**FUNCITY MANAGEMENT SYSTEM**

**A MINI-PROJECT REPORT**

**Submitted by**

**SKANDAN KAMAL 230701322**

**SIDHARTH SUBRAMONIAN 230701315**

**In partial fulfillment of the award of the degree**

**of**

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

****

**RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI**

**An Autonomous Institute**

**CHENNAI**

**NOVEMBER 2024**

BONAFIDE CERTIFICATE

Certified that this project “FUNCITY MANAGEMENT SYSTEM” is the

bonafide work of “SKANDAN KAMAL, SIDHARTH SUBRAMONIAN” who carried out the project work under my supervision.

SIGNATURE SIGNATURE

MR G SARAVANA GOKUL MRS V JANANEE

ASSISTANT PROFESSOR ASSISTANT PROFESSOR

Dept. of Computer Science and Engg, Dept. of Computer Science and Engg,

Rajalakshmi Engineering College Rajalakshmi Engineering College

Chennai Chennai

This mini project report is submitted for the viva voce examination to be held on \_\_\_\_\_\_

TABLE OF CONTENTS

S.NO TITLE PAGE

Abstract

1. Introduction 05
2. Scope of Project 06
3. ER Diagrams 10
4. Code Implementation 11
5. Output 31
6. Conclusion 46

**ABSTRACT:**

The FunCity Management System is a Java-based mini-project aimed at streamlining operations for amusement parks or recreational facilities. This system is designed to manage various aspects of FunCity, including visitor entry, ticketing, ride reservations, and facility management. Built with an object-oriented approach, the project leverages Java's robust features to deliver a scalable and efficient solution.

The system comprises several core modules, including user registration, ticket purchase, ride scheduling, and payment handling. The backend handles data storage and retrieval, ensuring seamless integration with various functionalities. The project also includes an administrator interface for managing rides, tracking visitor statistics, and generating reports for analysis.

This mini-project highlights the use of advanced Java concepts such as collections, exception handling, and multi-threading to provide a responsive and user-friendly application. Through this system, amusement park administrators can enhance operational efficiency, improve visitor experience, and gain insights into park activities, making FunCity Management a modernized solution for the recreational industry.

**INTRODUCTION**:

The FunCity Management System is a mini-project developed using Java to manage the operations of an amusement park or recreational facility efficiently. It is designed to handle key functionalities such as visitor entry management, ticket bookings, ride scheduling, and administrative tasks, offering a seamless experience for both users and administrators.

The project adopts a modular design, ensuring easy navigation and scalability. Visitors can explore rides, book tickets, and reserve slots, while administrators can oversee park operations, update ride information, and track revenue. Java's rich features, including object-oriented programming, exception handling, and file handling, make the system robust and reliable.

This system aims to automate repetitive tasks, reduce errors, and enhance the overall experience of visiting and managing a recreational park. It is a practical demonstration of applying programming skills to solve real-world problems, making it an ideal project for students and budding developers exploring Java application development

**SCOPE OF THE PROJECT**

The FunCity Management System is a comprehensive mini-project designed to automate and streamline the operations of an amusement park or recreational facility. The scope of this project covers a wide range of functionalities aimed at enhancing the experience for both park visitors and administrators.

1. **Effortless Ticket Management**:

* Visitors can book tickets online or on-site, reducing waiting time and ensuring convenience.
* Availability of multiple ticket options, such as daily passes, family packages, and ride-specific tickets.

2**. Ride Reservations**:

* Allows visitors to view, select, and book rides based on their preferences.
* Real-time updates on ride availability and scheduling changes.

3**. Personalized Experience**:

* Notifications and updates about special offers, discounts, and upcoming events.
* User accounts to track booking history and manage personal preferences.

**Administrator-Centric Scope**:

1. **Operational Efficiency**:

- Centralized system to manage all park operations, including rides, attractions, and visitor data.

- Tools to monitor park capacity and prevent overcrowding.

2. **Revenue Tracking and Analytics**:

- Reports on ticket sales, ride popularity, and overall revenue generation.

- Insights to help park administrators optimize operations and improve services.

3. **Facility Management**:

- Easy addition, removal, or modification of rides and attractions.

- Maintenance scheduling and monitoring for rides.

**Technical Scope**:

1. **Scalability**:

- Modular design allows for easy addition of new features, rides, and functionalities.

2. **Portability**:

- Can be deployed on different platforms and environments, including local servers or cloud infrastructure.

3. **Security**:

- Incorporates secure data handling practices to protect user and park information.

**Future Enhancements**

1. Integration with advanced payment gateways for cashless transactions.

2. Implementation of real-time monitoring systems using IoT devices for ride safety.

3. Mobile application support for enhanced user accessibility.

4. Augmented and virtual reality modules to provide immersive ride previews.

The FunCity Management System is a scalable and robust solution tailored to address the needs of modern recreational facilities. It not only simplifies management tasks but also enhances the visitor experience, making it a practical and impactful project.

**Key Features of the funcity management System in JAVA**:

1**. User Management**:

- Registration and login functionality for visitors and administrators.

- Secure password handling and user authentication.

2. **Ticket Booking System**:

- Easy booking of tickets for park entry and individual rides.

- Flexible ticket categories.

3. **Ride Scheduling and Reservation**:

- Display of available rides and their timings.

- Option to reserve slots for popular rides in advance.

4. **Payment Integration**:

- Simple payment system with multiple options.

- Generation of digital receipts for transactions.

5**. Administrative Dashboard**:

- Add, update, or remove rides and attractions.

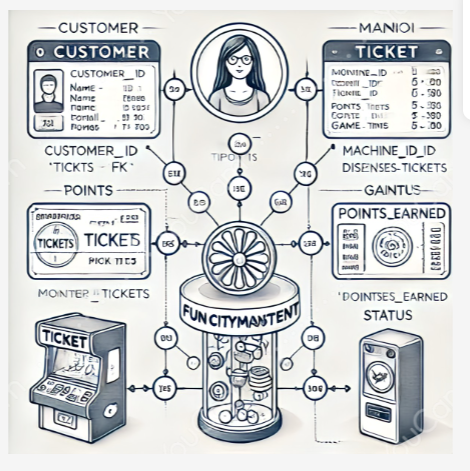
- Manage visitor data and monitor park capacity in real-time.

6. **Real-Time Notifications**:

Alerts for ride availability and scheduling changes.

Notifications for users about upcoming bookings and offers.

**ER DIAGRAM**



**Code implementation:**

**Main:**

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

public class Main extends JFrame {

private JTabbedPane tabbedPane;

public Main() {

setTitle("Fun City Management System");

setSize(600, 500);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

tabbedPane = new JTabbedPane();

tabbedPane.add("Customer Management", new CustomerPanel());

tabbedPane.add("Machine Management", new MachinePanel());

tabbedPane.add("Ticket Management", new TicketPanel());

tabbedPane.add("Points Management", new PointsPanel());

tabbedPane.add("Game Management", new GamePanel());

add(tabbedPane);

}

public static void main(String[] args) {

// Show the login page first

SwingUtilities.invokeLater(() -> new LoginPage().setVisible(true));

}

}

class LoginPage extends JFrame {

private JTextField usernameField;

private JPasswordField passwordField;

private JButton loginButton;

private JLabel messageLabel;

public LoginPage() {

setTitle("Login Page");

setSize(400, 300);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setLayout(new GridBagLayout());

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(5, 5, 5, 5);

JLabel usernameLabel = new JLabel("Username:");

usernameField = new JTextField(20);

JLabel passwordLabel = new JLabel("Password:");

passwordField = new JPasswordField(20);

loginButton = new JButton("Login");

messageLabel = new JLabel("");

gbc.gridx = 0; gbc.gridy = 0; gbc.anchor = GridBagConstraints.EAST;

add(usernameLabel, gbc);

gbc.gridx = 1; gbc.gridy = 0; gbc.anchor = GridBagConstraints.WEST;

add(usernameField, gbc);

gbc.gridx = 0; gbc.gridy = 1; gbc.anchor = GridBagConstraints.EAST;

add(passwordLabel, gbc);

gbc.gridx = 1; gbc.gridy = 1; gbc.anchor = GridBagConstraints.WEST;

add(passwordField, gbc);

gbc.gridx = 1; gbc.gridy = 2; gbc.anchor = GridBagConstraints.CENTER;

add(loginButton, gbc);

gbc.gridx = 1; gbc.gridy = 3; gbc.anchor = GridBagConstraints.CENTER;

add(messageLabel, gbc);

loginButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String username = usernameField.getText();

String password = new String(passwordField.getPassword());

boolean loginSuccess = LoginManagement.validateLogin(username, password);

if (loginSuccess) {

messageLabel.setText("Login Successful");

messageLabel.setForeground(Color.GREEN);

new Main().setVisible(true);

dispose();

} else {

messageLabel.setText("Invalid username or password");

messageLabel.setForeground(Color.RED);

}

}

});

}

}

class LoginManagement {

public static boolean validateLogin(String username, String password) {

return "admin".equals(username) && "password".equals(password);

}

}

class MachinePanel extends JPanel {

private JTextField locationField, ticketCountField;

private JCheckBox dispensesCheckbox;

private JComboBox<String> statusComboBox;

private DefaultTableModel tableModel;

private JTable machineTable;

public MachinePanel() {

setLayout(new BorderLayout(10, 10));

JPanel inputPanel = new JPanel();

inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y\_AXIS));

inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

JPanel locationPanel = createInputPanel("Location:", locationField = new JTextField());

JPanel ticketCountPanel = createInputPanel("Ticket Count:", ticketCountField = new JTextField());

JPanel dispensesPanel = createInputPanel("Dispenses Tickets:", dispensesCheckbox = new JCheckBox());

JPanel statusPanel = createInputPanel("Status:", statusComboBox = new JComboBox<>(new String[]{"Operational", "Maintenance"}));

inputPanel.add(locationPanel);

inputPanel.add(ticketCountPanel);

inputPanel.add(dispensesPanel);

inputPanel.add(statusPanel);

JPanel buttonPanel = new JPanel();

JButton addButton = new JButton("Add Machine");

JButton viewButton = new JButton("View Machines");

buttonPanel.add(addButton);

buttonPanel.add(viewButton);

addButton.addActionListener(new AddMachineAction());

viewButton.addActionListener(new ViewMachinesAction());

inputPanel.add(buttonPanel);

add(inputPanel, BorderLayout.NORTH);

tableModel = new DefaultTableModel(new String[]{"Machine ID", "Location", "Ticket Count", "Dispenses", "Status"}, 0);

machineTable = new JTable(tableModel);

add(new JScrollPane(machineTable), BorderLayout.CENTER);

}

private JPanel createInputPanel(String labelText, JComponent field) {

JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));

JLabel label = new JLabel(labelText);

label.setPreferredSize(new Dimension(100, 20));

field.setPreferredSize(new Dimension(200, 25));

panel.add(label);

panel.add(field);

return panel;

}

private class AddMachineAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

String location = locationField.getText();

int ticketCount = Integer.parseInt(ticketCountField.getText());

boolean dispensesTickets = dispensesCheckbox.isSelected();

String status = (String) statusComboBox.getSelectedItem();

MachineManagement.addMachine(location, ticketCount, dispensesTickets, status);

JOptionPane.showMessageDialog(MachinePanel.this, "Machine added successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

locationField.setText("");

ticketCountField.setText("");

dispensesCheckbox.setSelected(false);

statusComboBox.setSelectedIndex(0);

}

}

private class ViewMachinesAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

tableModel.setRowCount(0);

Object[][] machines = MachineManagement.getMachines();

for (Object[] machine : machines) {

tableModel.addRow(machine);

}

}

}

}

class TicketPanel extends JPanel {

private JTextField customerIdField, machineIdField;

private DefaultTableModel tableModel;

private JTable ticketTable;

public TicketPanel() {

setLayout(new BorderLayout(10, 10));

JPanel inputPanel = new JPanel();

inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y\_AXIS));

inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

JPanel customerIdPanel = createInputPanel("Customer ID:", customerIdField = new JTextField());

JPanel machineIdPanel = createInputPanel("Machine ID:", machineIdField = new JTextField());

inputPanel.add(customerIdPanel);

inputPanel.add(machineIdPanel);

JPanel buttonPanel = new JPanel();

JButton addButton = new JButton("Add Ticket");

JButton viewButton = new JButton("View Tickets");

buttonPanel.add(addButton);

buttonPanel.add(viewButton);

addButton.addActionListener(new AddTicketAction());

viewButton.addActionListener(new ViewTicketsAction());

inputPanel.add(buttonPanel);

add(inputPanel, BorderLayout.NORTH);

tableModel = new DefaultTableModel(new String[]{"Ticket ID", "Customer ID", "Machine ID"}, 0);

ticketTable = new JTable(tableModel);

add(new JScrollPane(ticketTable), BorderLayout.CENTER);

}

private JPanel createInputPanel(String labelText, JTextField field) {

JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));

JLabel label = new JLabel(labelText);

label.setPreferredSize(new Dimension(100, 20));

field.setPreferredSize(new Dimension(200, 25));

panel.add(label);

panel.add(field);

return panel;

}

private class AddTicketAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

int customerId = Integer.parseInt(customerIdField.getText());

int machineId = Integer.parseInt(machineIdField.getText());

TicketManagement.addTicket(customerId, machineId);

JOptionPane.showMessageDialog(TicketPanel.this, "Ticket added successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

customerIdField.setText("");

machineIdField.setText("");

}

}

private class ViewTicketsAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

tableModel.setRowCount(0);

Object[][] tickets = TicketManagement.getTickets();

for (Object[] ticket : tickets) {

tableModel.addRow(ticket);

}

}

}

}

class PointsPanel extends JPanel {

private JTextField customerIdField, pointsField;

private DefaultTableModel tableModel;

private JTable pointsTable;

public PointsPanel() {

setLayout(new BorderLayout(10, 10));

JPanel inputPanel = new JPanel();

inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y\_AXIS));

inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

JPanel customerIdPanel = createInputPanel("Customer ID:", customerIdField = new JTextField());

JPanel pointsPanel = createInputPanel("Points Earned:", pointsField = new JTextField());

inputPanel.add(customerIdPanel);

inputPanel.add(pointsPanel);

JPanel buttonPanel = new JPanel();

JButton addButton = new JButton("Add Points");

JButton viewButton = new JButton("View Points");

buttonPanel.add(addButton);

buttonPanel.add(viewButton);

addButton.addActionListener(new AddPointsAction());

viewButton.addActionListener(new ViewPointsAction());

inputPanel.add(buttonPanel);

add(inputPanel, BorderLayout.NORTH);

tableModel = new DefaultTableModel(new String[]{"Points ID", "Customer ID", "Points"}, 0);

pointsTable = new JTable(tableModel);

add(new JScrollPane(pointsTable), BorderLayout.CENTER);

}

private JPanel createInputPanel(String labelText, JTextField field) {

JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));

JLabel label = new JLabel(labelText);

label.setPreferredSize(new Dimension(100, 20));

field.setPreferredSize(new Dimension(200, 25));

panel.add(label);

panel.add(field);

return panel;

}

private class AddPointsAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

int customerId = Integer.parseInt(customerIdField.getText());

int pointsEarned = Integer.parseInt(pointsField.getText());

PointsManagement.addPoints(customerId, pointsEarned);

JOptionPane.showMessageDialog(PointsPanel.this, "Points added successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

customerIdField.setText("");

pointsField.setText("");

}

}

private class ViewPointsAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

tableModel.setRowCount(0);

Object[][] pointsData = PointsManagement.getPoints();

for (Object[] points : pointsData) {

tableModel.addRow(points);

}

}

}

}

class GamePanel extends JPanel {

private JTextField nameField, locationField, statusField;

private DefaultTableModel tableModel;

private JTable gameTable;

public GamePanel() {

setLayout(new BorderLayout(10, 10));

JPanel inputPanel = new JPanel();

inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y\_AXIS));

inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

JPanel namePanel = createInputPanel("Name:", nameField = new JTextField());

JPanel locationPanel = createInputPanel("Location:", locationField = new JTextField());

JPanel statusPanel = createInputPanel("Status:", statusField = new JTextField());

inputPanel.add(namePanel);

inputPanel.add(locationPanel);

inputPanel.add(statusPanel);

JPanel buttonPanel = new JPanel();

JButton addButton = new JButton("Add Game");

JButton viewButton = new JButton("View Games");

buttonPanel.add(addButton);

buttonPanel.add(viewButton);

addButton.addActionListener(new AddGameAction());

viewButton.addActionListener(new ViewGamesAction());

inputPanel.add(buttonPanel);

add(inputPanel, BorderLayout.NORTH);

tableModel = new DefaultTableModel(new String[]{"Game ID", "Name", "Location", "Status"}, 0);

gameTable = new JTable(tableModel);

JScrollPane scrollPane = new JScrollPane(gameTable);

add(scrollPane, BorderLayout.CENTER);

}

private JPanel createInputPanel(String labelText, JTextField textField) {

JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));

JLabel label = new JLabel(labelText);

label.setPreferredSize(new Dimension(80, 20));

textField.setPreferredSize(new Dimension(200, 25));

panel.add(label);

panel.add(textField);

return panel;

}

private class AddGameAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

String name = nameField.getText().trim();

String location = locationField.getText().trim();

String status = statusField.getText().trim();

if (name.isEmpty() || location.isEmpty() || status.isEmpty()) {

JOptionPane.showMessageDialog(GamePanel.this, "Please fill in all fields.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

GameManagement.addGame(name, location, status);

JOptionPane.showMessageDialog(GamePanel.this, "Game added successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

nameField.setText("");

locationField.setText("");

statusField.setText("");

}

}

private class ViewGamesAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

tableModel.setRowCount(0);

Object[][] games = GameManagement.getGames();

for (Object[] game : games) {

tableModel.addRow(game);

}

}

}

}

class CustomerPanel extends JPanel {

private JTextField nameField;

private JTextField emailField;

private JTextField pointsField;

private JTextField balanceField;

private JTable customerTable;

private DefaultTableModel tableModel;

public CustomerPanel() {

setLayout(new BorderLayout(10, 10));

JPanel inputPanel = new JPanel();

inputPanel.setLayout(new BoxLayout(inputPanel, BoxLayout.Y\_AXIS));

inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); // Adding some padding around the form

JPanel namePanel = createInputPanel("Name:", nameField = new JTextField());

JPanel emailPanel = createInputPanel("Email:", emailField = new JTextField());

JPanel pointsPanel = createInputPanel("Points:", pointsField = new JTextField());

JPanel balancePanel = createInputPanel("Balance:", balanceField = new JTextField());

inputPanel.add(namePanel);

inputPanel.add(emailPanel);

inputPanel.add(pointsPanel);

inputPanel.add(balancePanel);

JPanel buttonPanel = new JPanel();

JButton addButton = new JButton("Add Customer");

JButton viewButton = new JButton("View Customers");

buttonPanel.add(addButton);

buttonPanel.add(viewButton);

addButton.addActionListener(new AddCustomerAction());

viewButton.addActionListener(new ViewCustomersAction());

inputPanel.add(buttonPanel);

add(inputPanel, BorderLayout.NORTH);

tableModel = new DefaultTableModel(new String[]{"Customer ID", "Name", "Email", "Points", "Balance"}, 0);

customerTable = new JTable(tableModel);

JScrollPane scrollPane = new JScrollPane(customerTable);

add(scrollPane, BorderLayout.CENTER);

}

private JPanel createInputPanel(String labelText, JTextField textField) {

JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT));

JLabel label = new JLabel(labelText);

label.setPreferredSize(new Dimension(80, 20));

textField.setPreferredSize(new Dimension(200, 25));

panel.add(label);

panel.add(textField);

return panel;

}

private class AddCustomerAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

String name = nameField.getText().trim();

String email = emailField.getText().trim();

String pointsStr = pointsField.getText().trim();

String balanceStr = balanceField.getText().trim();

if (name.isEmpty() || email.isEmpty() || pointsStr.isEmpty() || balanceStr.isEmpty()) {

JOptionPane.showMessageDialog(CustomerPanel.this, "Please fill in all fields.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

try {

int points = Integer.parseInt(pointsStr);

double balance = Double.parseDouble(balanceStr);

CustomerManagement.addCustomer(name, email, points, balance);

JOptionPane.showMessageDialog(CustomerPanel.this, "Customer added successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

nameField.setText("");

emailField.setText("");

pointsField.setText("");

balanceField.setText("");

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(CustomerPanel.this, "Please enter valid numeric values for points and balance.", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

private class ViewCustomersAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

tableModel.setRowCount(0);

Object[][] customers = CustomerManagement.getCustomers();

for (Object[] customer : customers) {

tableModel.addRow(customer);

}

}

}

}

**Ticket Management:**

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class TicketManagement {

    public static void addTicket(int customerId, int machineId) {

        String sql = "INSERT INTO Ticket (customer\_id, machine\_id) VALUES (?, ?)";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, customerId);

            pstmt.setInt(2, machineId);

            pstmt.executeUpdate();

            System.out.println("Ticket added successfully!");

        } catch (SQLException e) {

            System.out.println("Error adding ticket: " + e.getMessage());

        }

    }

    public static void viewTickets() {

        String sql = "SELECT \* FROM Ticket";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                System.out.println("Ticket ID: " + rs.getInt("ticket\_id"));

                System.out.println("Customer ID: " + rs.getInt("customer\_id"));

                System.out.println("Machine ID: " + rs.getInt("machine\_id"));

                System.out.println("Issue Time: " + rs.getTimestamp("issue\_time"));

                System.out.println("----------------------");

            }

        } catch (SQLException e) {

            System.out.println("Error viewing tickets: " + e.getMessage());

        }

    }

    public static void deleteTicket(int ticketId) {

        String sql = "DELETE FROM Ticket WHERE ticket\_id = ?";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, ticketId);

            pstmt.executeUpdate();

            System.out.println("Ticket deleted successfully!");

        } catch (SQLException e) {

            System.out.println("Error deleting ticket: " + e.getMessage());

        }

    }

    // Method to get all tickets from the database

    public static Object[][] getTickets() {

        String sql = "SELECT \* FROM Ticket";

        List<Object[]> ticketList = new ArrayList<>();

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                Object[] ticket = new Object[]{

                    rs.getInt("ticket\_id"),

                    rs.getInt("customer\_id"),

                    rs.getInt("machine\_id"),

                    rs.getTimestamp("issue\_time")

                };

                ticketList.add(ticket);

            }

        } catch (SQLException e) {

            System.out.println("Error retrieving tickets: " + e.getMessage());

        }

        return ticketList.toArray(new Object[0][]); // Convert list to 2D array

    }

}

**Game Management:**

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class GameManagement {

    public static void addGame(String name, String location, String status) {

        String sql = "INSERT INTO Game (name, location, status) VALUES (?, ?, ?)";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, name);

            pstmt.setString(2, location);

            pstmt.setString(3, status);

            pstmt.executeUpdate();

            System.out.println("Game added successfully!");

        } catch (SQLException e) {

            System.out.println("Error adding game: " + e.getMessage());

        }

    }

    public static void viewGames() {

        String sql = "SELECT \* FROM Game";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                System.out.println("Game ID: " + rs.getInt("game\_id"));

                System.out.println("Name: " + rs.getString("name"));

                System.out.println("Location: " + rs.getString("location"));

                System.out.println("Status: " + rs.getString("status"));

                System.out.println("----------------------");

            }

        } catch (SQLException e) {

            System.out.println("Error viewing games: " + e.getMessage());

        }

    }

    public static Object[][] getGames() {

        String sql = "SELECT \* FROM Game";

        List<Object[]> gameList = new ArrayList<>();

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                Object[] game = new Object[] {

                    rs.getInt("game\_id"),

                    rs.getString("name"),

                    rs.getString("location"),

                    rs.getString("status")

                };

                gameList.add(game);

            }

        } catch (SQLException e) {

            System.out.println("Error retrieving games: " + e.getMessage());

        }

        return gameList.toArray(new Object[0][]);

    }

}

**MachineManagement:**

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class MachineManagement {

    public static void addMachine(String location, int ticketCount, boolean dispensesTickets, String status) {

        String sql = "INSERT INTO Machine (location, ticket\_count, dispenses\_tickets, status) VALUES (?, ?, ?, ?)";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, location);

            pstmt.setInt(2, ticketCount);

            pstmt.setBoolean(3, dispensesTickets);

            pstmt.setString(4, status);

            pstmt.executeUpdate();

            System.out.println("Machine added successfully!");

        } catch (SQLException e) {

            System.out.println("Error adding machine: " + e.getMessage());

        }

    }

    public static void viewMachines() {

        String sql = "SELECT \* FROM Machine";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                System.out.println("Machine ID: " + rs.getInt("machine\_id"));

                System.out.println("Location: " + rs.getString("location"));

                System.out.println("Ticket Count: " + rs.getInt("ticket\_count"));

                System.out.println("Dispenses Tickets: " + rs.getBoolean("dispenses\_tickets"));

                System.out.println("Status: " + rs.getString("status"));

                System.out.println("----------------------");

            }

        } catch (SQLException e) {

            System.out.println("Error viewing machines: " + e.getMessage());

        }

    }

    public static void deleteMachine(int machineId) {

        String sql = "DELETE FROM Machine WHERE machine\_id = ?";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, machineId);

            pstmt.executeUpdate();

            System.out.println("Machine deleted successfully!");

        } catch (SQLException e) {

            System.out.println("Error deleting machine: " + e.getMessage());

        }

    }

    public static Object[][] getMachines() {

        String sql = "SELECT \* FROM Machine";

        List<Object[]> machineList = new ArrayList<>();

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                Object[] machine = new Object[]{

                    rs.getInt("machine\_id"),

                    rs.getString("location"),

                    rs.getInt("ticket\_count"),

                    rs.getBoolean("dispenses\_tickets"),

                    rs.getString("status")

                };

                machineList.add(machine);

            }

        } catch (SQLException e) {

            System.out.println("Error retrieving machines: " + e.getMessage());

        }

        return machineList.toArray(new Object[0][]);

    }

}

**PointsManagement:**

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class PointsManagement {

    public static void addPoints(int customerId, int pointsEarned) {

        String sql = "INSERT INTO Points (customer\_id, points\_earned) VALUES (?, ?)";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, customerId);

            pstmt.setInt(2, pointsEarned);

            pstmt.executeUpdate();

            System.out.println("Points added successfully!");

        } catch (SQLException e) {

            System.out.println("Error adding points: " + e.getMessage());

        }

    }

    // Method to view all points records in the database

    public static void viewPoints() {

        String sql = "SELECT \* FROM Points";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                System.out.println("Point ID: " + rs.getInt("point\_id"));

                System.out.println("Customer ID: " + rs.getInt("customer\_id"));

                System.out.println("Points Earned: " + rs.getInt("points\_earned"));

                System.out.println("Earned Time: " + rs.getTimestamp("earned\_time"));

                System.out.println("----------------------");

            }

        } catch (SQLException e) {

            System.out.println("Error viewing points: " + e.getMessage());

        }

    }

    // Method to delete points by their ID

    public static void deletePoints(int pointId) {

        String sql = "DELETE FROM Points WHERE point\_id = ?";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, pointId);

            pstmt.executeUpdate();

            System.out.println("Points record deleted successfully!");

        } catch (SQLException e) {

            System.out.println("Error deleting points: " + e.getMessage());

        }

    }

    // Method to get all points records from the database

    public static Object[][] getPoints() {

        String sql = "SELECT \* FROM Points";

        List<Object[]> pointsList = new ArrayList<>();

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                Object[] points = new Object[]{

                    rs.getInt("point\_id"),

                    rs.getInt("customer\_id"),

                    rs.getInt("points\_earned"),

                    rs.getTimestamp("earned\_time")

                };

                pointsList.add(points);

            }

        } catch (SQLException e) {

            System.out.println("Error retrieving points records: " + e.getMessage());

        }

        return pointsList.toArray(new Object[0][]); // Convert list to 2D array

    }

}

**Database:**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class Database{

    private static final String URL = "jdbc:mysql://localhost:3306/FunCityManagement";

    private static final String USER = "root";

    private static final String PASSWORD = "12345";

    public static Connection getConnection() {

        Connection conn = null;

        try {

            conn = DriverManager.getConnection(URL, USER, PASSWORD);

            System.out.println("Connected to database!");

        } catch (SQLException e) {

            System.out.println("Error connecting to database: " + e.getMessage());

        }

        return conn;

    }

}

**CustomerManagement:**

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class CustomerManagement {

    public static void addCustomer(String name, String email, int points, double balance) {

        String sql = "INSERT INTO Customer (name, email, points, balance) VALUES (?, ?, ?, ?)";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, name);

            pstmt.setString(2, email);

            pstmt.setInt(3, points);

            pstmt.setDouble(4, balance);

            pstmt.executeUpdate();

            System.out.println("Customer added successfully!");

        } catch (SQLException e) {

            System.out.println("Error adding customer: " + e.getMessage());

        }

    }

    public static void viewCustomers() {

        String sql = "SELECT \* FROM Customer";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                System.out.println("Customer ID: " + rs.getInt("customer\_id"));

                System.out.println("Name: " + rs.getString("name"));

                System.out.println("Email: " + rs.getString("email"));

                System.out.println("Points: " + rs.getInt("points"));

                System.out.println("Balance: " + rs.getDouble("balance"));

                System.out.println("----------------------");

            }

        } catch (SQLException e) {

            System.out.println("Error viewing customers: " + e.getMessage());

        }

**}**

    public static void deleteCustomer(int customerId) {

        String sql = "DELETE FROM Customer WHERE customer\_id = ?";

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, customerId);

            pstmt.executeUpdate();

            System.out.println("Customer deleted successfully!");

        } catch (SQLException e) {

            System.out.println("Error deleting customer: " + e.getMessage());

        }

    }

    public static Object[][] getCustomers() {

        String sql = "SELECT \* FROM Customer";

        List<Object[]> customerList = new ArrayList<>();

        try (Connection conn = Database.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql);

             ResultSet rs = pstmt.executeQuery()) {

            while (rs.next()) {

                Object[] customer = new Object[]{

                    rs.getInt("customer\_id"),

                    rs.getString("name"),

                    rs.getString("email"),

                    rs.getInt("points"),

                    rs.getDouble("balance")

                };

                customerList.add(customer);

            }

        } catch (SQLException e) {

            System.out.println("Error retrieving customers: " + e.getMessage());

        }

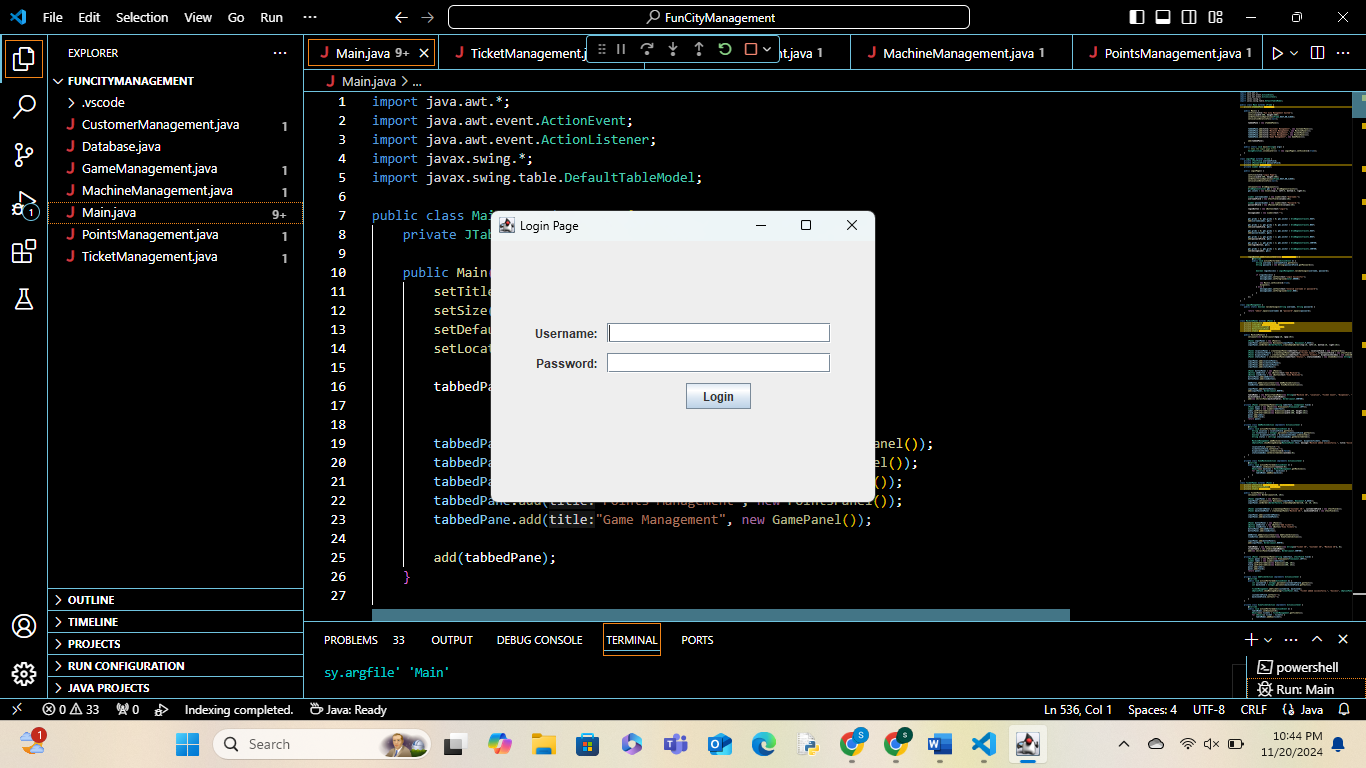
return customerList.toArray(new Object[0][]);

    }

}

**OUTPUT:**

**1)Login Page**

****

**2)Table View**

**A computer screen shot of a computer

Description automatically generated**

**3)Adding Record in the table:**

**A screenshot of a computer

Description automatically generated**

**4)Viewing the record being added**

**A screenshot of a computer

Description automatically generated**

**Conclusion:**

The **FunCity Management System** is a comprehensive Java-based mini-project designed to simplify the operations of an amusement park. By incorporating JDBC connectivity, the system provides seamless interaction with the database, ensuring secure and efficient management of visitor information, ticket bookings, rides, and reward points.

This project demonstrates the effective use of **object-oriented programming principles** combined with **relational database management** to deliver a functional and user-friendly system. The modular design makes it scalable, allowing future enhancements like real-time ride availability tracking, mobile app integration, and advanced reporting functionalities.