Author

[Company name]  [Company address]

[Document title]

Project Report: SkyRunner Game

1. Introduction

The SkyRunner project is a Java-based game designed to provide an engaging driving experience. The project involves multiple components, including UI management, game logic, media handling, and data interactions. This report details the project's structure, data flow, implementation of OOP concepts, and the overall working of the application.

2. Project Structure

2.1 Package Organization

The project is organized into several packages, each responsible for different aspects of the application:

com.example.skyrunner.screens: Contains the Game class that manages the main game loop, rendering, and user interactions.

com.example.skyrunner.database: Manages database operations (e.g., saving scores, retrieving player data).

com.example.skyrunner.models: Includes data models like PlayerCar and TrafficCar, representing game entities.

com.example.skyrunner.viewfactory: Provides UI management functionalities, such as UIManager, which handles UI components and layout.

3. Flow of Data

3.1 Initialization

Loading Resources: The Game class loads images and sounds from resources.

Setting Up UI: UIManager sets up and configures the user interface components.

Creating Game Entities: Instances of PlayerCar and TrafficCar are created and initialized.

3.2 Game Loop

Start Game: The start() method in the Game class initiates the game loop using AnimationTimer.

Update State: The update() method updates game state, including player and traffic car positions, score, distance, and collision detection.

Render Game: The render() method draws game elements on the Canvas.

3.3 User Interaction

Input Handling: User input (e.g., keyboard or mouse events) is processed to control the player car and interact with the game.

Game State Changes: Input can lead to changes in game state, such as moving the car, pausing the game or updating the score.

3.4 Game State Management

Pause/Resume: The game can be paused and resumed using the togglePause() method in the Game class.

End Game: On game over, the final score is saved and displayed, and the game loop is stopped.

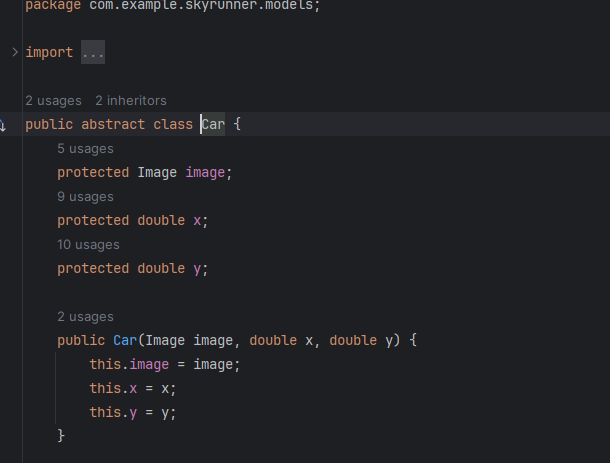
4. Object-Oriented Programming Concepts

4.1 Encapsulation

Implementation: The Game class encapsulates game logic, rendering, and state management. It provides public methods for starting, pausing, and stopping the game while keeping internal details private.

4.2 Inheritance and Polymorphism

Usage: Although the Game class does not use inheritance or polymorphism directly, the project structure allows for easy extension. For example, new types of cars or game elements can be introduced by extending base classes.



A screen shot of a computer program

Description automatically generated

4.3 Composition

Implementation: The Game class uses composition to include various components:

PlayerCar and TrafficCar: Car entities used in the game.

UIManager: Manages the user interface.

Canvas and GraphicsContext: For rendering game elements.

A screenshot of a computer program

Description automatically generated

5. Working of the Project

5.1 Initialization

Resource Loading: Images and sounds are loaded into memory.

UI Setup: UI components are created and added to the scene.

Game Entities: Player and traffic cars are initialized.

5.2 Game Loop

Game Start: The AnimationTimer starts, calling the update() and render() methods at each frame.

State Update: Game logic updates based on elapsed time and user input.

Rendering: Game elements are drawn on the Canvas, and the UI is updated.

5.3 User Interaction

Input Processing: User inputs are processed to control the player car and interact with the game.

Game State Changes: Actions such as pausing the game or colliding with obstacles affect the game state.

5.4 Game Over

End Conditions: The game ends based on specific conditions (e.g., collision).

Final Score: The final score is saved and displayed.

Game Loop Stop: The game loop is stopped, and resources are cleaned up.

6. Conclusion

The SkyRunner project demonstrates a well-structured Java application with a clear separation of concerns and effective use of OOP principles. The project’s organization into packages and classes allows for modular development and maintenance. The game’s functionality, including resource management, state updates, and user interaction, is efficiently handled through the Game class and associated components.