept: Inheritance
   Used in: projectile class/ waterboom class / orangeboom class/DrawPanel class
   Description: This concept was used to inherit the attributes and the methods from one class to another class by using the extends keyword.
   e.g., projectile class >>>> waterboom class / orangeboom class
   JPanel >>>> DrawPanel

```
public class waterboom extends projectile {

               <snip>
      }
```

```
public class orangeboom extends projectile{

                <snip>
      }
```

```
public class DrawPanel extends JPanel implements ActionListener {

                 <snip>
}
```

5. Concept: Polymorphism
   Used in: waterboom class / orangeboom class
   Description: This concept was used to perform different method under the inherit from the super class, for example, in the orangeboom class, it will perform a explode() method with the different print line from the projectile class.

```
public class orangeboom extends projectile{

            public orangeboom (String theText, int x, int y, int dx, 
int dy, Color projectileColor) {
            super (theText,x,y,dx,dy,projectileColor);

            this.explodeTime = 400;

      }

      @Override
      public void explode() {
            // This is polymorphism
            System.out.println("ORANGE MONSTER EXPLODE!!!!!!!!! \n");


      }
}
```

6. Concept: Override method
   Used in: waterboom class / orangeboom class
   Description: This concept was used to override or implement a method declared in a supertype. For example, In the orangeboom class, use override method on explode(), to override the explode() method used in projectile class.

```
// In orangeboom Class
public class orangeboom extends projectile{

            <snip>

      }

      @Override
      public void explode() {
            // This is polymorphism
            System.out.println("ORANGE MONSTER EXPLODE!!!!!!!!! \n");


      }
}


// The method declared in projectile

abstract void explode();
```

7. Java Keyword: abstract
   Used in: projectile class
   Description: This keyword is used to identify a non-access modifier. Abstract keyword can used to assign the abstract class and the abstract method.

```
// Abstract Class
public abstract class projectile extends Thread {

// Abstract Method
      abstract void explode();
}
```

8. Java keyword: private, public, and protected
   Used in: projectile class, mainpanel_projectile class
   Description: Keyword of private is an access modifier, making the attributes, methods and constructors can only be accessible within the declared class. For example, in the projectile class, the attribute with private keyword can only accessible in the projectile class. Keyword of public is another access modifier to make the attributes, methods and constructors can be accessible by any other class. Keyword of protected is one of the access modifier to make the attributes, methods and constructors can be accessible in the same package and subclasses.

```
// Private keyword
public abstract class projectile extends Thread {

      private int x = 0;
      private int y = 0;
      private int dx = 0;
      private int dy = 0;
      private int ddx = 0;
      private int ddy = -1;

}

// Public keyword
public class mainpanel_projectile {

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

}

// Protected keyword
public abstract class projectile extends Thread {

      <snip>

      protected int explodeTime = 400;
      protected String projectileName;
      protected Color projectileColor;
```

9. Concept: Exceptions (try-catch-finally) and throw: throws

Used in: projectile class
Description: try statement and catch statement they come into a pair; this pair of statement is used to catch the exception. For example, in the run() method, when multiple thread start to run, this pair of statement is used to catch the interrupted exception in the while loop, if one of the thread out of the condition of while loop, the exception will appear and catch by the pair of statement and implement with the explode method, then the loop will continue to run and print the x,y,dx,dy coordinate until all the thread run and out of the condition.

```
public void run() {

            System.out.println("Thread start");

            while (y <= 450) {

             // Execptions
                  try {
                        Thread.sleep(100);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  } finally {
                        System.out.println(" Not Yet!!");
                  }


                  this.x = this.x + this.dx;
                  this.y = this.y + this.dy;
                  //simulation of the gravity
                  this.dy = this.dy + 1;

                  System.out.println("Projectile emission==>   " + 
projectiles[0] + "  |  " + projectiles[1]);
                  System.out.println("x= " + x + " | " + "y= " + y + " 
| " + "dx= " + dx + " | " + "dy= " + dy);

            }

            explode();

      }
```

10. Concept: Arrays or arraylist: show several display objects using an array
    Used in: projectile class
    Description: This concept is use to store multiple values in a single variable. For example, the projectiles array in the projectile class is used to store multiple name of the projectile.

```
public abstract class projectile extends Thread {
          <snip>
      private String[] projectiles = {"orangeboom", "waterboom"};

}
```