Even Semester (2023)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:

Surname:	**Given Name:**	**Student ID Number:**
Christoffer	Christoffer Raffaelo Wijaya	2602177051

Course Code : COMP6048001 **Course Name :** Data Structure

Class : L2CC **Lecturer :Jude Joseph Lamug Martinez, MCS**

Type of Assignments: Final Project

Submission Pattern

Due Date : 16 June 20223 **Submission Date :** 15 June 2023

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating, and collusion as academic offences which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Christoffer Raffaelo Wijaya

Table of Contents

Project Specification

Overview

This program is about a bomberman game. The goal of the game is to bomb the wall and prevent it from getting hurt by the enemies. These are only games that can be played by children. In this game, there are 5 enemies that can go horizontal while the bomberman can go either vertical or horizontal and put bombs to break the wall.

Program Output and Input

It has methods for initializing the grid, placing bombs, checking for valid positions, and printing the grid. The main method serves as the entry point to the program and handles the user input using a scanner project.You can compile and run the Java code to run the program. To position the bomb on the grid, input the X and Y coordinates when prompted. After each placement of a bomb, the application updates the grid and displays it.

Library Used

1. Swing
    - Used to make the GUI of the game.
2. Action Event
    - Used for clicking buttons which refers to action.
3. File
    - Used to retrieve elements from an external file.

Solution Design

Files that are involved are AbstractCharacter.java, AbstractPowerUp.java, Bomb.java, BombCounterPU.java, BombermanComponent.java, BombermanFrame.java, BombRadiusPU.java, Enemy.java, Engine.java, Explosion.java, Floor.java and Player.java.

AbstractCharacter.java

It represents the abstract base class for all characters in the game, such as players, enemies, or power-ups. It would define common properties and behaviors that are shared among these character types.

```
public class AbstractCharacter
    // character in game
{
    1 usage
    private final static int SIZE = 30;
    // size of character
    4 usages
    private int x;
    // x coordinate
    4 usages
    private int y;
    // y coordinate
    3 usages
    private int pixelsPerStep;
    // the number of the character walk per step

    2 usages
    protected AbstractCharacter(int x, int y, int pixelsPerStep) {
        // initializes the character's position (x and y) and the number of pixels it moves per step
    this.x = x;
    this.y = y;
    this.pixelsPerStep = pixelsPerStep;
    }

    // Enum to represent different move directions
    38 usages
    public enum Move
    {
    7 usages
    DOWN( deltaX: 0,  deltaY: 1),
    7 usages
    UP( deltaX: 0,  deltaY: -1),
    5 usages
    RIGHT( deltaX: 1,  deltaY: 0),
    7 usages
    LEFT( deltaX: -1,  deltaY: 0);
```

AbstractPowerUp.java

The majority of powerup types that extend AbstractPowerup derive from this class, and they all share its contents. BombCounterPU and BombRadiusPU are classes that extend this one. The x and y coordinates that will determine where the powerup will be placed are required by the constructor.

```java
public class AbstractPowerup
{
    // Constants are static by definition.
    1 usage
    private final static int POWERUP_SIZE = 30;
    // variable with a value of 30.
    // size of the power-up
    2 usages
    private final int x;
    // initial position
    2 usages
    private final int y;
    // initial position
    1 usage
    private String name = null;

    2 usages
    public AbstractPowerup(int x, int y) {
    this.x = x;
    this.y = y;
    }

    1 usage  2 overrides
    public void addToPlayer(Player player) {
    }

    2 usages
    public int getPowerupSize() { return POWERUP_SIZE; }

    2 usages
    public int getX() { return x; }

    2 usages
    public int getY() { return y; }

    2 usages  2 overrides
    public String getName() { return name; }
}
```

Bomb.java

It represents the objects that are placed on the game grid and explode after a certain time. The Bomb class has instance variables to represent the bomb's position (x and y), the remaining time on the bomb's timer (timer), and whether the bomb has exploded.

```
public class Bomb
    // bomb class
{
    // Constants are static by definition.
    1 usage
    private final static int BOMBSIZE = 30;
    // size of the bomb
    1 usage
    private final static int STARTCOUNTDOWN = 100;
    // countdown of 100
    2 usages
    private int timeToExplosion = STARTCOUNTDOWN;
    // time to explode
    2 usages
    private final int rowIndex;
    // return the row of the bomb's position
    2 usages
    private final int colIndex;
    // return the column of the bomb's position
    2 usages
    private int explosionRadius;
    // the bomb's explosion radius.
    3 usages
    private boolean playerLeft;
    // set to false
```

BombCounterPU.java

The BombCounterPU class could represent a power-up that increases the maximum number of bombs a player can place on the game grid. It would be a subclass of an AbstractPowerup class, which provides common functionality for power-ups in the game.This class extends AbstractPowerup and receives fundamental methods such as getters for its coordinates and size. This class has an addToPlayer-method which adjusts the bombCount of the player.

```
public class BombCounterPU extends AbstractPowerup
    //  inheriting its properties and methods.
{

    1 usage
    public BombCounterPU(int rowIndex, int colIndex) { super(colIndex, rowIndex); }
    // takes the row index and column index as arguments and calls the constructor of the superclass (AbstractPowerup)
    // using super(colIndex, rowIndex) to initialize the position of the power-up.

    1 usage
    public void addToPlayer(Player player) {
    int currentBombCount = player.getBombCount();
    // increases the bomb count of the player by one,
        // and updates it using the setBombCount method of the Player class.
    player.setBombCount(currentBombCount + 1);
    }

    2 usages
    public String getName() {
        // It returns the name of the power-up, which is set to "BombCounter".
    final String name = "BombCounter";
    return name;
    }
}
```

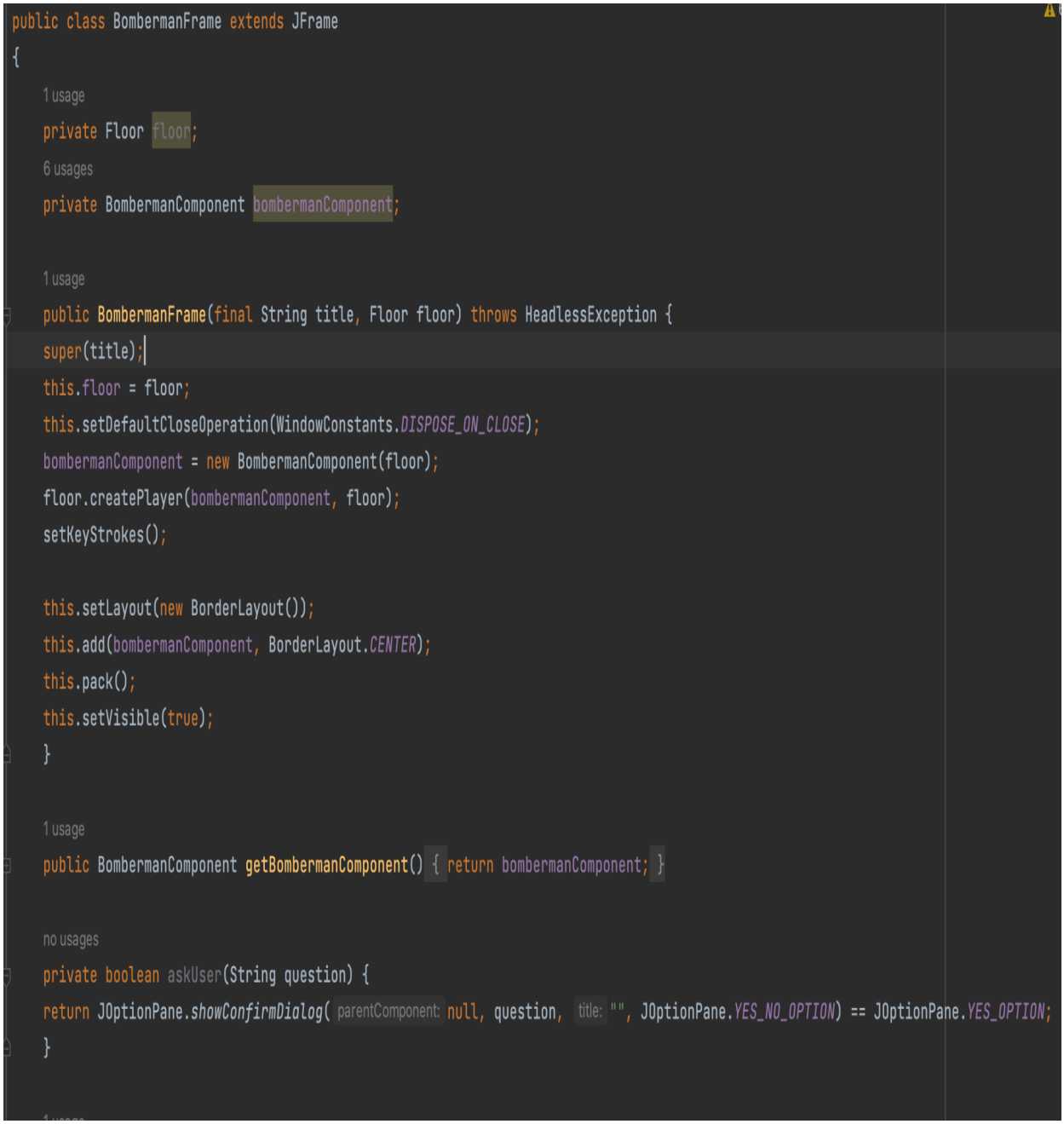
BombermanComponent.java

The BombermanComponent class could represent a graphical component responsible for rendering the game's grid and interacting with the user interface. It would be a subclass of JComponent and implement the FloorListener interface to receive notifications about changes in the game grid.

```
public class BombermanComponent extends JComponent implements FloorListener
{
    // Constants are static by definition.
    108 usages
    private final static int SQUARE_SIZE = 40;
    28 usages
    private final static int CHARACTER_ADJUSTMENT_FOR_PAINT = 15;
    25 usages
    private final static int SQUARE_MIDDLE = SQUARE_SIZE/2;
    6 usages
    private final static int BOMB_ADJUSTMENT_1 =5;
    2 usages
    private final static int BOMB_ADJUSTMENT_2 =10;
    // Defining painting parameters
    no usages
    private final static int PAINT_PARAMETER_13 = 13;
    3 usages
    private final static int PAINT_PARAMETER_15 = 15;
    no usages
    private final static int PAINT_PARAMETER_17 = 17;
    2 usages
    private final static int PAINT_PARAMETER_18 = 18;
    1 usage
    private final static int PAINT_PARAMETER_19 = 19;
    5 usages
    private final static int PAINT_PARAMETER_20 = 20;
    no usages
    private final static int PAINT_PARAMETER_24 = 24;
    17 usages
    private final Floor floor;
    5 usages
    private final AbstractMap<FloorTile, Color> colorMap;
```

BombermanFrame.java

It could represent the main graphical frame or window that contains the game components and provides the user interface. It would be a subclass of JFrame, which is a Swing class for creating windows.



```
public class BombermanFrame extends JFrame
{
    1 usage
    private Floor floor;
    6 usages
    private BombermanComponent bombermanComponent;

    1 usage
    public BombermanFrame(final String title, Floor floor) throws HeadlessException {
    super(title);
    this.floor = floor;
    this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    bombermanComponent = new BombermanComponent(floor);
    floor.createPlayer(bombermanComponent, floor);
    setKeyStrokes();

    this.setLayout(new BorderLayout());
    this.add(bombermanComponent, BorderLayout.CENTER);
    this.pack();
    this.setVisible(true);
    }

    1 usage
    public BombermanComponent getBombermanComponent() { return bombermanComponent; }

    no usages
    private boolean askUser(String question) {
    return JOptionPane.showConfirmDialog( parentComponent: null, question, title: "", JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION;
    }
```

BombRadiusPU.java

This class extends AbstractPowerup and receives fundamental methods such as getters for its coordinates and size. This class has an addToPlayer-method which adjusts the bombRadius of the player.

```java
public class BombRadiusPU extends AbstractPowerup
{

    1 usage
    public BombRadiusPU(int rowIndex, int colIndex) { super(colIndex, rowIndex); }

    1 usage
    public void addToPlayer(Player player) {
       // takes a Player object as a parameter and increases the player's explosion radius by 1.
        int currentExplosionRadius = player.getExplosionRadius();
    player.setExplosionRadius(currentExplosionRadius + 1);
    }

    2 usages
    public String getName() {
     //   returns the name of the power-up as a string. In this case, the name is "BombRadius".
    final String name = "BombRadius";
    return name;
    }
}
```

Enemy.java

The Enemy class could stand in for the game's foes or enemies. It would be a subclass of AbstractCharacter, inheriting all of its traits and characteristics.

```
public class Enemy extends AbstractCharacter
{
    9 usages
    private Move currentDirection;

    2 usages
    public Enemy(int x, int y, boolean vertical) {
        // new instance of the Enemy class
        super(x, y, pixelsPerStep: 1);
        // enemy moves vertically or horizontally.
        currentDirection = randomDirection(vertical);
    }

    2 usages
    public void changeDirection() {
        if (currentDirection == Move.DOWN) {
            currentDirection = Move.UP;
        } else if (currentDirection == Move.UP) {
            currentDirection = Move.DOWN;
        } else if (currentDirection == Move.LEFT) {
            currentDirection = Move.RIGHT;
        } else {
            currentDirection = Move.LEFT;
        }
    }
```

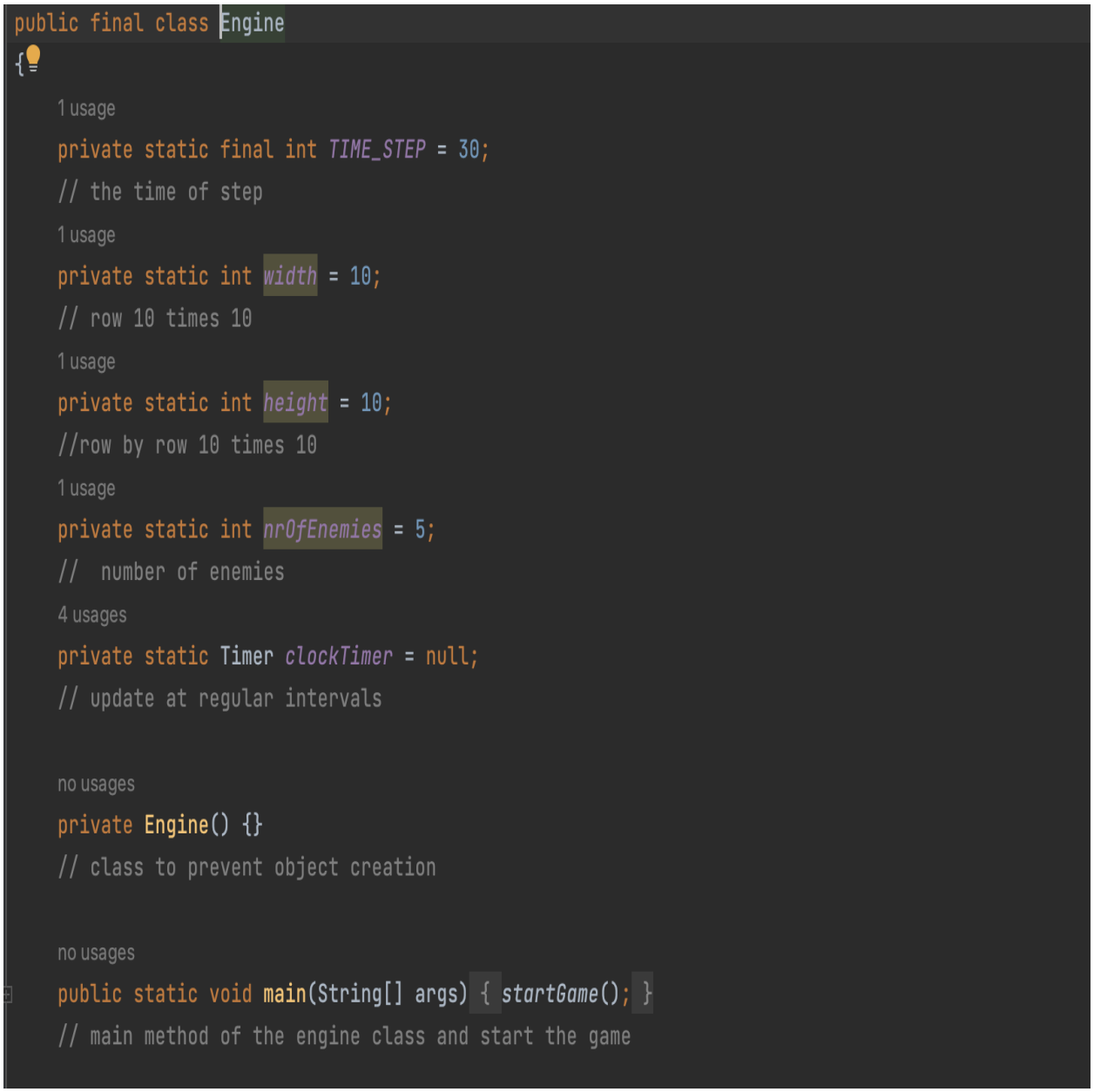
Engine.java

It could stand in for the game engine, which controls game logic and handles user input, game state updates, and coordination of interactions between various game pieces.

```java
public final class Engine
{
    1 usage
    private static final int TIME_STEP = 30;
    // the time of step
    1 usage
    private static int width = 10;
    // row 10 times 10
    1 usage
    private static int height = 10;
    //row by row 10 times 10
    1 usage
    private static int nrOfEnemies = 5;
    //  number of enemies
    4 usages
    private static Timer clockTimer = null;
    // update at regular intervals

    no usages
    private Engine() {}
    // class to prevent object creation

    no usages
    public static void main(String[] args) { startGame(); }
    // main method of the engine class and start the game
```

Explosion.java

This class represents the "fireballs" or explosions that have the power to demolish BREAKABLEBLOCKs and kill Players or Enemies. For logic and painting, it requires a row- and column-index. Its duration is the number of timesteps it will last before being eliminated.



```
public final class Engine
{
    1 usage
    private static final int TIME_STEP = 30;
    // the time of step
    1 usage
    private static int width = 10;
    // row 10 times 10
    1 usage
    private static int height = 10;
    //row by row 10 times 10
    1 usage
    private static int nrOfEnemies = 5;
    //  number of enemies
    4 usages
    private static Timer clockTimer = null;
    // update at regular intervals


    no usages
    private Engine() {}
    // class to prevent object creation


    no usages
    public static void main(String[] args) { startGame(); }
    // main method of the engine class and start the game
```

Floor.java

The game grid's individual tiles or cells could be represented by the Floor class. Normally, it would be a component of a bigger Grid class that controls the entire game board.

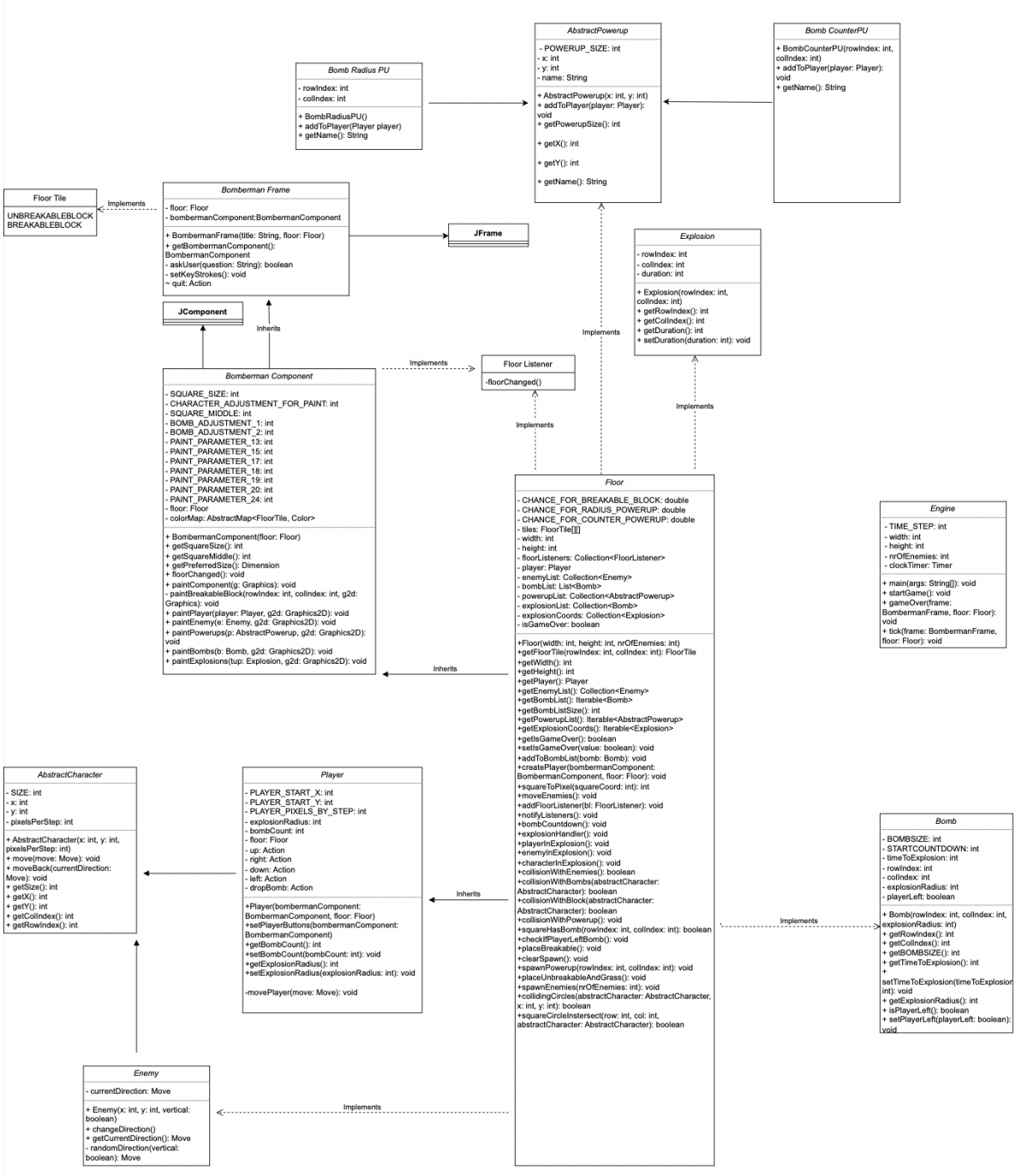
```
public class Floor {
    // Constants are static by definition.
    1 usage
    private final static double CHANCE_FOR_BREAKABLE_BLOCK = 0.4;
    1 usage
    private final static double CHANCE_FOR_RADIUS_POWERUP = 0.2;
    1 usage
    private final static double CHANCE_FOR_COUNTER_POWERUP = 0.8;
    13 usages
    private final FloorTile[][] tiles;
    8 usages
    private int width;
    8 usages
    private int height;
    2 usages
    private Collection<FloorListener> floorListeners = new ArrayList<>();
    7 usages
    private Player player = null;
    8 usages
    private Collection<Enemy> enemyList = new ArrayList<>();
    8 usages
    private List<Bomb> bombList= new ArrayList<>();
    5 usages
    private Collection<AbstractPowerup> powerupList = new ArrayList<>();
    3 usages
    private Collection<Bomb> explosionList= new ArrayList<>();
    7 usages
    private Collection<Explosion> explosionCoords= new ArrayList<>();
    5 usages
    private boolean isGameOver = false;
```

Player.java

The Player class could serve as the player's avatar or gaming character. Typically, it would be a subclass of AbstractCharacter, inheriting all of its common traits and behaviors.

```
public class Player extends AbstractCharacter
{

    1 usage
    private final static int PLAYER_START_X = 60;
    1 usage
    private final static int PLAYER_START_Y = 60;
    1 usage
    private final static int PLAYER_PIXELS_BY_STEP = 4;
    3 usages
    private int explosionRadius;
    3 usages
    private int bombCount;
    12 usages
    private Floor floor;

    1 usage
    public Action up = new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        movePlayer(Move.UP);

    }
    };

    1 usage
    public Action right = new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        movePlayer(Move.RIGHT);

    }
    };

    1 usage
    public Action down = new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        movePlayer(Move.DOWN);

    }
```

Class Diagram



Evidence of working program