**Case Study**

**Kunal Sharma**

create database CARS;

use CARS;

CREATE TABLE Victims (

VictimID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DateOfBirth DATE,

Gender VARCHAR(10),

ContactInformation VARCHAR(255)

);

CREATE TABLE Suspects (

SuspectID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DateOfBirth DATE,

Gender VARCHAR(10),

ContactInformation VARCHAR(255)

);

CREATE TABLE LawEnforcementAgencies (

AgencyID INT PRIMARY KEY,

AgencyName VARCHAR(100),

Jurisdiction VARCHAR(100),

ContactInformation VARCHAR(255)

);

CREATE TABLE Officers (

OfficerID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

BadgeNumber VARCHAR(20),

OfficerRank VARCHAR(20),

ContactInformation VARCHAR(255),

AgencyID INT,

FOREIGN KEY (AgencyID) REFERENCES LawEnforcementAgencies(AgencyID)

);

CREATE TABLE Incidents (

IncidentID INT PRIMARY KEY,

IncidentType VARCHAR(50),

IncidentDate DATE,

LocationLatitude DECIMAL(9,6),

LocationLongitude DECIMAL(9,6),

Description TEXT,

Status VARCHAR(20),

VictimID INT,

SuspectID INT,

AgencyID INT,

FOREIGN KEY (VictimID) REFERENCES Victims(VictimID),

FOREIGN KEY (SuspectID) REFERENCES Suspects(SuspectID),

FOREIGN KEY (AgencyID) REFERENCES LawEnforcementAgencies(AgencyID)

);

CREATE TABLE Evidence (

EvidenceID INT PRIMARY KEY,

Description TEXT,

LocationFound VARCHAR(255),

IncidentID INT,

FOREIGN KEY (IncidentID) REFERENCES Incidents(IncidentID)

);

CREATE TABLE Reports (

ReportID INT PRIMARY KEY,

IncidentID INT,

ReportingOfficer INT,

ReportDate DATE,

ReportDetails TEXT,

Status VARCHAR(20),

FOREIGN KEY (IncidentID) REFERENCES Incidents(IncidentID),

FOREIGN KEY (ReportingOfficer) REFERENCES Officers(OfficerID)

);

package entity;

import java.sql.Date;

public class Incident {

private int incidentID;

private String incidentType;

private Date incidentDate;

private String location;

private String description;

private String status;

private int victimID;

private int suspectID;

public Incident() {

}

public Incident(int incidentID, String incidentType, Date incidentDate, String location, String description,

String status, int victimID, int suspectID) {

super();

this.incidentID = incidentID;

this.incidentType = incidentType;

this.incidentDate = incidentDate;

this.location = location;

this.description = description;

this.status = status;

this.victimID = victimID;

this.suspectID = suspectID;

}

public int getIncidentID() {

return incidentID;

}

public void setIncidentID(int incidentID) {

this.incidentID = incidentID;

}

public String getIncidentType() {

return incidentType;

}

public void setIncidentType(String incidentType) {

this.incidentType = incidentType;

}

public Date getIncidentDate() {

return incidentDate;

}

public void setIncidentDate(Date incidentDate) {

this.incidentDate = incidentDate;

}

public String getLocation() {

return location;

}

public void setLocation(String location) {

this.location = location;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

public int getVictimID() {

return victimID;

}

public void setVictimID(int victimID) {

this.victimID = victimID;

}

public int getSuspectID() {

return suspectID;

}

public void setSuspectID(int suspectID) {

this.suspectID = suspectID;

}

*@Override*

public String toString() {

return "Incident [incidentID=" + incidentID + ", incidentType=" + incidentType + ", incidentDate="

+ incidentDate + ", location=" + location + ", description=" + description + ", status=" + status

+ ", victimID=" + victimID + ", suspectID=" + suspectID + "]";

}

}

package entity;

import java.util.Date;

public class Report {

private int reportID;

private int incidentID; // Foreign key linking to Incidents

private int reportingOfficer; // Foreign key linking to Officers

private Date reportDate;

private String reportDetails;

private String status;

public Report() {

//super();

}

public Report(int reportID, int incidentID, int reportingOfficer, Date reportDate, String reportDetails,

String status) {

super();

this.reportID = reportID;

this.incidentID = incidentID;

this.reportingOfficer = reportingOfficer;

this.reportDate = reportDate;

this.reportDetails = reportDetails;

this.status = status;

}

public int getReportID() {

return reportID;

}

public void setReportID(int reportID) {

this.reportID = reportID;

}

public int getIncidentID() {

return incidentID;

}

public void setIncidentID(int incidentID) {

this.incidentID = incidentID;

}

public int getReportingOfficer() {

return reportingOfficer;

}

public void setReportingOfficer(int reportingOfficer) {

this.reportingOfficer = reportingOfficer;

}

public Date getReportDate() {

return reportDate;

}

public void setReportDate(Date reportDate) {

this.reportDate = reportDate;

}

public String getReportDetails() {

return reportDetails;

}

public void setReportDetails(String reportDetails) {

this.reportDetails = reportDetails;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

*@Override*

public String toString() {

return "Report [reportID=" + reportID + ", incidentID=" + incidentID + ", reportingOfficer=" + reportingOfficer

+ ", reportDate=" + reportDate + ", reportDetails=" + reportDetails + ", status=" + status + "]";

}

}

package entity;

public class Evidence {

private int evidenceID;

private String description;

private String locationFound;

private int incidentID;

public Evidence() {

}

public Evidence(int evidenceID, String description, String locationFound, int incidentID) {

super();

this.evidenceID = evidenceID;

this.description = description;

this.locationFound = locationFound;

this.incidentID = incidentID;

}

public int getEvidenceID() {

return evidenceID;

}

public void setEvidenceID(int evidenceID) {

this.evidenceID = evidenceID;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public String getLocationFound() {

return locationFound;

}

public void setLocationFound(String locationFound) {

this.locationFound = locationFound;

}

public int getIncidentID() {

return incidentID;

}

public void setIncidentID(int incidentID) {

this.incidentID = incidentID;

}

*@Override*

public String toString() {

return "Evidence [evidenceID=" + evidenceID + ", description=" + description + ", locationFound="

+ locationFound + ", incidentID=" + incidentID + "]";

}

}

package entity;

public class LawEnforcementAgency {

private int agencyID;

private String agencyName;

private String jurisdiction;

private String contactInformation;

public LawEnforcementAgency() {

}

public int getAgencyID() {

return agencyID;

}

public void setAgencyID(int agencyID) {

this.agencyID = agencyID;

}

public String getAgencyName() {

return agencyName;

}

public void setAgencyName(String agencyName) {

this.agencyName = agencyName;

}

public String getJurisdiction() {

return jurisdiction;

}

public void setJurisdiction(String jurisdiction) {

this.jurisdiction = jurisdiction;

}

public String getContactInformation() {

return contactInformation;

}

public void setContactInformation(String contactInformation) {

this.contactInformation = contactInformation;

}

public LawEnforcementAgency(int agencyID, String agencyName, String jurisdiction, String contactInformation) {

super();

this.agencyID = agencyID;

this.agencyName = agencyName;

this.jurisdiction = jurisdiction;

this.contactInformation = contactInformation;

}

*@Override*

public String toString() {

return "LawEnforcementAgency [agencyID=" + agencyID + ", agencyName=" + agencyName + ", jurisdiction="

+ jurisdiction + ", contactInformation=" + contactInformation + "]";

}

}

package entity;

public class Officer {

private int officerID;

private String firstName;

private String lastName;

private String badgeNumber;

private String rank;

private String contactInformation;

private int agencyID;

public Officer() {

}

public int getOfficerID() {

return officerID;

}

public void setOfficerID(int officerID) {

this.officerID = officerID;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getBadgeNumber() {

return badgeNumber;

}

public void setBadgeNumber(String badgeNumber) {

this.badgeNumber = badgeNumber;

}

public String getRank() {

return rank;

}

public void setRank(String rank) {

this.rank = rank;

}

public String getContactInformation() {

return contactInformation;

}

public void setContactInformation(String contactInformation) {

this.contactInformation = contactInformation;

}

public int getAgencyID() {

return agencyID;

}

public void setAgencyID(int agencyID) {

this.agencyID = agencyID;

}

public Officer(int officerID, String firstName, String lastName, String badgeNumber, String rank,

String contactInformation, int agencyID) {

super();

this.officerID = officerID;

this.firstName = firstName;

this.lastName = lastName;

this.badgeNumber = badgeNumber;

this.rank = rank;

this.contactInformation = contactInformation;

this.agencyID = agencyID;

}

*@Override*

public String toString() {

return "Officer [officerID=" + officerID + ", firstName=" + firstName + ", lastName=" + lastName

+ ", badgeNumber=" + badgeNumber + ", rank=" + rank + ", contactInformation=" + contactInformation

+ ", agencyID=" + agencyID + "]";

}

}

package dao;

import c.myexceptions.IncidentNumberNotFoundException;

import entity.Incident;

import entity.Report;

import java.util.Collection;

import java.util.Date;

import java.util.List;

public interface ICrimeAnalysisService {

boolean createIncident(Incident incident);

boolean updateIncidentStatus(String status, int incidentId);

Collection<Incident> getIncidentsInDateRange(Date startDate, Date endDate);

Collection<Incident> searchIncidents(String incidentType, String status);

Report generateIncidentReport(Incident incident);

Incident getIncidentDetails(int incidentId) throws IncidentNumberNotFoundException;

boolean updateIncidentDetails(Incident incident) throws IncidentNumberNotFoundException;

List<Incident> getAllIncidents();

}

package util;

import java.io.IOException;

import java.io.InputStream;

import java.util.Properties;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnection {

private static Connection *connection*;

static {

try {

Properties properties = new Properties();

InputStream input = DBConnection.class.getClassLoader().getResourceAsStream("db.properties");

properties.load(input);

String url = properties.getProperty("db.url");

String username = properties.getProperty("db.username");

String password = properties.getProperty("db.password");

*initializeConnection*(url, username, password);

} catch (IOException | SQLException e) {

e.printStackTrace();

throw new ExceptionInInitializerError("Error initializing database connection.");

}

}

public static Connection getConnection() {

return *connection*;

}

private static void initializeConnection(String url, String username, String password) throws SQLException {

*connection* = DriverManager.*getConnection*(url, username, password);

}

}

package util;

import java.io.IOException;

import java.io.InputStream;

import java.util.Properties;

public class PropertyUtil {

private static final String ***PROPERTY\_FILE*** = "db.properties";

public static String getPropertyString() {

Properties properties = new Properties();

try (InputStream input = PropertyUtil.class.getClassLoader().getResourceAsStream(***PROPERTY\_FILE***)) {

if (input == null) {

System.***out***.println("Sorry, unable to find " + ***PROPERTY\_FILE***);

return null;

}

properties.load(input);

return *buildConnectionString*(properties);

} catch (IOException e) {

e.printStackTrace();

return null;

}

}

private static String buildConnectionString(Properties properties) {

String url = properties.getProperty("db.url");

String username = properties.getProperty("db.username");

String password = properties.getProperty("db.password");

return url + "?user=" + username + "&password=" + password;

}

}

driver\_class\_name=com.mysql.cj.jdbc.Driver

db.url=jdbc:mysql://localhost:3306/CARS

db.username=root

db.password=pass123

package dao;

import c.myexceptions.IncidentNumberNotFoundException;

import entity.Incident;

import entity.Report;

import util.DBConnection;

import java.sql.\*;

import java.util.ArrayList;

import java.util.Collection;

import java.sql.Date;

import java.util.List;

public class CrimeAnalysisServiceImpl implements ICrimeAnalysisService {

private static Connection *connection*;

public CrimeAnalysisServiceImpl() {

*connection* = DBConnection.*getConnection*();

}

*@Override*

public boolean createIncident(Incident incident) {

String query = "INSERT INTO Incidents (IncidentId,Incidentdate, Location, Description, Status, VictimID, SuspectID,Incidenttype) " +

"VALUES (?, ?, ?, ?, ?, ?, ?,?)";

try (PreparedStatement preparedStatement = *connection*.prepareStatement(query)) {

preparedStatement.setInt(1, incident.getIncidentID());

if (incident.getIncidentDate() != null) {

preparedStatement.setDate(2, new java.sql.Date(incident.getIncidentDate().getTime()));

} else {

preparedStatement.setDate(2, new java.sql.Date(System.*currentTimeMillis*()));

}

preparedStatement.setString(3, incident.getLocation());

preparedStatement.setString(4, incident.getDescription());

preparedStatement.setString(5, incident.getStatus());

preparedStatement.setInt(6, incident.getVictimID());

preparedStatement.setInt(7, incident.getSuspectID());

preparedStatement.setString(8, incident.getIncidentType());

int affectedRows = preparedStatement.executeUpdate();

return affectedRows > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

*@Override*

public boolean updateIncidentStatus(String status, int incidentId) {

String query = "UPDATE Incidents SET Status = ? WHERE IncidentID = ? ";

try (PreparedStatement preparedStatement = *connection*.prepareStatement(query)) {

preparedStatement.setString(1, status);

preparedStatement.setInt(2, incidentId);

int affectedRows = preparedStatement.executeUpdate();

return affectedRows > 0;

} catch (SQLException e) {

System.***err***.println("Error updating incident status: " + e.getMessage());

System.***err***.println("SQL State: " + e.getSQLState());

System.***err***.println("Error Code: " + e.getErrorCode());

e.printStackTrace();

return false;

}

}

*@Override*

public Collection<Incident> getIncidentsInDateRange(Date startDate, Date endDate) {

List<Incident> incidents = new ArrayList<>();

String query = "SELECT \* FROM Incidents WHERE IncidentDate BETWEEN ? AND ?";

try (PreparedStatement preparedStatement = *connection*.prepareStatement(query)) {

preparedStatement.setDate(1, new java.sql.Date(startDate.getTime()));

preparedStatement.setDate(2, new java.sql.Date(endDate.getTime()));

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

Incident incident = new Incident();

incident.setIncidentID(resultSet.getInt("IncidentID"));

incident.setIncidentType(resultSet.getString("IncidentType"));

incident.setIncidentDate(resultSet.getDate("IncidentDate"));

incident.setLocation(resultSet.getString("Location"));

incident.setDescription(resultSet.getString("Description"));

incident.setStatus(resultSet.getString("Status"));

incident.setVictimID(resultSet.getInt("VictimID"));

incident.setSuspectID(resultSet.getInt("SuspectID"));

incidents.add(incident);

}

} catch (SQLException e) {

e.printStackTrace();

}

return incidents;

}

*@Override*

public Collection<Incident> searchIncidents(String incidentType, String status) {

List<Incident> incidents = new ArrayList<>();

String query = "SELECT \* FROM Incidents WHERE IncidentType = ? AND Status = ?";

try (PreparedStatement preparedStatement = *connection*.prepareStatement(query)) {

preparedStatement.setString(1, incidentType);

preparedStatement.setString(2, status);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

Incident incident = new Incident();

incident.setIncidentID(resultSet.getInt("IncidentID"));

incident.setIncidentType(resultSet.getString("IncidentType"));

incident.setIncidentDate(resultSet.getDate("IncidentDate"));

incident.setLocation(resultSet.getString("Location"));

incident.setDescription(resultSet.getString("Description"));

incident.setStatus(resultSet.getString("Status"));

incident.setVictimID(resultSet.getInt("VictimID"));

incident.setSuspectID(resultSet.getInt("SuspectID"));

incidents.add(incident);

}

} catch (SQLException e) {

e.printStackTrace();

}

return incidents;

}

*@Override*

public Report generateIncidentReport(Incident incident) {

Report report = new Report();

// Set basic information for the report

report.setIncidentID(incident.getIncidentID());

report.setReportingOfficer(1);

report.setReportDate(incident.getIncidentDate());

report.setReportDetails("Placeholder report details");

report.setStatus("Pending"); // You can set an appropriate status

return new Report();

return new Report();

}

*@Override*

public Incident getIncidentDetails(int incidentId) throws IncidentNumberNotFoundException {

String query = "SELECT \* FROM Incidents WHERE IncidentID = ?";

try (PreparedStatement preparedStatement = *connection*.prepareStatement(query)) {

preparedStatement.setInt(1, incidentId);

ResultSet resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {

Incident incident = new Incident();

incident.setIncidentID(resultSet.getInt("IncidentID"));

incident.setIncidentType(resultSet.getString("IncidentType"));

incident.setIncidentDate(resultSet.getDate("IncidentDate"));

incident.setLocation(resultSet.getString("Location"));

incident.setDescription(resultSet.getString("Description"));

incident.setStatus(resultSet.getString("Status"));

incident.setVictimID(resultSet.getInt("VictimID"));

incident.setSuspectID(resultSet.getInt("SuspectID"));

return incident;

} else {

throw new IncidentNumberNotFoundException("Incident with ID " + incidentId + " not found.");

}

} catch (SQLException e) {

e.printStackTrace();

throw new IncidentNumberNotFoundException("Error retrieving incident details.");

}

}

*@Override*

public boolean updateIncidentDetails(Incident incident) throws IncidentNumberNotFoundException {

String query = "UPDATE Incidents SET IncidentType = ?, IncidentDate = ?, Location = ?, Description = ?, " +

"Status = ?, VictimID = ?, SuspectID = ? WHERE IncidentID = ?";

try (PreparedStatement preparedStatement = *connection*.prepareStatement(query)) {

preparedStatement.setString(1, incident.getIncidentType());

preparedStatement.setDate(2, new java.sql.Date(incident.getIncidentDate().getTime()));

preparedStatement.setString(3, incident.getLocation());

preparedStatement.setString(4, incident.getDescription());

preparedStatement.setString(5, incident.getStatus());

preparedStatement.setInt(6, incident.getVictimID());

preparedStatement.setInt(7, incident.getSuspectID());

preparedStatement.setInt(8, incident.getIncidentID());

int affectedRows = preparedStatement.executeUpdate();

if (affectedRows > 0) {

return true;

} else {

throw new IncidentNumberNotFoundException("Incident with ID " + incident.getIncidentID() + " not found.");

}

} catch (SQLException e) {

e.printStackTrace();

throw new IncidentNumberNotFoundException("Error updating incident details.");

}

}

*@Override*

public List<Incident> getAllIncidents() {

List<Incident> incidents = new ArrayList<>();

String query = "SELECT \* FROM Incidents";

try (Statement statement = *connection*.createStatement()) {

ResultSet resultSet = statement.executeQuery(query);

while (resultSet.next()) {

Incident incident = new Incident();

incident.setIncidentID(resultSet.getInt("IncidentID"));

incident.setIncidentType(resultSet.getString("IncidentType"));

incident.setIncidentDate(resultSet.getDate("IncidentDate"));

incident.setLocation(resultSet.getString("Location"));

incident.setDescription(resultSet.getString("Description"));

incident.setStatus(resultSet.getString("Status"));

incident.setVictimID(resultSet.getInt("VictimID"));

incident.setSuspectID(resultSet.getInt("SuspectID"));

incidents.add(incident);

}

} catch (SQLException e) {

e.printStackTrace();

}

return incidents;

}

}

package c.myexceptions;

public class IncidentNumberNotFoundException extends Exception {

public IncidentNumberNotFoundException(String message) {

super(message);

}

}

package Main;

import dao.CrimeAnalysisServiceImpl;

import dao.ICrimeAnalysisService;

import entity.Incident;

import entity.Report;

import c.myexceptions.IncidentNumberNotFoundException;

import java.sql.Date;

import java.util.Collection;

import java.util.Scanner;

public class MainModule {

public static void main(String[] args) {

ICrimeAnalysisService crimeAnalysisService = new CrimeAnalysisServiceImpl();

Scanner scanner = new Scanner(System.***in***);

try {

int choice;

do {

System.***out***.println("====||C.A.R.S||===\n");

System.***out***.println("1. Create Incident");

System.***out***.println("2. Update Incident Status");

System.***out***.println("3. Get Incidents in Date Range");

System.***out***.println("4. Search Incidents");

System.***out***.println("5. Generate Incident Report");

System.***out***.println("6. Get Incident Details");

System.***out***.println("7. Get All Incidents");

System.***out***.println("8. Exit");

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

choice = scanner.nextInt();

scanner.nextLine();

switch (choice) {

case 1:

// Call createIncident method

{

int incidentID;

String incidentType;

Date incidentDate;

String location;

String description;

String status;

int victimID;

int suspectID;

System.***out***.print("Enter incident ID: ");

incidentID = scanner.nextInt();

System.***out***.print("Enter incident Type: ");

incidentType = scanner.next();

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

String dateString = scanner.next();

incidentDate = Date.*valueOf*(dateString);

System.***out***.print("Enter location: ");

location = scanner.next();

System.***out***.print("Enter description: ");

description = scanner.next();

System.***out***.print("Enter status: ");

status = scanner.next();

System.***out***.print("Enter victim ID: ");

victimID = scanner.nextInt();

System.***out***.print("Enter suspect ID: ");

suspectID = scanner.nextInt();

Incident newIncident = new Incident(incidentID, incidentType, incidentDate, location,

description, status, victimID, suspectID);

System.***out***.println("Incident value: " + newIncident);

boolean incidentCreated = crimeAnalysisService.createIncident(newIncident);

if (incidentCreated) {

System.***out***.println("Incident created successfully.");

} else {

System.***out***.println("Failed to create incident.");

}

break;

}

case 2:

// Call updateIncidentStatus method

System.***out***.print("Enter incident ID: ");

int incidentId = scanner.nextInt();

scanner.nextLine(); // Consume the newline character

System.***out***.print("Enter new status: ");

String newStatus = scanner.nextLine();

boolean statusUpdated = crimeAnalysisService.updateIncidentStatus(newStatus, incidentId);

if (statusUpdated) {

System.***out***.println("Incident status updated successfully.");

} else {

System.***out***.println("Failed to update incident status.");

}

break;

case 3:

// Call getIncidentsInDateRange method

System.***out***.print("Enter start date (yyyy-MM-dd): ");

String startDateString = scanner.nextLine();

System.***out***.print("Enter end date (yyyy-MM-dd): ");

String endDateString = scanner.nextLine();

// Convert date strings to Date objects

Date startDate = java.sql.Date.*valueOf*(startDateString);

Date endDate = java.sql.Date.*valueOf*(endDateString);

Collection<Incident> incidentsInRange = crimeAnalysisService.getIncidentsInDateRange(startDate, endDate);

// Handle the collection as needed

for (Incident incident : incidentsInRange) {

System.***out***.println(incident);

}

break;

case 4:

// Call searchIncidents method

System.***out***.print("Enter incident type: ");

String incidentType = scanner.nextLine();

System.***out***.print("Enter status: ");

String status = scanner.nextLine();

Collection<Incident> searchResults = crimeAnalysisService.searchIncidents(incidentType, status);

for (Incident incident : searchResults) {

System.***out***.println(incident);

}

break;

case 5:

// Call generateIncidentReport method

System.***out***.print("Enter incident ID: ");

int reportIncidentId = scanner.nextInt();

scanner.nextLine();

Incident reportIncident = crimeAnalysisService.getIncidentDetails(reportIncidentId);

if (reportIncident != null) {

Report incidentReport = crimeAnalysisService.generateIncidentReport(reportIncident);

System.***out***.println("Incident value: " +incidentReport);

} else {

System.***out***.println("Incident not found.");

}

break;

case 6:

// Call getIncidentDetails method

System.***out***.print("Enter incident ID: ");

int detailsIncidentId = scanner.nextInt();

scanner.nextLine();

try {

Incident incidentDetails = crimeAnalysisService.getIncidentDetails(detailsIncidentId);

System.***out***.println("Incident value: " +incidentDetails);

} catch (IncidentNumberNotFoundException e) {

System.***out***.println("Incident Number Not Found: " + e.getMessage());

}

break;

//case 7:

case 7:

// Call getAllIncidents method

Collection<Incident> allIncidents = crimeAnalysisService.getAllIncidents();

// Handle the collection as needed

for (Incident incident : allIncidents) {

System.***out***.println(incident);

}

break;

case 8:

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

break;

default:

System.***out***.println("Invalid choice. Please enter a number between 1 and 9.");

break;

}

} while (choice != 9);

} catch (Exception e) {

e.printStackTrace();

} finally {

scanner.close();

}

}

}

package dao;

import static org.junit.jupiter.api.Assertions.\*;

import java.text.SimpleDateFormat;

import java.time.LocalDate;

import java.util.Collection;

import java.util.List;

import java.sql.Date;

import org.junit.jupiter.api.AfterAll;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import c.myexceptions.IncidentNumberNotFoundException;

import entity.Incident;

class CrimeAnalysisServiceImplTest {

static CrimeAnalysisServiceImpl *cr*;

*@BeforeAll*

static void setUpBeforeClass() throws Exception {

*cr*=new CrimeAnalysisServiceImpl();

}

*@AfterAll*

static void tearDownAfterClass() throws Exception {

*cr*=null;

}

*@Test*

void testCrimeAnalysisServiceImpl() {

CrimeAnalysisServiceImpl crimeAnalysisService = new CrimeAnalysisServiceImpl();

// Assert

*assertNotNull*(crimeAnalysisService, "CrimeAnalysisServiceImpl instance should not be null");

}

*@Test*

void testCreateIncident() {

//Date d = new Date(2002-02-01);

Incident i=new Incident(6,"Roberry",Date.*valueOf*("2003-02-01"),"Assam","Eye","Open",1,1);

boolean result=*cr*.createIncident(i);

*assertTrue*(result, "Incident creation should be successful");

}

*@Test*

void testUpdateIncidentStatus() {

boolean result = *cr*.updateIncidentStatus("Open", 1);

*assertTrue*(result, "Incident status should be updated successfully");

}

*@Test*

void testGetIncidentsInDateRange() {

Date startDate = Date.*valueOf*("2002-02-01");/\* your start date \*/;

Date endDate = Date.*valueOf*("2002-03-01");/\* your end date \*/;

// Act

Collection<Incident> incidentsInRange = *cr*.getIncidentsInDateRange(startDate, endDate);

// Assert

*assertNotNull*(incidentsInRange, "Collection should not be null");

*assertTrue*(incidentsInRange.isEmpty(), "Collection should not be empty");

}

*@Test*

void testSearchIncidents() {

Incident testIncident = new Incident(2, "Robbery", Date.*valueOf*("2002-05-01"), "Test Location", "Test Description", "Open", 1, 2);

*cr*.createIncident(testIncident);

Collection<Incident> result = *cr*.searchIncidents("Robbery", "Open");

*assertNotNull*(result, "Result should not be null");

*assertEquals*(1, result.size(), "Result size should be 1");

}

*@Test*

void testGetIncidentDetails() {

Incident testIncident = new Incident(3, "Test Incident", Date.*valueOf*("2002-05-01"), "Test Location", "Test Description", "Open", 1, 2);

*cr*.createIncident(testIncident);

try {

Incident retrievedIncident = *cr*.getIncidentDetails(testIncident.getIncidentID());

*assertNotNull*(retrievedIncident, "Incident details should not be null");

*assertEquals*(testIncident.getIncidentID(), retrievedIncident.getIncidentID(), "Incident ID should match");

*assertEquals*(testIncident.getIncidentType(), retrievedIncident.getIncidentType(), "Incident type should match");

// Add additional assertions for other incident properties

} catch (IncidentNumberNotFoundException e) {

*fail*("Exception not expected: " + e.getMessage());

}

}

*@Test*

void testUpdateIncidentDetails() {

Incident testIncident = new Incident(4, "Test Incident", Date.*valueOf*("2002-01-02"), "Test Location", "Test Description", "Open", 1, 2);

*cr*.createIncident(testIncident);

testIncident.setIncidentType("Updated Incident");

testIncident.setDescription("Updated Description");

boolean result;

try {

result = *cr*.updateIncidentDetails(testIncident);

*assertTrue*(result, "Incident details should be updated successfully");

} catch (IncidentNumberNotFoundException e) {

e.printStackTrace();

}

}

*@Test*

void testGetAllIncidents() {

List<Incident> allIncidents = *cr*.getAllIncidents();

*assertNotNull*(allIncidents, "All incidents should not be null");

*assertEquals*(5, allIncidents.size(), "There should be two incidents");

}

}