ALLIANCE UNIVERSITY

Private University established in Karnataka State by Act No.34 of year 2010
Recognized by the University Grants Commission (UGC), New Delhi

Project Report

Bachelor of Computer Applications

Semester - II

Lab-Object Oriented programming Java

Project Title: Snake Game using Java

Submitted By:

1. **Soham Sen – 2411021240008**
2. **Keshav Shekhawat - 2411021240066**

Submitted To:

VEERA RAGHAVAIAH KATHULA *Dr. J Lenin*
Mentor/Faculty Signature

Department of Computer Application

Alliance University

Chandapura- Anekal Main Road, Anekal

Bengaluru – 562 106

April 2025

📚 Table of Contents

1.Introduction to Java

1.1 What is Java?

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. Developed by James Gosling and his team at Sun Microsystems in 1995, Java is now owned and maintained by Oracle Corporation.

It is widely known for its portability, security, and simplicity, making it ideal for beginners and professionals alike. One of Java's key strengths is the "Write Once, Run Anywhere" capability, which means once you write Java code, it can run on any device or platform that has a Java Virtual Machine (JVM) installed.

1.2 History and Evolution

Java's development began in 1991, initially for interactive television. The team originally called it Oak, but the name was changed to Java (named after Java coffee) due to copyright issues.

Timeline:

- 1995: Java 1.0 released. Applets became popular.
- 1998: Java 2 introduced; the platform was divided into SE, EE, and ME.
- 2006: Sun made Java open source.
- 2010: Oracle acquired Sun Microsystems and took over Java.
- 2020–Present: Java 17+ introduced new modern features like Records, Sealed Classes, Pattern Matching, etc.

1.3 Key Features of Java

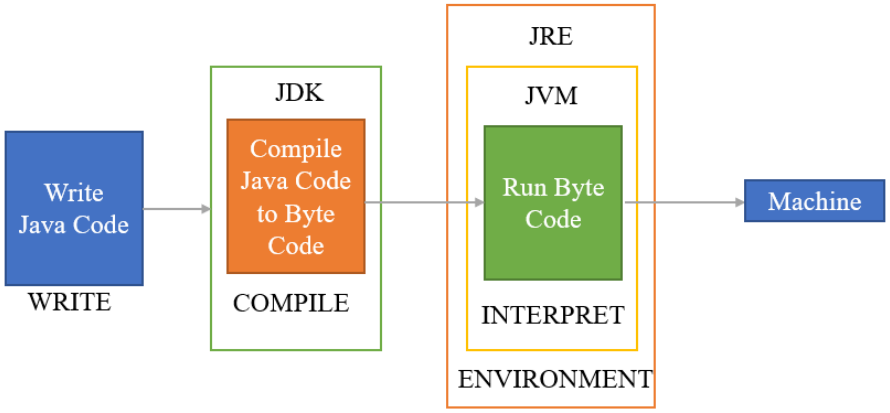
Feature	Description
Simple	Easy to learn and use; syntax like C++ but with removed complexities.
Object-Oriented	Everything in Java is treated as an object with classes and inheritance.
Platform-Independent	Java programs run on any platform with a JVM.
Secure	Features like bytecode verification, exception handling, and memory safety.
Robust	Strong memory management and runtime error checking.
Multithreaded	Supports multiple threads for performing tasks concurrently.
High Performance	Uses Just-In-Time (JIT) compilers for faster execution.
Distributed	Supports RMI and EJB for distributed computing.
Portable	Java bytecode can be carried and run across platforms.

1.4 Java Architecture

Understanding how Java works internally is important. Here's how the architecture looks:

1. Source Code: You write code using .java extension.
2. Compiler: Converts code to bytecode (.class files).
3. JVM (Java Virtual Machine): Executes bytecode on any platform.

4. JRE (Java Runtime Environment): Contains JVM and libraries to run applications.
5. JDK (Java Development Kit): Includes compiler (javac), JRE, and tools for development.



1.5 Java Editions

Java is available in various editions tailored for specific types of development:

Edition	Description
Java SE (Standard Edition)	Core functionality – used for desktops and basics.
Java EE (Enterprise Edition)	Web and enterprise apps – includes servlets, JSP.
Java ME (Micro Edition)	For embedded systems, small devices, smart cards.
JavaFX	Used to build GUI-rich desktop applications.

This project is based on Java SE using core features like OOP, GUI, and File Handling.

1.6 Java Programming Paradigm

Java follows several programming paradigms:

- Procedural (method-based logic flow)
- Object-Oriented (core paradigm: class, object, inheritance, etc.)
- Event-Driven (for GUI-based applications using event listeners)
- Multithreaded (for concurrent execution of code blocks)

1.7 Core Concepts in Java

Here are a few essential concepts every Java learner should know:

a. Class and Object

- A class is a blueprint for creating objects.
- An object is an instance of a class.

b. Inheritance

Allows a class to inherit properties and methods from another class.

c. Encapsulation

Hides internal data using private variables and provides access through methods (getters/setters).

d. Polymorphism

Allows one interface to be used for a general class of actions.

e. Abstraction

Hides complex implementation details and shows only essential features.

1.8 Java Syntax Example

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, Java World!");
    }
}
```

This is the most basic Java program. It demonstrates:

- Class creation
- Main method (entry point)
- Printing output

1.9 Applications of Java

Java is used in:

- Android App Development
- Web Applications (Banking, eCommerce)
- Scientific Computing
- Game Development
- Enterprise Applications
- Embedded Systems
- Cloud-based Services

Many large organizations like Google, Amazon, and LinkedIn rely on Java for their core backend systems.

1.10 Why Java is Ideal for Students

- Easy to learn
- Strong community support
- Rich documentation and libraries
- Can be used in both academic and real-world projects
- Foundation for learning more advanced languages like Kotlin or frameworks like Spring Boot.

Conclusion

Java is not just a programming language—it's a complete development platform that supports a wide range of applications. Its simplicity, powerful features, and cross-platform capabilities make it one of the best languages to begin a programming career with. Learning Java lays a solid foundation for various fields such as app development, data science, backend development, and cloud computing.

1. Objective of the Project:

The objective of this project is to develop a classic **Snake Game** using Java with an enhanced user experience through **login authentication** and personalized **score tracking**. The project aims to provide an engaging, interactive application that allows users to **register**, **log in securely**, and track their **high scores** over time. It demonstrates core Java programming concepts such as GUI development with **Swing**, **event handling**, **file input/output operations**, and basic **data management**. The project not only serves as an entertaining game but also as a learning tool to showcase how Java can be used to build functional, user-centric desktop applications.

2. About the Project

Title: Snake Game using Java

Type: Desktop Scale Video Game using only Java concepts

Domain: Game Development (Major), User Authentication Systems (Minor), Data Handling (Partially).

Target Users: Casual Gamers, Students i.e., Beginners in Programming, Developers using Authentication and Score Management Systems.

3. Project Description

This project is a classic Snake game implemented in Java with an integrated user login and high score tracking system. The **LoginFrame.java** file handles user registration and login functionality, securely managing credentials and initializing the game upon successful authentication. Once logged in, users are directed to the **Snake.java** window, which sets up the main game interface. The core gameplay logic resides in **Board.java**, including snake movement, collision detection, score tracking, and a dynamically rendered leaderboard. The project uses **Swing** for the graphical interface and **local file storage** to persist user data and scores.

4. Technologies Used:

Java (JDK) – The primary programming language used for the game and the login system.

Java Swing – Used to build the GUI (Login Screen, Game Window).

File I/O – To store user credentials and high scores locally.

5. System Requirements:

Operating System –

- Windows 7 or later, macOS, or any modern Linux distributions.

Java Runtime –

- Java SE Runtime Environment (JRE) 8 or higher,
    - Or install Java Development Kit if compiling from the source.

Processor –

- 1 GHz (Single-Core) or higher

RAM –

- 512 MB RAM (1 GB is recommended).

Storage –

- Around 10 – 20 MB of free disk space –
    - To store .jar or .class files.
    - Small files for user data and score records

Display –

- 800 x 600 – resolution or higher (calculated based on size of the game screen).

6. Modules and Features:

6.1 Modules:-

- **Java.awt:** The Java Abstract Window Toolkit package is one of the original Java GUI libraries. It provides the basic building blocks for graphical user interfaces, such as windows, buttons, labels, and layout managers. It also includes classes for handling graphics, events, colors, fonts, and more.
- **Java.swing:** The Java Swing package builds on top of AWT to provide a more powerful and flexible set of GUI components. It includes features like frames, panels, buttons, text fields, tables, and dialog boxes with better appearance and cross-platform consistency.
- **Java.util:** The Java Utilities package is a core part of the Java standard library, offering a wide range of utility classes and data

structures. It includes collections such as ArrayList, HashMap, HashSet, and more, as well as classes for date/time, random number generation, and formatting. This package is essential for managing data efficiently in any Java application.

- **Java.io:** The Java InputOutput package provides classes for performing input and output operations, including reading from and writing to files, managing data streams, and serializing objects. It supports both byte-based and character-based I/O, making it crucial for file handling, data storage, and communication in Java programs.
- **Java.nio:** The Java New InputOutput package is a more modern and scalable alternative to java.io. It introduces buffers, channels, and selectors to support high-performance and non-blocking I/O operations. java.nio is especially useful for large-scale or real-time applications that require efficient file and network handling.

6.2 Features:-

- **Interactive Gameplay:**
  - The core of the project is a classic Snake game where the player controls a growing snake, collects apples, and avoids collisions.
  - The game includes scoring, smooth movement, and visual feedback using `javax.swing` and `java.awt`.
- **User Login and Registration:**
  - Users can register with a username and password, and log in securely to play.
  - The system uses file storage (`java.io`) to save and validate user credentials, enabling personalized gameplay sessions.
- **High Score Tracking & Leaderboard:**
  - Each player's high score is saved locally and displayed at the end of the game.

- A real-time leaderboard shows the top scores of all registered users using efficient file handling and sorting (`java.util`, `java.nio`).

7. Working of the Project:

There are three main parts to this project –

- **Board.java**,
- **Snake.java**, and
- **LoginFrame.java**.

7.1. Board.java -

```
package Snake_Game;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Random;
import java.io.IOException;
import java.util.Map;
import java.util.HashMap;
import java.nio.file.*;
import java.util.stream.Collectors;

public class Board extends JPanel implements ActionListener {

    private final int B_WIDTH = 300;
    private final int B_HEIGHT = 300;
    private final int DOT_SIZE = 10;
    private final int ALL_DOTS = 900;
    private final int RAND_POS = 29;
    private final int DELAY = 140;

    private final int x[] = new int[ALL_DOTS];
    private final int y[] = new int[ALL_DOTS];

    private int dots;
    private int apple_x;
    private int apple_y;
```

```
private boolean leftDirection = false;
private boolean rightDirection = true;
private boolean upDirection = false;
private boolean downDirection = false;
private boolean inGame = true;

private Timer timer;
private Image ball;
private Image apple;
private Image head;

private int score = 0;
private String username;

public Board(String username) {
    this.username = username;
    initBoard();
}

private void initBoard() {
    addKeyListener(new TAdapter());
    setBackground(Color.black);
    setFocusable(true);

    setPreferredSize(new Dimension(B_WIDTH, B_HEIGHT));
    loadImages();
    initGame();
}

private void loadImages() {
    ImageIcon iid = new ImageIcon("src/resources/dot.png");
    ball = iid.getImage();

    ImageIcon iia = new ImageIcon("src/resources/apple.png");
    apple = iia.getImage();

    ImageIcon iih = new ImageIcon("src/resources/head.png");
    head = iih.getImage();
}
```

```
private void initGame() {
    dots = 3;
    score = 0;
    inGame = true;

    for (int i = 0; i < dots; i++) {
        x[i] = 50 - i * DOT_SIZE;
        y[i] = 50;
    }

    locateApple();
    timer = new Timer(DELAY, this);
    timer.start();
}


@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    if (inGame) {
        doDrawing(g);
    }
    else {
        gameOver(g);
    }
    Toolkit.getDefaultToolkit().sync();
}

private void doDrawing(Graphics g) {
    if (inGame) {
        g.drawImage(apple, apple_x, apple_y, this);
        for (int z = 0; z < dots; z++) {
            if (z == 0) {
                g.drawImage(head, x[z], y[z], this);
            }
            else {
                g.drawImage(ball, x[z], y[z], this);
            }
        }

        g.setColor(Color.white);
```

```
            g.drawString("Score: " + score, 10, 20);

            Toolkit.getDefaultToolkit().sync();

        }
    }

    private JButton playAgainButton;
    private JButton exitButton;

    private void gameOver(Graphics g) {
        Font titleFont = new Font("Arial", Font.BOLD, 22);
        Font regularFont = new Font("Arial", Font.PLAIN, 16);

        g.setColor(Color.WHITE);

        // Game Over title
        g.setFont(titleFont);
        String msg = "Game Over";
        FontMetrics fm = g.getFontMetrics();
        g.drawString(msg, (B_WIDTH - fm.stringWidth(msg)) / 2, 100);

        // Final score
        g.setFont(regularFont);
        fm = g.getFontMetrics();
        String finalScore = "Final Score: " + score;
        g.drawString(finalScore, (B_WIDTH - fm.stringWidth(finalScore))
/ 2, 140);

        // High score
        try {
            int highScore = ScoreManager.getHighScore(username);
            String highScoreStr = "High Score: " + highScore;
            g.drawString(highScoreStr, (B_WIDTH -
fm.stringWidth(highScoreStr)) / 2, 170);

            // Leaderboard
            g.drawString("--- Leaderboard ---", (B_WIDTH -
fm.stringWidth("--- Leaderboard ---")) / 2, 200);

            int yOffset = 225;
```

```
            Map<String, Integer> leaderboard =
ScoreManager.getLeaderboard();
            for (Map.Entry<String, Integer> entry :
leaderboard.entrySet()) {
                String entryStr = entry.getKey() + ": " +
entry.getValue();
                g.drawString(entryStr, (B_WIDTH -
fm.stringWidth(entryStr)) / 2, yOffset);
                yOffset += 20;
            }

            // Restart instructions
            String restart = "Press SPACE to play again or ESC to exit";
            g.drawString(restart, (B_WIDTH - fm.stringWidth(restart)) /
2, yOffset + 20);

        }
        catch (IOException e) {
            g.drawString("Error loading high score", 10, 170);
        }
    }

    private void checkApple() {
        if ((x[0] == apple_x) && (y[0] == apple_y)) {
            dots++;
            score += 10;
            locateApple();
        }
    }

    private void move() {
        for (int z = dots; z > 0; z--) {
            x[z] = x[(z - 1)];
            y[z] = y[(z - 1)];
        }

        if (leftDirection) {
            x[0] -= DOT_SIZE;
        }

        if (rightDirection) {
```

```
            x[0] += DOT_SIZE;
        }

        if (upDirection) {
            y[0] -= DOT_SIZE;
        }

        if (downDirection) {
            y[0] += DOT_SIZE;
        }
    }

    private void checkCollision() {
        for (int z = dots; z > 0; z--) {
            if ((z > 4) && (x[0] == x[z]) && (y[0] == y[z])) {
                inGame = false;
            }
        }

        if (x[0] >= B_WIDTH) {
            x[0] = 0;
        }
        if (x[0] < 0) {
            x[0] = B_WIDTH - DOT_SIZE;
        }
        if (y[0] >= B_HEIGHT) {
            y[0] = 0;
        }
        if (y[0] < 0) {
            y[0] = B_HEIGHT - DOT_SIZE;
        }


        if (!inGame) {
            timer.stop();
            gameOver(getGraphics());
            try {
                ScoreManager.saveHighScore(username, score);
            }
            catch (IOException e) {
                e.printStackTrace();
```

```
        }
    }
}

private void locateApple() {
    int r = (int) (Math.random() * RAND_POS);
    apple_x = ((r * DOT_SIZE));
    r = (int) (Math.random() * RAND_POS);
    apple_y = ((r * DOT_SIZE));
}

@Override
public void actionPerformed(ActionEvent e) {
    if (inGame) {
        checkApple();
        checkCollision();
        move();
    }
    repaint();
}

private class TAdapter extends KeyAdapter {
    @Override
    public void keyPressed(KeyEvent e) {
        int key = e.getKeyCode();
        if ((key == KeyEvent.VK_LEFT) && (!rightDirection)) {
            leftDirection = true;
            upDirection = false;
            downDirection = false;
        }

        if ((key == KeyEvent.VK_RIGHT) && (!leftDirection)) {
            rightDirection = true;
            upDirection = false;
            downDirection = false;
        }

        if ((key == KeyEvent.VK_UP) && (!downDirection)) {
            upDirection = true;
            rightDirection = false;
            leftDirection = false;
```

```
            }

            if ((key == KeyEvent.VK_DOWN) && (!upDirection)) {
                downDirection = true;
                rightDirection = false;
                leftDirection = false;
            }
            if (!inGame) {
                if (key == KeyEvent.VK_SPACE) {
                    initGame();
                    repaint();
                }
                else if (key == KeyEvent.VK_ESCAPE) {
                    System.exit(0);
                }
            }
        }
    }
}
```

7.2. Snake.java -

```
package Snake_Game;

import java.awt.EventQueue;
import javax.swing.JFrame;

public class Snake  extends JFrame {
    private String username;

    public Snake(String username) {
        this.username = username;
        initUI();
    }

    public void setUsername(String username) {
        this.username = username;
    }

    private void initUI() {
        add(new Board(username));
        pack();
```

```
        setResizable(false);
        setTitle("Snake");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

7.3. LoginFrame.java -

```
package Snake_Game;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.Files;
import java.util.stream.Collectors;
import java.io.IOException;

class User {
    private String username;
    private String password;

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }
    public String getPassword() {
        return password;
    }
}

class UserManager {
    private static final String USER_FILE = "users.txt";
```

```
    public static void saveUser(User user) throws IOException {
        try (FileWriter fw = new FileWriter(USER_FILE, true)) {
            fw.write(user.getUsername() + "," + user.getPassword() +
"\n");
        }
    }

    public static boolean validateLogin(String username, String
password) throws IOException {
        try (BufferedReader br = new BufferedReader(new
FileReader(USER_FILE))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(",");
                if (parts.length == 2 && parts[0].equals(username) &&
parts[1].equals(password)) {
                    return true;
                }
            }
        }
        return false;
    }
}

class ScoreManager {
    private static final String SCORE_FILE = "scores.txt";

    public static int getHighScore(String username) throws IOException {
        Path scoreFile = Paths.get("highscores.txt");
        if (!Files.exists(scoreFile)) return 0;

        return Files.lines(scoreFile)
                .map(line -> line.split(":"))
                .filter(parts -> parts.length == 2 &&
parts[0].equals(username))
                .mapToInt(parts -> {
                    try {
                        return Integer.parseInt(parts[1]);
                    }
                    catch (NumberFormatException e) {
                        return 0;
```

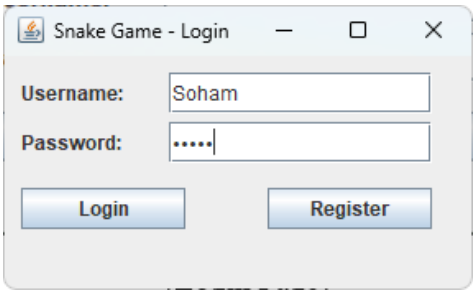
```
                    }
                })
                .max()
                .orElse(0);
    }


    public static void saveHighScore(String username, int score) throws
IOException {
        Path scoreFile = Paths.get("highscores.txt");
        Map<String, Integer> scores = new HashMap<>();

        if (Files.exists(scoreFile)) {
            Files.lines(scoreFile)
                    .map(line -> line.split(":"))
                    .filter(parts -> parts.length == 2)
                    .forEach(parts -> {
                        try {
                            scores.put(parts[0],
Math.max(scores.getOrDefault(parts[0], 0), Integer.parseInt(parts[1])));
                        }
                        catch (NumberFormatException e) {
                        }
                    });
        }

        // Update or insert the current user's score
        scores.put(username, Math.max(scores.getOrDefault(username, 0),
score));

        // Saves to the file
        BufferedWriter writer = Files.newBufferedWriter(scoreFile);
        for (Map.Entry<String, Integer> entry : scores.entrySet()) {
            writer.write(entry.getKey() + ":" + entry.getValue());
            writer.newLine();
        }
        writer.close();
    }

    public static Map<String, Integer> getLeaderboard() throws
IOException {
```

```
        Path scoreFile = Paths.get("highscores.txt");
        Map<String, Integer> scores = new HashMap<>();

        if (!Files.exists(scoreFile)) {
            return scores;
        }

        for (String line : Files.readAllLines(scoreFile)) {
            String[] parts = line.split(":");
            if (parts.length == 2) {
                try {
                    String user = parts[0];
                    int score = Integer.parseInt(parts[1]);
                    scores.put(user, Math.max(scores.getOrDefault(user,
0), score));
                }
                catch (NumberFormatException ignored) {}
            }
        }

        // Sort scores by value descending
        return scores.entrySet()
                .stream()
                .sorted((e1, e2) -> Integer.compare(e2.getValue(),
e1.getValue()))
                .collect(Collectors.toMap(
                        Map.Entry::getKey,
                        Map.Entry::getValue,
                        (a, b) -> a,
                        LinkedHashMap::new
                ));
    }
}

public class LoginFrame extends JFrame {
    public LoginFrame() {
        setTitle("Snake Game - Login");
        setSize(300, 180);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
```

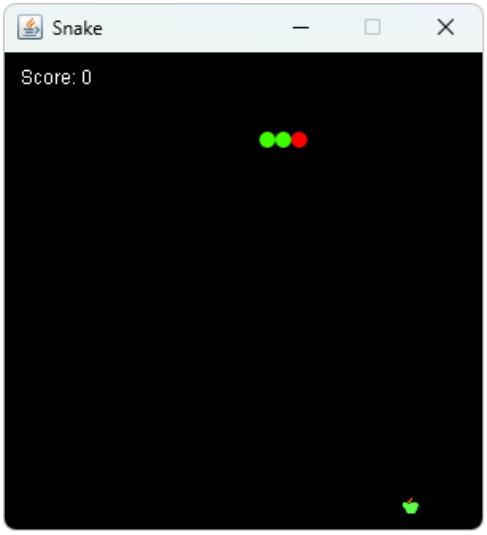
```
        JPanel panel = new JPanel();
        add(panel);
        placeComponents(panel);
    }
    private void placeComponents(JPanel panel) {
        panel.setLayout(null);

        JLabel userLabel = new JLabel("Username:");
        userLabel.setBounds(10, 10, 80, 25);
        panel.add(userLabel);

        JTextField userText = new JTextField(20);
        userText.setBounds(100, 10, 160, 25);
        panel.add(userText);

        JLabel passwordLabel = new JLabel("Password:");
        passwordLabel.setBounds(10, 40, 80, 25);
        panel.add(passwordLabel);

        JPasswordField passwordText = new JPasswordField(20);
        passwordText.setBounds(100, 40, 160, 25);
        panel.add(passwordText);

        JButton loginButton = new JButton("Login");
        loginButton.setBounds(10, 80, 100, 25);
        panel.add(loginButton);

        JButton registerButton = new JButton("Register");
        registerButton.setBounds(160, 80, 100, 25);
        panel.add(registerButton);

        loginButton.addActionListener(e -> {
            String user = userText.getText();
            String pass = new String(passwordText.getPassword());
            try {
                if (UserManager.validateLogin(user, pass)) {
                    JOptionPane.showMessageDialog(null, "Login
Successful!");
                    dispose();
                    SwingUtilities.invokeLater(() -> {
                        Snake game = new Snake(user);
```

```
                        game.setVisible(true);
                    });
                }
                else {
                    JOptionPane.showMessageDialog(null, "Invalid
credentials!");
                }
            }
            catch (IOException ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(null, "Error reading user
data.");
            }
        });
        registerButton.addActionListener(e -> {
            String user = userText.getText();
            String pass = new String(passwordText.getPassword());
            try {
                if (!user.isEmpty() && !pass.isEmpty()) {
                    UserManager.saveUser(new User(user, pass));
                    JOptionPane.showMessageDialog(null, "Registration
Successful!");
                }
                else {
                    JOptionPane.showMessageDialog(null, "Username and
Password cannot be empty.");
                }
            }
            catch (IOException ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(null, "Error saving
user.");
            }
        });
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new
LoginFrame().setVisible(true));
    }
}
```

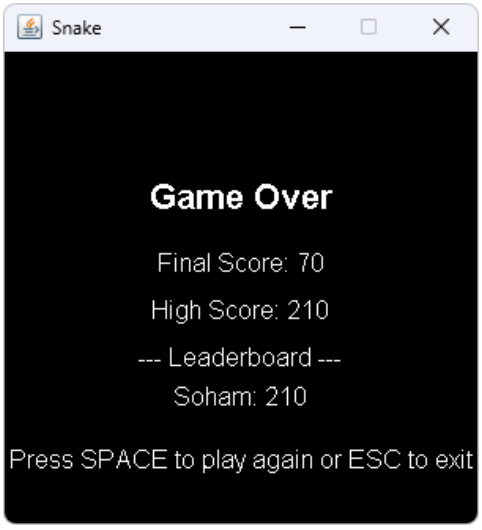
8. Screenshots:



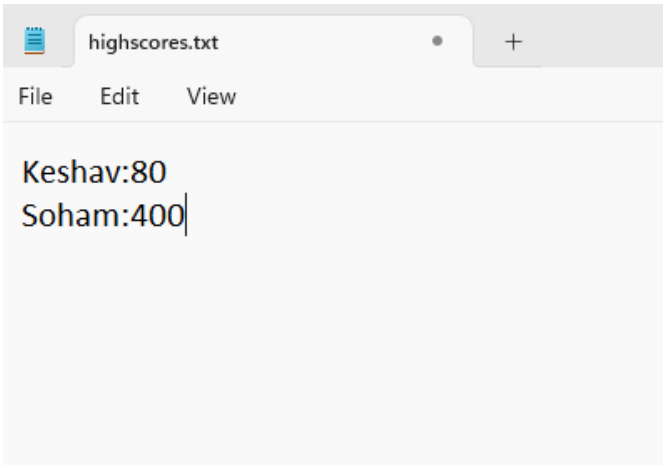
(Login Page)



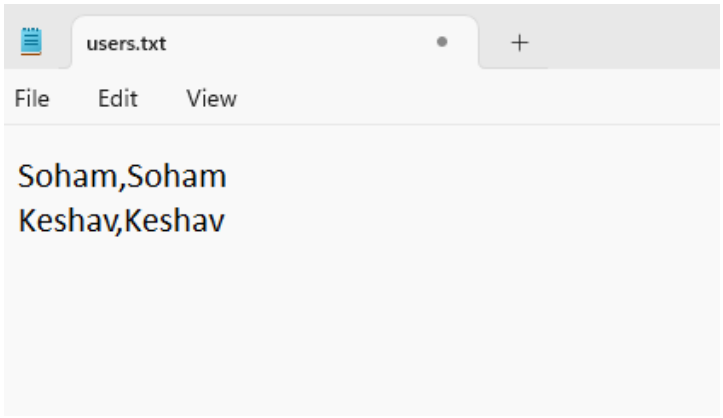
(Game Screen)



(Game Over Screen)



(High Scores Text File)



(User Credentials Text File)

9. Sample Code Snippets:

The below snippet is from the file **Snake.java**, where it acceps a **username** during initialization, passes it onto the **board** component, and sets up the **UI**. It also contains a method to update the username if the need ever arises.

```
package Snake_Game;

import java.awt.EventQueue;
import javax.swing.JFrame;

public class Snake extends JFrame {  2 usages
    private String username;  3 usages

    public Snake(String username) {  1 usage
        this.username = username;
        initUI();
    }

    public void setUsername(String username) {  no usages
        this.username = username;
    }

    private void initUI() {  1 usage
        add(new Board(username));
        pack();
        setResizable(false);
        setTitle("Snake");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

The below snippet defines a **ScoreManager** class with a method **getHighScore()** that reads from a file (**highscores.txt**) and returns the highest score for a specific user. It filters lines by username, parses the score, handles invalid numbers gracefully, and returns the highest score using Java's **Streams** and **Files** API.

```
class ScoreManager {
    private static final String SCORE_FILE = "scores.txt";

    public static int getHighScore(String username) throws IOException {
        Path scoreFile = Paths.get( first: "highscores.txt");
        if (!Files.exists(scoreFile)) return 0;

        return Files.lines(scoreFile)
                .map( String line -> line.split( regex: ":"))
                .filter( String[] parts -> parts.length == 2 && parts[0].equals(username))
                .mapToInt( String[] parts -> {
                    try {
                        return Integer.parseInt(parts[1]);
                    }
                    catch (NumberFormatException e) {
                        return 0;
                    }
                })
                .max()
                .orElse( other: 0);
    }
```

The below code snippet defines a **UserManager** class with methods to handle user data. The **saveUser()** method appends a user's username and password to a file named **users.txt**. The **validateLogin()** method reads through the file, splits each line by a comma, and checks if any entry matches both the provided username and password, returning true if a match is found.

```
class UserManager {
    private static final String USER_FILE = "users.txt";

    public static void saveUser(User user) throws IOException {
        try (FileWriter fw = new FileWriter(USER_FILE,  append: true)) {
            fw.write( str: user.getUsername() + "," + user.getPassword() + "\n");
        }
    }

    public static boolean validateLogin(String username, String password) throws IOException {
        try (BufferedReader br = new BufferedReader(new FileReader(USER_FILE))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split( regex: ",");
                if (parts.length == 2 && parts[0].equals(username) && parts[1].equals(password)) {
                    return true;
                }
            }
        }
        return false;
    }
}
```

The code snippet below is the initial setup for the **Board** class of the Snake game, which extends **JPanel** and implements **ActionListener**, indicating it handles **GUI rendering** and **event-based actions**. It imports key Java libraries for GUI components, event handling, file I/O, and utilities. The class defines constants for the game board's dimensions, dot size, and timing delay. It initializes arrays **x[]** and **y[]** to store the coordinates of the snake's body segments. The code also includes variables to track the apple's position, the snake's movement direction, and the game's state (**inGame**). Lastly, it declares a **Timer** for game progression and **Image** variables to hold graphics for the snake and apple.

```java
package Snake_Game;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Random;
import java.io.IOException;
import java.util.Map;
import java.util.HashMap;
import java.nio.file.*;
import java.util.stream.Collectors;

public class Board extends JPanel implements ActionListener {  1 usage

    private final int B_WIDTH = 300;  9 usages
    private final int B_HEIGHT = 300;  3 usages
    private final int DOT_SIZE = 10;  9 usages
    private final int ALL_DOTS = 900;  2 usages
    private final int RAND_POS = 29;  2 usages
    private final int DELAY = 140;  1 usage

    private final int x[] = new int[ALL_DOTS];  14 usages
    private final int y[] = new int[ALL_DOTS];  14 usages

    private int dots;  6 usages
    private int apple_x;  3 usages
    private int apple_y;  3 usages

    private boolean leftDirection = false;  5 usages
    private boolean rightDirection = true;  5 usages
    private boolean upDirection = false;  5 usages
    private boolean downDirection = false;  5 usages
    private boolean inGame = true;  7 usages

    private Timer timer;  3 usages
    private Image ball;  2 usages
    private Image apple;  2 usages
    private Image head;  2 usages
```

10. Conclusion:

This Java-based Snake Game project successfully combines the classic arcade game with modern features like **user authentication** and **score tracking**. Utilizing **Java Swing** for the graphical interface and **file I/O** for data persistence, it offers an engaging experience with a personalized touch through login and high score saving.

The implementation of a leaderboard adds a competitive element, encouraging repeated play. The project demonstrates core Java concepts including **event handling**, **GUI development**, **collections**, and **file management**.

Overall, it is a well-rounded application suitable for both learning and entertainment, showcasing how fundamental Java packages can be used to build interactive, user-driven desktop applications.

11. References:

- www.geeksforgeeks.org/java-awt-tutorial/
- www.geeksforgeeks.org/introduction-to-java-swing/
- www.guru99.com/java-swing-gui.html
- www.tutorialspoint.com/how-can-we-create-a-login-form-in-java