Java Programming

Advanced Data Access with JDBC

Review of Lecture 9

- **Exception Handling:**
  - **Exceptions** are exceptional events that disrupt the normal flow of a program
  - Java Exception **model** – termination:
    - The block that causes the exception expires.
- **try block:**
  - Code that may generate exceptions
- **catch block**
  - To handle exceptions
  - Takes an Exception object as argument
- **finally block**
  - Executes always – clean up operations
- **throws clause:**
  - Indicates exceptions thrown by a method
- **throw** statement:
  - To throw an exception

Review of Lecture 9

- **Checked exceptions**
  - The compiler checks the code for exception handling
- **Unchecked exceptions** – RuntimeException objects – need exception handling code
- **Exception hierarchy**
  - Throwable interface
  - Exception class
  - RuntimeException class
  - ArithmeticException class
  - InputMismatchException class
  - NullPonterException

- **Using JDBC**
  - JDBC drivers
  - Class.**forName** method
  - **DriverManager** class
    - **getConnection** method
  - **Connection** interface
    - **createStatement** method
  - **Statement** interface
    - **executeQuery** method
    - **executeUpdate** method
  - **ResultSet** interface
    - **next()** method
  - Database metadata

Lesson 10 Objectives

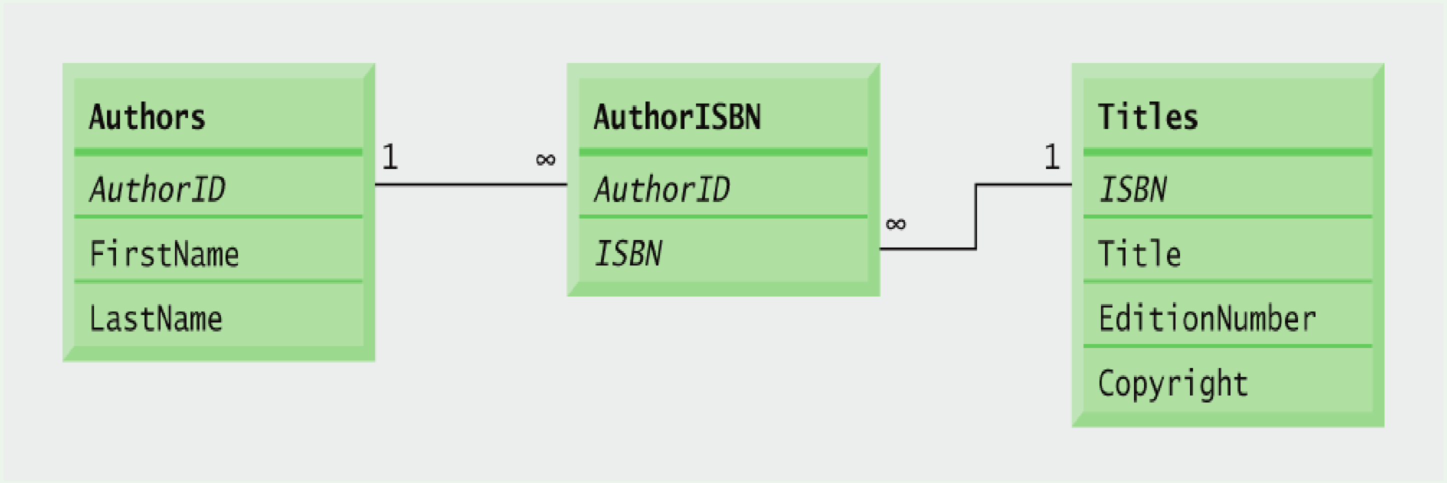
- Develop Java applications that insert, update or delete database records.
- Use PreparedStatements.
- Use the RowSet interface to create a disconnected recordset.

Sample Books database

- Tables:
  - **`authors`**
    - `authorID`, `firstName`, `lastName`
  - **`titles`**
    - `isbn`, `title`, `editionNumber`, `copyright`, `publisherID`, `imageFile`, `price`
  - **`authorISBN`**
    - `authorID`, `isbn`

Entity-relationship (ER) diagram

- Table relationships in the books database:



Books database example

```
CREATE TABLE authors (
   authorID INT NOT NULL,
   firstName varchar (20) NOT NULL,
   lastName varchar (30) NOT NULL,
   PRIMARY KEY (authorID)
);
```

Books database example

```
CREATE TABLE titles (
   isbn varchar (20) NOT NULL,
   title varchar (100) NOT NULL,
   editionNumber INT NOT NULL,
   copyright varchar (4) NOT NULL,
   PRIMARY KEY (isbn)
);
```

Books database example

```
CREATE TABLE authorISBN (
   authorID INT NOT NULL,
   isbn varchar (20) NOT NULL,
   FOREIGN KEY (authorID) REFERENCES authors
(authorID),
   FOREIGN KEY (isbn) REFERENCES titles (isbn)
);
```

Books database example

```
INSERT INTO authors  VALUES (1, 'Harvey','Deitel');

INSERT INTO authors VALUES (2, 'Paul','Deitel');

INSERT INTO authors VALUES (3, 'Andrew','Goldberg');

INSERT INTO authors VALUES (4, 'David','Choffnes');
```

Books database example

```
INSERT INTO titles (isbn,title,editionNumber,copyright)
VALUES ('0131869000','Visual Basic 2005 How to Program',3,'2006');

INSERT INTO titles (isbn,title,editionNumber,copyright) VALUES
('0131525239','Visual C# 2005 How to Program',2,'2006');
INSERT INTO titles (isbn,title,editionNumber,copyright) VALUES ('0132222205','Java
How to Program',7,'2007');
INSERT INTO titles (isbn,title,editionNumber,copyright) VALUES ('0131857576','C++
How to Program',5,'2005');
INSERT INTO titles (isbn,title,editionNumber,copyright) VALUES ('0132404168','C
How to Program',5,'2007');
INSERT INTO titles (isbn,title,editionNumber,copyright) VALUES
('0131450913','Internet & World Wide Web How to Program',3,'2004');
INSERT INTO titles (isbn,title,editionNumber,copyright) VALUES
('0131828274','Operating Systems',3,'2004');
```

Books database example

```
INSERT INTO authorISBN (authorID,isbn) VALUES (1,'0131869000');
INSERT INTO authorISBN (authorID,isbn) VALUES   (2,'0131869000');
INSERT INTO authorISBN (authorID,isbn) VALUES   (1,'0131525239');
INSERT INTO authorISBN (authorID,isbn) VALUES   (2,'0131525239');
INSERT INTO authorISBN (authorID,isbn) VALUES   (1,'0132222205');
INSERT INTO authorISBN (authorID,isbn) VALUES   (2,'0132222205');
INSERT INTO authorISBN (authorID,isbn) VALUES   (1,'0131857576');
INSERT INTO authorISBN (authorID,isbn) VALUES   (2,'0131857576');
INSERT INTO authorISBN (authorID,isbn) VALUES   (1,'0132404168');
INSERT INTO authorISBN (authorID,isbn) VALUES   (2,'0132404168');
INSERT INTO authorISBN (authorID,isbn) VALUES   (1,'0131450913');
INSERT INTO authorISBN (authorID,isbn) VALUES   (2,'0131450913');
INSERT INTO authorISBN (authorID,isbn) VALUES   (3,'0131450913');
INSERT INTO authorISBN (authorID,isbn) VALUES   (1,'0131828274');
INSERT INTO authorISBN (authorID,isbn) VALUES   (2,'0131828274');
INSERT INTO authorISBN (authorID,isbn) VALUES   (4,'0131828274');
```

DisplayAuthorsTable.java example

```
// load the driver class
Class.forName( DRIVER );
// establish connection to database
conn = DriverManager.getConnection( DATABASE_URL, "user", "password" );
st = conn.createStatement();
rs = st.executeQuery("SELECT * FROM authors");
ResultSetMetaData md = rs.getMetaData();
//create columns headers
for( int i=1;i <= md.getColumnCount();i++)
{
	columns.addElement(md.getColumnName(i));
}
```

DisplayQueryResults.java example

- ResultSetTableModel class: uses a table model.
- Provides implementations for the following three methods:
  - public int getRowCount();
  - public int getColumnCount();
  - public Object getValueAt(int row, int column);

Scrollable Result Sets

- With the JDBC 2.X APIs and higher, you will be able to do the following:
- Scroll forward and backward in a result set or move to a specific row
- Make updates to database tables using methods in the Java programming language instead of using SQL commands.

Create a scrollable ResultSet object

- TYPE_FORWARD_ONLY - cursor may move only forward.
- TYPE_SCROLL_INSENSITIVE - scrollable cursor but generally **not sensitive** to changes made by others.
- TYPE_SCROLL_SENSITIVE - scrollable cursor and generally **sensitive** to changes made by others.

```
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);

ResultSet srs = stmt.executeQuery("SELECT * FROM
STUDENTS");
```

Inserting a new row to a ResultSet

- The first step will be to move the cursor to the insert row, by calling the method **moveToInsertRow**.
- The next step is to set a value for each column in the row:

rs.**moveToInsertRow()**; //create a buffer for the new row

rs.**updateString**(1,”Toronto”); // populate the first field

rs.**updateString**(2,”Centennial”); //populate the second field

rs.**insertRow()**; //Insert the contents of the insert row into table

Updating an existing row

- The updateRow() method is provided to update an existing row in a table.
- The following code shows how to update the current row for the same RecordSet object mentioned above:

```
rs.updateString(1,”HP Campus”); //update the first field
rs.updateString(2,”Centennial College”); //update the second field
rs.updateRow();
```

Deleting a row

- The method **deleteRow()** deletes the current row from this ResultSet object and from the underlying database.
- This method cannot be called when the cursor is on the insert row.
- The following statement deletes the current row:

rs.**deleteRow()**;

Navigating through records

When a new **ResultSet** object is created it maintains a cursor that gets positioned before the first row.

- The method **next()** can be used to **move the cursor to the next row** if there is one.
  - It returns false if there are no more records

```
while(rs.next())
{
    //access columns here
}
```

Navigating through records

- Here are other navigational methods:
  - **first()** - Moves the cursor to the first row in this ResultSet object.
  - **last()** - Moves the cursor to the last row in this ResultSet object.
  - **previous()** - Moves the cursor to the previous row in this ResultSet object
  - **beforeFirst()** - Moves the cursor to the front of this ResultSet object, just before the first row
  - **afterLast()** - Moves the cursor to the end of this ResultSet object, just after the last row

Navigating through records

- **isAfterLast()** - Indicates whether the cursor is after the last row in this ResultSet object.
- **isBeforeFirst()** - Indicates whether the cursor is before the first row in this ResultSet object.
- The method **absolute(int row)** moves the cursor to the given row number in this ResultSet object.
- The method **relative(int rows)** moves the cursor a relative number of rows, either positive or negative.

Navigating through records

Example:

```
srs.absolute(4);
int rowNum = srs.getRow(); // rowNum should be 4
srs.relative(-3);
int rowNum = srs.getRow(); // rowNum should be 1
srs.relative(2);
int rowNum = srs.getRow(); // rowNum should be 3
```

`RowSet` Interface

- Interface **`RowSet`**
  - Configures the database connection automatically
  - Prepares query statements automatically
  - Provides **set** methods to specify the properties needed to establish a connection
  - Part of the **`javax.sql`** package
- Two types of **`RowSet`**
  - **Connected** `RowSet`
    - Connects to database once and remain connected
  - **Disconnected** `RowSet`
    - Connects to database, executes a query and then closes connection

RowSet Interface

- Package `javax.sql.rowset`
  - **`JdbcRowSet`**
    - **Connected** `RowSet`
    - Wrapper around a `ResultSet`
    - **Scrollable** and **updatable** by default

```
// connect to database books and query database
try (JdbcRowSet rowSet =
        RowSetProvider.newFactory().createJdbcRowSet())
{
//
}
```

JdbcRowSetTest.java example

```
try (JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet())
{
        // specify JdbcRowSet properties
        rowSet.setUrl(DATABASE_URL);
        rowSet.setUsername(USERNAME);
        rowSet.setPassword(PASSWORD);
        rowSet.setCommand("SELECT * FROM authors"); // set query
        rowSet.execute(); // execute query
                ……..
}
catch (SQLException sqlException)
{
        sqlException.printStackTrace();
}
```

Using Prepared Statements with JDBC

- To improve the **performance** when performing the same operation multiple times, use a **PreparedStatement** object.
- Better security
- A PreparedStatement is *precompiled* by the DBMS.
- You may also pass arguments to a prepared statement.
- To create a prepared statement use the method **prepareStatement** instead of the method createStatement.

Using Prepared Statements with JDBC

- **PreparedStatement** pst = conn.prepareStatement("Insert into Authors (authorID, firstname, lastname) VALUES(?,?,?)");

Using Prepared Statements with JDBC

- The IN arguments, indicated by '?', can be filled by **set*XXX*** methods.

```
//populate the fields
pst.setInt(1, 5);
pst.setString(2, "Sam");
pst.setString(3, "Malone");
```

- Execute the prepared statement using executeUpdate method:

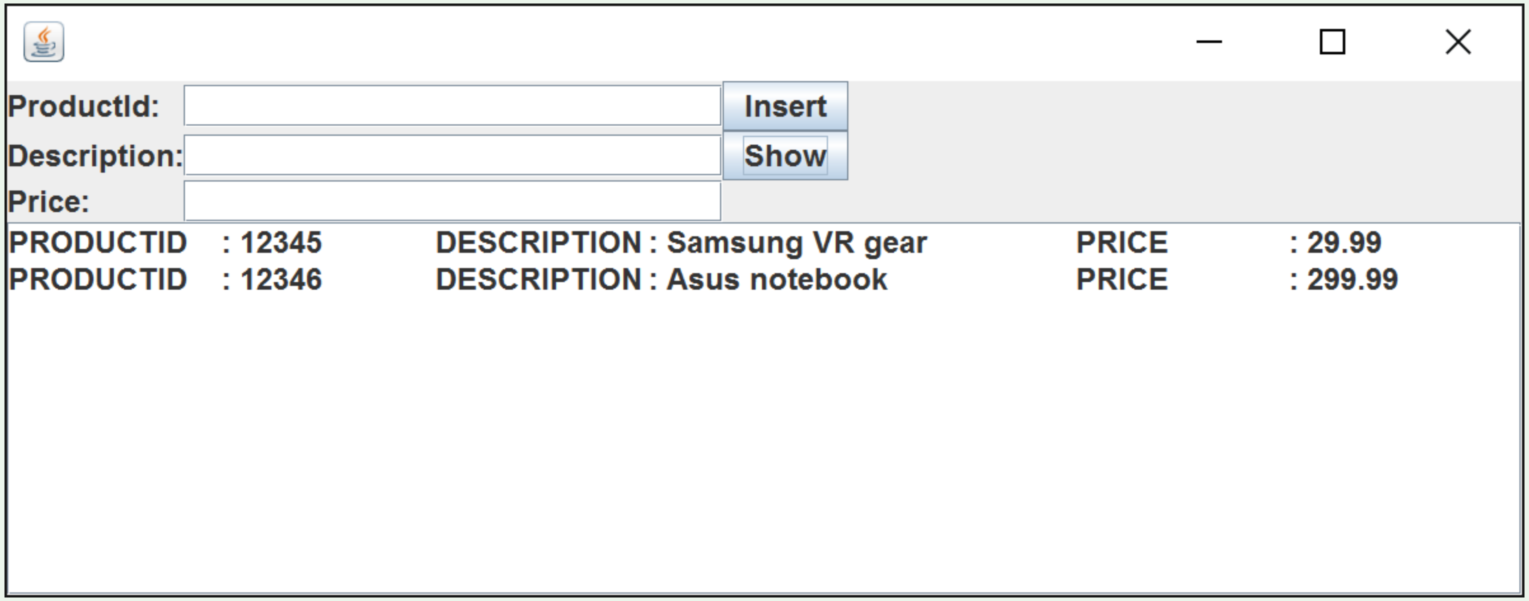
```
int val = pst.executeUpdate(); //returns the row count
```

PreparedStatementTest Example

```
try {
	// load the driver class
	Class.forName( DRIVER );
	// establish connection to database
	conn = DriverManager.getConnection( DATABASE_URL, "user", "password" );
	pst = conn.prepareStatement("Insert into Authors (authorID, firstname, lastname)
	VALUES(?,?,?)");
	//populate the fields
	pst.setInt(1, 5);
	pst.setString(2, "Sam");
	pst.setString(3, "Malone");
	int val = pst.executeUpdate(); //returns the row count
	pst.close();
}
catch (SQLException e) { e.printStackTrace(); }
…..
```

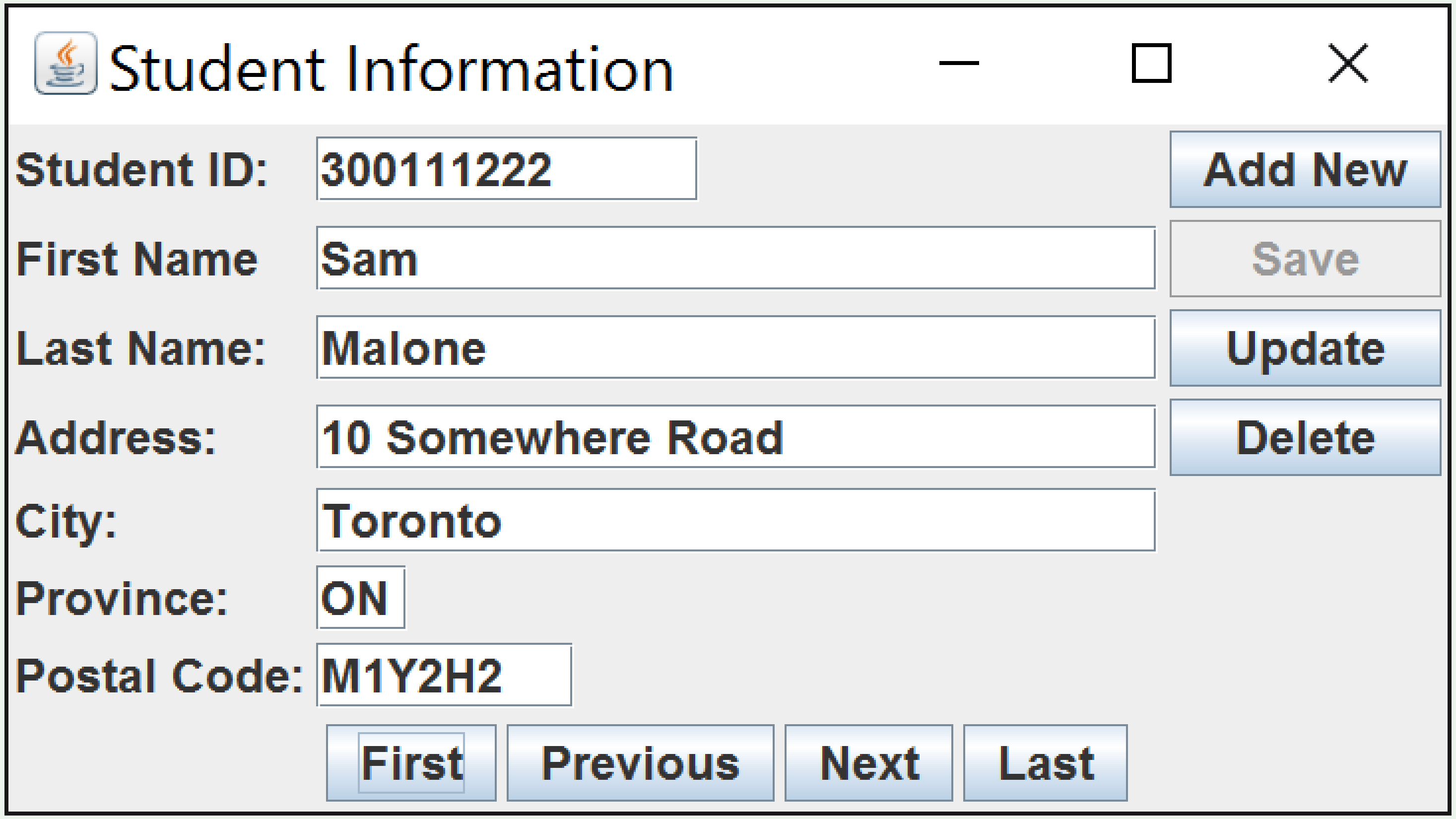
PreparedStatementTestUI Example

- Using Product table



```
PRODUCTID    : 12345        DESCRIPTION : Samsung VR gear        PRICE        : 29.99
PRODUCTID    : 12346        DESCRIPTION : Asus notebook          PRICE        : 299.99
```

Student Information Application


Student Information

Student ID:	300111222
First Name	Sam
Last Name:	Malone
Address:	10 Somewhere Road
City:	Toronto
Province:	ON
Postal Code:	M1Y2H2

Student Information Application

- Student class - represents a Customer object
- StudentData class – implements data access tier
- StudentScreen – UI, uses a GridBagLayout

CallableStatement

- A CallableStatement object provides a way to call stored procedures in a standard way for all DBMSs.
- A stored procedure is stored in a database; the *call* to the stored procedure is what a CallableStatement object contains

```
String sql = "{call getTestData(?, ?)}";
CallableStatement cstmt2 = con.prepareCall(sql,
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
```

References

- Textbook
- https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html
- https://docs.oracle.com/javase/tutorial/jdbc/index.html
- http://www.java2s.com/Tutorials/Java/JDBC_How_to/index.htm