Database Applications (2)

Han-fen Hu

UNLV

Outline

- ResultSet Navigation Methods
- Data Access Object Pattern
- Database Transactions

Scrollable Result Sets (1)

- By default, a **`ResultSet`** object is created with a read-only concurrency level and the cursor is limited to forward movement
- A *scrollable result set* can be created with the **`Connection`** object's method

```
conn.createStatement(type, concur);

conn.prepareStatement(sql, type, concur);
```

  - ***`sql`*** is the SQL command with parameters
  - ***`type`*** is a constant for the scrolling type
  - **`concur`** is a constant for the concurrency level

UNLV

The `ResultSet` Scrolling Types

- **`ResultSet.TYPE_FORWARD_ONLY`**
  - Default scrolling type
  - Cursor moves forward only
- **`ResultSet.TYPE_SCROLL_INSENSITIVE`**
  - Cursor moves both forward and backward
  - Changes made to the database do not appear
- **`ResultSet.TYPE_SCROLL_SENSITIVE`**
  - Cursor moves both forward and backward
  - Changes made to the database appear as soon as they are made

UNLV

The `ResultSet` Concurrency Levels

- **`ResultSet.CONCUR_READ_ONLY`**
  - Default concurrency level
  - Read-only version of data from the database
  - Cannot change database by altering result set
- **`ResultSet.CONCUR_UPDATEABLE`**
  - Result set is updateable
  - Changes can be made to the result set and saved to the database
  - Uses methods that allow changes to be made to the database without issuing SQL statements

UNLV

Scrollable Result Sets (2)

- Example 1:
  - Creates a scrollable result set that is read-only and insensitive to database changes

```
Statement stmt =
   conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                        ResultSet.CONCUR_READ_ONLY);
```

- Example 2:
  - Creates a scrollable result set that is updatable and insensitive to database changes

```
PreparedStatement prepStmt =
             conn.prepareStatement(sql,
             ResultSet.TYPE_SCROLL_INSENSITIVE,
             ResultSet.CONCUR_UPDATABLE);
```

`ResultSet` Navigation Methods (1)

- **`first()`**
  - Moves the cursor to the first row
- **`last()`**
  - Moves the cursor to the last row
- **`next()`**
  - Moves the cursor to the next row
- **`previous()`**
  - Moves the cursor to the previous row

`ResultSet` Navigation Methods (2)

- **`relative(`*`rows`*`)`**
  - Moves the cursor the number specified by the ***`rows`*** argument relative to the current row
    - A positive ***`rows`*** value will move the cursor forward
    - A negative ***`rows`*** value will move the cursor backward
- **`absolute(`*`rows`*`)`**
  - Moves the cursor to the row number specified by the ***`rows`*** argument
    - A **`rows`** value of 1 will move the cursor to the first row
    - A **`rows`** value of 2 will move cursor to the second row
    - And so on until the last row

UNLV

Application of `ResultSet` Navigation Methods

- **`ResultSet`** navigation methods can be used to determine the number of rows in a result set
- Example:

```
resultSet.last();                    // Move to the last row
int numRows = resultSet.getRow();    // Get the current row number
resultSet.first();                   // Move back to the first row
```

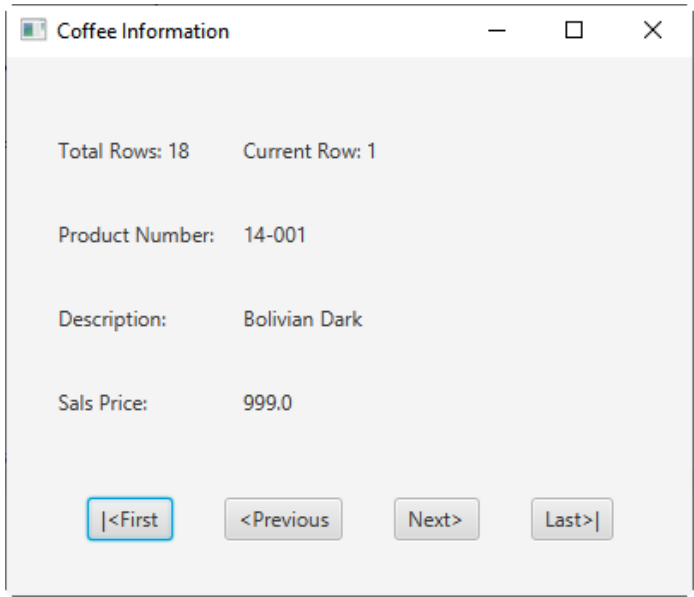
  - Move cursor to last row
  - Get the last row's number and store the value
  - Move back to the first row

Database Application in JavaFX

- Option 1: Whenever a JavaFX application is launched:
  - Calls the start (javafx.stage.Stage) method
    - Open the database connection
  - Waits for the application to finish
  - Calls the stop() method
    - Close the database connection
- Option 2: Open and close the database connect in each event listener

Lab (1)

- CoffeeBrowser.fxml
- CoffeeBrowserController.java
- CoffeeBrowser.java
  - The application will display the database query result one record at a time. The four buttons can be used to switch records.

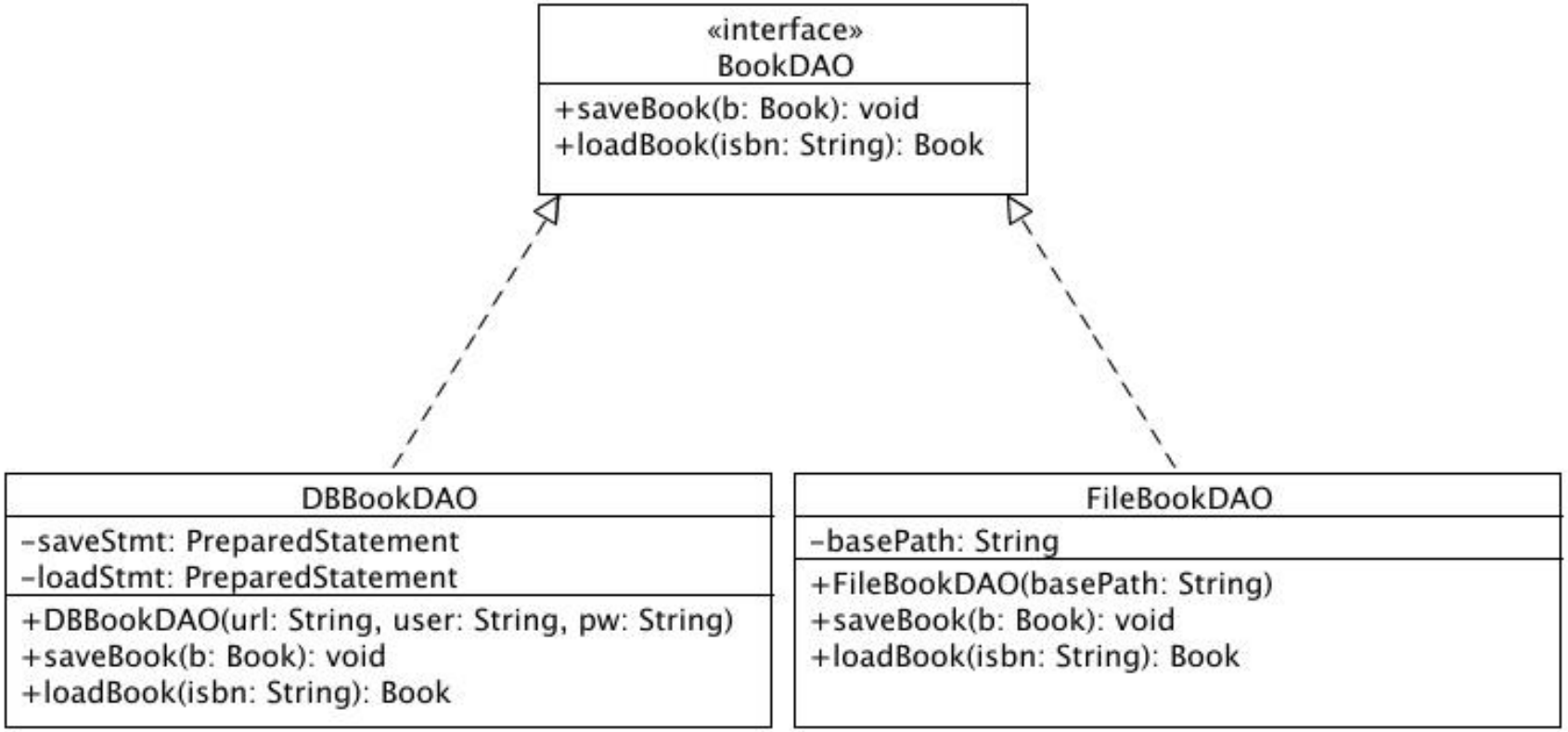


Data Access Object (DAO) Pattern

- Access to persistent storage varies by the type of storage (relational databases, object-oriented databases, flat files, and so forth) and the vendor implementation
  - Access to data varies depending on the source of the data.
- Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services

UNLV

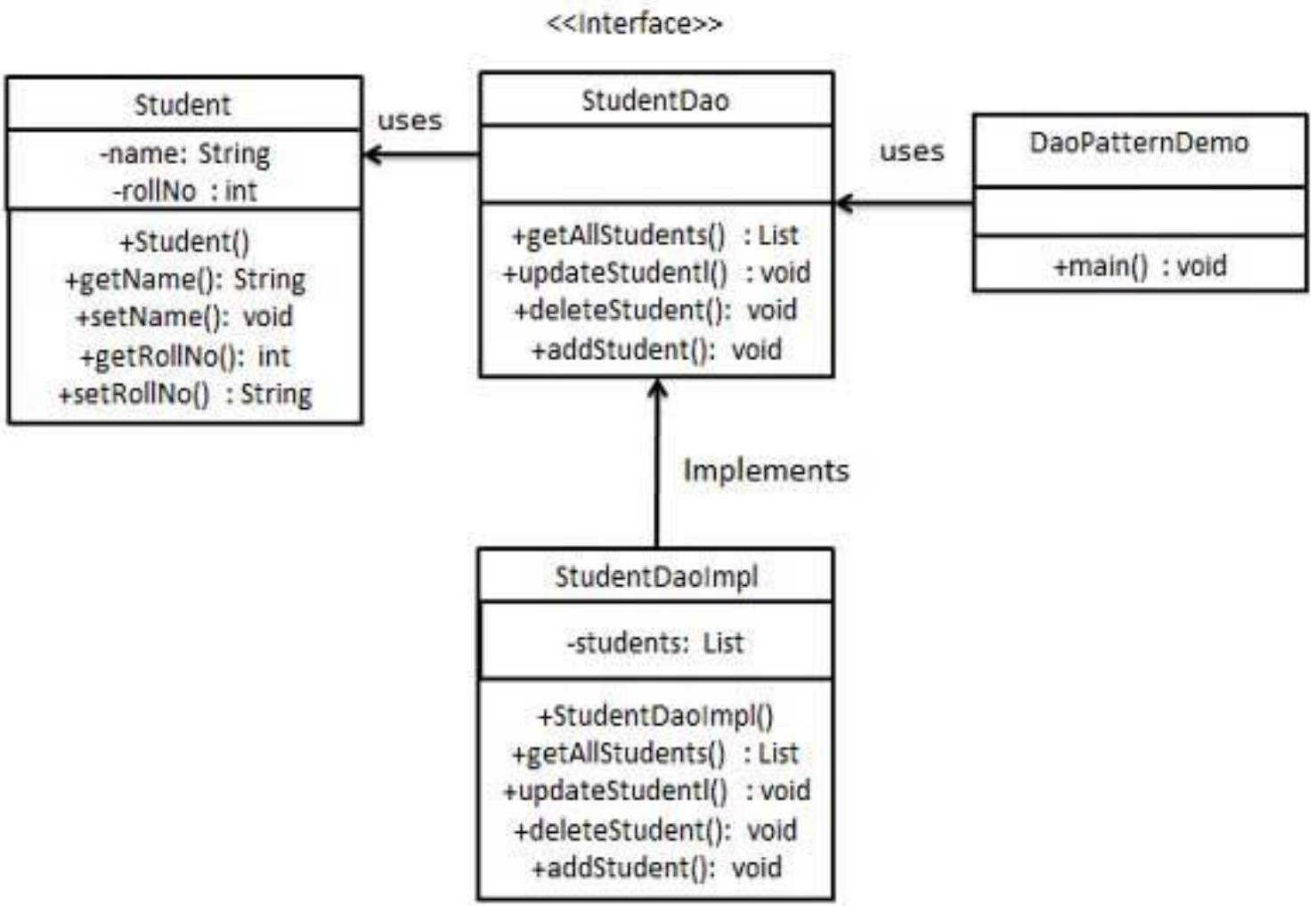
DAO for Different Data Sources



Why DAO?

- Use a Data Access Object (DAO) to abstract and encapsulate all access to the data source.
  - DAO manages the connection with the data source to obtain and store data.
  - DAO implements the access mechanism required to work with the data source.
- The Data Access Object (DAO) pattern provides an abstraction layer between the business logic tier and the persistent storage tier
  - Changes made to the data source should not modify the business objects; only the data access objects themselves would need to change.

DAO Components (1): Example



http://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm

UNLV

DAO Components (2)

- Data Model
  - Contains get/set methods to store data retrieved using DAO class
- Data Access Object Interface
  - Defines the standard operations to be performed on a model object(s)
- Data Access Object concrete class
  - Implements above interface.
  - Get data from a data source which can be database / xml or any other storage mechanism.

Steps of Implementing DAO Pattern

- Create Data Model (model class)
- Create DAO Interface
- Create concrete class implementing above interface
  - Either retrieve the data from databases or files
- Use the DAO in applications

Lab (2)

- CoffeeDBConstants.java
- CoffeeDBUtil.java
- Customer.java
- CustomerDAO.java
- CustomerDAOImpl.java

Use DAO Pattern in JavaFX (1)

- The JavaFX application does not have to deal with database connection and operations
- In the action listener methods, instantiate object of the DAO concrete class, and then call the methods to insert, update, delete, and query.

Lab (3)

- CustomerInserter.java
- CustomerInserter.fxml
- CustomerInserterController.java
  - It allows the user to enter and save the data.

Use DAO Pattern in JavaFX (2)

- To display the list of object in a TableView, the ArrayList need to be converted into a ObservableList first

```
List<Customer> customerList =
        new ArrayList<Customer>();
ObservableList<Customer> observableCustomers =
        FXCollections.observableArrayList(customerList);
```

Lab (4)

- CustomerQuery.java
- CustomerQuery.fxml
- CustomerQueryController.java
  - It allows user to enter a state and displays customer in the state

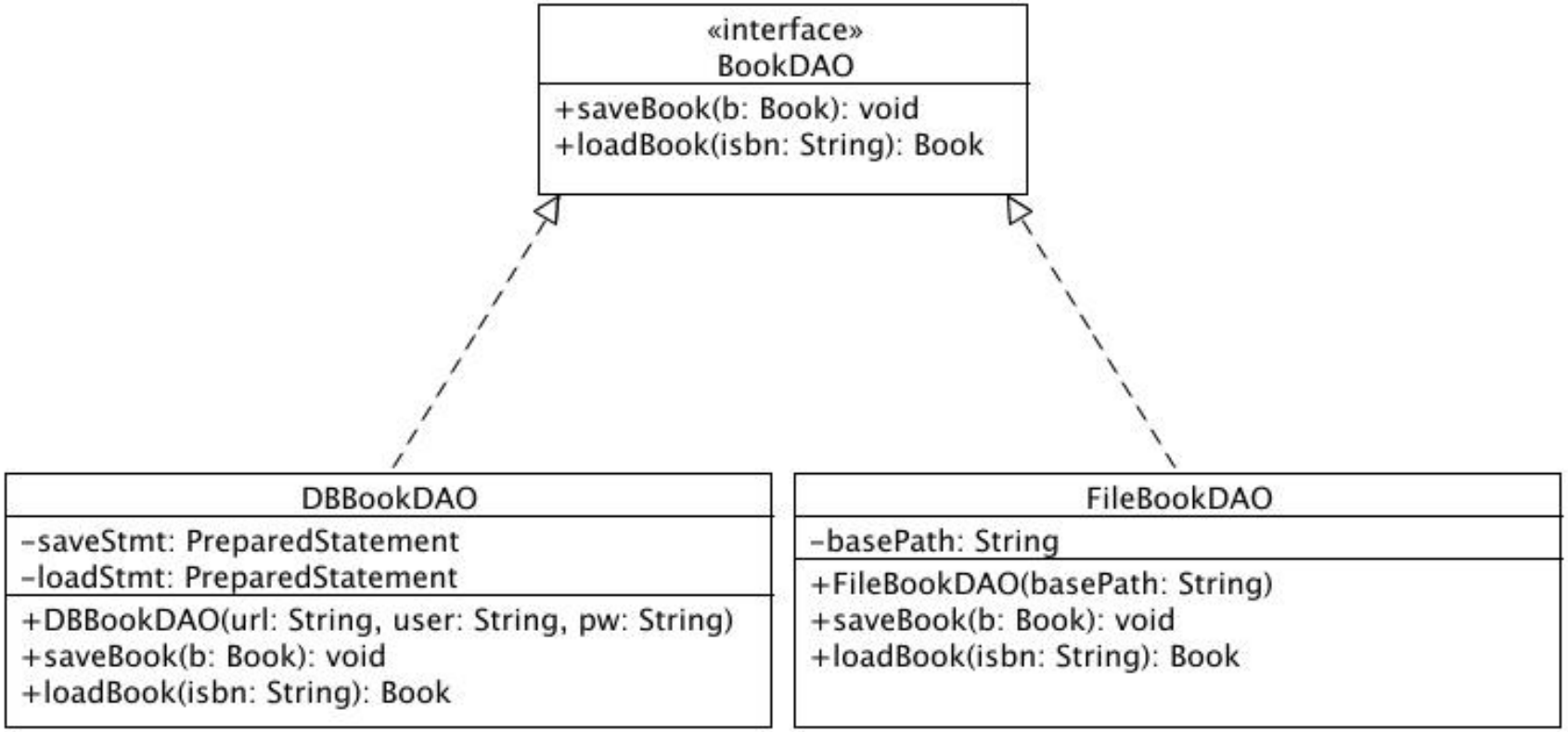
When to Use DAO?

- Need to access a persistent storage more than one time
- Want to separate a data resource's client interface from its data access mechanisms
- Want to provide a generic interface
- In a larger project, different teams work on different parts of the application: the DAO pattern allows clean separation of concerns.

MVC Design Pattern

- Model
  - Data Model (model class)
  - DAO
- Visual
  - fxml
- Control
  - Controller and application class

DAO for Different Data Sources



How to Plan a DAO?

- Insertion, update, and deletion
  - Insert a record
  - Update a record
  - Delete by the primary key (e.g., ID)
  - Delete by other columns
- Data retrieval
  - Get all records
  - Find by the primary key (e.g., ID)
  - Find by other columns

Exercise

- Design a DAO structure for the table Coffee of the CoffeeShopData
  - Data Model: Coffee
  - Data Access Object Interface: CoffeeDAO
  - Data Access Object Implementation: CoffeeDAOImpl
- Create a program to demonstrate the use of the DAO pattern, you can consider a program for
  - Inserting a record,
  - Query the coffee by a certain column,
  - Or any other applications that you can think of

Transactions

- An operation that requires multiple database updates is known as a *transaction*.
  - For a transaction to be complete, all of the steps involved in the transaction must be performed.
  - If any single step within a transaction fails, none of the steps in the transaction should be performed.
- When you write transaction-processing code, there are two concepts you must understand:
  - Commit: making a permanent change to a database
  - Rollback: refers to undoing changes to a database

JDBC Auto Commit Mode

- By default, the JDBC **`Connection`** class operates in auto commit mode.
- In *auto commit* mode
  - All updates that are made to the database are made permanent as soon as they are executed.
- When auto commit mode is turned off
  - Changes do not become permanent until a commit command is executed
  - A rollback command can be used to undo changes

JDBC Transaction Methods

- To turn auto commit mode off
  - Call the **`Connection`** class's **`setAutoCommit`** method
  - Pass the argument **`false`**

```
conn.setAutoCommit(false);
```

- To execute a commit command
  - Call the **`Connection`** class's **`commit`** method

```
conn.commit();
```

- To execute a rollback command
  - Call the **`Connection`** class's **`rollback`** method

```
conn.rollback();
```

JDBC Transaction Example

```
conn.setAutoCommit(false);
// Attempt the transaction
try {
    // Update the inventory records.
    stmt.executeUpdate(updateStatement);
    // Add the order to the UnpaidOrder table.
    stmt.executeUpdate(insertStatement);
……..
    // Commit all these updates.
    conn.commit();
}
catch (SQLException ex) {
    // Roll back the changes
    conn.rollback();
}
```

The `commit` method is called in the `try` block

The `rollback` method is called in the `catch` block

References

- Data Access Object Pattern
  http://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm
- Write once, persist anywhere
  http://www.javaworld.com/article/2074052/design-patterns/write-once--persist-anywhere.html
- Core J2EE Patterns - Data Access Object
  http://www.oracle.com/technetwork/java/dataaccessobject-138824.html
- Best Practice Software Engineering - Data Access Object
  http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/dao.html