University of Nottingham

Developing maintainable software

Course Work 03 - Part 01

Table of Contents

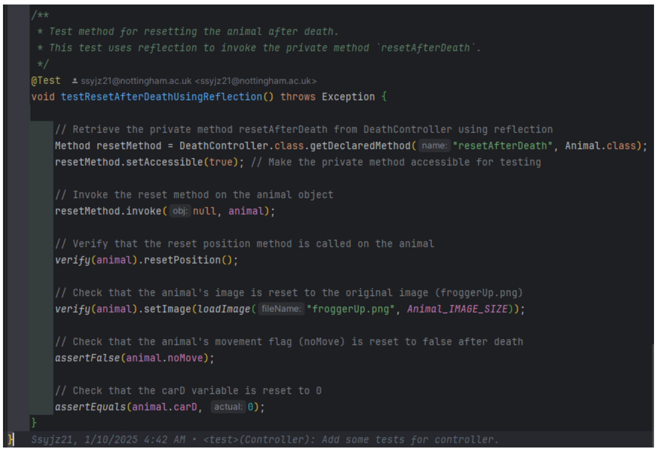
Jirui Zhang (20513930)

1-12-2025

Test

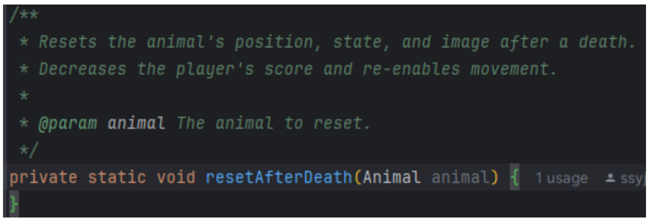
Example

- ### Initial Test



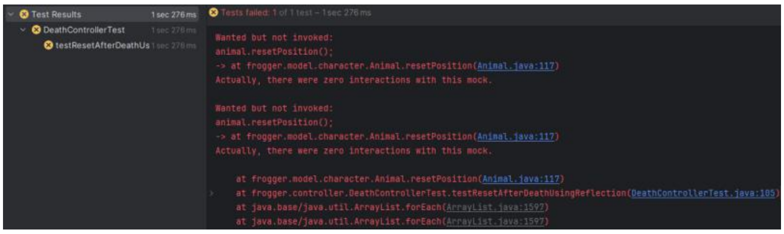
```
/**
 * Test method for resetting the animal after death.
 * This test uses reflection to invoke the private method `resetAfterDeath`.
 */
@Test  ssyjz21@nottingham.ac.uk <ssyjz21@nottingham.ac.uk>
void testResetAfterDeathUsingReflection() throws Exception {

    // Retrieve the private method resetAfterDeath from DeathController using reflection
    Method resetMethod = DeathController.class.getDeclaredMethod( name: "resetAfterDeath", Animal.class);
    resetMethod.setAccessible(true); // Make the private method accessible for testing

    // Invoke the reset method on the animal object
    resetMethod.invoke( obj: null, animal);

    // Verify that the reset position method is called on the animal
    verify(animal).resetPosition();

    // Check that the animal's image is reset to the original image (froggerUp.png)
    verify(animal).setImage(loadImage( fileName: "froggerUp.png", Animal_IMAGE_SIZE));

    // Check that the animal's movement flag (noMove) is reset to false after death
    assertFalse(animal.noMove);

    // Check that the carD variable is reset to 0
    assertEquals(animal.carD,  actual: 0);
}
Ssyjz21, 1/10/2025 4:42 AM • <test>(Controller): Add some tests for controller.
```

- ### Stub method



```
/**
 * Resets the animal's position, state, and image after a death.
 * Decreases the player's score and re-enables movement.
 *
 * @param animal The animal to reset.
 */
private static void resetAfterDeath(Animal animal) {  1 usage  ssyj
}
```

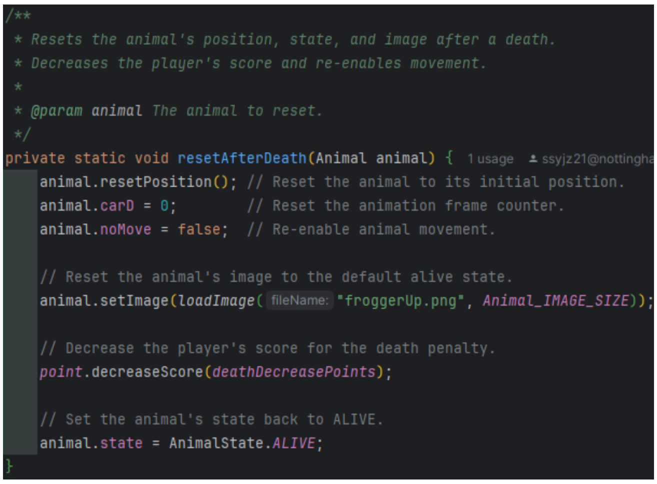
- ### Failure Message



Test Results 1 sec 276 ms
DeathControllerTest 1 sec 276 ms
testResetAfterDeathUs 1 sec 276 ms

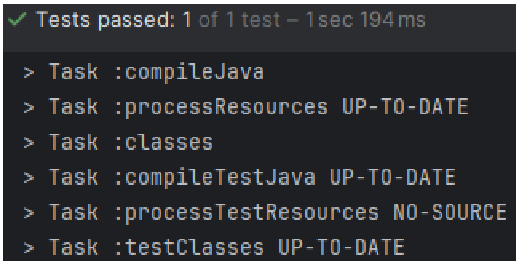
```
Tests failed: 1 of 1 test – 1 sec 276 ms

Wanted but not invoked:
animal.resetPosition();
-> at frogger.model.character.Animal.resetPosition(Animal.java:117)
Actually, there were zero interactions with this mock.

Wanted but not invoked:
animal.resetPosition();
-> at frogger.model.character.Animal.resetPosition(Animal.java:117)
Actually, there were zero interactions with this mock.

    at frogger.model.character.Animal.resetPosition(Animal.java:117)
    at frogger.controller.DeathControllerTest.testResetAfterDeathUsingReflection(DeathControllerTest.java:105)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
```

- ### Implement method



```
/**
 * Resets the animal's position, state, and image after a death.
 * Decreases the player's score and re-enables movement.
 *
 * @param animal The animal to reset.
 */
private static void resetAfterDeath(Animal animal) {  1 usage  ssyjz21@nottingha
    animal.resetPosition(); // Reset the animal to its initial position.
    animal.carD = 0;        // Reset the animation frame counter.
    animal.noMove = false;  // Re-enable animal movement.

    // Reset the animal's image to the default alive state.
    animal.setImage(loadImage( fileName: "froggerUp.png", Animal_IMAGE_SIZE));

    // Decrease the player's score for the death penalty.
    point.decreaseScore(deathDecreasePoints);

    // Set the animal's state back to ALIVE.
    animal.state = AnimalState.ALIVE;
}
```

- ### Success Message



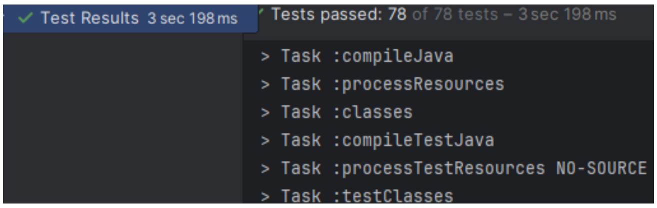
```
✓ Tests passed: 1 of 1 test – 1 sec 194 ms

> Task :compileJava
> Task :processResources UP-TO-DATE
> Task :classes
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
```

- ### Explanation

- This Unit Test tests a method resetting the animal's position, state, and image after a death and no return value. It is a case-based test using Reflection for testing private methods.

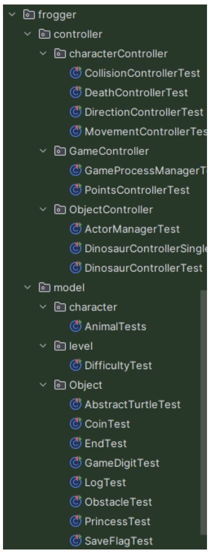
Overall

- ### Success Tests (Overall)



```
Tests passed: 78 of 78 tests – 3 sec 198 ms

> Task :compileJava
> Task :processResources
> Task :classes
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
```

- ### Structure:



```
frogger
  controller
    characterController
      CollisionControllerTest
      DeathControllerTest
      DirectionControllerTest
      MovementControllerTes
    GameController
      GameProcessManagerT
      PointsControllerTest
    ObjectController
      ActorManagerTest
      DinosaurControllerSingl
      DinosaurControllerTest
  model
    character
      AnimalTests
    level
      DifficultyTest
    Object
      AbstractTurtleTest
      CoinTest
      EndTest
      GameDigitTest
      LogTest
      ObstacleTest
      PrincessTest
      SaveFlagTest
```

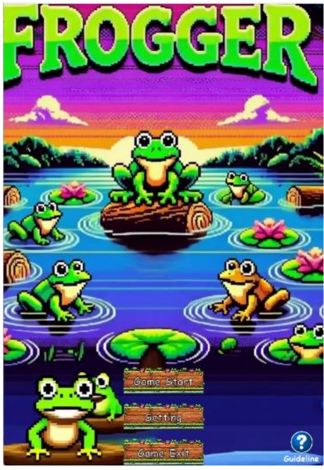
- ### Explanation

This game is test-driven refactoring. All 78 tests are normal case-based unit tests for only classes in game controllers and models, because View/UI should be tested using End-to-End Tests and classes in utility package are only used for auxiliary functions (for example SQL code), so there is no need to test them. And almost all of them use Mockito to mock the necessary classes.

Additional Features

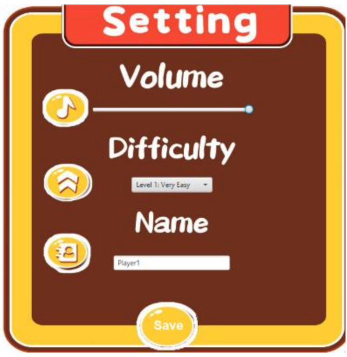
Game Start

- **Start UI:** Dynamic game background.
- **Guideline:** Click the **lower right button** to open the guideline.
- **Game Start & Exit:** Start / Exit the game.

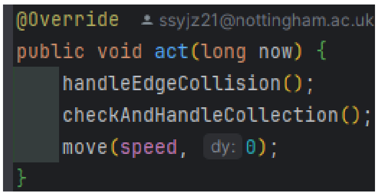


Setting

- **Volume:** Change the game Volume.
- **Difficulty:** Choose the game's five difficulties.
- **Name:** Change the name, players want to be stored in database.
- **Save:** Save the change you made and close the setting page.



During the Game



```
@Override  ssyjz21@nottingham.ac.uk
public void act(long now) {
    handleEdgeCollision();
    checkAndHandleCollection();
    move(speed,  dy: 0);
}
```

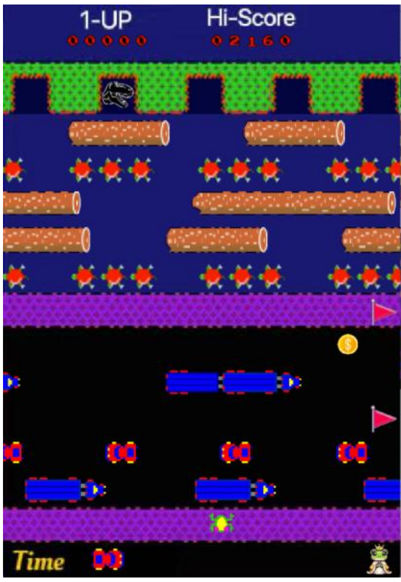
- **Coin:** Increasing score after picking.
- **Flag:** Save Point.
- **Time:** In the bottom row, the player must reach the end

before the car hits the princess.

- **Dinosaur:** The dinosaur will randomly appear in the

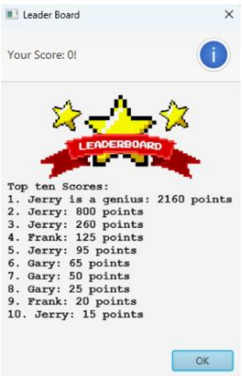
end that is not activated, and the frog will die if they touch it.

- **Hi-Score:** The highest score in database, it will change only when the record is broken.



After Game

- **Leader Board:** The Top ten player and their score in database.
- **Bad End:** The princess is hit by the car.
- **Happy End:** The player passes the game in limited time.



Your Score: 0!

```
Top ten Scores:
1. Jerry is a genius: 2160 points
2. Jerry: 800 points
3. Jerry: 260 points
4. Frank: 125 points
5. Jerry: 95 points
6. Gary: 65 points
7. Gary: 50 points
8. Gary: 25 points
9. Frank: 20 points
10. Jerry: 15 points
```

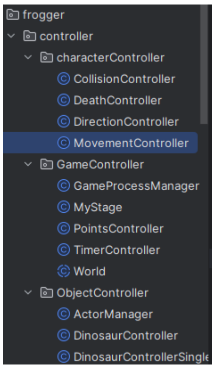
Addition (Can be found in README)

- **Database:** A database file that stores the player's name and score.
- **Game Audio:** All interactions in game have their corresponding sound effect.
- **Game Video:** A video quickly shows the process of the whole game.

For a more detailed description of the game, please see the guidelines and live video in Readme. Codes can be found on Gitlab.

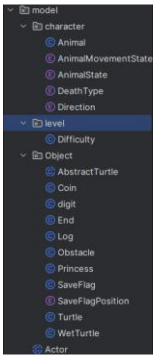
Refactor

Meaningful Package



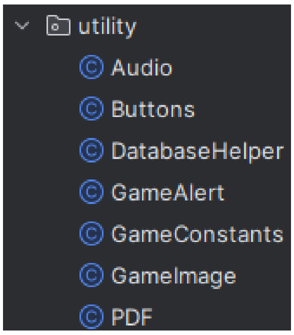
```
frogger
  controller
    characterController
      CollisionController
      DeathController
      DirectionController
      MovementController
    GameController
      GameProcessManager
      MyStage
      PointsController
      TimerController
      World
    ObjectController
      ActorManager
      DinosaurController
      DinosaurControllerSingle
```

controller: Handles character / game / object Logic.



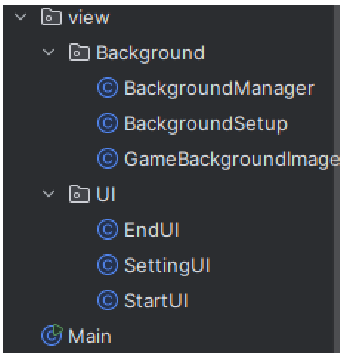
```
model
  character
    Animal
    AnimalMovementState
    AnimalState
    DeathType
    Direction
  level
    Difficulty
  Object
    AbstractTurtle
    Coin
    digit
    End
    Log
    Obstacle
    Princess
    SaveFlag
    SaveFlagPosition
    Turtle
    WetTurtle
  Actor
```

model: Defining and maintaining data structures, objects.



```
utility
  Audio
  Buttons
  DatabaseHelper
  GameAlert
  GameConstants
  GameImage
  PDF
```

utility: Some utility classes and methods that are not included in MVC architecture.



```
view
  Background
    BackgroundManager
    BackgroundSetup
    GameBackgroundImage
  UI
    EndUI
    SettingUI
    StartUI
  Main
```

view: Contains JavaFX files for UI and background

MVC architecture

At present, the code has been refactored into the MVC structure.

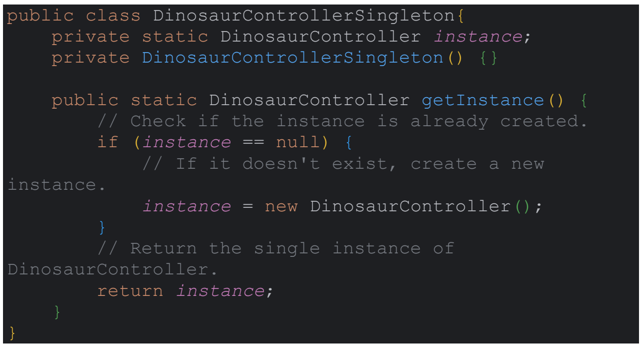
- **Model:** Currently it is responsible for handling all resources files store in the Resource package (including image, audio, video, database), It's also responsible for defining the main character, levels and objects in this frogger game.
- **View:** Currently, View is responsible for handling four UI interfaces (Start, Setting, Main, End) and background Image, also responding to user actions during games.
- **Controller:** Currently, it is responsible for handling Logic for main character, Objects and the whole game, every class handles a single logic, and they are all independent.

Design patterns

At present, there are some design patterns in the project, three examples will be shown.

- **Singleton Pattern**

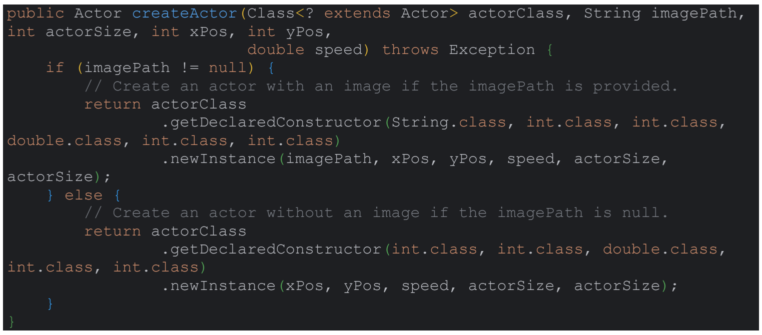
This DinosaurControllerSingleton ensures that only a single DinosaurController gets created in Game and provides a global access point for the entire game.



```
public class DinosaurControllerSingleton{
    private static DinosaurController instance;
    private DinosaurControllerSingleton() {}

    public static DinosaurController getInstance() {
        // Check if the instance is already created.
        if (instance == null) {
            // If it doesn't exist, create a new
instance.
            instance = new DinosaurController();
        }
        // Return the single instance of
DinosaurController.
        return instance;
    }
}
```

- **Factory & Strategy Pattern**

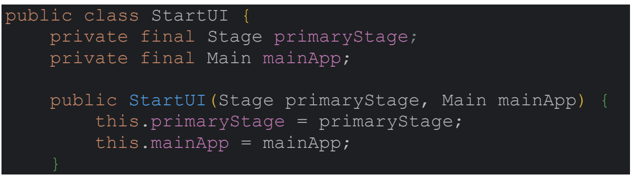
The createActor method dynamically creates an instance of an Actor object by parameterising it using reflection.



```
public Actor createActor(Class<? extends Actor> actorClass, String imagePath,
int actorSize, int xPos, int yPos,
                         double speed) throws Exception {
    if (imagePath != null) {
        // Create an actor with an image if the imagePath is provided.
        return actorClass
                .getDeclaredConstructor(String.class, int.class, int.class,
double.class, int.class, int.class)
                .newInstance(imagePath, xPos, yPos, speed, actorSize,
actorSize);
    } else {
        // Create an actor without an image if the imagePath is null.
        return actorClass
                .getDeclaredConstructor(int.class, int.class, double.class,
int.class, int.class)
                .newInstance(xPos, yPos, speed, actorSize, actorSize);
    }
}
```

- **Dependency Injection**

The StartUI class receives the primaryStage and mainApp dependencies through the constructor and assigns them to class member variables.

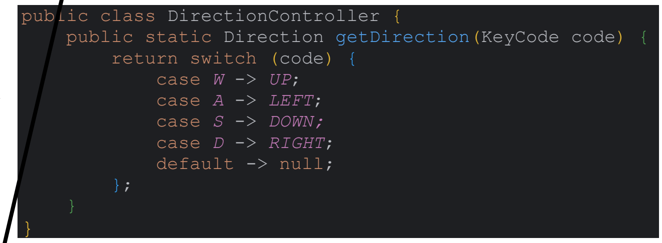


```
public class StartUI {
    private final Stage primaryStage;
    private final Main mainApp;

    public StartUI(Stage primaryStage, Main mainApp) {
        this.primaryStage = primaryStage;
        this.mainApp = mainApp;
    }
```

SOLID principles

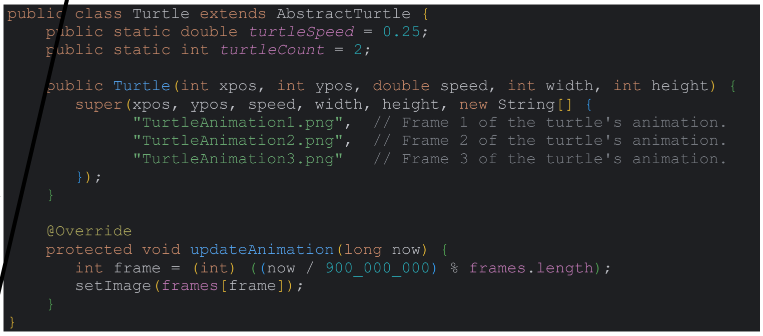
At present, the game follows the SOLID principles, some examples will be given.

- **SRP[1]:** Each class has a single responsibility.
  - ○ This DirectionController Class is only responsible for control the direction of the frog.



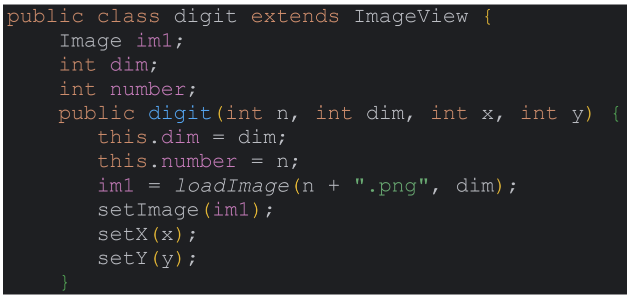
```
public class DirectionController {
    public static Direction getDirection(KeyCode code) {
        return switch (code) {
            case W -> UP;
            case A -> LEFT;
            case S -> DOWN;
            case D -> RIGHT;
            default -> null;
        };
    }
}
```

- **OCP[2]:** It is open for extension but is not closed for modification.
  - ○ This Turtle only extends a functionality of animation but doesn't edit the abstract class.
- **LSP[3]:** All subclasses follow the behaviour contract of their parent class.
  - ○ All properties of the AbstractTurtle hold in the Turtle class.



```
public class Turtle extends AbstractTurtle {
    public static double turtleSpeed = 0.25;
    public static int turtleCount = 2;

    public Turtle(int xpos, int ypos, double speed, int width, int height) {
        super(xpos, ypos, speed, width, height, new String[] {
                "TurtleAnimation1.png",  // Frame 1 of the turtle's animation.
                "TurtleAnimation2.png",  // Frame 2 of the turtle's animation.
                "TurtleAnimation3.png"   // Frame 3 of the turtle's animation.
        });
    }

    @Override
    protected void updateAnimation(long now) {
        int frame = (int) ((now / 900_000_000) % frames.length);
        setImage(frames[frame]);
    }
}
```

- **ISP[4]:** Classes only implement interfaces with methods that they use.
  - ○ This Digit Class doesn't extend the Actors but ImageView because it never uses act().



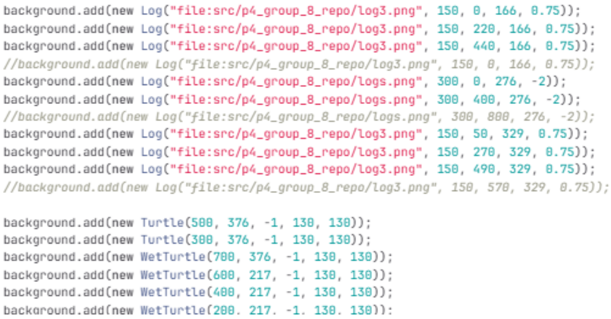
```
public class digit extends ImageView {
    Image im1;
    int dim;
    int number;
    public digit(int n, int dim, int x, int y) {
        this.dim = dim;
        this.number = n;
        im1 = loadImage(n + ".png", dim);
        setImage(im1);
        setX(x);
        setY(y);
    }
```

- **DIP[5]:** Classes depend on concrete implementations instead of abstractions.
  - ○ The StartUI Class uses Dependency Injection passing the specific dependencies. This avoids the details of the object creating its own dependencies.

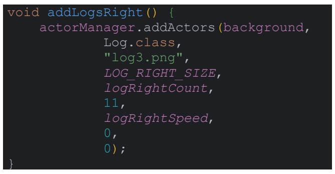
Code Smell

At present, the game eliminates / avoids code smell in many aspects. Here are some examples.

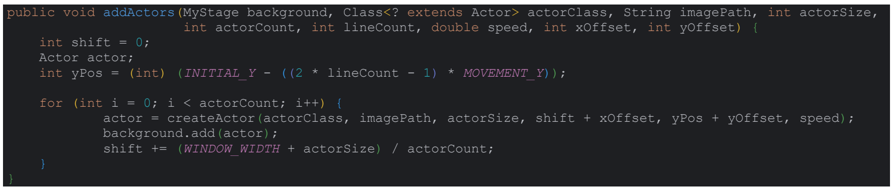
- **Duplicated Code**



```
background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 0, 166, 0.75));
background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 220, 166, 0.75));
background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 440, 166, 0.75));
//background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 0, 166, 0.75));
background.add(new Log("file:src/p4_group_8_repo/logs.png", 300, 0, 276, -2));
background.add(new Log("file:src/p4_group_8_repo/logs.png", 300, 400, 276, -2));
//background.add(new Log("file:src/p4_group_8_repo/logs.png", 300, 800, 276, -2));
background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 50, 329, 0.75));
background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 270, 329, 0.75));
background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 490, 329, 0.75));
//background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 570, 329, 0.75));

background.add(new Turtle(500, 376, -1, 130, 130));
background.add(new Turtle(300, 376, -1, 130, 130));
background.add(new WetTurtle(700, 376, -1, 130, 130));
background.add(new WetTurtle(600, 217, -1, 130, 130));
background.add(new WetTurtle(400, 217, -1, 130, 130));
background.add(new WetTurtle(200, 217, -1, 130, 130));
```



```
void addLogsRight() {
    actorManager.addActors(background,
            Log.class,
            "log3.png",
            LOG_RIGHT_SIZE,
            logRightCount,
            11,
            logRightSpeed,
            0,
            0);
}
```



```
public void addActors(MyStage background, Class<? extends Actor> actorClass, String imagePath, int actorSize,
                      int actorCount, int lineCount, double speed, int xOffset, int yOffset) {
    int shift = 0;
    Actor actor;
    int yPos = (int) (INITIAL_Y - ((2 * lineCount - 1) * MOVEMENT_Y));

    for (int i = 0; i < actorCount; i++) {
            actor = createActor(actorClass, imagePath, actorSize, shift + xOffset, yPos + yOffset, speed);
            background.add(actor);
            shift += (WINDOW_WIDTH + actorSize) / actorCount;
    }
}
```

○ In this example, Factory & Strategy Pattern is used to exact the duplicated code into a method to add actors in background which successfully avoid the code smell.

[1] Single Responsibility Principle

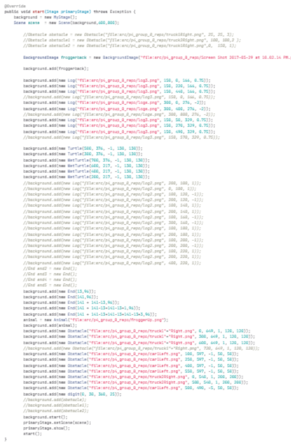
[2] Open/Closed Principle

[3] Leskov's Substitution Principle

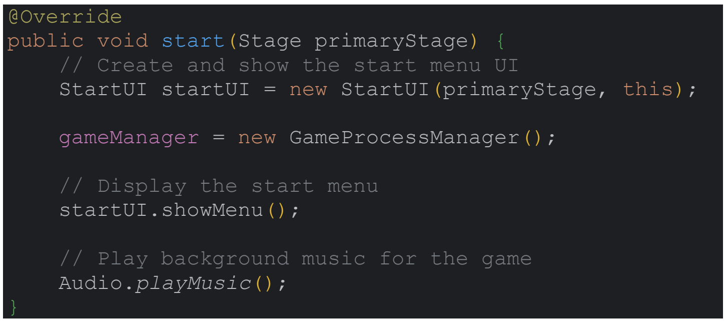
[4] Interface Segregation Principle

[5] Dependency Inversion Principle

- **Long methods & Class / Divergent Changes**



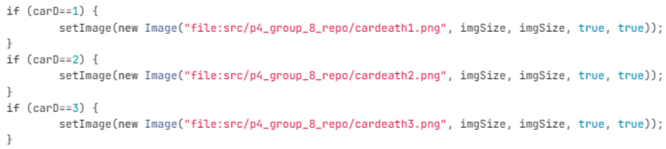
```
@Override
public void start(Stage primaryStage) throws Exception {
    background = new MyStage();
    Scene scene  = new Scene(background,600,800);

        //Obstacle obstacle = new Obstacle("file:src/p4_group_8_repo/truck1Right.png", 25, 25, 3);
        //Obstacle obstacle1 = new Obstacle("file:src/p4_group_8_repo/truck2Right.png", 100, 100,2 );
        //Obstacle obstacle2 = new Obstacle("file:src/p4_group_8_repo/truck1Right.png",0,  150, 1);

        BackgroundImage froggerback = new BackgroundImage("file:src/p4_group_8_repo/Screen Shot 2017-05-29 at 10.02.14 PM.png");

        background.add(froggerback);

        background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 0, 166, 0.75));
        background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 220, 166, 0.75));
        background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 440, 166, 0.75));
        //background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 0, 166, 0.75));
        background.add(new Log("file:src/p4_group_8_repo/logs.png", 300, 0, 276, -2));
        background.add(new Log("file:src/p4_group_8_repo/logs.png", 300, 400, 276, -2));
        //background.add(new Log("file:src/p4_group_8_repo/logs.png", 300, 800, 276, -2));
        background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 50, 329, 0.75));
        background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 270, 329, 0.75));
        background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 490, 329, 0.75));
        //background.add(new Log("file:src/p4_group_8_repo/log3.png", 150, 570, 329, 0.75));

        background.add(new Turtle(500, 376, -1, 130, 130));
        background.add(new Turtle(300, 376, -1, 130, 130));
        background.add(new WetTurtle(700, 376, -1, 130, 130));
        background.add(new WetTurtle(600, 217, -1, 130, 130));
        background.add(new WetTurtle(400, 217, -1, 130, 130));
        background.add(new WetTurtle(200, 217, -1, 130, 130));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 200, 100, 1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 0, 100, 1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 100, 120, -1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 200, 120, -1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 100, 140, 1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 200, 140, 1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 100, 160, -1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 300, 160, -1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 100, 180, 1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 200, 180, 1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 100, 200, -1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 200, 200, -1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 100, 220, 1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 200, 220, 1));
        //background.add(new Log("file:src/p4_group_8_repo/log2.png", 400, 220, 1));
        //End end2 = new End();
        //End end3 = new End();
        //End end4 = new End();
        //End end5 = new End();
        background.add(new End(13,96));
        background.add(new End(141,96));
        background.add(new End(141 + 141-13,96));
        background.add(new End(141 + 141-13+141-13+1,96));
        background.add(new End(141 + 141-13+141-13+141-13+3,96));
        animal = new Animal("file:src/p4_group_8_repo/froggerUp.png");
        background.add(animal);
        background.add(new Obstacle("file:src/p4_group_8_repo/truck1"+"Right.png", 0, 649, 1, 120, 120));
        background.add(new Obstacle("file:src/p4_group_8_repo/truck1"+"Right.png", 300, 649, 1, 120, 120));
        background.add(new Obstacle("file:src/p4_group_8_repo/truck1"+"Right.png", 600, 649, 1, 120, 120));
        //background.add(new Obstacle("file:src/p4_group_8_repo/truck1"+"Right.png", 720, 649, 1, 120, 120));
        background.add(new Obstacle("file:src/p4_group_8_repo/car1Left.png", 100, 597, -1, 50, 50));
        background.add(new Obstacle("file:src/p4_group_8_repo/car1Left.png", 250, 597, -1, 50, 50));
        background.add(new Obstacle("file:src/p4_group_8_repo/car1Left.png", 400, 597, -1, 50, 50));
        background.add(new Obstacle("file:src/p4_group_8_repo/car1Left.png", 550, 597, -1, 50, 50));
        background.add(new Obstacle("file:src/p4_group_8_repo/truck2Right.png", 0, 540, 1, 200, 200));
        background.add(new Obstacle("file:src/p4_group_8_repo/truck2Right.png", 500, 540, 1, 200, 200));
        background.add(new Obstacle("file:src/p4_group_8_repo/car1Left.png", 500, 490, -5, 50, 50));
        background.add(new digit(0, 30, 360, 25));
        //background.add(obstacle);
        //background.add(obstacle1);
        //background.add(obstacle2);
        background.start();
        primaryStage.setScene(scene);
        primaryStage.show();
        start();
}
```



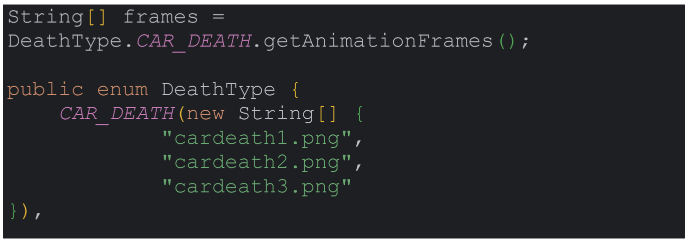
```
@Override
public void start(Stage primaryStage) {
    // Create and show the start menu UI
    StartUI startUI = new StartUI(primaryStage, this);

    gameManager = new GameProcessManager();

    // Display the start menu
    startUI.showMenu();

    // Play background music for the game
    Audio.playMusic();
}
```

- In this example, the responsibilities of main class and start method are separated into multiple classes and methods to decrease the method & class length.

- **Long Parameter Lists**



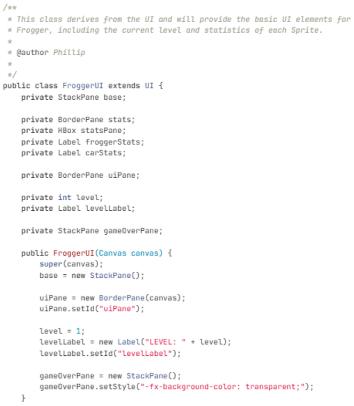
```
if (carD==1) {
        setImage(new Image("file:src/p4_group_8_repo/cardeath1.png", imgSize, imgSize, true, true));
}
if (carD==2) {
        setImage(new Image("file:src/p4_group_8_repo/cardeath2.png", imgSize, imgSize, true, true));
}
if (carD==3) {
        setImage(new Image("file:src/p4_group_8_repo/cardeath3.png", imgSize, imgSize, true, true));
}
```



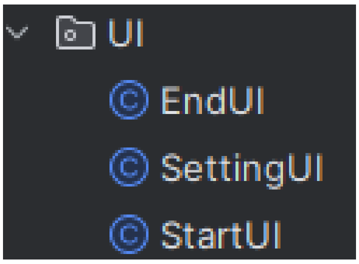
```
String[] frames =
DeathType.CAR_DEATH.getAnimationFrames();

public enum DeathType {
    CAR_DEATH(new String[] {
            "cardeath1.png",
            "cardeath2.png",
            "cardeath3.png"
}),
```

- In this example, a public enumerate is created to simplify the parameters lists of setImage for death animation. This could reduce the parameter lists length.

- **Lazy Class**

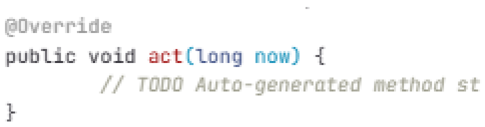


```
/**
 * This class derives from the UI and will provide the basic UI elements for
 * Frogger, including the current level and statistics of each Sprite.
 *
 * @author Phillip
 *
 */
public class FroggerUI extends UI {
    private StackPane base;

    private BorderPane stats;
    private HBox statsPane;
    private Label froggerStats;
    private Label carStats;

    private BorderPane uiPane;

    private int level;
    private Label levelLabel;

    private StackPane gameOverPane;

    public FroggerUI(Canvas canvas) {
        super(canvas);
        base = new StackPane();

        uiPane = new BorderPane(canvas);
        uiPane.setId("uiPane");

        level = 1;
        levelLabel = new Label("LEVEL: " + level);
        levelLabel.setId("levelLabel");

        gameOverPane = new StackPane();
        gameOverPane.setStyle("-fx-background-color: transparent;");
    }
```

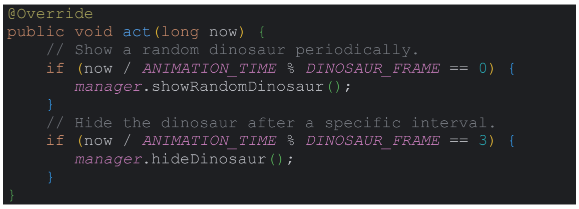


- In this example, the froggerUI class is no longer needed (no usage). It just been deleted, and three new UI classes with the similar functionality were created to replace it.

- **Speculative Generality**



```
@Override
public void act(long now) {
        // TODO Auto-generated method st
}
```



```
@Override
public void act(long now) {
    // Show a random dinosaur periodically.
    if (now / ANIMATION_TIME % DINOSAUR_FRAME == 0) {
        manager.showRandomDinosaur();
    }
    // Hide the dinosaur after a specific interval.
    if (now / ANIMATION_TIME % DINOSAUR_FRAME == 3) {
        manager.hideDinosaur();
    }
}
```

- In this example, the act() method in End Class is not implemented yet. A new feature is added in that method which can randomly show the dinosaur for those ends that are not activated.

Coding Conventions

1. Naming Conventions

- **Classes:** Pascal Case. e.g., EndUI **Methods and Variables:** camelCase. e.g., setEnd() hasDinosaur

2. Consistent Indentation and Formatting

- Proper indentation (2 spaces).
- Consistent use of braces {} and bracket ().
- Unified comment Format.
- Addition blank lines between statements.

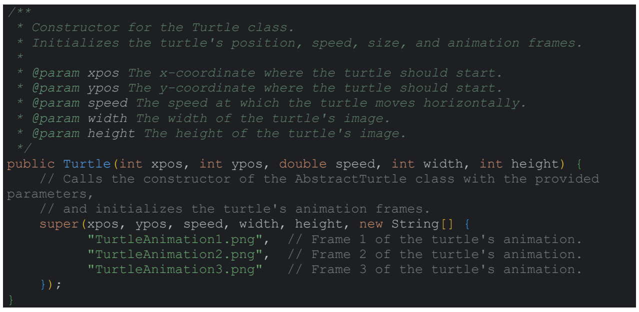
3. Code Documentation

- Meaningful comments explain the code.
- Javadoc used before classes and methods.

4. Use of Industry Standards

- Follows Java standard libraries (java.util).
- Ensures compatibility with JavaFX.

Example Method:



```java
/**
 * Constructor for the Turtle class.
 * Initializes the turtle's position, speed, size, and animation frames.
 *
 * @param xpos The x-coordinate where the turtle should start.
 * @param ypos The y-coordinate where the turtle should start.
 * @param speed The speed at which the turtle moves horizontally.
 * @param width The width of the turtle's image.
 * @param height The height of the turtle's image.
 */
public Turtle(int xpos, int ypos, double speed, int width, int height) {
    // Calls the constructor of the AbstractTurtle class with the provided
parameters,
    // and initializes the turtle's animation frames.
    super(xpos, ypos, speed, width, height, new String[] {
          "TurtleAnimation1.png",  // Frame 1 of the turtle's animation.
          "TurtleAnimation2.png",  // Frame 2 of the turtle's animation.
          "TurtleAnimation3.png"   // Frame 3 of the turtle's animation.
    });
}
```

Test Smell

➢ Avoidance

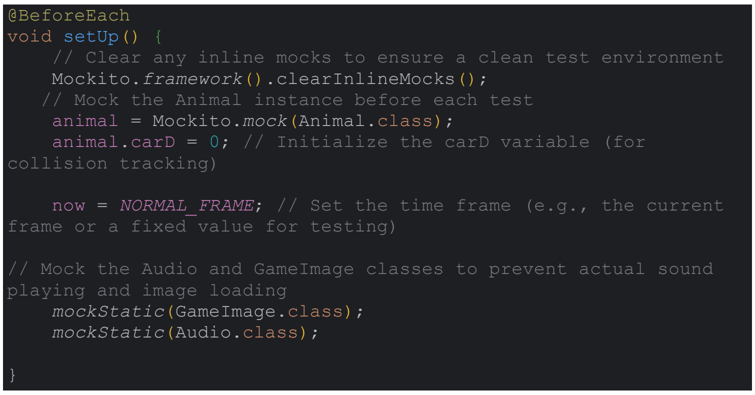
1. **Clear and Descriptive Test Names**

- Each test name clearly describes its purpose and scenario. e.g., `testAnimalInWater()`

2. **Independent Tests**

- Tests are isolated and do not depend on the execution order or shared state.

E.g.: Use clearInlineMocks to clear all mocks each Test.



```java
@BeforeEach
void setUp() {
    // Clear any inline mocks to ensure a clean test environment
    Mockito.framework().clearInlineMocks();
   // Mock the Animal instance before each test
    animal = Mockito.mock(Animal.class);
    animal.carD = 0; // Initialize the carD variable (for
collision tracking)

    now = NORMAL_FRAME; // Set the time frame (e.g., the current
frame or a fixed value for testing)

// Mock the Audio and GameImage classes to prevent actual sound
playing and image loading
    mockStatic(GameImage.class);
    mockStatic(Audio.class);

}
```

3. **Mocking and Stubbing**

- Use mocks for external dependencies to isolate the unit being tested, ensuring the tests focus solely on the logic under examination.

4. Meaningful Assertions

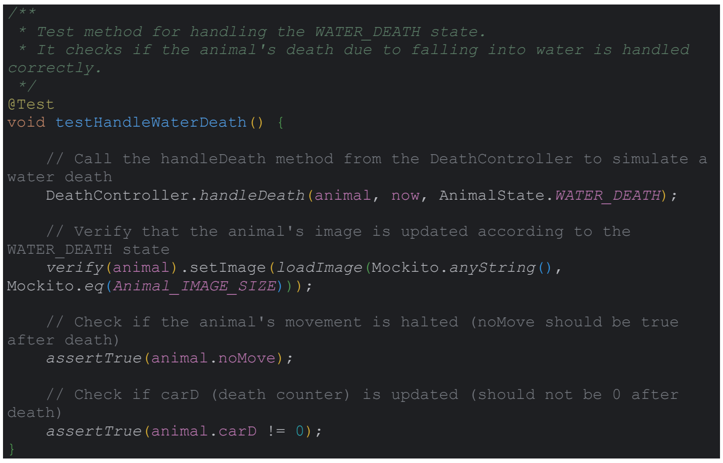
- Assertions verify the expected output clearly and completely.

5. **Readable and Maintainable Tests**

- Tests are concise, well-documented, and avoid redundant logic. Repeated setups are moved to helper methods or annotations like @BeforeEach/@Overide.

6. **Detailed documentation and comments**

Example Test:



```java
/**
 * Test method for handling the WATER_DEATH state.
 * It checks if the animal's death due to falling into water is handled
correctly.
 */
@Test
void testHandleWaterDeath() {

    // Call the handleDeath method from the DeathController to simulate a
water death
    DeathController.handleDeath(animal, now, AnimalState.WATER_DEATH);

    // Verify that the animal's image is updated according to the
WATER_DEATH state
    verify(animal).setImage(loadImage(Mockito.anyString(),
Mockito.eq(Animal_IMAGE_SIZE)));

    // Check if the animal's movement is halted (noMove should be true
after death)
    assertTrue(animal.noMove);

    // Check if carD (death counter) is updated (should not be 0 after
death)
    assertTrue(animal.carD != 0);
}
```