

**Group Members: Fatima Khalid (SP24-BSE-132)**

**Aleesha Batool (SP24-BSE-131)**

**Sara Riaz (SP24-BSE-110)**

**Course name: Object Oriented Programming**

**Submitted to: Muhammad Shahid Bhatti**

**Lab Assignment: 04**

**Due Date: 12-12-2024**

**Housing Society Plot Management System:**

**Source code:**

**HelloApplication Class:**

package com.example.hspsm;  
  
import javafx.animation.PauseTransition;  
import javafx.application.Application;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.print.PrinterJob;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.layout.GridPane;  
import javafx.scene.layout.HBox;  
import javafx.scene.layout.VBox;  
import javafx.scene.paint.Color;  
import javafx.scene.text.Font;  
import javafx.scene.text.FontWeight;  
import javafx.scene.text.Text;  
import javafx.scene.text.TextAlignment;  
import javafx.stage.Stage;  
  
import java.time.LocalDate;  
import java.util.\*;  
  
import java.io.\*;  
  
import static com.example.hspsm.Admin.*analyzePlotStatistics*;  
import static com.example.hspsm.Admin.*generateReports*;  
  
public class HelloApplication extends Application{  
 public static int *userCount* = 1;  
 @Override  
 public void start(Stage stage) throws IOException {  
 welcomeScreen(stage);  
 stage.show();  
 }  
  
 public void welcomeScreen(Stage stage) {  
 VBox vBox = new VBox();  
 vBox.setAlignment(Pos.*CENTER*);  
 vBox.setSpacing(30);  
 vBox.setPadding(new Insets(20));  
 vBox.setStyle("-fx-background-color: linear-gradient(to bottom, #34495e, #2c3e50);");  
  
 Text welcome = new Text("Welcome to Housing Society Plot Management System");  
 welcome.setTextAlignment(TextAlignment.*CENTER*);  
 welcome.setFill(Color.*WHITE*);  
 welcome.setStyle("-fx-font-size: 28px; -fx-font-weight: bold; -fx-font-family: Arial;");  
  
  
 Button nextButton = new Button("Next");  
 nextButton.setStyle("-fx-background-color: #1abc9c; -fx-text-fill: white; -fx-padding: 10 20; -fx-font-size: 14px; -fx-font-family: Arial;");  
 nextButton.setOnAction(e -> *loginScreen*(stage));  
  
  
 vBox.getChildren().addAll(welcome, nextButton);  
  
 Scene scene = new Scene(vBox, 800, 600);  
 stage.setScene(scene);  
 stage.setTitle("Welcome Screen");  
 }  
 public static void loginScreen(Stage stage) {  
 VBox vBox = new VBox();  
 vBox.setAlignment(Pos.*CENTER*);  
 vBox.setSpacing(20);  
 vBox.setPadding(new Insets(20));  
 vBox.setStyle("-fx-background-color: #ecf0f1;");  
  
 Label usernameLabel = new Label("Username:");  
 Label passwordLabel = new Label("Password:");  
 usernameLabel.setStyle("-fx-text-fill: #34495e; -fx-font-size: 18px; -fx-font-family: Arial; -fx-font-weight: bold;");  
 passwordLabel.setStyle("-fx-text-fill: #34495e; -fx-font-size: 18px; -fx-font-family: Arial; -fx-font-weight: bold;");  
  
  
 TextField usernameField = new TextField();  
 usernameField.setPromptText("Enter your username");  
 usernameField.setStyle("-fx-font-size: 16px; -fx-padding: 10px; -fx-background-color: #fff; -fx-border-color: #ccc; -fx-border-radius: 5px;");  
  
 PasswordField passwordField = new PasswordField();  
 passwordField.setPromptText("Enter your password");  
 passwordField.setStyle("-fx-font-size: 16px; -fx-padding: 10px; -fx-background-color: #fff; -fx-border-color: #ccc; -fx-border-radius: 5px;");  
  
  
 Button login = new Button("Login");  
 Button admin = new Button("Login as Admin");  
 Button register = new Button("Sign Up");  
  
  
 String buttonStyle = "-fx-background-color: #3498db; -fx-text-fill: white; -fx-padding: 12px 30px; -fx-font-size: 16px; -fx-font-family: Arial; -fx-border-radius: 5px;";  
  
 login.setStyle(buttonStyle);  
 admin.setStyle(buttonStyle);  
 register.setStyle(buttonStyle);  
  
  
 login.setOnMouseEntered(e -> login.setStyle("-fx-background-color: #2980b9; -fx-text-fill: white; -fx-padding: 12px 30px; -fx-font-size: 16px; -fx-font-family: Arial; -fx-border-radius: 5px;"));  
 login.setOnMouseExited(e -> login.setStyle(buttonStyle));  
  
 admin.setOnMouseEntered(e -> admin.setStyle("-fx-background-color: #2980b9; -fx-text-fill: white; -fx-padding: 12px 30px; -fx-font-size: 16px; -fx-font-family: Arial; -fx-border-radius: 5px;"));  
 admin.setOnMouseExited(e -> admin.setStyle(buttonStyle));  
  
 register.setOnMouseEntered(e -> register.setStyle("-fx-background-color: #f39c12; -fx-text-fill: white; -fx-padding: 12px 30px; -fx-font-size: 16px; -fx-font-family: Arial; -fx-border-radius: 5px;"));  
 register.setOnMouseExited(e -> register.setStyle(buttonStyle));  
  
  
 Text invalidMessage = new Text();  
 invalidMessage.setTextAlignment(TextAlignment.*CENTER*);  
 invalidMessage.setFill(Color.*RED*);  
 invalidMessage.setFont(Font.*font*("Arial", FontWeight.*BOLD*, 14));  
  
  
 List<User> users = *loadUsers*();  
 login.setOnAction(e -> {  
 boolean isValidUser = users.stream().anyMatch(user ->  
 user.getUsername().equals(usernameField.getText()) &&  
 user.getPassword().equals(passwordField.getText()));  
  
 if (isValidUser) {  
 *buyerDashboard*(stage);  
 } else {  
 invalidMessage.setText("Invalid Username or Password");  
 }  
 });  
  
 register.setOnAction(e -> *registerUser*(stage));  
 admin.setOnAction(e -> *adminLoginScreen*(stage));  
  
  
 GridPane inputGrid = new GridPane();  
 inputGrid.setAlignment(Pos.*CENTER*);  
 inputGrid.setHgap(10);  
 inputGrid.setVgap(10);  
 inputGrid.add(usernameLabel, 0, 0);  
 inputGrid.add(usernameField, 1, 0);  
 inputGrid.add(passwordLabel, 0, 1);  
 inputGrid.add(passwordField, 1, 1);  
  
  
 vBox.getChildren().addAll(inputGrid, invalidMessage, login, admin, register);  
  
 Scene scene = new Scene(vBox, 400, 450);  
 stage.setScene(scene);  
 stage.setTitle("Login Screen");  
 }  
 public static void adminDashboardScene(Stage stage) {  
  
 VBox vBox = new VBox();  
 vBox.setAlignment(Pos.*CENTER*);  
 vBox.setSpacing(15);  
 vBox.setPadding(new Insets(20));  
  
  
 Label titleLabel = new Label("Admin Dashboard");  
 titleLabel.setStyle("-fx-font-size: 24px; -fx-font-weight: bold; -fx-text-fill: #333333;");  
  
  
 Button manageUsersButton = *createStyledButton*("Manage Users", "#4CAF50");  
 Button managePlotsButton = *createStyledButton*("Manage Plots", "#4CAF50");  
 Button managePaymentsButton = *createStyledButton*("Manage Payments", "#4CAF50");  
 Button generateReportsButton = *createStyledButton*("Generate Reports", "#4CAF50");  
  
 Button logoutButton = *createStyledButton*("Logout", "#F44336");  
  
  
 vBox.getChildren().addAll(  
 titleLabel,  
 manageUsersButton,  
 managePlotsButton,  
 managePaymentsButton,  
 generateReportsButton,  
 logoutButton  
 );  
  
  
 manageUsersButton.setOnAction(e -> *UserManagementScene*(stage));  
 managePlotsButton.setOnAction(e -> *managePlotsScene*(stage));  
 managePaymentsButton.setOnAction(e -> *managePaymentsScene*(stage));  
 generateReportsButton.setOnAction(e -> *generateReportScene*(stage));  
  
 logoutButton.setOnAction(e -> *loginScreen*(stage));  
  
  
 Scene scene = new Scene(vBox, 800, 600);  
 stage.setScene(scene);  
 stage.setTitle("Admin Dashboard");  
 }  
  
  
 private static Button createStyledButton(String text, String color) {  
 Button button = new Button(text);  
 button.setStyle(  
 "-fx-background-color: " + color + "; " +  
 "-fx-text-fill: white; " +  
 "-fx-font-size: 14; " +  
 "-fx-padding: 10 20; " +  
 "-fx-background-radius: 5;"  
 );  
 return button;  
 }  
  
  
 public static void UserManagementScene(Stage stage) {  
 VBox vBox = new VBox();  
 vBox.setAlignment(Pos.*CENTER*);  
 vBox.setSpacing(10);  
  
 Label titleLabel = new Label("User Management");  
 titleLabel.setStyle("-fx-font-size: 20px; -fx-font-weight: bold;");  
 vBox.getChildren().add(titleLabel);  
  
  
 TableView<User> userTable = new TableView<>();  
 userTable.setItems(*loadUsers*());  
  
 TableColumn<User, String> idColumn = new TableColumn<>("User ID");  
 idColumn.setCellValueFactory(new PropertyValueFactory<>("userId"));  
  
 TableColumn<User, String> usernameColumn = new TableColumn<>("Username");  
 usernameColumn.setCellValueFactory(new PropertyValueFactory<>("username"));  
  
 TableColumn<User, String> roleColumn = new TableColumn<>("Role");  
 roleColumn.setCellValueFactory(new PropertyValueFactory<>("role"));  
  
 TableColumn<User, String> emailColumn = new TableColumn<>("Email");  
 emailColumn.setCellValueFactory(new PropertyValueFactory<>("email"));  
  
 TableColumn<User, String> phoneColumn = new TableColumn<>("Phone");  
 phoneColumn.setCellValueFactory(new PropertyValueFactory<>("phoneNumber"));  
  
 userTable.getColumns().addAll(idColumn, usernameColumn, roleColumn, emailColumn, phoneColumn);  
  
  
 Button addButton = new Button("Add User");  
 Button editButton = new Button("Edit User");  
 Button deleteButton = new Button("Delete User");  
  
  
 addButton.setOnAction(e -> {  
 Stage addStage = new Stage();  
 VBox addVBox = new VBox(10);  
 addVBox.setAlignment(Pos.*CENTER*);  
  
 TextField usernameField = new TextField();  
 usernameField.setPromptText("Username");  
  
 PasswordField passwordField = new PasswordField();  
 passwordField.setPromptText("Password");  
  
 TextField roleField = new TextField();  
 roleField.setPromptText("Role (Admin/Buyer)");  
  
 TextField emailField = new TextField();  
 emailField.setPromptText("Email");  
  
 TextField phoneField = new TextField();  
 phoneField.setPromptText("Phone Number");  
  
 Button saveButton = new Button("Save");  
 saveButton.setOnAction(event -> {  
 ObservableList<User> users = userTable.getItems();  
 users.add(new User(usernameField.getText(), passwordField.getText(), roleField.getText(), emailField.getText(), phoneField.getText()));  
 *saveUsers*(users);  
 addStage.close();  
 });  
  
 addVBox.getChildren().addAll(usernameField, passwordField, roleField, emailField, phoneField, saveButton);  
 addStage.setScene(new Scene(addVBox, 300, 400));  
 addStage.setTitle("Add User");  
 addStage.show();  
 });  
  
  
 editButton.setOnAction(e -> {  
 User selectedUser = userTable.getSelectionModel().getSelectedItem();  
 if (selectedUser != null) {  
 Stage editStage = new Stage();  
 VBox editVBox = new VBox(10);  
 editVBox.setAlignment(Pos.*CENTER*);  
  
 TextField emailField = new TextField(selectedUser.getEmail());  
 emailField.setPromptText("Email");  
  
 TextField phoneField = new TextField(selectedUser.getPhoneNumber());  
 phoneField.setPromptText("Phone Number");  
  
 TextField roleField = new TextField(selectedUser.getRole());  
 roleField.setPromptText("Role (Admin/Buyer)");  
  
 Button saveButton = new Button("Save Changes");  
 saveButton.setOnAction(event -> {  
 selectedUser.setEmail(emailField.getText());  
 selectedUser.setPhoneNumber(phoneField.getText());  
 selectedUser.setRole(roleField.getText());  
 userTable.refresh();  
 *saveUsers*(userTable.getItems());  
 editStage.close();  
 });  
  
 editVBox.getChildren().addAll(emailField, phoneField, roleField, saveButton);  
 editStage.setScene(new Scene(editVBox, 300, 300));  
 editStage.setTitle("Edit User");  
 editStage.show();  
 }  
 });  
  
  
 deleteButton.setOnAction(e -> {  
 User selectedUser = userTable.getSelectionModel().getSelectedItem();  
 if (selectedUser != null) {  
 userTable.getItems().remove(selectedUser);  
 *saveUsers*(userTable.getItems());  
 }  
 });  
  
  
 Button backButton = new Button("Back");  
 backButton.setOnAction(e -> *adminDashboardScene*(stage));  
  
  
 HBox buttonBox = new HBox(10, addButton, editButton, deleteButton);  
 buttonBox.setAlignment(Pos.*CENTER*);  
  
 vBox.getChildren().addAll(userTable, buttonBox, backButton);  
  
 Scene scene = new Scene(vBox, 800, 600);  
 stage.setScene(scene);  
 stage.setTitle("User Management");  
 }  
  
 public static void managePlotsScene(Stage stage) {  
 VBox vBox = new VBox();  
 vBox.setAlignment(Pos.*CENTER*);  
 vBox.setSpacing(10);  
  
 Label titleLabel = new Label("Plot Management");  
 titleLabel.setStyle("-fx-font-size: 20px; -fx-font-weight: bold;");  
 vBox.getChildren().add(titleLabel);  
  
  
 TableView<Plot> plotTable = new TableView<>();  
 plotTable.setItems(*loadPlots*());  
  
 TableColumn<Plot, Integer> idColumn = new TableColumn<>("Plot ID");  
 idColumn.setCellValueFactory(new PropertyValueFactory<>("plotId"));  
  
 TableColumn<Plot, String> numberColumn = new TableColumn<>("Plot Number");  
 numberColumn.setCellValueFactory(new PropertyValueFactory<>("plotNumber"));  
  
 TableColumn<Plot, Double> lengthColumn = new TableColumn<>("Length");  
 lengthColumn.setCellValueFactory(new PropertyValueFactory<>("length"));  
  
 TableColumn<Plot, Double> widthColumn = new TableColumn<>("Width");  
 widthColumn.setCellValueFactory(new PropertyValueFactory<>("width"));  
  
 TableColumn<Plot, Double> areaColumn = new TableColumn<>("Total Area");  
 areaColumn.setCellValueFactory(new PropertyValueFactory<>("totalArea"));  
  
 TableColumn<Plot, String> locationColumn = new TableColumn<>("Location");  
 locationColumn.setCellValueFactory(new PropertyValueFactory<>("location"));  
  
 TableColumn<Plot, String> gpsColumn = new TableColumn<>("GPS Coordinates");  
 gpsColumn.setCellValueFactory(new PropertyValueFactory<>("gpsCoordinates"));  
  
 TableColumn<Plot, String> statusColumn = new TableColumn<>("Status");  
 statusColumn.setCellValueFactory(new PropertyValueFactory<>("status"));  
  
 TableColumn<Plot, Double> priceUnitColumn = new TableColumn<>("Price Per Unit");  
 priceUnitColumn.setCellValueFactory(new PropertyValueFactory<>("pricePerUnit"));  
  
 TableColumn<Plot, Double> priceColumn = new TableColumn<>("Total Price");  
 priceColumn.setCellValueFactory(new PropertyValueFactory<>("totalPrice"));  
  
 TableColumn<Plot, String> developmentColumn = new TableColumn<>("Development Status");  
 developmentColumn.setCellValueFactory(new PropertyValueFactory<>("developmentStatus"));  
  
 plotTable.getColumns().addAll(idColumn, numberColumn, lengthColumn, widthColumn, areaColumn, locationColumn, gpsColumn, statusColumn, priceUnitColumn, priceColumn, developmentColumn);  
  
  
 Button addButton = new Button("Add Plot");  
 Button editButton = new Button("Edit Plot");  
 Button deleteButton = new Button("Delete Plot");  
  
  
 addButton.setOnAction(e -> {  
 Stage addStage = new Stage();  
 VBox addVBox = new VBox(10);  
 addVBox.setAlignment(Pos.*CENTER*);  
  
 TextField numberField = new TextField();  
 numberField.setPromptText("Plot Number");  
  
 TextField lengthField = new TextField();  
 lengthField.setPromptText("Length");  
  
 TextField widthField = new TextField();  
 widthField.setPromptText("Width");  
  
 TextField locationField = new TextField();  
 locationField.setPromptText("Location");  
  
 TextField gpsField = new TextField();  
 gpsField.setPromptText("GPS Coordinates");  
  
 TextField statusField = new TextField();  
 statusField.setPromptText("Status");  
  
 TextField priceUnitField = new TextField();  
 priceUnitField.setPromptText("Price Per Unit");  
  
 TextField developmentField = new TextField();  
 developmentField.setPromptText("Development Status");  
  
 Button saveButton = new Button("Save");  
 saveButton.setOnAction(event -> {  
 try {  
 int newId = plotTable.getItems().size() + 1;  
 double length = Double.*parseDouble*(lengthField.getText());  
 double width = Double.*parseDouble*(widthField.getText());  
 double pricePerUnit = Double.*parseDouble*(priceUnitField.getText());  
  
 Plot newPlot = new Plot(newId, numberField.getText(), length, width, locationField.getText(), gpsField.getText(), statusField.getText(), pricePerUnit, developmentField.getText());  
 ObservableList<Plot> plots = plotTable.getItems();  
 plots.add(newPlot);  
 *savePlots*(plots);  
 addStage.close();  
 } catch (NumberFormatException ex) {  
 System.*out*.println("Invalid input values.");  
 }  
 });  
  
 addVBox.getChildren().addAll(numberField, lengthField, widthField, locationField, gpsField, statusField, priceUnitField, developmentField, saveButton);  
 addStage.setScene(new Scene(addVBox, 400, 500));  
 addStage.setTitle("Add Plot");  
 addStage.show();  
 });  
  
  
 editButton.setOnAction(e -> {  
 Plot selectedPlot = plotTable.getSelectionModel().getSelectedItem();  
 if (selectedPlot != null) {  
 Stage editStage = new Stage();  
 VBox editVBox = new VBox(10);  
 editVBox.setAlignment(Pos.*CENTER*);  
  
 TextField numberField = new TextField(selectedPlot.getPlotNumber());  
 numberField.setPromptText("Plot Number");  
  
 TextField lengthField = new TextField(String.*valueOf*(selectedPlot.getLength()));  
 lengthField.setPromptText("Length");  
  
 TextField widthField = new TextField(String.*valueOf*(selectedPlot.getWidth()));  
 widthField.setPromptText("Width");  
  
 TextField locationField = new TextField(selectedPlot.getLocation());  
 locationField.setPromptText("Location");  
  
 TextField gpsField = new TextField(selectedPlot.getGpsCoordinates());  
 gpsField.setPromptText("GPS Coordinates");  
  
 TextField statusField = new TextField(selectedPlot.getStatus());  
 statusField.setPromptText("Status");  
  
 TextField priceUnitField = new TextField(String.*valueOf*(selectedPlot.getPricePerUnit()));  
 priceUnitField.setPromptText("Price Per Unit");  
  
 TextField developmentField = new TextField(selectedPlot.getDevelopmentStatus());  
 developmentField.setPromptText("Development Status");  
  
 Button saveButton = new Button("Save Changes");  
 saveButton.setOnAction(event -> {  
 try {  
 selectedPlot.setPlotNumber(numberField.getText());  
 selectedPlot.setLength(Double.*parseDouble*(lengthField.getText()));  
 selectedPlot.setWidth(Double.*parseDouble*(widthField.getText()));  
 selectedPlot.setLocation(locationField.getText());  
 selectedPlot.setGpsCoordinates(gpsField.getText());  
 selectedPlot.setStatus(statusField.getText());  
 selectedPlot.setPricePerUnit(Double.*parseDouble*(priceUnitField.getText()));  
 selectedPlot.setTotalArea(selectedPlot.getLength() \* selectedPlot.getWidth());  
 selectedPlot.setTotalPrice(selectedPlot.getTotalArea() \* selectedPlot.getPricePerUnit());  
 selectedPlot.setDevelopmentStatus(developmentField.getText());  
  
 plotTable.refresh();  
 *savePlots*(plotTable.getItems());  
 editStage.close();  
 } catch (NumberFormatException ex) {  
 System.*out*.println("Invalid input values.");  
 }  
 });  
  
 editVBox.getChildren().addAll(numberField, lengthField, widthField, locationField, gpsField, statusField, priceUnitField, developmentField, saveButton);  
 editStage.setScene(new Scene(editVBox, 400, 500));  
 editStage.setTitle("Edit Plot");  
 editStage.show();  
 }  
 });  
  
  
 deleteButton.setOnAction(e -> {  
 Plot selectedPlot = plotTable.getSelectionModel().getSelectedItem();  
 if (selectedPlot != null) {  
 plotTable.getItems().remove(selectedPlot);  
 *savePlots*(plotTable.getItems());  
 }  
 });  
  
  
 Button backButton = new Button("Back");  
 backButton.setOnAction(e -> *adminDashboardScene*(stage));  
  
  
 HBox buttonBox = new HBox(10, addButton, editButton, deleteButton);  
 buttonBox.setAlignment(Pos.*CENTER*);  
  
 vBox.getChildren().addAll(plotTable, buttonBox, backButton);  
  
 Scene scene = new Scene(vBox, 1000, 700);  
 stage.setScene(scene);  
 stage.setTitle("Plot Management");  
 }  
  
 public static void managePaymentsScene(Stage stage){  
 VBox layout = new VBox(10);  
 layout.setPadding(new Insets(10));  
  
 Button backButton = new Button("Back");  
 backButton.setOnAction(e -> *adminDashboardScene*(stage));  
  
 ListView<Payment> paymentListView = new ListView<>();  
 paymentListView.getItems().setAll(*loadPayments*());  
  
 Button addPaymentButton = new Button("Add Payment");  
 addPaymentButton.setOnAction(e -> {  
  
 System.*out*.println("Adding a new payment...");  
 });  
  
  
 Button removePaymentButton = new Button("Remove Payment");  
 removePaymentButton.setOnAction(e -> {  
 Payment selectedPayment = paymentListView.getSelectionModel().getSelectedItem();  
 if (selectedPayment != null) {  
 ObservableList<Payment> payments = *loadPayments*();  
 payments.remove(selectedPayment);  
 *savePayments*(payments);  
 paymentListView.getItems().setAll(*loadPayments*());  
 System.*out*.println("Payment removed: " + selectedPayment);  
 } else {  
 System.*out*.println("Please select a payment to remove.");  
 }  
 });  
  
  
 Button updatePaymentButton = new Button("Update Payment");  
 updatePaymentButton.setOnAction(e -> {  
 Payment selectedPayment = paymentListView.getSelectionModel().getSelectedItem();  
 if (selectedPayment != null) {  
  
 System.*out*.println("Updating payment: " + selectedPayment);  
 } else {  
 System.*out*.println("Please select a payment to update.");  
 }  
 });  
  
  
 layout.getChildren().addAll(paymentListView, addPaymentButton, removePaymentButton, updatePaymentButton,backButton);  
  
  
 Scene scene = new Scene(layout, 400, 400);  
 stage.setScene(scene);  
 stage.setTitle("Manage Payments");  
 stage.show();  
 }  
 public static void generateReportScene(Stage stage){  
 VBox layout = new VBox(10);  
 layout.setPadding(new Insets(10));  
  
  
 String report = *generateReports*();  
 String plotStatistics = *analyzePlotStatistics*();  
  
  
 final TextArea reportTextArea = new TextArea(report + "\n\n" + plotStatistics);  
 reportTextArea.setEditable(false);  
 reportTextArea.setWrapText(true);  
  
  
 Button printButton = new Button("Print Report");  
 reportTextArea.setEditable(false);  
  
  
 reportTextArea.setText(report);  
  
 printButton.setOnAction(e -> {  
  
 String contentToPrint = reportTextArea.getText();  
 if (!contentToPrint.isEmpty()) {  
 *print*(contentToPrint);  
 } else {  
 Alert alert = new Alert(Alert.AlertType.*WARNING*, "No report to print!");  
 alert.show();  
 }  
 });  
  
  
 Button closeButton = new Button("Close");  
 closeButton.setOnAction(e -> {  
 stage.close();  
 });  
  
  
 layout.getChildren().addAll(reportTextArea, printButton, closeButton);  
  
  
 Scene scene = new Scene(layout, 600, 400);  
 stage.setScene(scene);  
 stage.setTitle("Generate Report");  
 stage.show();  
 }  
 private static void print(String content) {  
  
 PrinterJob printerJob = PrinterJob.*createPrinterJob*();  
  
 if (printerJob == null) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "No printers found. Please check your printer setup.");  
 alert.show();  
 return;  
 }  
  
  
 Text printableContent = new Text(content);  
 printableContent.setWrappingWidth(500);  
  
  
 boolean proceed = printerJob.showPrintDialog(null);  
  
 if (proceed) {  
  
 boolean success = printerJob.printPage(printableContent);  
  
 if (success) {  
 printerJob.endJob();  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*, "Printing complete.");  
 alert.show();  
 } else {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*, "Failed to print.");  
 alert.show();  
 }  
 } else {  
 // User cancelled the print dialog  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*, "Printing cancelled.");  
 alert.show();  
 }  
 }  
 public static void buyerDashboard(Stage stage) {  
 // Main VBox layout  
 VBox vBox = new VBox();  
 vBox.setAlignment(Pos.*CENTER*);  
 vBox.setSpacing(15);  
 vBox.setPadding(new Insets(20));  
  
 // Create buttons  
 Button viewPlots = new Button("View Available Plots");  
 Button requestPlot = new Button("Request Plot");  
 Button ownershipDetails = new Button("Ownership Details");  
 Button trackPaymentStatus = new Button("Track Payment Status");  
 Button updatePreference = new Button("Update Preference");  
 Button viewMap = new Button("View Map"); // New Map button  
 Button exit = new Button("Exit");  
 Button logout = new Button("Logout");  
  
 // Set button styles  
 String buttonStyle = "-fx-background-color: #4CAF50; -fx-text-fill: white; -fx-padding: 10 20; -fx-font-size: 14;";  
 viewPlots.setStyle(buttonStyle);  
 requestPlot.setStyle(buttonStyle);  
 ownershipDetails.setStyle(buttonStyle);  
 trackPaymentStatus.setStyle(buttonStyle);  
 updatePreference.setStyle(buttonStyle);  
 viewMap.setStyle(buttonStyle); // Style for the new button  
 logout.setStyle("-fx-background-color: #2196F3; -fx-text-fill: white; -fx-padding: 10 20; -fx-font-size: 14;");  
 exit.setStyle("-fx-background-color: #F44336; -fx-text-fill: white; -fx-padding: 10 20; -fx-font-size: 14;");  
  
 // Set button actions  
 viewPlots.setOnAction(e -> *viewPlots*(stage));  
 requestPlot.setOnAction(e -> *requestPlot*(stage));  
 ownershipDetails.setOnAction(e -> *ownershipDetails*(stage));  
 trackPaymentStatus.setOnAction(e -> *trackPaymentStatus*(stage));  
 updatePreference.setOnAction(e -> *updatePreference*(stage));  
  
 viewMap.setOnAction(e -> {  
 stage.setScene(ViewMap.*getMainScene*(stage));  
 });  
  
 logout.setOnAction(e -> {  
 *loginScreen*(stage);  
 });  
 exit.setOnAction(e -> stage.close());  
  
 // Add buttons to VBox  
 vBox.getChildren().addAll(  
 viewPlots,  
 requestPlot,  
 ownershipDetails,  
 trackPaymentStatus,  
 updatePreference,  
 viewMap, // Added the view map button  
 logout,  
 exit  
 );  
  
 // Create scene and set on stage  
 Scene scene = new Scene(vBox, 400, 500);  
 stage.setScene(scene);  
 stage.setTitle("Buyer Dashboard");  
 }  
  
 public static void adminLoginScreen(Stage stage) {  
 // VBox for main layout  
 VBox vBox = new VBox();  
 vBox.setAlignment(Pos.*CENTER*);  
 vBox.setSpacing(20);  
 vBox.setPadding(new Insets(20));  
  
 // Labels and fields  
 Label usernameLabel = new Label("Username: ");  
 Label passwordLabel = new Label("Password: ");  
 TextField usernameField = new TextField();  
 usernameField.setPromptText("Enter admin username");  
 PasswordField passwordField = new PasswordField();  
 passwordField.setPromptText("Enter admin password");  
  
 // Login button  
 Button login = new Button("Login");  
 login.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white; -fx-padding: 10 20;");  
  
 // Error message text  
 Text invalidMessage = new Text();  
 invalidMessage.setTextAlignment(TextAlignment.*CENTER*);  
 invalidMessage.setFill(Color.*RED*);  
 invalidMessage.setFont(Font.*font*("Arial", FontWeight.*BOLD*, 12));  
  
 // Load users and handle admin login  
 User userObj = new User();  
 List<User> users = userObj.loadUsers();  
 login.setOnAction(e -> {  
 boolean isAdmin = "Admin".equals(usernameField.getText()) && "admin".equals(passwordField.getText());  
 if (isAdmin) {  
 *adminDashboardScene*(stage);  
 } else {  
 invalidMessage.setText("Invalid Username or Password");  
 }  
 });  
  
 // Layout for username and password inputs  
 GridPane inputGrid = new GridPane();  
 inputGrid.setAlignment(Pos.*CENTER*);  
 inputGrid.setHgap(10);  
 inputGrid.setVgap(10);  
 inputGrid.add(usernameLabel, 0, 0);  
 inputGrid.add(usernameField, 1, 0);  
 inputGrid.add(passwordLabel, 0, 1);  
 inputGrid.add(passwordField, 1, 1);  
  
 // Add components to the VBox  
 vBox.getChildren().addAll(inputGrid, invalidMessage, login);  
  
 // Scene and stage setup  
 Scene scene = new Scene(vBox, 400, 400);  
 stage.setScene(scene);  
 stage.setTitle("Admin Login");  
 }  
 public static void registerUser(Stage stage) {  
 VBox vBox = new VBox();  
 vBox.setAlignment(Pos.*CENTER*);  
 vBox.setSpacing(15);  
 vBox.setPadding(new Insets(10));  
 vBox.setStyle("-fx-background-color: linear-gradient(to bottom, #8e44ad, #3498db);");  
  
 Label username = new Label("Username:");  
 Label password = new Label("Password:");  
 Label email = new Label("Email:");  
 Label phoneNumber = new Label("Phone Number:");  
 Label preferredLocation = new Label("Preferred Location:");  
 Label preferredSize = new Label("Preferred Size:");  
 Label budget = new Label("Budget:");  
  
 username.setStyle("-fx-text-fill: white; -fx-font-size: 16px; -fx-font-family: Arial;");  
 password.setStyle("-fx-text-fill: white; -fx-font-size: 16px; -fx-font-family: Arial;");  
 email.setStyle("-fx-text-fill: white; -fx-font-size: 16px; -fx-font-family: Arial;");  
 phoneNumber.setStyle("-fx-text-fill: white; -fx-font-size: 16px; -fx-font-family: Arial;");  
 preferredLocation.setStyle("-fx-text-fill: white; -fx-font-size: 16px; -fx-font-family: Arial;");  
 preferredSize.setStyle("-fx-text-fill: white; -fx-font-size: 16px; -fx-font-family: Arial;");  
 budget.setStyle("-fx-text-fill: white; -fx-font-size: 16px; -fx-font-family: Arial;");  
  
 TextField usernameField = new TextField();  
 PasswordField passwordField = new PasswordField();  
 TextField emailField = new TextField();  
 TextField phoneNumberField = new TextField();  
 TextField preferredLocationField = new TextField();  
 TextField preferredSizeField = new TextField();  
 TextField budgetField = new TextField();  
  
 Button register = new Button("Register");  
 register.setStyle("-fx-background-color: #1abc9c; -fx-text-fill: white; -fx-padding: 10 20; -fx-font-size: 14px; -fx-font-family: Arial;");  
  
 register.setOnAction(e -> {  
 List<User> users = *loadUsers*();  
 List<Buyer> buyers = *loadBuyers*();  
 Buyer buyer = new Buyer(usernameField.getText(), passwordField.getText(), "Buyer", emailField.getText(), phoneNumberField.getText(), preferredLocationField.getText(), Double.*parseDouble*(preferredSizeField.getText()), Double.*parseDouble*(budgetField.getText()));  
 buyers.add(buyer);  
 users.add(buyer);  
 *saveBuyers*(FXCollections.*observableArrayList*(buyers));  
 *saveUsers*(FXCollections.*observableArrayList*(users));  
 *buyerDashboard*(stage);  
 });  
  
 vBox.getChildren().addAll(username, usernameField, password, passwordField, email, emailField, phoneNumber, phoneNumberField, preferredLocation, preferredLocationField, preferredSize, preferredSizeField, budget, budgetField, register);  
  
 Scene scene = new Scene(vBox, 500, 600);  
 stage.setScene(scene);  
 stage.setTitle("Register User");  
 }  
  
 private static void viewPlots(Stage stage){  
 VBox vBox = new VBox();  
 Button exit = new Button("Exit");  
 TableView plotTable = new TableView<>();  
 ObservableList<Plot> plots = *loadPlots*();  
  
 // Define columns  
 TableColumn<Plot, Integer> plotIdColumn = new TableColumn<>("Plot ID");  
 plotIdColumn.setCellValueFactory(new PropertyValueFactory<>("plotId"));  
  
 TableColumn<Plot, String> plotNumberColumn = new TableColumn<>("Plot Number");  
 plotNumberColumn.setCellValueFactory(new PropertyValueFactory<>("plotNumber"));  
  
 TableColumn<Plot, Double> lengthColumn = new TableColumn<>("Length");  
 lengthColumn.setCellValueFactory(new PropertyValueFactory<>("length"));  
  
 TableColumn<Plot, Double> widthColumn = new TableColumn<>("Width");  
 widthColumn.setCellValueFactory(new PropertyValueFactory<>("width"));  
  
 TableColumn<Plot, Double> totalAreaColumn = new TableColumn<>("Total Area");  
 totalAreaColumn.setCellValueFactory(new PropertyValueFactory<>("totalArea"));  
  
 TableColumn<Plot, String> locationColumn = new TableColumn<>("Location");  
 locationColumn.setCellValueFactory(new PropertyValueFactory<>("location"));  
  
 TableColumn<Plot, String> gpsCoordinatesColumn = new TableColumn<>("GPS Coordinates");  
 gpsCoordinatesColumn.setCellValueFactory(new PropertyValueFactory<>("gpsCoordinates"));  
  
 TableColumn<Plot, String> statusColumn = new TableColumn<>("Status");  
 statusColumn.setCellValueFactory(new PropertyValueFactory<>("status"));  
  
 TableColumn<Plot, Double> pricePerUnitColumn = new TableColumn<>("Price per Unit");  
 pricePerUnitColumn.setCellValueFactory(new PropertyValueFactory<>("pricePerUnit"));  
  
 TableColumn<Plot, Double> totalPriceColumn = new TableColumn<>("Total Price");  
 totalPriceColumn.setCellValueFactory(new PropertyValueFactory<>("totalPrice"));  
  
 TableColumn<Plot, String> developmentStatusColumn = new TableColumn<>("Development Status");  
 developmentStatusColumn.setCellValueFactory(new PropertyValueFactory<>("developmentStatus"));  
  
 // Add columns to TableView  
 plotTable.getColumns().addAll(  
 plotIdColumn, plotNumberColumn, lengthColumn, widthColumn, totalAreaColumn,  
 locationColumn, gpsCoordinatesColumn, statusColumn, pricePerUnitColumn,  
 totalPriceColumn, developmentStatusColumn  
 );  
  
 // Add sample data  
 ObservableList<Plot> plotData = FXCollections.*observableArrayList*(  
 new Plot(1, "P001", 50.0, 30.0, "Sector A", "27.2046,77.4977", "Available", 2000.0, "Developed"),  
 new Plot(2, "P002", 60.0, 40.0, "Sector B", "28.7041,77.1025", "Available", 2200.0, "Under Development")  
 );  
 for(Plot plot: plotData){  
 if(!(plot.getStatus().equals("Available")))  
 plotData.remove(plot);  
 }  
 plotTable.setItems(plotData);  
 exit.setOnAction(e->{  
 *buyerDashboard*(stage);  
 });  
 vBox.getChildren().addAll(plotTable,exit);  
 Scene scene = new Scene(vBox, 800, 800);  
 stage.setScene(scene);  
 stage.setTitle("Login Screen");  
 }  
 public static void requestPlot(Stage stage) {  
 VBox vBox = new VBox();  
 Label label = new Label("Request a Plot");  
 TextField plotIdField = new TextField();  
 plotIdField.setPromptText("Enter Plot ID");  
 Button submitButton = new Button("Submit");  
 Button backButton = new Button("Back");  
  
 submitButton.setOnAction(e -> {  
 // Placeholder for request plot logic  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*, "Plot request submitted!");  
 alert.showAndWait();  
 ObservableList<Plot> plots = *loadPlots*();  
 for (Plot plot : plots) {  
 if (plot.getPlotId()==Integer.*parseInt*(plotIdField.getText())) {  
 plot.setStatus("Reserved");  
 break;  
 }  
 }  
 *savePlots*(plots);  
 });  
  
 backButton.setOnAction(e -> {  
 *buyerDashboard*(stage);  
 });  
  
 vBox.getChildren().addAll(label, plotIdField, submitButton, backButton);  
 Scene scene = new Scene(vBox, 400, 300);  
 stage.setScene(scene);  
 stage.setTitle("Request Plot");  
 }  
  
 public static void ownershipDetails(Stage stage) {  
 VBox vBox = new VBox();  
 Label label = new Label("Ownership Details");  
 TableView<Document> tableView = new TableView<>();  
  
 // Define columns for the TableView  
 TableColumn<Document, Integer> documentIdColumn = new TableColumn<>("Document ID");  
 documentIdColumn.setCellValueFactory(new PropertyValueFactory<>("documentId"));  
  
 TableColumn<Document, Integer> buyerIdColumn = new TableColumn<>("Buyer ID");  
 buyerIdColumn.setCellValueFactory(new PropertyValueFactory<>("buyerId"));  
  
 TableColumn<Document, Integer> plotIdColumn = new TableColumn<>("Plot ID");  
 plotIdColumn.setCellValueFactory(new PropertyValueFactory<>("plotId"));  
  
 TableColumn<Document, String> documentTypeColumn = new TableColumn<>("Document Type");  
 documentTypeColumn.setCellValueFactory(new PropertyValueFactory<>("documentType"));  
  
 TableColumn<Document, LocalDate> uploadDateColumn = new TableColumn<>("Upload Date");  
 uploadDateColumn.setCellValueFactory(new PropertyValueFactory<>("uploadDate"));  
  
 // Add columns to the TableView  
 tableView.getColumns().addAll(documentIdColumn, buyerIdColumn, plotIdColumn, documentTypeColumn, uploadDateColumn);  
  
 // Load ownership documents and set them in the TableView  
 List<Document> documents = *loadDocuments*();  
 ObservableList<Document> ownershipDocs = FXCollections.*observableArrayList*();  
  
 for (Document doc : documents) {  
 if ("Ownership".equalsIgnoreCase(doc.getDocumentType())) {  
 ownershipDocs.add(doc);  
 }  
 }  
  
 tableView.setItems(ownershipDocs);  
  
 // Back button to return to buyerDashboard  
 Button backButton = new Button("Back");  
 backButton.setOnAction(e -> *buyerDashboard*(stage));  
  
 vBox.getChildren().addAll(label, tableView, backButton);  
 Scene scene = new Scene(vBox, 800, 600);  
 stage.setScene(scene);  
 stage.setTitle("Ownership Details");  
 }  
  
 public static void trackPaymentStatus(Stage stage) {  
 VBox vBox = new VBox();  
 Label label = new Label("Track Payment Status");  
  
 // Input fields for plot ID (to track payments)  
 TextField plotIdField = new TextField();  
 plotIdField.setPromptText("Enter Plot ID");  
  
 Button searchButton = new Button("Search Payments");  
  
 // TableView for displaying payment history  
 TableView<Payment> tableView = new TableView<>();  
  
 // Define columns for the TableView  
 TableColumn<Payment, Integer> paymentIdColumn = new TableColumn<>("Payment ID");  
 paymentIdColumn.setCellValueFactory(new PropertyValueFactory<>("paymentId"));  
  
 TableColumn<Payment, Integer> buyerIdColumn = new TableColumn<>("Buyer ID");  
 buyerIdColumn.setCellValueFactory(new PropertyValueFactory<>("buyerId"));  
  
 TableColumn<Payment, Integer> plotIdColumn = new TableColumn<>("Plot ID");  
 plotIdColumn.setCellValueFactory(new PropertyValueFactory<>("plotId"));  
  
 TableColumn<Payment, Double> amountPaidColumn = new TableColumn<>("Amount Paid");  
 amountPaidColumn.setCellValueFactory(new PropertyValueFactory<>("amountPaid"));  
  
 TableColumn<Payment, Double> outstandingBalanceColumn = new TableColumn<>("Outstanding Balance");  
 outstandingBalanceColumn.setCellValueFactory(new PropertyValueFactory<>("outstandingBalance"));  
  
 TableColumn<Payment, String> paymentMethodColumn = new TableColumn<>("Payment Method");  
 paymentMethodColumn.setCellValueFactory(new PropertyValueFactory<>("paymentMethod"));  
  
 TableColumn<Payment, LocalDate> paymentDateColumn = new TableColumn<>("Payment Date");  
 paymentDateColumn.setCellValueFactory(new PropertyValueFactory<>("paymentDate"));  
  
 // Add columns to the TableView  
 tableView.getColumns().addAll(paymentIdColumn, buyerIdColumn, plotIdColumn, amountPaidColumn, outstandingBalanceColumn, paymentMethodColumn, paymentDateColumn);  
  
 // Action for search button  
 searchButton.setOnAction(e -> {  
 String plotIdText = plotIdField.getText();  
 if (plotIdText.isEmpty()) {  
 *showAlert*("Error", "Please enter a Plot ID.");  
 return;  
 }  
  
 try {  
 int plotId = Integer.*parseInt*(plotIdText);  
  
 // Fetch payment history for the entered plot ID  
 List<Payment> paymentList = new ArrayList<>();  
 List<Payment> payments = *loadPayments*();  
 for(Payment payment: payments){  
 if(payment.getPlotId()==plotId){  
 paymentList.add(payment);  
 }  
 }  
  
 if (paymentList.isEmpty()) {  
 *showAlert*("No Payments Found", "No payment records found for the given Plot ID.");  
 } else {  
 ObservableList<Payment> paymentData = FXCollections.*observableArrayList*(paymentList);  
 tableView.setItems(paymentData);  
 }  
 } catch (NumberFormatException ex) {  
 *showAlert*("Invalid Input", "Please enter a valid Plot ID.");  
 }  
 });  
  
 // Back button to return to buyerDashboard  
 Button backButton = new Button("Back");  
 backButton.setOnAction(e -> *buyerDashboard*(stage));  
  
 vBox.getChildren().addAll(label, plotIdField, searchButton, tableView, backButton);  
 Scene scene = new Scene(vBox, 800, 600);  
 stage.setScene(scene);  
 stage.setTitle("Track Payment Status");  
 }  
  
 private static void showAlert(String title, String message) {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle(title);  
 alert.setHeaderText(null);  
 alert.setContentText(message);  
 alert.showAndWait();  
 }  
  
 public static void updatePreference(Stage stage) {  
 VBox vBox = new VBox();  
 Label label = new Label("Update Preferences");  
  
 // Fields to input preferred location, size, and budget  
 TextField preferredLocationField = new TextField();  
 preferredLocationField.setPromptText("Enter preferred location");  
  
 TextField preferredSizeField = new TextField();  
 preferredSizeField.setPromptText("Enter preferred size (in square meters)");  
  
 TextField budgetField = new TextField();  
 budgetField.setPromptText("Enter budget");  
  
 Button updateButton = new Button("Update");  
 Button backButton = new Button("Back");  
  
 updateButton.setOnAction(e -> {  
 // Retrieve the input values  
 String preferredLocation = preferredLocationField.getText();  
 String preferredSizeText = preferredSizeField.getText();  
 String budgetText = budgetField.getText();  
  
 // Validate the inputs  
 if (preferredLocation.isEmpty() || preferredSizeText.isEmpty() || budgetText.isEmpty()) {  
 *showAlert*("Error", "Please fill all the fields.");  
 return;  
 }  
  
 double preferredSize = 0;  
 double budget = 0;  
  
 try {  
 preferredSize = Double.*parseDouble*(preferredSizeText);  
 budget = Double.*parseDouble*(budgetText);  
 } catch (NumberFormatException ex) {  
 *showAlert*("Error", "Please enter valid numbers for size and budget.");  
 return;  
 }  
  
 // Placeholder: Logic for updating preferences (e.g., storing them in a database)  
 // For now, just show a success message  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*, "Preferences updated successfully!");  
 alert.showAndWait();  
  
 // After updating, go back to the buyer dashboard  
 *buyerDashboard*(stage);  
 });  
  
 backButton.setOnAction(e -> {  
 *buyerDashboard*(stage);  
 });  
  
 vBox.getChildren().addAll(label, preferredLocationField, preferredSizeField, budgetField, updateButton, backButton);  
 Scene scene = new Scene(vBox, 400, 300);  
 stage.setScene(scene);  
 stage.setTitle("Update Preferences");  
 }  
 // Load Users as ObservableList  
 public static ObservableList<User> loadUsers() {  
 ObservableList<User> users = FXCollections.*observableArrayList*();  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Users.ser"))) {  
 List<User> userList = (List<User>) inputStream.readObject();  
 users.addAll(userList); // Add all items to ObservableList  
 *userCount* = users.size() + 1; // Assuming `userCount` is declared elsewhere.  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Users file not found. Starting with an empty list.");  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return users;  
 }  
  
 // Save Users  
 public static void saveUsers(ObservableList<User> users) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Users.ser"))) {  
 outputStream.writeObject(new ArrayList<>(users)); // Convert ObservableList to ArrayList  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Load Buyers as ObservableList  
 public static ObservableList<Buyer> loadBuyers() {  
 ObservableList<Buyer> buyers = FXCollections.*observableArrayList*();  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Buyers.ser"))) {  
 List<Buyer> buyerList = (List<Buyer>) inputStream.readObject();  
 buyers.addAll(buyerList);  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Buyers file not found. Starting with an empty list.");  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return buyers;  
 }  
  
 // Save Buyers  
 public static void saveBuyers(ObservableList<Buyer> buyers) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Buyers.ser"))) {  
 outputStream.writeObject(new ArrayList<>(buyers));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Load Plots as ObservableList  
 public static ObservableList<Plot> loadPlots() {  
 ObservableList<Plot> plots = FXCollections.*observableArrayList*();  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Plots.ser"))) {  
 List<Plot> plotList = (List<Plot>) inputStream.readObject();  
 plots.addAll(plotList);  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Plots file not found. Starting with an empty list.");  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return plots;  
 }  
  
 // Save Plots  
 public static void savePlots(ObservableList<Plot> plots) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Plots.ser"))) {  
 outputStream.writeObject(new ArrayList<>(plots));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Load Payments as ObservableList  
 public static ObservableList<Payment> loadPayments() {  
 ObservableList<Payment> payments = FXCollections.*observableArrayList*();  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Payments.ser"))) {  
 List<Payment> paymentList = (List<Payment>) inputStream.readObject();  
 payments.addAll(paymentList);  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Payments file not found. Starting with an empty list.");  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return payments;  
 }  
  
 // Save Payments  
 public static void savePayments(ObservableList<Payment> payments) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Payments.ser"))) {  
 outputStream.writeObject(new ArrayList<>(payments));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Load Documents as ObservableList  
 public static ObservableList<Document> loadDocuments() {  
 ObservableList<Document> documents = FXCollections.*observableArrayList*();  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Documents.ser"))) {  
 List<Document> documentList = (List<Document>) inputStream.readObject();  
 documents.addAll(documentList);  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Documents file not found. Starting with an empty list.");  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return documents;  
 }  
  
 // Save Documents  
 public static void saveDocuments(ObservableList<Document> documents) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Documents.ser"))) {  
 outputStream.writeObject(new ArrayList<>(documents));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
  
}

**ViewMap:**

package com.example.hspsm;  
  
import javafx.geometry.Insets;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.GridPane;  
import javafx.scene.layout.Pane;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
import javafx.scene.text.Text;  
import javafx.stage.Stage;  
  
import java.util.Random;  
  
public class ViewMap {  
  
 public static Scene getMainScene(Stage primaryStage) {  
 MainScene mainScene = new MainScene(primaryStage);  
 return mainScene.getScene();  
 }  
}  
  
class MainScene {  
 private final Scene scene;  
  
 public MainScene(Stage primaryStage) {  
 GridPane blockPane = new GridPane();  
 blockPane.setHgap(15);  
 blockPane.setVgap(15);  
 blockPane.setPadding(new Insets(20, 20, 20, 20));  
  
 for (int i = 1; i <= 5; i++) {  
 Button block = new Button("Welcome User \nMap View\nBlock " + i);  
 block.setPrefSize(180, 180);  
 block.setStyle("-fx-font-size: 14px; -fx-font-weight: bold; " +  
 "-fx-background-color: linear-gradient(to bottom, #4facfe, #00f2fe); " +  
 "-fx-text-fill: white; " +  
 "-fx-background-radius: 20px; " +  
 "-fx-border-radius: 20px; " +  
 "-fx-padding: 10px;");  
  
 block.setOnAction(e -> {  
 PlotScene plotScene = new PlotScene(primaryStage);  
 primaryStage.setScene(plotScene.getScene());  
 });  
  
 block.setOnMouseEntered(e -> block.setStyle("-fx-font-size: 14px; -fx-font-weight: bold; " +  
 "-fx-background-color: linear-gradient(to bottom, #00f2fe, #4facfe); " +  
 "-fx-text-fill: white; " +  
 "-fx-background-radius: 20px; " +  
 "-fx-border-radius: 20px; " +  
 "-fx-padding: 10px;"));  
 block.setOnMouseExited(e -> block.setStyle("-fx-font-size: 14px; -fx-font-weight: bold; " +  
 "-fx-background-color: linear-gradient(to bottom, #4facfe, #00f2fe); " +  
 "-fx-text-fill: white; " +  
 "-fx-background-radius: 20px; " +  
 "-fx-border-radius: 20px; " +  
 "-fx-padding: 10px;"));  
  
 blockPane.add(block, (i - 1) % 3, (i - 1) / 3);  
 }  
  
 this.scene = new Scene(blockPane, 600, 500);  
 }  
  
 public Scene getScene() {  
 return scene;  
 }  
}  
  
class PlotScene {  
 private final Scene scene;  
  
 public PlotScene(Stage primaryStage) {  
 Pane plotPane = new Pane();  
 int plotSize = 150;  
 int rows = 4;  
 int cols = 6;  
 int roadWidth = 25;  
  
 Random random = new Random();  
 Color[] colors = {Color.*LIGHTGREEN*, Color.*LIGHTBLUE*, Color.*LIGHTCORAL*, Color.*LIGHTYELLOW*, Color.*LIGHTPINK*};  
  
 plotPane.setStyle("-fx-background-color: #F0F0F0;");  
  
 for (int row = 0; row < rows; row++) {  
 for (int col = 0; col < cols; col++) {  
 int plotNumber = row \* cols + col + 1;  
  
 double x = col \* plotSize;  
 double y = row \* (plotSize + roadWidth);  
  
 Rectangle plot = new Rectangle(x, y, plotSize, plotSize);  
 plot.setFill(colors[random.nextInt(colors.length)]);  
 plot.setStroke(Color.*BLACK*);  
 plot.setArcHeight(10);  
 plot.setArcWidth(10);  
  
 String dimensionsText = "Plot " + plotNumber + "\nW: " + plotSize + " H: " + plotSize;  
 Text plotText = new Text(x + plotSize / 4, y + plotSize / 2 + 10, dimensionsText);  
 plotText.setStyle("-fx-font-family: Arial; -fx-font-size: 14px; -fx-font-weight: bold; -fx-fill: #333333;");  
  
 plotPane.getChildren().addAll(plot, plotText);  
 }  
  
 if (row < rows - 1) {  
 Rectangle hRoad = new Rectangle(0, (row + 1) \* (plotSize + roadWidth) - roadWidth, cols \* plotSize, roadWidth);  
 hRoad.setFill(Color.*GRAY*);  
 plotPane.getChildren().add(hRoad);  
 }  
 }  
  
 this.scene = new Scene(plotPane, cols \* plotSize, rows \* (plotSize + roadWidth));  
 }  
  
 public Scene getScene() {  
 return scene;  
 }  
}

**Admin Class:**

package com.example.hspsm;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
import java.util.Scanner;  
  
import static com.example.hspsm.Payment.*PaymentFileName*;  
import static com.example.hspsm.Plot.*PlotFileName*;  
  
public class Admin extends User{  
 private String adminId;  
 private static int *adminCount* = 1;  
  
 public Admin(String email, String phoneNumber) {  
 super("Admin", "admin", "Admin", email, phoneNumber);  
 this.adminId = String.*format*("admin%04d",*adminCount*++);  
 }  
  
 public String getAdminId() {  
 return adminId;  
 }  
  
 public void setAdminId(String adminId) {  
 this.adminId = adminId;  
 }  
  
 public static void addPlot(Plot plot){  
 List<Plot> plots = *loadPlots*();  
 plots.add(plot);  
 *savePlots*(plots);  
 }  
  
 public static void removePlot(int plotId){  
 List<Plot> plots = *loadPlots*();  
 plots.removeIf(plot -> plot.getPlotId()==plotId);  
 *savePlots*(plots);  
 System.*out*.println("Plot removed successfully");  
 }  
  
 public static String generateReports(){  
 List<Plot> plots = *loadPlots*();  
 List<Payment> payments = *loadPayments*();  
  
 int totalPlots = plots.size();  
 int soldPlots = 0;  
 int availablePlots = 0;  
 double totalRevenue = 0;  
 double popularArea = 0;  
 int maxCount = 0;  
 List<Double> soldPlotAreas = new ArrayList<>();  
  
 for (Plot plot : plots) {  
 if ("Sold".equalsIgnoreCase(plot.getStatus())) {  
 soldPlots++;  
 soldPlotAreas.add(plot.getTotalArea());  
 } else if ("Available".equalsIgnoreCase(plot.getStatus())) {  
 availablePlots++;  
 }  
 }  
  
 for (Payment payment : payments) {  
 totalRevenue += payment.getAmountPaid();  
 }  
  
 for (double area : soldPlotAreas) {  
 int count = Collections.*frequency*(soldPlotAreas, area);  
 if (count > maxCount) {  
 maxCount = count;  
 popularArea = area;  
 }  
 }  
  
 return String.*format*(  
 "--- Report ---\n" +  
 "Total Plots: %d\n" +  
 "Sold Plots: %d\n" +  
 "Available Plots: %d\n" +  
 "Popular Plot Area: %.2f sq. meters\n" +  
 "Total Revenue: $%.2f\n",  
 totalPlots, soldPlots, availablePlots, popularArea, totalRevenue  
 );  
 }  
 public static String analyzePlotStatistics() {  
 List<Plot> plots = *loadPlots*();  
 int totalPlots = plots.size();  
 int soldPlots = 0;  
 int availablePlots = 0;  
 List<Double> plotAreasSold = new ArrayList<>();  
 for (Plot plot : plots) {  
 if ("Sold".equals(plot.getStatus())) {  
 soldPlots++;  
 plotAreasSold.add(plot.getTotalArea());  
 } else if ("Available".equals(plot.getStatus())) {  
 availablePlots++;  
 }  
 }  
 double popularArea = 0;  
 int maxCount = 0;  
 for (double area : plotAreasSold) {  
 int count = Collections.*frequency*(plotAreasSold, area);  
 if (count > maxCount) {  
 maxCount = count;  
 popularArea = area;  
 }  
 }  
 return String.*format*("Plot Statistics:\nTotal Plots: %d\nSold Plots: %d\nAvailable Plots: %d\nPopular Plot Area: %.2f",  
 totalPlots, soldPlots, availablePlots, popularArea);  
 }  
 public static void updatePlotDetails(Plot plot){  
 plot.updatePlotDetails();  
 }  
  
// public static void manageUsers(){  
// Scanner scanner = new Scanner(System.in);  
// List<User> users = loadUsers();  
//  
// while (true) {  
// System.out.println("\n--- User Management Menu ---");  
// System.out.println("1. Add New User");  
// System.out.println("2. Delete User");  
// System.out.println("3. Update User");  
// System.out.println("4. View All Users");  
// System.out.println("5. Search User by ID");  
// System.out.println("6. Exit");  
// System.out.print("Enter your choice: ");  
// int choice = scanner.nextInt();  
//  
// switch (choice) {  
// case 1 -> addUser(users, scanner);  
// case 2 -> deleteUser(users, scanner);  
// case 3 -> updateUser(users, scanner);  
// case 4 -> viewAllUsers(users);  
// case 5 -> searchUser(users, scanner);  
// case 6 -> {  
// saveUsers(users);  
// System.out.println("Exiting User Management.");  
// return;  
// }  
// default -> System.out.println("Invalid choice! Please try again.");  
// }  
// }  
// }  
//  
// private static void addUser(List<User> users, Scanner scanner) {  
// scanner.nextLine(); // Consume newline  
// System.out.print("Enter Username: ");  
// String username = scanner.nextLine();  
// System.out.print("Enter Password: ");  
// String password = scanner.nextLine();  
// System.out.print("Enter Role (Admin/Buyer): ");  
// String role = scanner.nextLine();  
// System.out.print("Enter Email: ");  
// String email = scanner.nextLine();  
// System.out.print("Enter Phone Number: ");  
// String phoneNumber = scanner.nextLine();  
//  
// users.add(new User(username, password, role, email, phoneNumber));  
// System.out.println("User added successfully!");  
// }  
//  
// private static void deleteUser(List<User> users, Scanner scanner) {  
// System.out.print("Enter User ID to delete: ");  
// String userId = scanner.nextLine();  
//  
// boolean found = users.removeIf(user -> user.getUserId().equals(userId));  
// if (found) {  
// System.out.println("User deleted successfully!");  
// } else {  
// System.out.println("User not found!");  
// }  
// }  
//  
// private static void updateUser(List<User> users, Scanner scanner) {  
// System.out.print("Enter User ID to update: ");  
// String userId = scanner.nextLine();  
// scanner.nextLine(); // Consume newline  
//  
// for (User user : users) {  
// if (user.getUserId().equals(userId)) {  
// System.out.print("Enter new Email (leave blank to keep unchanged): ");  
// String email = scanner.nextLine();  
// if (!email.isBlank()) user.setEmail(email);  
//  
// System.out.print("Enter new Phone Number (leave blank to keep unchanged): ");  
// String phoneNumber = scanner.nextLine();  
// if (!phoneNumber.isBlank()) user.setPhoneNumber(phoneNumber);  
//  
// System.out.print("Enter new Role (leave blank to keep unchanged): ");  
// String role = scanner.nextLine();  
// if (!role.isBlank()) user.setRole(role);  
//  
// System.out.println("User updated successfully!");  
// return;  
// }  
// }  
// System.out.println("User not found!");  
// }  
//  
// private static void viewAllUsers(List<User> users) {  
// if (users.isEmpty()) {  
// System.out.println("No users to display.");  
// return;  
// }  
// System.out.println("\n--- All Users ---");  
// for (User user : users) {  
// System.out.println("ID: " + user.getUserId() +  
// "\nUsername: " + user.getUsername() +  
// "\nRole: " + user.getRole() +  
// "\nEmail: " + user.getEmail() +  
// "\nPhone: " + user.getPhoneNumber() +  
// "\nRegistered On: " + user.getRegistrationDate());  
// }  
// }  
//  
// private static void searchUser(List<User> users, Scanner scanner) {  
// System.out.print("Enter User ID to search: ");  
// String userId = scanner.nextLine();  
//  
// for (User user : users) {  
// if (user.getUserId().equals(userId)) {  
// System.out.println("\nUser Details:");  
// System.out.println("ID: " + user.getUserId());  
// System.out.println("Username: " + user.getUsername());  
// System.out.println("Role: " + user.getRole());  
// System.out.println("Email: " + user.getEmail());  
// System.out.println("Phone: " + user.getPhoneNumber());  
// System.out.println("Registered On: " + user.getRegistrationDate());  
// return;  
// }  
// }  
// System.out.println("User not found!");  
// }  
  
// public static List<User> loadUsers() {  
// List<User> users = null;  
// try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(UserFileName))) {  
// users = (List<User>) inputStream.readObject();  
// } catch (FileNotFoundException e) {  
// System.out.println("Users file not found. Starting with an empty list.");  
// users = new ArrayList<>();  
// } catch (IOException | ClassNotFoundException e) {  
// e.printStackTrace();  
// }  
// return users;  
// }  
//  
// public static void saveUsers(List<User> users) {  
// try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(UserFileName))) {  
// outputStream.writeObject(users);  
// } catch (IOException e) {  
// e.printStackTrace();  
// }  
// }  
 public static List<Payment> loadPayments(){  
 List<Payment> payments = null;  
 try(ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(*PaymentFileName*))){  
 payments=(List<Payment>) inputStream.readObject();  
 }  
 catch (IOException e){  
 e.printStackTrace();  
 }  
 catch (ClassNotFoundException e){  
 e.printStackTrace();  
 }  
 return payments;  
 }  
 public static List<Plot> loadPlots() {  
 List<Plot> plots = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(*PlotFileName*))) {  
 plots = (List<Plot>) inputStream.readObject();  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Users file not found. Starting with an empty list.");  
 plots = new ArrayList<>();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return plots;  
 }  
  
 public static void savePlots(List<Plot> plots) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(*PlotFileName*))) {  
 outputStream.writeObject(plots);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Buyer Class:**

package com.example.hspsm;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
import static com.example.hspsm.Document.*DocumentFileName*;  
import static com.example.hspsm.Payment.*PaymentFileName*;  
import static com.example.hspsm.Plot.*PlotFileName*;  
  
public class Buyer extends User implements Serializable {  
 public static String *BuyerFileName*= "Buyers.ser";  
 private static int *buyerIdCounter* = 0;  
 private int buyerId;  
 private String preferredLocation;  
 private double preferredSize;  
 private double budget;  
  
 public Buyer(String username, String password, String role, String email, String phoneNumber, String preferredLocation, double preferredSize, double budget) {  
 super(username, password, role, email, phoneNumber);  
 this.buyerId = ++*buyerIdCounter*;  
 this.preferredLocation = preferredLocation;  
 this.preferredSize = preferredSize;  
 this.budget = budget;  
 }  
  
 public double getBudget() {  
 return budget;  
 }  
  
 public void setBudget(double budget) {  
 this.budget = budget;  
 }  
  
 public int getBuyerId() {  
 return buyerId;  
 }  
  
 public void setBuyerId(int buyerId) {  
 this.buyerId = buyerId;  
 }  
  
 public String getPreferredLocation() {  
 return preferredLocation;  
 }  
  
 public void setPreferredLocation(String preferredLocation) {  
 this.preferredLocation = preferredLocation;  
 }  
  
 public double getPreferredSize() {  
 return preferredSize;  
 }  
  
 public void setPreferredSize(double preferredSize) {  
 this.preferredSize = preferredSize;  
 }  
  
 public List<Plot> viewAvailablePlots() {  
 List<Plot> plots = loadPlots();  
 List<Plot> availablePlots = new ArrayList<>();  
 for (Plot plot : plots) {  
 if (plot.getStatus().equals("Available"))  
 availablePlots.add(plot);  
 }  
 return availablePlots;  
 }  
  
 public void requestPlot(int plotId) {  
 List<Plot> plots = loadPlots();  
 for (Plot plot : plots) {  
 if (plot.getPlotId() == plotId) {  
 plot.setStatus("Reserved");  
 break;  
 }  
 }  
 savePlots(plots);  
  
 }  
  
 public List<String> trackPaymentStatus() {  
 List<Payment> payments = loadPayments();  
 List<String> buyerPayment = new ArrayList<>();  
 for (Payment payment : payments) {  
 if (payment.getBuyerId() == buyerId) {  
 buyerPayment.add(payment.toString());  
 }  
 }  
 return buyerPayment;  
 }  
  
 public List<String> getOwnershipDetails() {  
 List<String> ownershipDetails = new ArrayList<>();  
 List<Document> documents = loadDocuments();  
 for (Document document : documents) {  
 if (document.getBuyerId() == this.buyerId)  
 ownershipDetails.add(document.toString());  
 }  
 return ownershipDetails;  
 }  
  
 private List<Plot> loadPlots() {  
 List<Plot> plots = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(*PlotFileName*))) {  
 plots = (List<Plot>) inputStream.readObject();  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Plots file not found. Starting with an empty list.");  
 plots = new ArrayList<>();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return plots;  
 }  
  
 private void savePlots(List<Plot> plots) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(*PlotFileName*))) {  
 outputStream.writeObject(plots);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private List<Document> loadDocuments() {  
 List<Document> documents = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(*DocumentFileName*))) {  
 documents = (List<Document>) inputStream.readObject();  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Document file not found. Starting with an empty list.");  
 documents = new ArrayList<>();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return documents;  
 }  
  
 public List<Payment> loadPayments() {  
 List<Payment> payments = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(*PaymentFileName*))) {  
 payments = (List<Payment>) inputStream.readObject();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return payments;  
 }  
  
 public List<Buyer> loadBuyers() {  
 List<Buyer> buyers = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(*BuyerFileName*))) {  
 buyers = (List<Buyer>) inputStream.readObject();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return buyers;  
 }  
  
 public void saveBuyers(List<Buyer> buyers) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(*BuyerFileName*))) {  
 outputStream.writeObject(buyers);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Analytics Class:**

package com.example.hspsm;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
  
public class Analytics {  
  
 private int totalPlotsSold;  
 private double totalRevenue;  
 private int remainingPlots;  
 private String popularPlotSize;  
 private List<Buyer> frequentBuyers;  
  
 public Analytics(int totalPlotsSold, double totalRevenue, int remainingPlots, String popularPlotSize, List<Buyer> frequentBuyers) {  
 this.totalPlotsSold = totalPlotsSold;  
 this.totalRevenue = totalRevenue;  
 this.remainingPlots = remainingPlots;  
 this.popularPlotSize = popularPlotSize;  
 this.frequentBuyers = frequentBuyers;  
 }  
  
 public List<Buyer> getFrequentBuyers() {  
 return frequentBuyers;  
 }  
  
 public void setFrequentBuyers(List<Buyer> frequentBuyers) {  
 this.frequentBuyers = frequentBuyers;  
 }  
  
 public String getPopularPlotSize() {  
 return popularPlotSize;  
 }  
  
 public void setPopularPlotSize(String popularPlotSize) {  
 this.popularPlotSize = popularPlotSize;  
 }  
  
 public int getRemainingPlots() {  
 return remainingPlots;  
 }  
  
 public void setRemainingPlots(int remainingPlots) {  
 this.remainingPlots = remainingPlots;  
 }  
  
 public int getTotalPlotsSold() {  
 return totalPlotsSold;  
 }  
  
 public void setTotalPlotsSold(int totalPlotsSold) {  
 this.totalPlotsSold = totalPlotsSold;  
 }  
  
 public double getTotalRevenue() {  
 return totalRevenue;  
 }  
  
 public void setTotalRevenue(double totalRevenue) {  
 this.totalRevenue = totalRevenue;  
 }  
  
 public String generateSalesReport(){  
 List<Payment> payments = loadPayments();  
 double totalRevenue = 0;  
 List<Integer> soldPlots = new ArrayList<>();  
 for(Payment payment: payments){  
 totalRevenue+= payment.getAmountPaid();  
 if(!(soldPlots.contains(payment.getPlotId())))  
 soldPlots.add(payment.getPlotId());  
 }  
 return String.*format*("Sales Report\nTotal Revenue: %f\nTotal Plots Sold: %d\n", totalRevenue,soldPlots.size());  
 }  
 public String analyzePlotStatistics(){  
 List<Plot> plots = loadPlots();  
 int totalPlots = plots.size();  
 int soldPlots = 0;  
 int availablePlots=0;  
 List<Double> plotAreasSold = new ArrayList<>();  
 for(Plot plot: plots){  
 if("Sold".equals(plot.getStatus())){  
 soldPlots++;  
 plotAreasSold.add(plot.getTotalArea());  
 } else if ("Available".equals(plot.getStatus())) {  
 availablePlots++;  
 }  
 }  
 double popularArea = 0;  
 int maxCount =0;  
 for(double area: plotAreasSold){  
 int count = Collections.*frequency*(plotAreasSold, area);  
 if(count> maxCount){  
 maxCount = count;  
 popularArea = area;  
 }  
 }  
 return String.*format*("Plot Statistics:\nTotal Plots: %d\nSold Plots: %d\nAvailable Plots: %d\nPopular Plot Area: %.2f",  
 totalPlots, soldPlots, availablePlots, popularArea);  
 }  
 public List<Buyer> getBuyerActivity(){  
 List<Payment> payments = loadPayments();  
 List<Buyer> buyers = loadBuyers();  
 List<Buyer> frequentBuyers = new ArrayList<>();  
 List<Integer> paymentCounts = new ArrayList<>();  
  
 for(Buyer buyer: buyers){  
 int count = 0;  
 for(Payment payment: payments){  
 if(payment.getBuyerId()==buyer.getBuyerId()){  
 count++;  
 }  
 }  
 paymentCounts.add(count);  
 }  
  
 for(int i =0; i<paymentCounts.size(); i++){  
 for(int j = i+1; j<paymentCounts.size(); j++){  
 if(paymentCounts.get(j)>paymentCounts.get(i)){  
 int tempCount = paymentCounts.get(i);  
 paymentCounts.set(i, paymentCounts.get(j));  
 paymentCounts.set(j, tempCount);  
  
 Buyer tempBuyer = buyers.get(i);  
 buyers.set(i, buyers.get(j));  
 buyers.set(j, tempBuyer);  
 }  
 }  
 frequentBuyers.addAll(buyers);  
 }  
 return frequentBuyers;  
 }  
 private List<Plot> loadPlots() {  
 List<Plot> plots = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Plots.ser"))) {  
 plots = (List<Plot>) inputStream.readObject();  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Plots file not found. Starting with an empty list.");  
 plots = new ArrayList<>();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return plots;  
 }  
 public List<Payment> loadPayments(){  
 List<Payment> payments = null;  
 try(ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Payment.ser"))){  
 payments=(List<Payment>) inputStream.readObject();  
 }  
 catch (IOException e){  
 e.printStackTrace();  
 }  
 catch (ClassNotFoundException e){  
 e.printStackTrace();  
 }  
 return payments;  
 }  
  
 public void savePayments(List<Payment> payments){  
 try(ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Payment.ser"))){  
 outputStream.writeObject(payments);  
 }  
 catch (IOException e){  
 e.printStackTrace();  
 }  
 }  
 public List<Buyer> loadBuyers() {  
 List<com.example.hspsm.Buyer> buyers = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Buyer.ser"))) {  
 buyers = (List<com.example.hspsm.Buyer>) inputStream.readObject();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return buyers;  
 }  
}

**Document Class:**

package com.example.hspsm;  
  
import java.io.\*;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Document implements Serializable {  
 public static String *DocumentFileName* = "Documents.ser";  
 private int documentId;  
 private int plotId;  
 private int buyerId;  
 private String documentType;// ownership, sales agreement  
 private LocalDate uploadDate;  
  
 public Document(int documentId, int plotId, int buyerId, String documentType, LocalDate uploadDate) {  
 this.documentId = documentId;  
 this.plotId = plotId;  
 this.buyerId = buyerId;  
 this.documentType = documentType;  
 this.uploadDate = uploadDate;  
 }  
  
 public int getDocumentId() {  
 return documentId;  
 }  
  
 public void setDocumentId(int documentId) {  
 this.documentId = documentId;  
 }  
  
 public String getDocumentType() {  
 return documentType;  
 }  
  
 public void setDocumentType(String documentType) {  
 this.documentType = documentType;  
 }  
  
 public int getPlotId() {  
 return plotId;  
 }  
  
 public void setPlotId(int plotId) {  
 this.plotId = plotId;  
 }  
  
 public LocalDate getUploadDate() {  
 return uploadDate;  
 }  
  
 public void setUploadDate(LocalDate uploadDate) {  
 this.uploadDate = uploadDate;  
 }  
  
 public int getBuyerId() {  
 return buyerId;  
 }  
  
 public void setBuyerId(int buyerId) {  
 this.buyerId = buyerId;  
 }  
  
 public void uploadDocument(){  
 List<Document> documents =loadDocuments();  
 documents.add(this);  
 saveDocuments(documents);  
 }  
  
 public String viewDocument(int documentId) {  
 List<Document> documents = loadDocuments();  
 for (Document document : documents) {  
 if (document.getDocumentId() == documentId) {  
 return document.toString();  
 }  
 }  
 return "Document Not Found";  
 }  
 public void deleteDocument(int documentId){  
 List<Document> documents = loadDocuments();  
 for (Document document : documents) {  
 if (document.getDocumentId() == documentId) {  
 documents.remove(document);  
 break;  
 }  
 }  
 saveDocuments(documents);  
 }  
  
  
 @Override  
 public String toString() {  
 return "Document Details"+  
 "\nDocument Id: " + documentId +  
 "\nBuyer Id: " + buyerId +  
 "\nPlot Id:" + plotId +  
 "\nDocument Type='" + documentType +  
 "\nUpload Date=" + uploadDate +  
 '}';  
 }  
  
 public List<Document> loadDocuments() {  
 List<Document> documents = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Documents.ser"))) {  
 documents = (List<Document>) inputStream.readObject();  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Document file not found. Starting with an empty list.");  
 documents = new ArrayList<>();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return documents;  
 }  
  
 public void saveDocuments(List<Document> plots) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Documents.ser"))) {  
 outputStream.writeObject(plots);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Map Class:**

package com.example.hspsm;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Map implements Serializable {  
  
 private int mapId;  
 private String societyLayout;  
 private List<Plot> plotMarkers;  
  
 public Map(int mapId, String societyLayout, List<Plot> plotMarkers) {  
 this.mapId = mapId;  
 this.societyLayout = societyLayout;  
 this.plotMarkers = plotMarkers;  
 }  
  
 public int getMapId() {  
 return mapId;  
 }  
  
 public void setMapId(int mapId) {  
 this.mapId = mapId;  
 }  
  
 public List<Plot> getPlotMarkers() {  
 return plotMarkers;  
 }  
  
 public void setPlotMarkers(List<Plot> plotMarkers) {  
 this.plotMarkers = plotMarkers;  
 }  
  
 public String getSocietyLayout() {  
 return societyLayout;  
 }  
  
 public void setSocietyLayout(String societyLayout) {  
 this.societyLayout = societyLayout;  
 }  
  
 public static Map loadMap(){  
 Map map = null;  
 try(ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Maps.ser"))){  
 map = (Map) inputStream.readObject();  
 }  
 catch (FileNotFoundException e){  
 System.*out*.println("Map File not Found, Creating a new Map");  
 map = new Map(1, "Default Layout", new ArrayList<>());  
 }  
 catch (IOException e){  
 e.printStackTrace();  
 }  
 catch (ClassNotFoundException e){  
 e.printStackTrace();  
 }  
 return map;  
 }  
  
 public static void markPlot(int plotId, String status){  
 Map map = *loadMap*();  
 if(map ==null){  
 System.*out*.println("Map not loaded.");  
 return;  
 }  
 boolean found = false;  
 for(Plot plot: map.getPlotMarkers()){  
 if(plot.getPlotId()==plotId){  
 plot.setStatus(status);  
 found = true;  
 break;  
 }  
 }  
 if(found){  
 *saveMap*(map);  
 System.*out*.println("Plot ID " + plotId + " marked as " + status + ".");  
 } else {  
 System.*out*.println("Plot ID " + plotId + " not found.");  
 }  
 }  
  
 public static Plot getPlotDetailsFromMap(String location){  
 Map map = *loadMap*();  
 if (map == null) {  
 System.*out*.println("Map not loaded.");  
 return null;  
 }  
  
 for (Plot plot : map.getPlotMarkers()) {  
 if (plot.getLocation().equalsIgnoreCase(location)) {  
 return plot;  
 }  
 }  
  
 System.*out*.println("No plot found at location: " + location);  
 return null;  
 }  
 private static void saveMap(Map map) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Map.ser"))) {  
 outputStream.writeObject(map);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Notification Class:**

package com.example.hspsm;  
  
import java.io.Serializable;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.io.\*;  
  
public class Notification implements Serializable {  
 private int notificationId;  
 private int userId;  
 private String message;  
 private LocalDate notificationDate;  
 private String status;// read, unread  
  
 public Notification(int notificationId, int userId, String message, String status) {  
 this.notificationId = notificationId;  
 this.userId = userId;  
 this.message = message;  
 this.notificationDate = LocalDate.*now*();  
 this.status = status;  
 }  
  
 public String getMessage() {  
 return message;  
 }  
  
 public void setMessage(String message) {  
 this.message = message;  
 }  
  
 public LocalDate getNotificationDate() {  
 return notificationDate;  
 }  
  
 public void setNotificationDate(LocalDate notificationDate) {  
 this.notificationDate = notificationDate;  
 }  
  
 public int getNotificationId() {  
 return notificationId;  
 }  
  
 public void setNotificationId(int notificationId) {  
 this.notificationId = notificationId;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
  
 public int getUserId() {  
 return userId;  
 }  
  
 public void setUserId(int userId) {  
 this.userId = userId;  
 }  
  
 public static void sendNotification(int userId, String message){  
 List<Notification> notifications = *loadNotifications*();  
 int newNotificationId = notifications.isEmpty() ? 1 : notifications.get(notifications.size() - 1).getNotificationId() + 1;  
  
 Notification newNotification = new Notification(newNotificationId, userId, message, "unread");  
 notifications.add(newNotification);  
 *saveNotifications*(notifications);  
  
 System.*out*.println("Notification sent successfully to User ID: " + userId);  
 }  
  
 public static List<Notification> getNotifications(int userId){  
 List<Notification> notifications = *loadNotifications*();  
 List<Notification> userNotifications = new ArrayList<>();  
  
 for (Notification notification : notifications) {  
 if (notification.getUserId() == userId) {  
 userNotifications.add(notification);  
 }  
 }  
  
 return userNotifications;  
 }  
  
 public static void markAsRead(int notificationId){  
 List<Notification> notifications = *loadNotifications*();  
 boolean found = false;  
  
 for (Notification notification : notifications) {  
 if (notification.getNotificationId() == notificationId) {  
 notification.setStatus("read");  
 found = true;  
 break;  
 }  
 }  
  
 if (found) {  
 *saveNotifications*(notifications);  
 System.*out*.println("Notification ID " + notificationId + " marked as read.");  
 } else {  
 System.*out*.println("Notification ID " + notificationId + " not found.");  
 }  
 }  
 private static List<Notification> loadNotifications() {  
 List<Notification> notifications = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Notifications.ser"))) {  
 notifications = (List<Notification>) inputStream.readObject();  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Notifications file not found. Starting with an empty list.");  
 notifications = new ArrayList<>();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return notifications;  
 }  
  
 // Helper method to save notifications to the file  
 private static void saveNotifications(List<Notification> notifications) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Notifications.ser"))) {  
 outputStream.writeObject(notifications);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Payment Class:**

package com.example.hspsm;  
  
import java.io.\*;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Payment {  
 public static String *PaymentFileName*= "Payments.ser";  
 private static int *paymentCount* = 0;  
 private int paymentId;  
 private int plotId;  
 private int buyerId;  
 private LocalDate paymentDate;  
 private double amountPaid;  
 private String paymentMethod;// cash, bank transfer, card  
 private double outstandingBalance;  
  
 public Payment(int plotId, int buyerId, double amountPaid, String paymentMethod) {  
 this.paymentId=++*paymentCount*;  
 this.plotId = plotId;  
 this.buyerId = buyerId;  
 this.paymentDate = LocalDate.*now*();  
 this.amountPaid = amountPaid;  
 this.paymentMethod = paymentMethod;  
 this.outstandingBalance = getOutstandingBalance();  
 }  
  
 public LocalDate getPaymentDate() {  
 return paymentDate;  
 }  
  
 public void setPaymentDate(LocalDate paymentDate) {  
 this.paymentDate = paymentDate;  
 }  
  
 public int getPaymentId() {  
 return paymentId;  
 }  
  
 public void setPaymentId(int paymentId) {  
 this.paymentId = paymentId;  
 }  
  
 public String getPaymentMethod() {  
 return paymentMethod;  
 }  
  
 public void setPaymentMethod(String paymentMethod) {  
 this.paymentMethod = paymentMethod;  
 }  
  
 public int getPlotId() {  
 return plotId;  
 }  
  
 public void setPlotId(int plotId) {  
 this.plotId = plotId;  
 }  
  
 public double getAmountPaid() {  
 return amountPaid;  
 }  
  
 public void setAmountPaid(double amountPaid) {  
 this.amountPaid = amountPaid;  
 }  
  
 public int getBuyerId() {  
 return buyerId;  
 }  
  
 public void setBuyerId(int buyerId) {  
 this.buyerId = buyerId;  
 }  
  
 public double getOutstandingBalance() {  
 return outstandingBalance;  
 }  
  
 public void setOutstandingBalance(double outstandingBalance) {  
 this.outstandingBalance = outstandingBalance;  
 }  
  
 public void recordPayment(){  
 List<Payment> payments = loadPayments();  
 payments.add(this);  
 savePayments(payments);  
  
 }  
  
 public List<Payment> trackPaymentHistory(int plotId){  
 List<Payment> paymentList = new ArrayList<>();  
 List<Payment> payments = loadPayments();  
 for(Payment payment: payments){  
 if(payment.getPlotId()==plotId){  
 paymentList.add(payment);  
 }  
 }  
 return paymentList;  
  
 }  
  
 public double getOutstandingBalance(int plotId){  
 List<Payment> payments = loadPayments();  
 double balance=0;  
 for(Payment payment: payments){  
 if(payment.getPlotId()==plotId){  
 balance= payment.getOutstandingBalance();  
 }  
 }  
 return balance;  
 }  
 public List<Payment> loadPayments(){  
 List<Payment> payments = null;  
 try(ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Payments.ser"))){  
 payments=(List<Payment>) inputStream.readObject();  
 }  
 catch (IOException e){  
 e.printStackTrace();  
 }  
 catch (ClassNotFoundException e){  
 e.printStackTrace();  
 }  
 return payments;  
 }  
  
 public void savePayments(List<Payment> payments){  
 try(ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Payments.ser"))){  
 outputStream.writeObject(payments);  
 }  
 catch (IOException e){  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 public String toString(){  
 return String.*format*("Payment ID: %d\nBuyer ID: %d\nPlot ID: %d\nPaid Amount: %.2f\nOutstanding Balance: %.2f\nPayment Method: %s\nPayment Date: %s",paymentId,buyerId,plotId,amountPaid,outstandingBalance,paymentMethod,paymentDate);  
 }  
}

**Plot Class:**

package com.example.hspsm;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Plot implements Serializable{  
 public static String *PlotFileName*= "Plots.ser";  
 private int plotId;  
 private String plotNumber;  
 private double length;  
 private double width;  
 private double totalArea;  
 private String location;  
 private String gpsCoordinates;  
 private String status;// Available, Reserved, Sold  
 private double pricePerUnit;  
 private double totalPrice;  
 private String developmentStatus; //Developed, Undeveloped  
  
 public Plot(int plotId, String plotNumber, double length, double width, String location, String gpsCoordinates, String status, double pricePerUnit, String developmentStatus) {  
 this.plotId = plotId;  
 this.plotNumber = plotNumber;  
 this.length = length;  
 this.width = width;  
 this.totalArea = calculateArea();  
 this.location = location;  
 this.gpsCoordinates = gpsCoordinates;  
 this.status = status;  
 this.pricePerUnit = pricePerUnit;  
 this.totalPrice = calculateTotalPrice();  
 this.developmentStatus = developmentStatus;  
 }  
  
 public String getDevelopmentStatus() {  
 return developmentStatus;  
 }  
  
 public void setDevelopmentStatus(String developmentStatus) {  
 this.developmentStatus = developmentStatus;  
 }  
  
 public String getGpsCoordinates() {  
 return gpsCoordinates;  
 }  
  
 public void setGpsCoordinates(String gpsCoordinates) {  
 this.gpsCoordinates = gpsCoordinates;  
 }  
  
 public double getLength() {  
 return length;  
 }  
  
 public void setLength(double length) {  
 this.length = length;  
 }  
  
 public String getLocation() {  
 return location;  
 }  
  
 public void setLocation(String location) {  
 this.location = location;  
 }  
  
 public int getPlotId() {  
 return plotId;  
 }  
  
 public void setPlotId(int plotId) {  
 this.plotId = plotId;  
 }  
  
 public String getPlotNumber() {  
 return plotNumber;  
 }  
  
 public void setPlotNumber(String plotNumber) {  
 this.plotNumber = plotNumber;  
 }  
  
 public double getPricePerUnit() {  
 return pricePerUnit;  
 }  
  
 public void setPricePerUnit(double pricePerUnit) {  
 this.pricePerUnit = pricePerUnit;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
  
 public double getTotalArea() {  
 return totalArea;  
 }  
  
 public void setTotalArea(double totalArea) {  
 this.totalArea = totalArea;  
 }  
  
 public double getTotalPrice() {  
 return totalPrice;  
 }  
  
 public void setTotalPrice(double totalPrice) {  
 this.totalPrice = totalPrice;  
 }  
  
 public double getWidth() {  
 return width;  
 }  
  
 public void setWidth(double width) {  
 this.width = width;  
 }  
  
 public double calculateArea(){  
 return length\*width;  
  
 }  
  
 public void updatePlotDetails(){  
 List<Plot> plots = loadPlots();  
 if(plots!=null){  
 for(Plot plot: plots){  
 if(plot.getPlotId()==this.plotId){  
 plot.setPlotNumber(this.plotNumber);  
 plot.setLength(this.length);  
 plot.setWidth(this.width);  
 plot.setPricePerUnit(this.pricePerUnit);  
 plot.setDevelopmentStatus(this.developmentStatus);  
 plot.setGpsCoordinates(this.gpsCoordinates);  
 break;  
 }  
 }  
 savePlots(plots);  
 }  
 }  
  
 public void changeStatus(String newStatus) {  
 List<Plot> plots = loadPlots();  
 if (plots != null) {  
 for (Plot plot : plots) {  
 if (plot.getPlotId() == this.plotId) {  
 plot.setStatus(newStatus);  
 break;  
 }  
 }  
 savePlots(plots);  
 }  
 }  
  
 public void getPlotDetails() {  
 List<Plot> plots = loadPlots();  
  
 if (plots != null) {  
 for (Plot plot : plots) {  
 if (plot.getPlotId() == this.plotId) {  
 System.*out*.println("Plot Details");  
 System.*out*.println("Plot Number: "+plot.getPlotNumber());  
 System.*out*.println("Length: "+plot.getLength());  
 System.*out*.println("Width: "+plot.getWidth());  
 System.*out*.println("Total Area: "+plot.getTotalArea());  
 System.*out*.println("Location: "+plot.getLocation());  
 System.*out*.println("GPS Coordinates: "+plot.getGpsCoordinates());  
 System.*out*.println("Price Per Unit: "+ plot.getPricePerUnit());  
 System.*out*.println("Total Price: "+plot.getTotalPrice());  
 System.*out*.println("Development Status: "+plot.getDevelopmentStatus());  
 System.*out*.println("Status: "+plot.getStatus());  
 return;  
 }  
 else  
 System.*out*.println("Plot not Found");  
 }  
 }  
 }  
 private List<Plot> loadPlots() {  
 List<Plot> plots = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Plots.ser"))) {  
 plots = (List<Plot>) inputStream.readObject();  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Users file not found. Starting with an empty list.");  
 plots = new ArrayList<>();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return plots;  
 }  
  
 private void savePlots(List<Plot> plots) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Plots.ser"))) {  
 outputStream.writeObject(plots);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public double calculateTotalPrice(){  
 return pricePerUnit\*totalArea;  
 }  
 @Override  
 public String toString() {  
 return String.*format*("Plot Number: %s\nLength: %f\nWidth: %f\nTotal Area: %f\nLocation: %s\nGPS Coordinates: %s\nPrice Per Unit: %f\nTotal Price: %f\nDevelopment Status: %s\nStatus: %s", plotNumber, length, width, totalArea, location, gpsCoordinates, pricePerUnit, totalPrice, developmentStatus, status);  
  
 }  
}

**Reservation Class:**

package com.example.hspsm;  
  
import java.io.\*;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Reservation {  
 private int reservationId;  
 private int plotId;  
 private int buyerId;  
 private LocalDate reservationDate;  
 private String status; // reserved , cancelled  
  
 public Reservation(int reservationId, int plotId, int buyerId, LocalDate reservationDate, String status) {  
 this.reservationId = reservationId;  
 this.plotId = plotId;  
 this.buyerId = buyerId;  
 this.reservationDate = reservationDate;  
 this.status = status;  
 }  
  
 public int getBuyerId() {  
 return buyerId;  
 }  
  
 public void setBuyerId(int buyerId) {  
 this.buyerId = buyerId;  
 }  
  
 public int getPlotId() {  
 return plotId;  
 }  
  
 public void setPlotId(int plotId) {  
 this.plotId = plotId;  
 }  
  
 public LocalDate getReservationDate() {  
 return reservationDate;  
 }  
  
 public void setReservationDate(LocalDate reservationDate) {  
 this.reservationDate = reservationDate;  
 }  
  
 public int getReservationId() {  
 return reservationId;  
 }  
  
 public void setReservationId(int reservationId) {  
 this.reservationId = reservationId;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
  
 public boolean reservePlot(int plotId){  
 boolean reserved =false;  
 List<Plot> plots = loadPlots();  
 for(Plot plot: plots){  
 if(plot.getPlotId()== plotId){  
 plot.setStatus("Reserved");  
 reserved = true;  
 savePlots(plots);  
 break;  
 }  
 }  
 return reserved;  
 }  
  
 public void cancelReservation(){  
 List<Plot> plots = loadPlots();  
 for(Plot plot: plots){  
 if(plot.getPlotId()== this.plotId){  
 plot.setStatus("Available");  
 savePlots(plots);  
 break;  
 }  
 }  
 }  
  
 public void getReservationDetails(){  
 List<Plot> plots = loadPlots();  
 for(Plot plot: plots){  
 if(plot.getStatus().equals("Reserved"))  
 System.*out*.println(plot);  
 }  
  
 }  
 private List<Plot> loadPlots() {  
 List<Plot> plots = null;  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream("Plots.ser"))) {  
 plots = (List<Plot>) inputStream.readObject();  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Users file not found. Starting with an empty list.");  
 plots = new ArrayList<>();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return plots;  
 }  
  
 private void savePlots(List<Plot> plots) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream("Plots.ser"))) {  
 outputStream.writeObject(plots);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**User Class:**

package com.example.hspsm;  
  
import java.io.\*;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
public class User implements Serializable {  
 public static String *UserFileName* = "Users.ser";  
 public static int *userCount* = 1;  
 private String userId;  
 private String username;  
 private String password;  
 private String role;//(admin, buyer, sales, visitor)  
 private String email;  
 private String phoneNumber;  
 private LocalDate registrationDate;  
  
 public User(String username, String password, String role, String email, String phoneNumber) {  
 this.userId= String.*format*("%05d",*userCount*++);  
 this.username = username;  
 this.password = password;  
 this.role = role;  
 this.email = email;  
 this.phoneNumber = phoneNumber;  
 this.registrationDate = LocalDate.*now*();  
 }  
 public User(){  
  
 }  
  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public String getPhoneNumber() {  
 return phoneNumber;  
 }  
  
 public void setPhoneNumber(String phoneNumber) {  
 this.phoneNumber = phoneNumber;  
 }  
  
 public LocalDate getRegistrationDate() {  
 return registrationDate;  
 }  
  
 public void setRegistrationDate(LocalDate registrationDate) {  
 this.registrationDate = registrationDate;  
 }  
  
 public String getRole() {  
 return role;  
 }  
  
 public void setRole(String role) {  
 this.role = role;  
 }  
  
 public String getUserId() {  
 return userId;  
 }  
  
 public void setUserId(String userId) {  
 this.userId = userId;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public void registerUser(){  
 List<User> users = loadUsers();  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter Username: ");  
 String username = scanner.nextLine();  
 System.*out*.print("Enter Password: ");  
 String password = scanner.nextLine();  
 System.*out*.print("Enter Role (Admin/Buyer): ");  
 String role = scanner.nextLine();  
 System.*out*.print("Enter Email: ");  
 String email = scanner.nextLine();  
 System.*out*.print("Enter Phone Number: ");  
 String phoneNumber = scanner.nextLine();  
 for(User user: users) {  
 if (user.getUsername().equals(this.username)) {  
 System.*out*.println("Username already exists. Registration failed.");  
 return;  
 }  
 }  
 User user = new User(username,password,role,email,phoneNumber);  
 users.add(user);  
 saveUsers(users);  
 }  
  
 public boolean loginUser(){  
 boolean found = false;  
 List<User> users = loadUsers();  
 for(User user: users){  
 if(username.equals(user.username)&& password.equals(user.password)){  
 found = true;  
 break;  
 }  
 }  
 return found;  
 }  
  
 public void updateProfile(){  
 List<User> users = loadUsers();  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter Username: ");  
 String username = scanner.nextLine();  
 System.*out*.print("Enter Password: ");  
 String password = scanner.nextLine();  
 System.*out*.print("Enter Role (Admin/Buyer): ");  
 String role = scanner.nextLine();  
 System.*out*.print("Enter Email: ");  
 String email = scanner.nextLine();  
 System.*out*.print("Enter Phone Number: ");  
 String phoneNumber = scanner.nextLine();  
 for(User user: users){  
 if(user.getUserId().equals(this.userId)){  
 user.setEmail(email);  
 user.setPhoneNumber(phoneNumber);  
 user.setPassword(password);  
 user.setUsername(username);  
 user.setRole(role);  
 break;  
 }  
 }  
 saveUsers(users);  
 }  
  
 public void viewProfile(){  
 List<User> users = loadUsers();  
 boolean found = false;  
 for(User user: users){  
 if(user.getUserId().equals(this.userId)){  
 System.*out*.println("User Profile");  
 System.*out*.println("UserName: "+user.getUsername());  
 System.*out*.println("Email: "+user.getEmail());  
 System.*out*.println("Phone Number: "+user.getPhoneNumber());  
 System.*out*.println("Role: "+user.getRole());  
 System.*out*.println("Registration Date: "+user.getRegistrationDate());  
 found = true;  
 break;  
 }  
 }  
 if(!found)  
 System.*out*.println("User not Found");  
 }  
 public List<User> loadUsers() {  
 List<User> users = new ArrayList<>();  
 try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(*UserFileName*))) {  
 users = (ArrayList<User>) inputStream.readObject();  
 *userCount*=users.size()+1;  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Users file not found. Starting with an empty list.");  
 users = new ArrayList<>();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 catch(NullPointerException e){  
 users = new ArrayList<>();  
 }  
 return users;  
 }  
  
 public void saveUsers(List<User> users) {  
 try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(*UserFileName*))) {  
 outputStream.writeObject(users);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}