**Case Study on Digital Asset Management Application**

**Name:** Kannan G

**Superset ID:** 5372153

**Create following tables in SQL Schema with appropriate class and write the unit test case for the Digital Asset Management application.**

**Schema Design:**

1. **employees** table:

employee\_id (Primary Key)

name

epartment.

email

password.

2. **assets** table:

asset\_id (Primary Key): Unique identifier for each asset.

name.

type: Type of the asset (e.g., laptop, vehicle, equipment).

serial\_number: Serial number or unique identifier of the asset.

purchase\_date.

location: Current location of the asset.

status: Status of the asset (e.g., in use, decommissioned, under maintenance).

owner\_id: (Foreign Key): References the employee who owns the asset.

3. **maintenance\_records** table:

maintenance\_id (Primary Key): Unique identifier for each maintenance record.

asset\_id (Foreign Key): References the asset for which maintenance was performed.

maintenance\_date.

description: Description of the maintenance activity.

cost: Cost associated with the maintenance.

4. **asset\_allocations** table:

allocation\_id (Primary Key): Unique identifier for each asset allocation.

asset\_id (Foreign Key): References the asset that is allocated.

employee\_id (Foreign Key): References the employee to whom the asset is allocated.

allocation\_date: Date when the asset was allocated.

return\_date: Date when the asset was returned (if applicable).

5. **reservations** table (to store order details):

reservation\_id (Primary Key): Unique identifier for each reservation.

asset\_id (Foreign Key): References the asset that is being reserved.

employee\_id (Foreign Key): References the employee who made the reservation.

reservation\_date: Date when the reservation was made.

start\_date: Date when the reserved asset is needed.

end\_date: Date when the reservation ends.

status: Status of the reservation (e.g., pending, approved, canceled).

A screen shot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A computer screen with white text

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

**Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors (default and parametrized) and getters, setters methods.**

**Employees.java:**

**package** entity;

**public** **class** Employees {

**private** **int** employeeId;

**private** String name;

**private** String department;

**private** String email;

**private** String password;

**public** Employees() {

// **TODO** Auto-generated constructor stub

}

**public** Employees(**int** employeeId, String name, String department, String email, String password) {

**super**();

**this**.employeeId = employeeId;

**this**.name = name;

**this**.department = department;

**this**.email = email;

**this**.password = password;

}

**public** **int** getEmployeeId() {

**return** employeeId;

}

**public** **void** setEmployeeId(**int** employeeId) {

**this**.employeeId = employeeId;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDepartment() {

**return** department;

}

**public** **void** setDepartment(String department) {

**this**.department = department;

}

**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;

}

}

**Asset.java:**

**package** entity;

**import** java.time.LocalDate;

**public** **class** Asset {

**private** **int** assetId;

**private** String name;

**private** String type;

**private** String serialNumber;

**private** LocalDate purchaseDate;

**private** String location;

**private** String status;

**private** **int** ownerId;

**public** Asset() {

// **TODO** Auto-generated constructor stub

}

**public** Asset(**int** assetId, String name, String type, String serialNumber, LocalDate purchaseDate, String location,

String status, **int** ownerId) {

**super**();

**this**.assetId = assetId;

**this**.name = name;

**this**.type = type;

**this**.serialNumber = serialNumber;

**this**.purchaseDate = purchaseDate;

**this**.location = location;

**this**.status = status;

**this**.ownerId = ownerId;

}

**public** **int** getAssetId() {

**return** assetId;

}

**public** **void** setAssetId(**int** assetId) {

**this**.assetId = assetId;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getType() {

**return** type;

}

**public** **void** setType(String type) {

**this**.type = type;

}

**public** String getSerialNumber() {

**return** serialNumber;

}

**public** **void** setSerialNumber(String serialNumber) {

**this**.serialNumber = serialNumber;

}

**public** LocalDate getPurchaseDate() {

**return** purchaseDate;

}

**public** **void** setPurchaseDate(LocalDate purchaseDate) {

**this**.purchaseDate = purchaseDate;

}

**public** String getLocation() {

**return** location;

}

**public** **void** setLocation(String location) {

**this**.location = location;

}

**public** String getStatus() {

**return** status;

}

**public** **void** setStatus(String status) {

**this**.status = status;

}

**public** **int** getOwnerId() {

**return** ownerId;

}

**public** **void** setOwnerId(**int** ownerId) {

**this**.ownerId = ownerId;

}

}

**MaintenanceRecords.java:**

**package** entity;

**import** java.time.LocalDate;

**public** **class** MaintenanceRecords {

**private** **int** maintenanceId;

**private** **int** assetId;

**private** LocalDate maintenanceDate;

**private** String description;

**private** **double** cost;

**public** MaintenanceRecords() {

// **TODO** Auto-generated constructor stub

}

**public** MaintenanceRecords(**int** maintenanceId, **int** assetId, LocalDate maintenanceDate, String description,

**double** cost) {

**super**();

**this**.maintenanceId = maintenanceId;

**this**.assetId = assetId;

**this**.maintenanceDate = maintenanceDate;

**this**.description = description;

**this**.cost = cost;

}

**public** **int** getMaintenanceId() {

**return** maintenanceId;

}

**public** **void** setMaintenanceId(**int** maintenanceId) {

**this**.maintenanceId = maintenanceId;

}

**public** **int** getAssetId() {

**return** assetId;

}

**public** **void** setAssetId(**int** assetId) {

**this**.assetId = assetId;

}

**public** LocalDate getMaintenanceDate() {

**return** maintenanceDate;

}

**public** **void** setMaintenanceDate(LocalDate maintenanceDate) {

**this**.maintenanceDate = maintenanceDate;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** **double** getCost() {

**return** cost;

}

**public** **void** setCost(**double** cost) {

**this**.cost = cost;

}

}

**AssetAllocations.java:**

**package** entity;

**import** java.time.LocalDate;

**public** **class** AssetAllocations {

**private** **int** allocationId;

**private** **int** assetId;

**private** **int** employeeId;

**private** LocalDate allocationDate;

**private** LocalDate returnDate;

**public** AssetAllocations() {

// **TODO** Auto-generated constructor stub

}

**public** AssetAllocations(**int** allocationId, **int** assetId, **int** employeeId, LocalDate allocationDate,

LocalDate returnDate) {

**super**();

**this**.allocationId = allocationId;

**this**.assetId = assetId;

**this**.employeeId = employeeId;

**this**.allocationDate = allocationDate;

**this**.returnDate = returnDate;

}

**public** **int** getAllocationId() {

**return** allocationId;

}

**public** **void** setAllocationId(**int** allocationId) {

**this**.allocationId = allocationId;

}

**public** **int** getAssetId() {

**return** assetId;

}

**public** **void** setAssetId(**int** assetId) {

**this**.assetId = assetId;

}

**public** **int** getEmployeeId() {

**return** employeeId;

}

**public** **void** setEmployeeId(**int** employeeId) {

**this**.employeeId = employeeId;

}

**public** LocalDate getAllocationDate() {

**return** allocationDate;

}

**public** **void** setAllocationDate(LocalDate allocationDate) {

**this**.allocationDate = allocationDate;

}

**public** LocalDate getReturnDate() {

**return** returnDate;

}

**public** **void** setReturnDate(LocalDate returnDate) {

**this**.returnDate = returnDate;

}

}

**Reservations.java:**

**package** entity;

**import** java.time.LocalDate;

**public** **class** Reservations {

**private** **int** reservationId;

**private** **int** assetId;

**private** **int** employeeId;

**private** LocalDate reservationDate;

**private** LocalDate startDate;

**private** LocalDate endDate;

**private** String status;

**public** Reservations() {

// **TODO** Auto-generated constructor stub

}

**public** Reservations(**int** reservationId, **int** assetId, **int** employeeId, LocalDate reservationDate, LocalDate startDate,

LocalDate endDate, String status) {

**super**();

**this**.reservationId = reservationId;

**this**.assetId = assetId;

**this**.employeeId = employeeId;

**this**.reservationDate = reservationDate;

**this**.startDate = startDate;

**this**.endDate = endDate;

**this**.status = status;

}

**public** **int** getReservationId() {

**return** reservationId;

}

**public** **void** setReservationId(**int** reservationId) {

**this**.reservationId = reservationId;

}

**public** **int** getAssetId() {

**return** assetId;

}

**public** **void** setAssetId(**int** assetId) {

**this**.assetId = assetId;

}

**public** **int** getEmployeeId() {

**return** employeeId;

}

**public** **void** setEmployeeId(**int** employeeId) {

**this**.employeeId = employeeId;

}

**public** LocalDate getReservationDate() {

**return** reservationDate;

}

**public** **void** setReservationDate(LocalDate reservationDate) {

**this**.reservationDate = reservationDate;

}

**public** LocalDate getStartDate() {

**return** startDate;

}

**public** **void** setStartDate(LocalDate startDate) {

**this**.startDate = startDate;

}

**public** LocalDate getEndDate() {

**return** endDate;

}

**public** **void** setEndDate(LocalDate endDate) {

**this**.endDate = endDate;

}

**public** String getStatus() {

**return** status;

}

**public** **void** setStatus(String status) {

**this**.status = status;

}

}

6. **Service Provider Interface/Abstract class:**

Keep the interfaces and implementation classes in package dao

Define an **AssetManagementService** interface/abstract class with methods for adding/removing asset and its management. The following methods will interact with database.

**a. Add Asset:**

**i. Method: boolean addAsset(Asset asset)**

ii. Description: Adds a new asset to the system.

**b. Update Asset:**

**i. Method: boolean updateAsset(Asset asset)**

ii. Description: Updates information about an existing asset.

**c. Delete Asset:**

**i. Method: boolean deleteAsset(int assetId)**

ii. Description: Deletes an asset from the system based on its ID.

**d. Allocate Asset:**

**i. Method: boolean allocateAsset(int assetId, int employeeId, String**

**allocationDate)**

ii. Description: Allocates an asset to an employee on a specified allocation date.

**e. Deallocate Asset:**

**i. Method: boolean deallocateAsset(int assetId, int employeeId, String**

**returnDate)**

ii. Description: Deallocates an asset from an employee on a specified return date.

**f. Perform Maintenance:**

**i. Method: boolean performMaintenance(int assetId, String maintenanceDate,**

**String description, double cost)**

ii. Description: Records maintenance activity for an asset, including the date,

description, and cost.

**g. Reserve Asset:**

i. Method: boolean reserveAsset(int assetId, int employeeId, String

reservationDate, String startDate, String endDate)

ii. Description: Reserves an asset for a specified employee for a specific period,

starting from the start date to the end date. The reservation is made on the

reservation date.

**h. Withdraw Reservation:**

i. Method: boolean withdrawReservation(int reservationId)

ii. Description: Withdraws a reservation for an asset identified by the reservation

ID. The reserved asset becomes available for allocation again.

**AssetManagementService.java:**

**package** dao;

**import** entity.Asset;

**import** myexceptions.AssetNotFoundException;

**import** myexceptions.AssetNotMaintainException;

**public** **interface** AssetManagementService {

**boolean** addAsset(Asset asset);

**boolean** updateAsset(Asset asset) **throws** AssetNotFoundException;

**boolean** deleteAsset(**int** assetId) **throws** AssetNotFoundException;

**boolean** allocateAsset(**int** assetId, **int** employeeId, String allocationDate)**throws** AssetNotMaintainException;

**boolean** deallocateAsset(**int** assetId, **int** employeeId, String returnDate)**throws** AssetNotFoundException;

**boolean** performMaintenance(**int** assetId, String maintenanceDate, String description, **double** cost)**throws** AssetNotFoundException;

**boolean** reserveAsset(**int** assetId, **int** employeeId, String reservationDate, String startDate, String endDate)**throws** AssetNotFoundException;

**boolean** withdrawReservation(**int** reservationId);

}

7. Implement the above interface in a class called **AssetManagementServiceImpl in package dao**.

AssetManagementServiceImpl.java:

**package** dao;

**import** java.sql.Connection;

**import** java.sql.Date;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.time.LocalDate;

**import** entity.Asset;

**import** myexceptions.AssetNotFoundException;

**import** myexceptions.AssetNotMaintainException;

**import** util.DBConnection;

**public** **class** AssetManagementServiceImpl **implements** AssetManagementService{

**private** Connection con;

**public** AssetManagementServiceImpl() {

// **TODO** Auto-generated constructor stub

con = DBConnection.*getConnection*();

}

@Override

**public** **boolean** addAsset(Asset asset) {

// **TODO** Auto-generated method stub

**try** {

PreparedStatement pstmt = con.prepareStatement("insert into assets(name, type, serial\_number, purchase\_date, location, status, owner\_id) values (?, ?, ?, ?, ?, ?, ?)");

pstmt.setString(1, asset.getName());

pstmt.setString(2, asset.getType());

pstmt.setString(3, asset.getSerialNumber());

pstmt.setDate(4, Date.*valueOf*(asset.getPurchaseDate()));

pstmt.setString(5, asset.getLocation());

pstmt.setString(6, asset.getStatus());

pstmt.setInt(7, asset.getOwnerId());

//System.out.println("Asset added successfully.");

**return** pstmt.executeUpdate() > 0;

} **catch** (SQLException e) {

System.***err***.println("Error adding asset: " + e.getMessage());

}

**return** **false**;

}

@Override

**public** **boolean** updateAsset(Asset asset) **throws** AssetNotFoundException{

// **TODO** Auto-generated method stub

**try** {

PreparedStatement checkStmt = con.prepareStatement("SELECT \* FROM assets WHERE asset\_id = ?");

checkStmt.setInt(1, asset.getAssetId());

ResultSet rs = checkStmt.executeQuery();

**if** (!rs.next()) {

**throw** **new** AssetNotFoundException("Asset with ID " + asset.getAssetId() + " not found.");

}

PreparedStatement pstmt = con.prepareStatement("update assets set name = ?, type = ?, serial\_number = ?, purchase\_date = ?, location = ?, status = ?, owner\_id = ? where asset\_id = ?");

pstmt.setString(1, asset.getName());

pstmt.setString(2, asset.getType());

pstmt.setString(3, asset.getSerialNumber());

pstmt.setDate(4, Date.*valueOf*(asset.getPurchaseDate()));

pstmt.setString(5, asset.getLocation());

pstmt.setString(6, asset.getStatus());

pstmt.setInt(7, asset.getOwnerId());

pstmt.setInt(8, asset.getAssetId());

//System.out.println("Asset updated successfully.");

**return** pstmt.executeUpdate() > 0;

} **catch** (SQLException e) {

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

}

**return** **false**;

}

@Override

**public** **boolean** deleteAsset(**int** assetId) **throws** AssetNotFoundException{

// **TODO** Auto-generated method stub

**try** {

PreparedStatement checkStmt = con.prepareStatement("select \* from assets where asset\_id = ?");

checkStmt.setInt(1, assetId);

ResultSet rs = checkStmt.executeQuery();

**if** (!rs.next()) {

**throw** **new** AssetNotFoundException("Asset with ID " + assetId + " not found.");

}

PreparedStatement pstmt = con.prepareStatement("delete from assets where asset\_id = ?");

pstmt.setInt(1, assetId);

pstmt.executeUpdate();

//System.out.println("Asset deleted successfully.");

// int rowsAffected = pstmt.executeUpdate();

// if (rowsAffected == 0) {

// throw new AssetNotFoundException("Asset with ID " + assetId + " not found.");

// }

**return** **true**;

} **catch** (SQLException e) {

System.***err***.println("Error deleting asset: " + e.getMessage());

}

**return** **false**;

}

@Override

**public** **boolean** allocateAsset(**int** assetId, **int** employeeId, String allocationDate) **throws** AssetNotMaintainException{

// **TODO** Auto-generated method stub

**try** {

// Check if asset has been maintained in last 2 years

PreparedStatement checkStmt = con.prepareStatement("select max(maintenance\_date) from maintenance\_records where asset\_id = ?");

checkStmt.setInt(1, assetId);

ResultSet rs = checkStmt.executeQuery();

**if** (rs.next()) {

Date lastMaintenance = rs.getDate(1);

**if** (lastMaintenance == **null** || lastMaintenance.toLocalDate().isBefore(LocalDate.*now*().minusYears(2))) {

**throw** **new** AssetNotMaintainException("Asset with ID " + assetId + " hasn't been maintained in 2 years.");

}

}

PreparedStatement pstmt = con.prepareStatement("insert into asset\_allocations(asset\_id, employee\_id, allocation\_date) values (?, ?, ?)");

pstmt.setInt(1, assetId);

pstmt.setInt(2, employeeId);

pstmt.setDate(3, Date.*valueOf*(allocationDate));

//System.out.println("Asset allocated successfully.");

**return** pstmt.executeUpdate() > 0;

} **catch** (SQLException e) {

System.***err***.println("Error allocating asset: " + e.getMessage());

}

**return** **false**;

}

@Override

**public** **boolean** deallocateAsset(**int** assetId, **int** employeeId, String returnDate) **throws** AssetNotFoundException{

// **TODO** Auto-generated method stub

**try** {

PreparedStatement checkStmt = con.prepareStatement("select \* from asset\_allocations where asset\_id = ? and employee\_id = ?");

checkStmt.setInt(1, assetId);

checkStmt.setInt(2, employeeId);

ResultSet rs = checkStmt.executeQuery();

**if** (!rs.next()) {

**throw** **new** AssetNotFoundException("No allocation found for Asset ID " + assetId + " and Employee ID " + employeeId);

}

PreparedStatement pstmt = con.prepareStatement("update asset\_allocations set return\_date = ? where asset\_id = ? and employee\_id = ?");

pstmt.setDate(1, Date.*valueOf*(returnDate));

pstmt.setInt(2, assetId);

pstmt.setInt(3, employeeId);

//System.out.println("Asset deallocated successfully.");

**return** pstmt.executeUpdate() > 0;

} **catch** (SQLException e) {

System.***err***.println("Error deallocating asset: " + e.getMessage());

}

**return** **false**;

}

@Override

**public** **boolean** performMaintenance(**int** assetId, String maintenanceDate, String description, **double** cost) **throws** AssetNotFoundException{

// **TODO** Auto-generated method stub

**try** {

PreparedStatement checkStmt = con.prepareStatement("select \* from assets where asset\_id = ?");

checkStmt.setInt(1, assetId);

ResultSet rs = checkStmt.executeQuery();

**if** (!rs.next()) {

**throw** **new** AssetNotFoundException("Asset with ID " + assetId + " not found.");

}

PreparedStatement pstmt = con.prepareStatement("insert into maintenance\_records(asset\_id, maintenance\_date, description, cost) values (?, ?, ?, ?)");

pstmt.setInt(1, assetId);

pstmt.setDate(2, Date.*valueOf*(maintenanceDate));

pstmt.setString(3, description);

pstmt.setDouble(4, cost);

//System.out.println("Maintenance record added successfully.");

**return** pstmt.executeUpdate() > 0;

} **catch** (SQLException e) {

System.***err***.println("Error performing maintenance: " + e.getMessage());

}

**return** **false**;

}

@Override

**public** **boolean** reserveAsset(**int** assetId, **int** employeeId, String reservationDate, String startDate, String endDate) **throws** AssetNotFoundException{

// **TODO** Auto-generated method stub

**try** {

PreparedStatement checkStmt = con.prepareStatement("select \* from assets where asset\_id = ?");

checkStmt.setInt(1, assetId);

ResultSet rs = checkStmt.executeQuery();

**if** (!rs.next()) {

**throw** **new** AssetNotFoundException("Asset with ID " + assetId + " not found.");

}

PreparedStatement pstmt = con.prepareStatement("insert into reservations(asset\_id, employee\_id, reservation\_date, start\_date, end\_date) values (?, ?, ?, ?, ?)");

pstmt.setInt(1, assetId);

pstmt.setInt(2, employeeId);

pstmt.setDate(3, Date.*valueOf*(reservationDate));

pstmt.setDate(4, Date.*valueOf*(startDate));

pstmt.setDate(5, Date.*valueOf*(endDate));

//System.out.println("Asset reserved successfully.");

**return** pstmt.executeUpdate() > 0;

} **catch** (SQLException e) {

System.***err***.println("Error reserving asset: " + e.getMessage());

}

**return** **false**;

}

@Override

**public** **boolean** withdrawReservation(**int** reservationId) {

// **TODO** Auto-generated method stub

**try** {

PreparedStatement pstmt = con.prepareStatement("update reservations set status = ? where reservation\_id = ?");

pstmt.setString(1, "canceled");

pstmt.setInt(2, reservationId);

//System.out.println("Reservation withdrawn successfully.");

**return** pstmt.executeUpdate() > 0;

} **catch** (SQLException e) {

System.***err***.println("Error withdrawing reservation: " + e.getMessage());

}

**return** **false**;

}

}

Connect your application to the SQL database:

8. Write code to establish a connection to your SQL database.

• Create a utility class **DBConnection** in a package **util** with a static variable **connection** of

Type **Connection** and a static method **getConnection()** which returns connection.

• Connection properties supplied in the connection string should be read from a property

file.

**DBPropertyUtil.java:**

package util;

import java.io.FileInputStream;

import java.io.IOException;

import java.util.Properties;

public class DBPropertyUtil {

public static String getPropertyString(String fileName) throws IOException {

String connStr = null;

Properties props = new Properties();

FileInputStream fis = new FileInputStream(fileName);

props.load(fis);

String protocol = props.getProperty("protocol");

String hostname = props.getProperty("hostname");

String port = props.getProperty("port");

String dbname = props.getProperty("dbname");

String user = props.getProperty("user");

String password = props.getProperty("password");

connStr=protocol+"//"+hostname+":"+port+"/"+dbname+"?user="+user+"&password="+password;

return connStr;}

}

**DBConnection.java:**

package util;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnection {

private static final String fileName = "db.properties";

private static Connection connection = null;

public static Connection getConnection() {

String connString = null;

try {

connString = DBPropertyUtil.getPropertyString(fileName);

}

catch(IOException e) {

System.out.println("Connection string creation failed");

e.printStackTrace();

}

if(connString != null) {

try {

connection = DriverManager.getConnection(connString);

}

catch(SQLException e) {

System.out.println("Error while establishisng DBConnection...");

e.printStackTrace();}}

return connection;

}

}

9. Create the exceptions in package **myexceptions** and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,

• **AssetNotFoundException**: throw this exception when employee enters an invalid asset

id which doesn’t exist in db

• **AssetNotMaintainException**: throw this exception when employee need the asset which

is not maintained for 2 years.

**AssetNotFoundException.java:**

**package** myexceptions;

**public** **class** AssetNotFoundException **extends** Exception{

**public** AssetNotFoundException(String message) {

// **TODO** Auto-generated constructor stub

**super**(message);

}

}

**AssetNotMaintainException.java:**

**package** myexceptions;

**public** **class** AssetNotMaintainException **extends** Exception{

**public** AssetNotMaintainException(String message) {

// **TODO** Auto-generated constructor stub

**super**(message);

}

}

10. Create class named **AssetManagementApp** with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.

• Add Asset:

• Update Asset:

• Delete Asset:

• Allocate Asset:

• Deallocate Asset:

• Perform Maintenance:

• Reserve Asset:

**AssetManagementApp.java:**

**package** main;

**import** java.sql.Connection;

**import** java.time.LocalDate;

**import** java.util.Scanner;

**import** dao.AssetManagementServiceImpl;

**import** entity.Asset;

**import** myexceptions.AssetNotFoundException;

**import** myexceptions.AssetNotMaintainException;

**import** util.DBConnection;

**public** **class** AssetManagementApp {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Connection con = DBConnection.*getConnection*();

**if**(con != **null**) {

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

System.***out***.println("Connected!");

}

**else** {

System.***out***.println("Failed to connect");

}

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

AssetManagementServiceImpl service = **new** AssetManagementServiceImpl();

**while** (**true**) {

System.***out***.println("--- Digital Asset Management System ---");

System.***out***.println("1. Add Asset");

System.***out***.println("2. Update Asset");

System.***out***.println("3. Delete Asset");

System.***out***.println("4. Allocate Asset");

System.***out***.println("5. Deallocate Asset");

System.***out***.println("6. Perform Maintenance");

System.***out***.println("7. Reserve Asset");

System.***out***.println("8. Withdraw Reservation");

System.***out***.println("9. Exit");

System.***out***.print("Provide Option:\t");

**int** choice = sc.nextInt();

**switch** (choice) {

**case** 1 -> {

Asset asset = **new** Asset();

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

sc.nextLine();

asset.setName(sc.nextLine());

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

asset.setType(sc.nextLine());

System.***out***.print("Enter Serial Number: ");

asset.setSerialNumber(sc.nextLine());

System.***out***.print("Enter Purchase Date (yyyy-mm-dd): ");

asset.setPurchaseDate(LocalDate.*parse*(sc.nextLine()));

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

asset.setLocation(sc.nextLine());

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

asset.setStatus(sc.nextLine());

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

asset.setOwnerId(sc.nextInt());

**if** (service.addAsset(asset))

System.***out***.println("Asset added successfully.");

**else**

System.***out***.println("Failed to add asset.");

}

**case** 2 -> {

**try** {

Asset asset = **new** Asset();

System.***out***.print("Enter Asset ID to update: ");

asset.setAssetId(sc.nextInt());

sc.nextLine();

System.***out***.print("Enter New Name: ");

asset.setName(sc.nextLine());

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

asset.setType(sc.nextLine());

System.***out***.print("Enter New Serial Number: ");

asset.setSerialNumber(sc.nextLine());

System.***out***.print("Enter New Purchase Date (yyyy-mm-dd): ");

asset.setPurchaseDate(LocalDate.*parse*(sc.nextLine()));

System.***out***.print("Enter New Location: ");

asset.setLocation(sc.nextLine());

System.***out***.print("Enter New Status: ");

asset.setStatus(sc.nextLine());

System.***out***.print("Enter New Owner Employee ID: ");

asset.setOwnerId(sc.nextInt());

**if** (service.updateAsset(asset))

System.***out***.println("Asset updated successfully.");

**else**

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

}

**catch** (AssetNotFoundException e) {

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

}

}

**case** 3 -> {

**try** {

System.***out***.print("Enter Asset ID to delete: ");

**int** assetId = sc.nextInt();

sc.nextLine();

**if** (service.deleteAsset(assetId))

System.***out***.println("Asset deleted successfully.");

**else**

System.***out***.println("Failed to delete asset.");

}

**catch** (AssetNotFoundException e) {

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

}

}

**case** 4 -> {

**try** {

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

**int** assetId = sc.nextInt();

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

**int** empId = sc.nextInt();

sc.nextLine();

System.***out***.print("Enter Allocation Date (yyyy-mm-dd): ");

String allocDate = sc.nextLine();

**if** (service.allocateAsset(assetId, empId, allocDate))

System.***out***.println("Asset allocated successfully.");

**else**

System.***out***.println("Failed to allocate asset.");

} **catch** (AssetNotMaintainException e) {

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

}

}

**case** 5 -> {

**try** {

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

**int** assetId = sc.nextInt();

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

**int** empId = sc.nextInt();

sc.nextLine();

System.***out***.print("Enter Return Date (yyyy-mm-dd): ");

String returnDate = sc.nextLine();

**if** (service.deallocateAsset(assetId, empId, returnDate))

System.***out***.println("Asset deallocated successfully.");

**else**

System.***out***.println("Failed to deallocate asset.");

}

**catch** (AssetNotFoundException e) {

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

}

}

**case** 6 -> {

**try** {

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

**int** assetId = sc.nextInt();

sc.nextLine();

System.***out***.print("Enter Maintenance Date (yyyy-mm-dd): ");

String mDate = sc.nextLine();

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

String desc = sc.nextLine();

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

**double** cost = sc.nextDouble();

**if** (service.performMaintenance(assetId, mDate, desc, cost))

System.***out***.println("Maintenance recorded successfully.");

**else**

System.***out***.println("Failed to record maintenance.");

} **catch** (AssetNotFoundException e) {

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

}

}

**case** 7 -> {

**try** {

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

**int** assetId = sc.nextInt();

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

**int** empId = sc.nextInt();

sc.nextLine();

System.***out***.print("Enter Reservation Date (yyyy-mm-dd): ");

String resDate = sc.nextLine();

System.***out***.print("Enter Start Date (yyyy-mm-dd): ");

String startDate = sc.nextLine();

System.***out***.print("Enter End Date (yyyy-mm-dd): ");

String endDate = sc.nextLine();

**if** (service.reserveAsset(assetId, empId, resDate, startDate, endDate))

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

**else**

System.***out***.println("Failed to reserve asset.");

} **catch** (AssetNotFoundException e) {

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

}

}

**case** 8 -> {

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

**int** resId = sc.nextInt();

sc.nextLine();

**if** (service.withdrawReservation(resId))

System.***out***.println("Reservation withdrawn successfully.");

**else**

System.***out***.println("Failed to withdraw reservation.");

}

**case** 9 -> {

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

sc.close();

System.*exit*(0);

}

**default** -> **throw** **new** IllegalArgumentException("Invalid Option: " + choice);

}

}

}

}

**Sample Output:**

**A screen shot of a computer

Description automatically generated**



A screenshot of a computer program

Description automatically generated

A screen shot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Unit Testing**

11. Create Unit test cases for **Digital Asset Management System** are essential to ensure the

correctness and reliability of your system. Following questions to guide the creation of Unit test cases:

• Write test case to test asset created successfully or not.

• Write test case to test asset is added to maintenance successfully or not.

• Write test case to test asset is reserved successfully or not.

• Write test case to test exception is thrown correctly or not when employee id or asset id

not found in database.

**assetTest.java:**

**package** test;

**import** **static** org.junit.Assert.*assertThrows*;

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

**import** org.junit.jupiter.api.Test;

**import** java.time.LocalDate;

**import** org.junit.jupiter.api.BeforeEach;

**import** dao.AssetManagementService;

**import** dao.AssetManagementServiceImpl;

**import** entity.Asset;

**import** myexceptions.AssetNotFoundException;

**public** **class** assetTest {

**private** AssetManagementService service;

@BeforeEach

**public** **void** setUp() {

service = **new** AssetManagementServiceImpl();

}

@Test

**public** **void** testAssetCreatedSuccessfully() {

Asset asset = **new** Asset();

asset.setName("HP Laptop");

asset.setType("laptop");

asset.setSerialNumber("HP123");

asset.setPurchaseDate(LocalDate.*now*());

asset.setLocation("IT Department");

asset.setStatus("in use");

asset.setOwnerId(6);

**boolean** result = service.addAsset(asset);

*assertTrue*(result, "Asset should be created successfully");

}

@Test

**public** **void** testAssetMaintenanceAddedSuccessfully() **throws** AssetNotFoundException {

**int** assetId = 1;

**boolean** result = service.performMaintenance(assetId, LocalDate.*now*().toString(), "Routine Check", 300.0);

*assertTrue*(result, "Maintenance should be added successfully");

}

@Test

**public** **void** testAssetReservedSuccessfully() **throws** AssetNotFoundException {

**int** assetId = 1;

**int** employeeId = 1;

**boolean** result = service.reserveAsset(

assetId, employeeId,

LocalDate.*now*().toString(),

LocalDate.*now*().plusDays(1).toString(),

LocalDate.*now*().plusDays(5).toString());

*assertTrue*(result, "Asset should be reserved successfully");

}

@Test

**public** **void** testAssetNotFoundExceptionThrown() {

**int** invalidAssetId = 9999;

Exception exception = *assertThrows*(AssetNotFoundException.**class**, () -> {

service.performMaintenance(invalidAssetId, LocalDate.*now*().toString(), "Testing", 100.0);

});

String expectedMessage = "not found";

String actualMessage = exception.getMessage();

*assertTrue*(actualMessage.toLowerCase().contains(expectedMessage));

}

}

**Sample Output:**

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated