J2EE Design Pattern
DAO Pattern

514770-1
Fall 2021
11/29/2021
Kyoung Shin Park
Computer Engineering
Dankook University

J2EE Design Pattern

- Model-View-Controller (MVC) Pattern
  - Decouples data representation(model), application behavior(controller), and presentation(view).
- Business Delegate Pattern
  - Decouples clients and business services, hiding the underlying implementation details of the business service.
- Composite Entity Pattern
  - Models a network of related business entities.
- Composite View Pattern
  - Separately manages layout and content of multiple composed views.
- Data Access Object (DAO) Pattern
  - Abstracts and encapsulates data access mechanisms.

J2EE Design Pattern

- Front Controller Pattern
  - Centralizes application request processing
- Intercepting Filter Pattern
  - Pre- and post-processes application requests
- Service Locator Pattern
  - Simplifies client access to enterprise business services
- Session Façade Pattern
  - Provides a single interface for the business services of your application.
- Data Transfer Object Pattern
  - Provides better maintainability by separating use cases from the object model, allows for reuse of entity beans across different applications. DAO uses DTO (or Transfer Object or Value Object) to transport data to and from its clients.

DAO(Data Access Object) Pattern

- DAO Pattern is used to isolate the application/business layer from the persistence layer (usually a relational **database**, but it could be any other persistence mechanism) using an abstract API.
- This API hides from the application all the complexities involved in performing CRUD operations in the underlying storage mechanism. This permits both layers to evolve **separately** without knowing anything about each other.
- DAO interface describes the standard actions to be performed on a model object(s).
- DAO concrete class implements a DAO interface. This class is accountable to get data from a data source which can be Xml/database or any other storage mechanism.

DAO Pattern

	Description
Pattern	DAO
Problem	You would use a database to store persistent data. When you change database, it will affect the application logic that works with the changed aspects of your database.
Solution	Use DAO to abstract away the details of persistence in an application (Abstraction, Encapsulation)
Result	Loose coupling, Extensibility

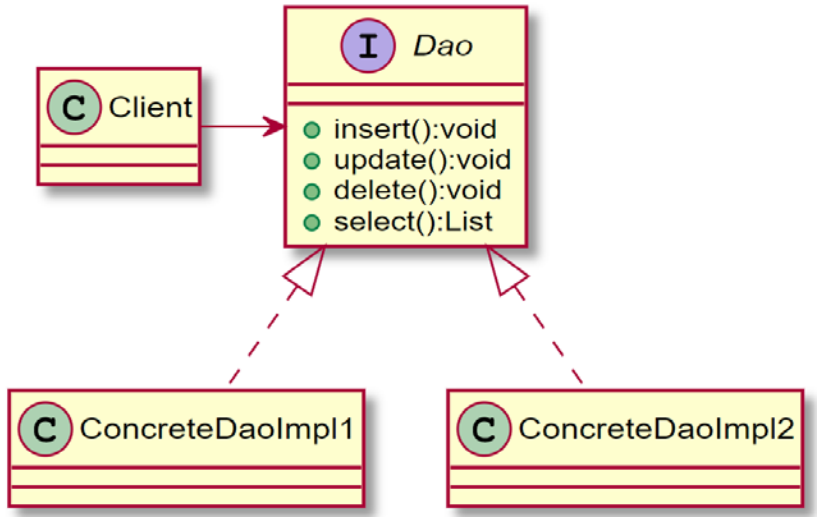
Problem

- Interfaces to data sources vary (relational database management system, XML, JSON, CSV file, etc) and even standard RDBMS/SQL interfaces can vary.
- If you change the persistent mechanism, it will affect the application logic (e.g., For example, change database from MySQL to PostgreSQL, change storage by file to database. SQL statements may be different depending on DBMS).

Design

Role	Design
Dao interface	Basic CRUD(Create, Read, Update, Delete) interface to the model
DaoImpl class	Concrete class implementing a DAO interface
Value Object (or Data Transfer Object)	Simple POJO(Plain old Java object) to store data using DAO

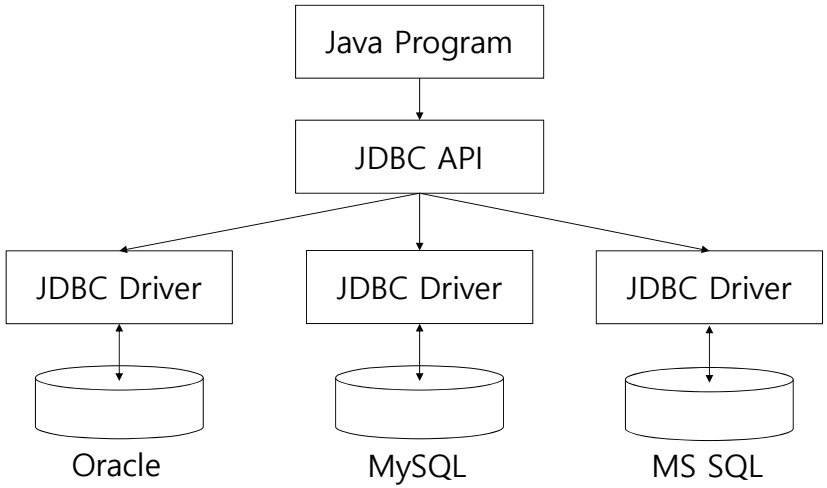
Design



JDBC(Java Database Connectivity)

- Database-Independent Standard Java API
  - Various databases can be used with the same interface
  - Various databases are connected and used through JDBC driver
- Main usage
  - Database connection
  - Execute SQL query
  - Modify the data returned as a result or print it to the screen

JDBC(Java Database Connectivity)



JDBC(Java Database Connectivity)

- DB Connection (including driver Loading)
  - Connect to the database using DriverManager.getConnection() – returns the Connection interface
  - Database URL is required
    - MySQL: "jdbc:mysql://localhost:3306/" + db_name
    - Oracle: "jdbc:oracle:thin:@//localhost:1521/" + db_name
  - If the JDBC driver is 4.0 or higher, it can be used automatically
  - For drivers prior to 4.0, the driver must be made available by using Class.forName() method and driver class name.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

JDBC(Java Database Connectivity)

- Connection interface
  - Maintains sessions connecting Java applications and database and executing queries
  - Is a factory that creates Statement or PreparedStatement that can be execute queries
  - createStatement()
    - Creates a statement interface object that can execute SQL queries
  - close()
    - Disconnects database

JDBC(Java Database Connectivity)

- Statement interface
  - Run SQL queries on the database
  - Is a factory that creates ResultSet interface object when executing SELECT query
  - executeQuery(String sql)
    - Executes a SELECT query and returns a ResultSet object
  - executeUpdate(String sql)
    - Executes the given sql query
    - Can create, drop, insert, update, delete, etc

JDBC(Java Database Connectivity)

- ResultSet interface
  - Cursor pointing to a row in the table
  - Cursor points before the first row
    - Uses next() to check if the data is real
  - next()
    - Moves the cursor to the next line from the current position (return true/false)
  - getInt(int columnIndex), getInt(String columnName)
    - Returns the data of a given column in the form of Integer
  - getString(int columnIndex), getString(String columnName)
    - Returns the data of a given column in the form of String

AddressBook

- Implement CRUD (Create, Read, Update, Delete)
- Address Book Management
  - User creates database and add data
  - Edit request
    - Check if the data exists
  - Delete request
    - Check if the data exists

AddressBook

- 3 Versions
  - AddressBookWithoutDao
    - Directly manipulate database without using DAO pattern
    - Database uses Sqlite
    - Database filename (addrbook.db)
    - Table schema (Table name: persons)
  - AddressBookWithDao
    - Develop using DAO pattern and database
  - AddressBookWithList
    - Use DAO pattern and no DB, but it is developed using ArrayList

AddressBookWithoutDao

- How to use JDBC

	Description
1	Download jar file and set classpath
2	import java.sql.*
3	Load a driver (sqlite is not needed)
4	DriverManager.getConnection()
5	SQL query using Statement, Get ResultSet
6	Close()

AddressBookWithoutDao

- Download Sqlite.jar file & Set classpath
  - Download the latest version https://github.com/xerial/sqlite-jdbc/releases
  - How to execute the Main class by specifying sqlite jar file in the command window,

```
java -classpath ".;./sqlite-jdbc-3.32.3.2.jar" Main
```

  - How to specify the jar file in the classpath as an option in the Eclipse IDE
    - Project->Build Path->Configure Build Path
    - Libraries tab->Classpath->Add External JARs
    - Add jar file

AddressBookWithoutDao

- Use DriverManager.getConnection() to connect a file and create a statement to execute SQL query.

```
Connection connection;
Statement statement;

// DB_FILE_NAME is the DB filename
connection = DriverManager.getConnection(
                    "jdbc:sqlite:" + DB_FILE_NAME);

statement = connection.createStatement();

// set timeout to 30 sec.
statement.setQueryTimeout(30);
```

AddressBookWithoutDao

- Execute query

```
statement.execute("SQL Query");
```

- Create a table

Name	Data Type	Description
ID	INTEGER	PRIMARY KEY
name	text	
address	text	

```
String table = " (ID INTEGER PRIMARY KEY
AUTOINCREMENT, name text, address text)";
statement.executeUpdate(
    "DROP TABLE IF EXISTS " + DB_TABLE_NAME);
statement.executeUpdate(
    "CREATE TABLE " + DB_TABLE_NAME + table);
```

AddressBookWithoutDao

- Data result

```
ResultSet rs;
rs = statement.executeQuery(
                  "SELECT * FROM persons");
while (rs.next()) {
    System.out.println("" + rs.getInt("ID")
     + ", " + rs.getString("name") + ", "
     + rs.getString("address"));
}
```

AddressBookWithoutDao

```
import java.sql.*;
import java.util.Properties;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class AddressBookWithoutDAO {
  final static String DB_FILE_NAME =
"addressbook.db";
  final static String DB_TABLE_NAME = "persons";

  public static void main (String[] args){
    Connection connection = null;
    ResultSet rs = null;
    Statement statement = null;
```

```
    try {
      connection = DriverManager.getConnection(
                        "jdbc:sqlite:" +
DB_FILE_NAME);
      statement = connection.createStatement();
      statement.setQueryTimeout(30);

      final String table = " (ID INTEGER PRIMARY
KEY AUTOINCREMENT, name text, address text)";

      // create table
      statement.executeUpdate(
           "DROP TABLE IF EXISTS " +
DB_TABLE_NAME);
      statement.executeUpdate(
           "CREATE TABLE " + DB_TABLE_NAME +
table);
      System.out.println("--- inserting...");
      statement.execute("INSERT INTO persons(name,
address) VALUES('S Kim','1 DKU')");
      statement.execute("INSERT INTO persons(name,
address) VALUES('J Lee','2 DKU')");
```

```
      System.out.println("--- finding all...");
      rs = statement.executeQuery(
     "SELECT * FROM persons WHERE id < 4 ORDER BY
id");
      while (rs.next()) {
        System.out.println(
          "" + rs.getInt("ID") + ", "
           + rs.getString("name") + ", "
           + rs.getString("address"));
      }

      System.out.println("--- updating...");
      statement.execute(
           "UPDATE persons SET name = 'K Park'
WHERE id = 1");

      System.out.println("--- see if updated...");
      rs = statement.executeQuery(
           "SELECT * FROM persons WHERE id = 1");
      while (rs.next()) {
        System.out.println(rs.getInt("id") + ", "
          + rs.getString("name") + ", "
          + rs.getString("address"));
      }
```

AddressBookWithoutDao

```
      System.out.println("--- deleting...");
      statement.execute(
             "DELETE FROM persons WHERE id = 1");

       System.out.println("--- finding all after
deleting...");
       rs = statement.executeQuery(
     "SELECT * FROM persons WHERE id < 4 ORDER BY
id");
       while (rs.next()) {
         System.out.println(rs.getInt("id") + ", "
           + rs.getString("name") + ", "
           + rs.getString("address"));
       }
    }
    catch (Exception e){
      e.printStackTrace();
    }
```

AddressBookWithoutDao

```
    finally {
      try {
        if (rs != null) { rs.close(); }
        if (statement != null) { statement.close(); }
        if (connection != null) { connection.close(); }
      }
      catch (Exception e) {
        e.printStackTrace();
      }
    }
  } // main
} // class
```

Person

- Person class

```
public class Person {
  private static int counter = 1;
  private int id;
  private String name;
  private String address;
  Person(String name, String address){
    this.name = name;
    this.address = address;
    id = counter;
    counter++;
  }
  Person(int id, String name, String address) {
        this.name = name;
        this.address = address;
        this.id = id;
  }
```

Person

```
  public String toString() {
    return "" + id + ", " + name + ", " +
address;
  }

  public String getName() { return name; }

  public void setName(String name) {
    this.name = name;
  }
  … // getter & setter for id and address
}
```

AddressBookWithDAO

- DAO interface
  - Specify CRUD of Person class

```
import java.util.List;

public interface PersonDao {
    public void insert(Person p);
    public List<Person> findAll();
    public Person findById(int id);
    public void update(Person p, int id);
    public void delete(int id);
    public void delete(Person p);
}
```

AddressBookWithDAO

```
public class PersonDaoImpl implements PersonDao {
  final static String DB_FILE_NAME = "addrbook.db";
  final static String DB_TABLE_NAME = "persons";

  Connection connection = null;
  ResultSet rs = null;
  Statement statement = null;
```

AddressBookWithDAO

```
  public PersonDaoImpl() {
    final String table = " (ID INTEGER PRIMARY KEY
AUTOINCREMENT, name text, address text)";
    try {
      connection = DriverManager.getConnection(
                    "jdbc:sqlite:" + DB_FILE_NAME);
      statement = connection.createStatement();

      // set timeout to 30 sec.
      statement.setQueryTimeout(30);

      // create table
      statement.executeUpdate("DROP TABLE IF EXISTS " +
DB_TABLE_NAME);
      statement.executeUpdate("CREATE TABLE " +
DB_TABLE_NAME + table);
    }
    catch (SQLException e) { e.printStackTrace(); }
  }
```

AddressBookWithDAO

```
  public void insert(Person p) {
    try {
      String fmt = "INSERT INTO %s VALUES(%d, '%s',
'%s')";
      String q = String.format(fmt, DB_TABLE_NAME,
             p.getId(), p.getName(), p.getAddress());

      statement.execute(q);
    }
    catch (SQLException e) { e.printStackTrace(); }
  }
```

AddressBookWithDAO

```
  public List<Person> findAll() {
    ArrayList<Person> persons = new ArrayList<Person>();

    try {
      rs = statement.executeQuery("SELECT * FROM " +
DB_TABLE_NAME);
      while (rs.next()) {
        persons.add(new Person(rs.getInt("id"),
        rs.getString("name"), rs.getString("address")));
      }
    }
    catch (SQLException e) { e.printStackTrace(); }
    return persons;
  }
```

AddressBookWithDAO

```
public Person findById(int id) {
  Person person = null;
  try {
    String fmt = "SELECT * FROM %s WHERE id = %d";
    String q = String.format(fmt, DB_TABLE_NAME, id);

    rs = statement.executeQuery(q);
    if (rs.next()) {
      person = new Person(rs.getInt("id"),
       rs.getString("name"), rs.getString("address"));
    }
  }
  catch (SQLException e) { e.printStackTrace(); }
  return person;
}
```

AddressBookWithDAO

```
  public void update(Person p, int id) {
    Person person = findById(id);

    if (person != null) {
      try {
        String fmt = "UPDATE %s SET name = '%s',
address = '%s' WHERE id = %d";
        String q = String.format(fmt, DB_TABLE_NAME,
               p.getName(), p.getAddress(), p.getId());

        statement.execute(q);
      }
      catch (SQLException e) { e.printStackTrace(); }
    }
  }
```

AddressBookWithDAO

```
public void delete(int id) {
  try {
    String fmt = "DELETE FROM %s WHERE id = %d";
    String q = String.format(fmt, DB_TABLE_NAME, id);

    statement.execute(q);
  }
  catch (SQLException e) { e.printStackTrace(); }
}

public void delete(Person p) {
  delete(p.getId());
}
```

AddressBookWithList

- DAO interface

```
import java.util.List;

public interface PersonDao {
    public void insert(Person p);
    public List<Person> findAll();
    public Person findById(int id);
    public void update(Person p, int id);
    public void delete(int id);
    public void delete(Person p);
}
```

AddressBookWithList

```
import java.util.ArrayList;
import java.util.List;

public class PersonDaoImpl implements PersonDao {
  List<Person> persons;

  public PersonDaoImpl() {
    persons = new ArrayList<Person>();
  }

  public void insert(Person p) {
    persons.add(p);
  }

  public List<Person> findAll() {
    return persons;
  }
```

```
public Person findById(int id) {
  int n = 0;
  for (Person pi : persons) {
    if (pi.getId() == id) {
      break;
    }
    n++;
  }
  if (n >= persons.size()) {
    return null;
  }
  return persons.get(n);
}

public void update(Person p, int id) {
  Person person = findById(id);
  if (person != null) {
    person.setName(p.getName());
    person.setAddress(p.getAddress());
  }
}
```

AddressBookWithList

```
  public void delete(int id) {
    Person person = findById(id);
    if (person != null) {
      persons.remove(person);
    }
  }

  public void delete(Person p) {
    persons.remove(p);
  }
}
```

AddressBookWithList

```
public class AddressBookWithList {
  public static void main(String[] args) {
    Person p;
    PersonDao personDao = new PersonDaoImpl();

    System.out.println("--- inserting...");
    p = new Person("S Kim", "1 DKU");
    personDao.insert(p);
    p = new Person("J Lee", "2 DKU");
    personDao.insert(p);

    System.out.println("--- finding all...");
    for (Person pi : personDao.findAll()) {
      System.out.println("reading... " + pi);
    }
    System.out.println("--- updating...");
    p = personDao.findById(0);
    p.setName("K Park");
    personDao.update(p, p.getId());

    System.out.println("--- see if updated...");
    p = personDao.findById(1);
    if (p != null) {
      System.out.println(p);
    }

    System.out.println("--- deleting...");
    personDao.delete(1);  // or
personDao.delete(p);

    System.out.println("--- finding all after
deleting...");
    for (Person pi : personDao.findAll()) {
      System.out.println("reading... " + pi);
    }
  }
}
```