**10.a) Cryptocurrency Transaction Ledger**

**Code :**

import java.io.\*;

import java.util.\*;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

// CryptoTransaction class implementing Serializable

class CryptoTransaction implements Serializable {

private String walletID;

private double transactionAmount;

private String timestamp;

public CryptoTransaction(String walletID, double transactionAmount) {

this.walletID = walletID;

this.transactionAmount = transactionAmount;

this.timestamp = getCurrentTimestamp();

}

private String getCurrentTimestamp() {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

return LocalDateTime.now().format(formatter);

}

public String toString() {

return "Wallet ID: " + walletID + ", Amount: " + transactionAmount + ", Timestamp: " + timestamp;

}

}

public class CryptoLedgerList {

static final String FILE\_NAME = "transactions\_list.dat";

// Method to add a transaction to the file

public static void addTransaction(CryptoTransaction transaction) {

ArrayList<CryptoTransaction> transactions = readTransactions();

transactions.add(transaction);

saveTransactions(transactions);

System.out.println("Transaction added successfully.\n");

}

// Method to save the transaction list to file

public static void saveTransactions(ArrayList<CryptoTransaction> transactions) {

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE\_NAME))) {

oos.writeObject(transactions);

} catch (IOException e) {

e.printStackTrace();

}

}

// Method to read all transactions from file

public static ArrayList<CryptoTransaction> readTransactions() {

ArrayList<CryptoTransaction> transactions = new ArrayList<>();

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE\_NAME))) {

transactions = (ArrayList<CryptoTransaction>) ois.readObject();

} catch (FileNotFoundException e) {

// Ignore if file doesn't exist yet

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

return transactions;

}

// Method to display all transactions

public static void displayTransactions() {

ArrayList<CryptoTransaction> transactions = readTransactions();

if (transactions.isEmpty()) {

System.out.println("No transactions found.\n");

} else {

for (CryptoTransaction t : transactions) {

System.out.println(t);

}

System.out.println();

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int choice;

do {

System.out.println("=== Crypto Transaction Ledger ===");

System.out.println("1. Add New Transaction");

System.out.println("2. View All Transactions");

System.out.println("3. Exit");

System.out.print("Enter your choice: ");

choice = sc.nextInt();

sc.nextLine(); // Consume newline

switch (choice) {

case 1:

System.out.print("Enter Wallet ID: ");

String walletID = sc.nextLine();

System.out.print("Enter Transaction Amount: ");

double amount = sc.nextDouble();

CryptoTransaction transaction = new CryptoTransaction(walletID, amount);

addTransaction(transaction);

break;

case 2:

displayTransactions();

break;

case 3:

System.out.println("Exiting program.");

break;

default:

System.out.println("Invalid choice! Try again.");

}

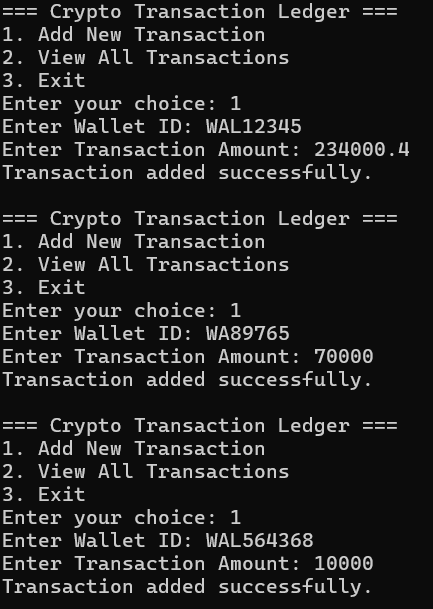
} while (choice != 3);

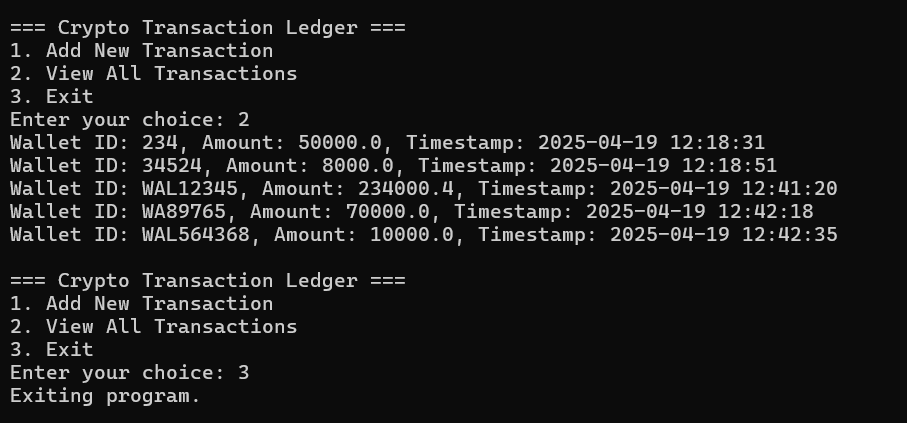
sc.close();

}

}

**Output:**

****

****

**10.b) Digital Certificate Management System**

**Code :**

import java.io.\*;

import java.util.\*;

// DigitalCertificate class implementing Serializable

class DigitalCertificate implements Serializable {

private static final long serialVersionUID = 1L;

private String holderName;

private String certificateID;

private String expiryDate;

public DigitalCertificate(String holderName, String certificateID, String expiryDate) {

this.holderName = holderName;

this.certificateID = certificateID;

this.expiryDate = expiryDate;

}

public String getHolderName() {

return holderName;

}

public String getCertificateID() {

return certificateID;

}

public String getExpiryDate() {

return expiryDate;

}

public void displayCertificate() {

System.out.println("Holder Name : " + holderName);

System.out.println("Certificate ID: " + certificateID);

System.out.println("Expiry Date : " + expiryDate);

}

}

public class DigitalCertificateManager {

private static final String FILE\_NAME = "certificates.dat";

// Method to save a certificate by updating the list

public static void saveCertificate(DigitalCertificate cert) {

List<DigitalCertificate> certList = new ArrayList<>();

// If file exists, load existing certificates first

File file = new File(FILE\_NAME);

if (file.exists()) {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {

certList = (List<DigitalCertificate>) ois.readObject();

} catch (IOException | ClassNotFoundException e) {

certList = new ArrayList<>();

}

}

// Add new cert and write back entire list

certList.add(cert);

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE\_NAME))) {

oos.writeObject(certList);

System.out.println("\nCertificate saved successfully!\n");

} catch (IOException e) {

System.out.println("Error saving certificate.");

e.printStackTrace();

}

}

// Method to load and display all certificates

public static void loadCertificates() {

File file = new File(FILE\_NAME);

if (!file.exists()) {

System.out.println("\nNo certificates found. Please issue some certificates first.\n");

return;

}

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {

List<DigitalCertificate> certList = (List<DigitalCertificate>) ois.readObject();

if (certList.isEmpty()) {

System.out.println("\nNo certificates available.\n");

return;

}

int count = 0;

for (DigitalCertificate cert : certList) {

System.out.println("\n----- Certificate " + (++count) + " -----");

cert.displayCertificate();

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

}

System.out.println("\nAll certificates loaded successfully.\n");

} catch (IOException | ClassNotFoundException e) {

System.out.println("Error loading certificates.");

e.printStackTrace();

}

}

// Main menu

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice;

do {

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

System.out.println(" Digital Certificate System ");

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

System.out.println("1. Issue New Certificate");

System.out.println("2. View All Certificates");

System.out.println("3. Exit");

System.out.print("Choose an option: ");

while (!scanner.hasNextInt()) {

System.out.print("Please enter a valid number (1-3): ");

scanner.next();

}

choice = scanner.nextInt();

scanner.nextLine(); // consume newline

switch (choice) {

case 1:

System.out.print("Enter Holder Name: ");

String name = scanner.nextLine();

System.out.print("Enter Certificate ID: ");

String id = scanner.nextLine();

System.out.print("Enter Expiry Date (YYYY-MM-DD): ");

String expiry = scanner.nextLine();

DigitalCertificate cert = new DigitalCertificate(name, id, expiry);

saveCertificate(cert);

break;

case 2:

loadCertificates();

break;

case 3:

System.out.println("\nExiting system. Goodbye!\n");

break;

default:

System.out.println("Invalid choice. Please try again.\n");

}

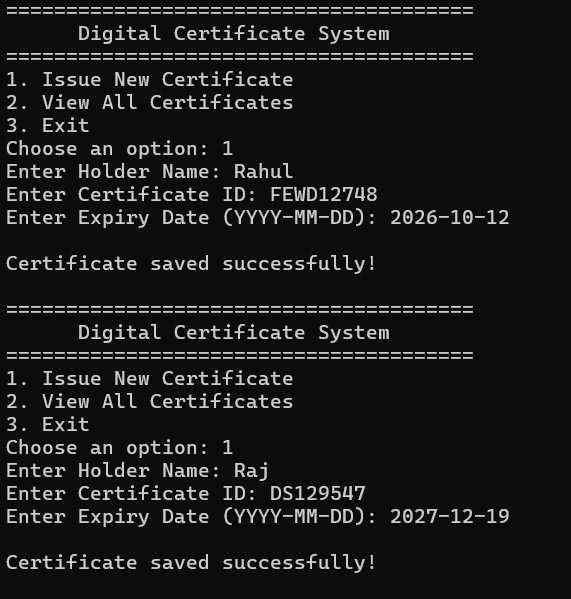
} while (choice != 3);

scanner.close();

}

}

**Output :**

** **

**11.a) Cybersecurity Intrusion Detection System**

**Code :**

import java.util.Scanner;

// Custom Exception Class

class UnauthorizedAccessException extends Exception {

public UnauthorizedAccessException(String message) {

super(message);

}

}

// Intrusion Detection System Class

class IntrusionDetection {

private int invalidAttempts = 0;

private final int threshold = 3;

// Method to validate login

public void validateLogin(String username, String password) throws UnauthorizedAccessException {

// Simulated correct credentials

String correctUsername = "admin";

String correctPassword = "password123";

if (username.equals(correctUsername) && password.equals(correctPassword)) {

System.out.println("Login successful. Welcome " + username + "!");

invalidAttempts = 0; // reset counter on success

System.exit(0);

} else {

invalidAttempts++;

System.out.println("Invalid login attempt #" + invalidAttempts);

// Check if invalid attempts exceed threshold

if (invalidAttempts >= threshold) {

throw new UnauthorizedAccessException("Unauthorized access detected! Too many failed attempts.");

}

}

}

}

// Main Class to run the program

public class CyberSecuritySystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

IntrusionDetection ids = new IntrusionDetection();

// Allow continuous login attempts

while (true) {

try {

System.out.print("Enter username: ");

String username = scanner.nextLine();

System.out.print("Enter password: ");

String password = scanner.nextLine();

// Validate the login attempt

ids.validateLogin(username, password);

} catch (UnauthorizedAccessException e) {

System.out.println(e.getMessage());

break; // Exit after breach detection

} finally {

System.out.println("Attempt logged.\n");

}

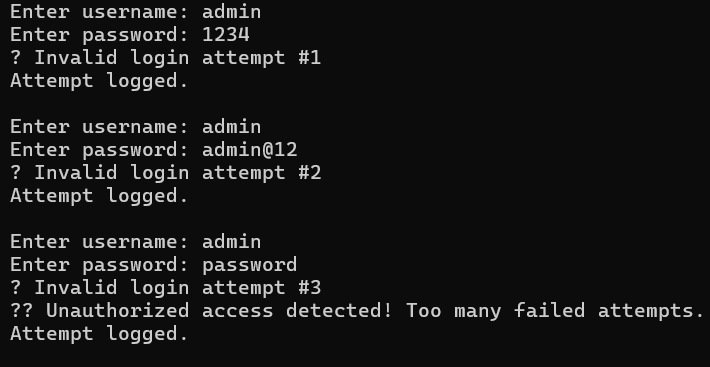
}

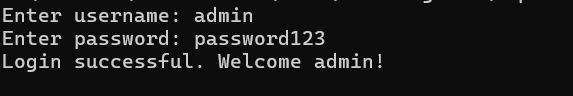
scanner.close();

}

}

**Output :**

****

****

**11.b) Autonomous Vehicle Error Handling System**

**Code :**

import java.util.Random;

// Base Exception for all car-related issues

class CarException extends Exception {

public CarException(String message) {

super(message);

}

}

// Specific Exceptions

class SensorFailureException extends CarException {

public SensorFailureException(String message) {

super(message);

}

}

class GPSFailureException extends CarException {

public GPSFailureException(String message) {

super(message);

}

}

class LowBatteryException extends CarException {

public LowBatteryException(String message) {

super(message);

}

}

// AutonomousCar class with multiple safety checks

class AutonomousCar {

private Random random = new Random();

private int batteryLevel = 100;

public void navigate() throws SensorFailureException, GPSFailureException, LowBatteryException {

System.out.println("\nNavigating...");

checkSensors();

checkGPS();

checkBattery();

System.out.println("Navigation successful. Car is moving safely.");

}

private void checkSensors() throws SensorFailureException {

if (random.nextInt(100) < 20) { // 20% chance of failure

throw new SensorFailureException("Obstacle Sensor Failure Detected!");

}

}

private void checkGPS() throws GPSFailureException {

if (random.nextInt(100) < 10) { // 10% chance of failure

throw new GPSFailureException("GPS Signal Lost!");

}

}

private void checkBattery() throws LowBatteryException {

batteryLevel -= random.nextInt(15); // Simulate battery drain

if (batteryLevel < 20) {

throw new LowBatteryException("Low Battery: " + batteryLevel + "% remaining.");

}

System.out.println("Battery Level: " + batteryLevel + "%");

}

}

// Main simulation class

public class AutonomousCarSystem {

public static void main(String[] args) {

AutonomousCar car = new AutonomousCar();

// Simulating continuous navigation attempts

for (int i = 1; i <= 5; i++) {

System.out.println("\n========= Journey Attempt #" + i + " =========");

try {

car.navigate();

} catch (SensorFailureException e) {

System.out.println("ERROR: " + e.getMessage());

System.out.println("Initiating emergency braking and rerouting.");

} catch (GPSFailureException e) {

System.out.println("ERROR: " + e.getMessage());

System.out.println("Switching to offline navigation mode.");

} catch (LowBatteryException e) {

System.out.println("ERROR: " + e.getMessage());

System.out.println("Redirecting to nearest charging station.");

break; // Stop further attempts on low battery

} finally {

System.out.println("Status report sent to control center.");

}

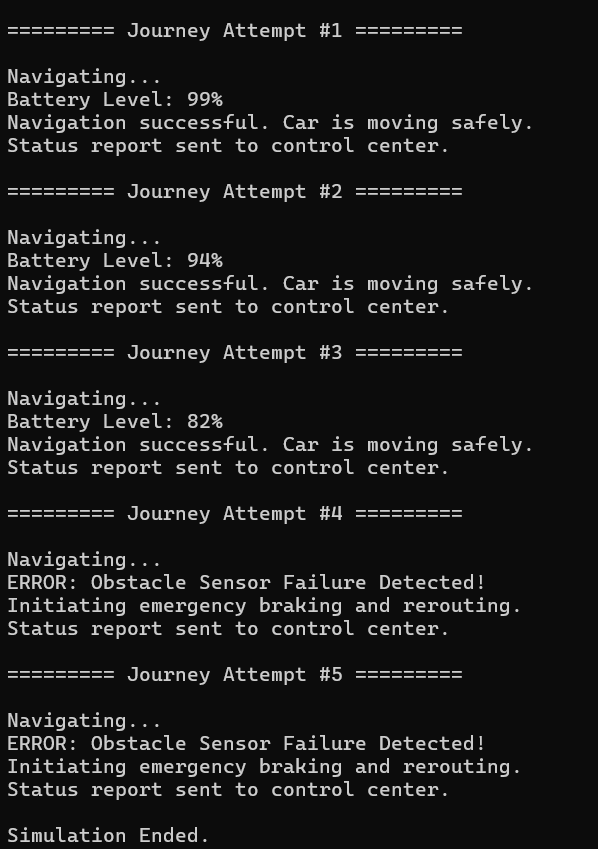
}

System.out.println("\nSimulation Ended.");

}

}

**Output :**

****