COP 4610L: Applications in the Enterprise Spring 2005

Introduction to JDBC – Part 2

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cop4610L/spr2005

School of Electrical Engineering and Computer Science
University of Central Florida

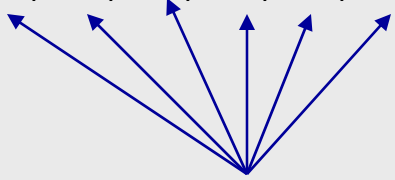
The `PreparedStatement` Interface

- In the previous set of notes, once we established a connection to a particular database, it was used to send an SQL statement from the application to the database.
- The `Statement` interface is used to execute static SQL statements that contain no parameters.
- The `PreparedStatement` interface, which extends the Statement interface, is used to execute a precompiled SQL statement with or without `IN` parameters.
- Since the SQL statements are precompiled, they are extremely efficient for repeated execution.

The PreparedStatement Interface
(cont.)

- A PreparedStatement object is created using the preparedStatement method in the Connection interface.

```
Statement pstmt = connection.preparedStatement
  ("insert into bikes (bikename, size, color,
      cost, purchased, mileage) +
       values ( ?, ?, ?, ?, ?, ?)" );
```



Placeholders for the values that will be dynamically provided by the user.

The `PreparedStatement` Interface
(cont.)

- As a subinterface of `Statement`, the `PreparedStatement` interface inherits all the methods defined in `Statement`. It also provides the methods for setting parameters in the object of `PreparedStatement`.
- These methods are used to set the values for the parameters before executing statements or procedures.
- In general, the set methods have the following signature:

```
setX (int parameterIndex, X value);
```

where `X` is the type of parameter and `parameterIndex` is the index of the parameter in the statement.

The `PreparedStatement` Interface (cont.)

- As an example, the method

```
setString( int parameterIndex, String value)
```

sets a String value to the specified parameter.

- Once the parameters are set, the prepared statement is executed like any other SQL statement where `executeQuery()` is used for SELECT statements and `executeUpdate()` is used for DDL or update commands.

- These two methods are similar to those found in the `Statement` interface except that they have no parameters since the SQL statements are already specified in the `preparedStatement` method when the object of a `PreparedStatement` is created.

FindBikeUsingPreparedStatement

```
import javax.swing.*;
import java.sql.*;
import java.awt.*;
import java.awt.event.*;

public class FindBikeUsingPreparedStatement extends JApplet {
  boolean isStandalone = false;
  private JTextField jtfbike = new JTextField(25);
  private JTextField jtfcost = new JTextField(6);
  private JButton jbtShowCost = new JButton("Show Bike Cost Info");

  // PreparedStatement for executing queries
  private PreparedStatement pstmt;

  /** Initialize the applet */
  public void init() {
    // Initialize database connection and create a PreparedStatement object
    initializeDB();

    jbtShowCost.addActionListener(
      new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jbtShowCost_actionPerformed(e);
      }
    });
```

PreparedStatement object

```
 JPanel jPanel1 = new JPanel();
  jPanel1.add(new JLabel("Bike Name"));
  jPanel1.add(jtfbike);
  jPanel1.add(jbtShowCost);
  this.getContentPane().add(jPanel1, BorderLayout.NORTH);
}

private void initializeDB() {
  try {
    // Load the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("Driver loaded");
    // Establish a connection
    Connection connection = DriverManager.getConnection
                          ("jdbc:mysql://localhost/bikedb", "root", "root");
    System.out.println("Database connected");

    String queryString = "select cost from bikes where bikename = ?";
    // Create a statement
    pstmt = connection.prepareStatement(queryString);
  }
  catch (Exception ex) {
    ex.printStackTrace();
  }
}
```

queryString contains the SQL statement with the ? Placeholder for the value to be determined at run-time.

Invoke the preparedStatement() method on the connection.

```
  private void jbtShowCost_actionPerformed(ActionEvent e) {
  String bikename = jtfbike.getText();
  String cost = jtfcost.getText();
  try {
    pstmt.setString(1, bikename);
    ResultSet rset = pstmt.executeQuery();
    if (rset.next()) {
      String price = rset.getString(1);
       // Display result in a dialog box
      JOptionPane.showMessageDialog(null, bikename + " cost $" + price);
    }
    else {    // Display result in a dialog box
      JOptionPane.showMessageDialog(null, "Bike Not Found");
    }
  }
  catch (SQLException ex) {
    ex.printStackTrace();     }
}
```

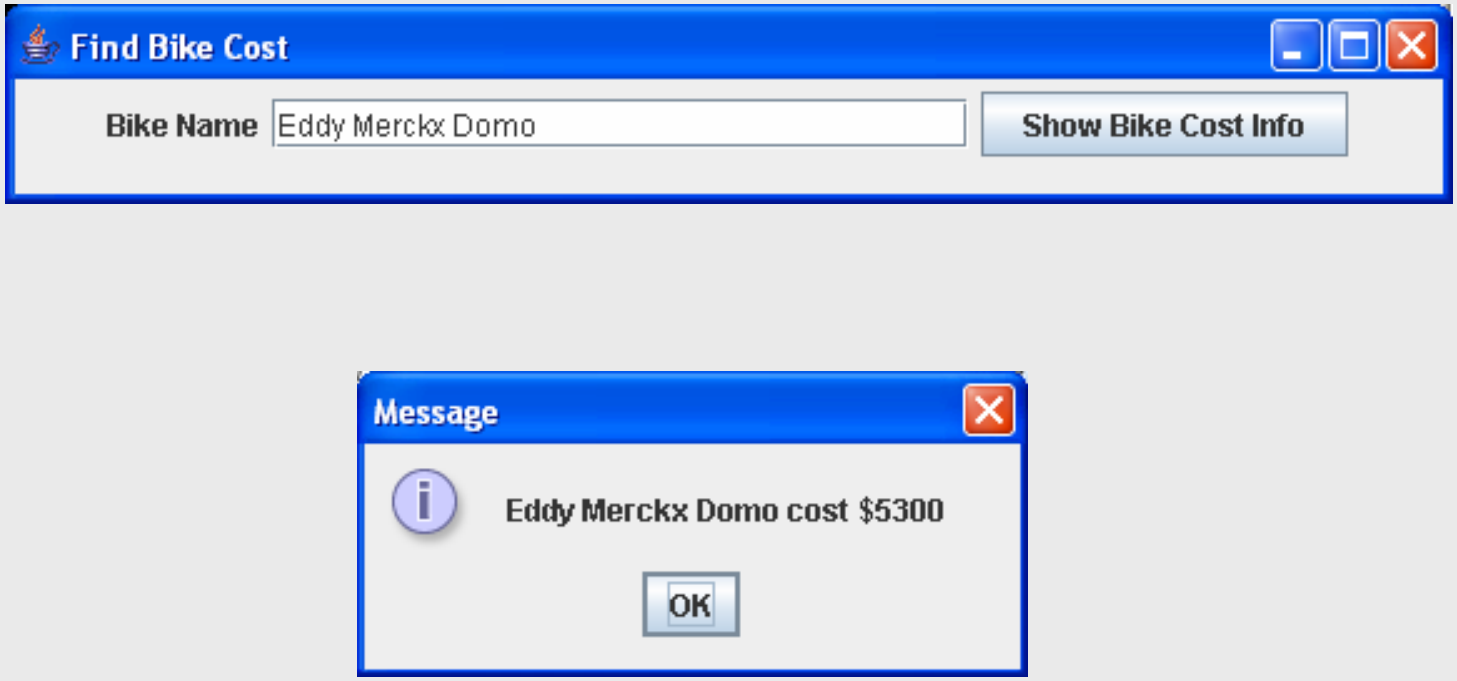
Set first parameter value for PreparedStatement object

Execute query using PreparedStatement object

Get data from results set returned by JDBC.

```
/** Main method */
  public static void main(String[] args) {
    FindBikeUsingPreparedStatement applet = new
FindBikeUsingPreparedStatement();
    JFrame frame = new JFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setTitle("Find Bike Cost");
    frame.getContentPane().add(applet, BorderLayout.CENTER);
    applet.init();    applet.start();    frame.setSize(580, 80);
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation((d.width - frame.getSize().width) / 2,
      (d.height - frame.getSize().height) / 2);
    frame.setVisible(true);
  } }
```


Output from `FindBikeUsingPreparedStatement`



The RowSet Interface (cont.)

- Interface `RowSet` provides several `set` methods that allow the programmer to specify the properties needed to establish a connection (such as the database URL, user name, password, etc.) and a create a `Statement` (such as a query).
- Interface `RowSet` also provides several `get` methods that return these properties.
- More information on these methods can be found at: http://java.sun.com/j2se/1.5.0/docs/api/javax/sql/RowSet.html

The RowSet Interface (cont.)

- RowSet is part of the javax.sql package.
- Although part of the Java 2 Standard Edition, the classes and interfaces of package javax.sql are most often used in the context of the Java 2 Platform Enterprise Edition (J2EE).
- We may get to some J2EE development later in the semester. You can learn more about J2EE at www.java.sun.com/j2ee.

Using the RowSet Interface

- There are two types of RowSet objects – connected and disconnected.
- A connected RowSet object connects to the database once and remains connected until the application terminates.
- A disconnected RowSet object connects to the database, executes a query to retrieve the data from the database and then closed the connection. A program may change the data in a disconnected RowSet while it is disconnected. Modified data can be updated to the database after a disconnected RowSet reestablishes the connection with the database.

Using the RowSet Interface (cont.)

- J2SE 5.0 package javax.sql.rowset contains two subinterfaces of `RowSet` – `JdbcRowSet` and `CachedRowSet`.

- `JdbcRowSet`, a connected `RowSet`, acts as a wrapper around a `ResultSet` object, and allows programmers to scroll through and update the rows in the `ResultSet` object. Recall that by default, a `ResultSet` object is non-scrollable and read only – you must explicitly set the result-set type constant to TYPE_SCROLL_INSENSITIVE and set the result set concurrency constant to CONCUR_UPDATEABLE to make a `ResultSet` object scrollable and updatable.

Using the RowSet Interface (cont.)

- A `JdbcRowSet` object is scrollable and updatable by default.
- `CachedRowSet,` a disconnected `RowSet,` caches the data of a ResultSet in memory and disconnects from the database. Like JdbcRowSet, a CachedRowSet object is scrollable and updatable by default.
- A CachedRowSet is also serializable, so it can be passed between Java applications through a network.
- However, a CachedRowSet has a limitation – the amount of data that can be stored in memory is limited.
- There are three other subinterfaces in this package (FilteredRowSet, WebRowSet, and JoinRowSet).

Using the RowSet Interface (cont.)

- The code example on the next couple of pages illustrates the use of the RowSet interface.
- Notice that unlike the TableSet version in the previous set of notes, the connection is made and the query executed automatically.

Class: JdbcRowSetTest – page 1

```
// Displaying the contents of the bikes table using JdbcRowSet.
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import javax.sql.rowset.JdbcRowSet;
import com.sun.rowset.JdbcRowSetImpl; // Sun's JdbcRowSet implementation

public class JdbcRowSetTest
{
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DATABASE_URL = "jdbc:mysql://localhost/bikedb";
   static final String USERNAME = "root";
   static final String PASSWORD = "root";

   // constructor connects to database, queries database, processes
   // results and displays results in window
   public JdbcRowSetTest()
   {
      // connect to database books and query database
      try
      {
         Class.forName( JDBC_DRIVER ); // load database driver class
```


Class: JdbcRowSetTest – page 2

```
// specify properties of JdbcRowSet
JdbcRowSet rowSet = new JdbcRowSetImpl();
rowSet.setUrl( DATABASE_URL ); // set database URL
rowSet.setUsername( USERNAME ); // set username
rowSet.setPassword( PASSWORD ); // set password
//set query

rowSet.setCommand( "SELECT bikename,size,purchased,cost FROM bikes" );
rowSet.execute(); // execute query

// process query results
ResultSetMetaData metaData = rowSet.getMetaData();
int numberOfColumns = metaData.getColumnCount();
System.out.println( "Bikes Table of bikedb Database:" );

// display rowset header
for ( int i = 1; i <= numberOfColumns; i++ )
   System.out.printf( "%-12s\t", metaData.getColumnName( i ) );
System.out.println();
```

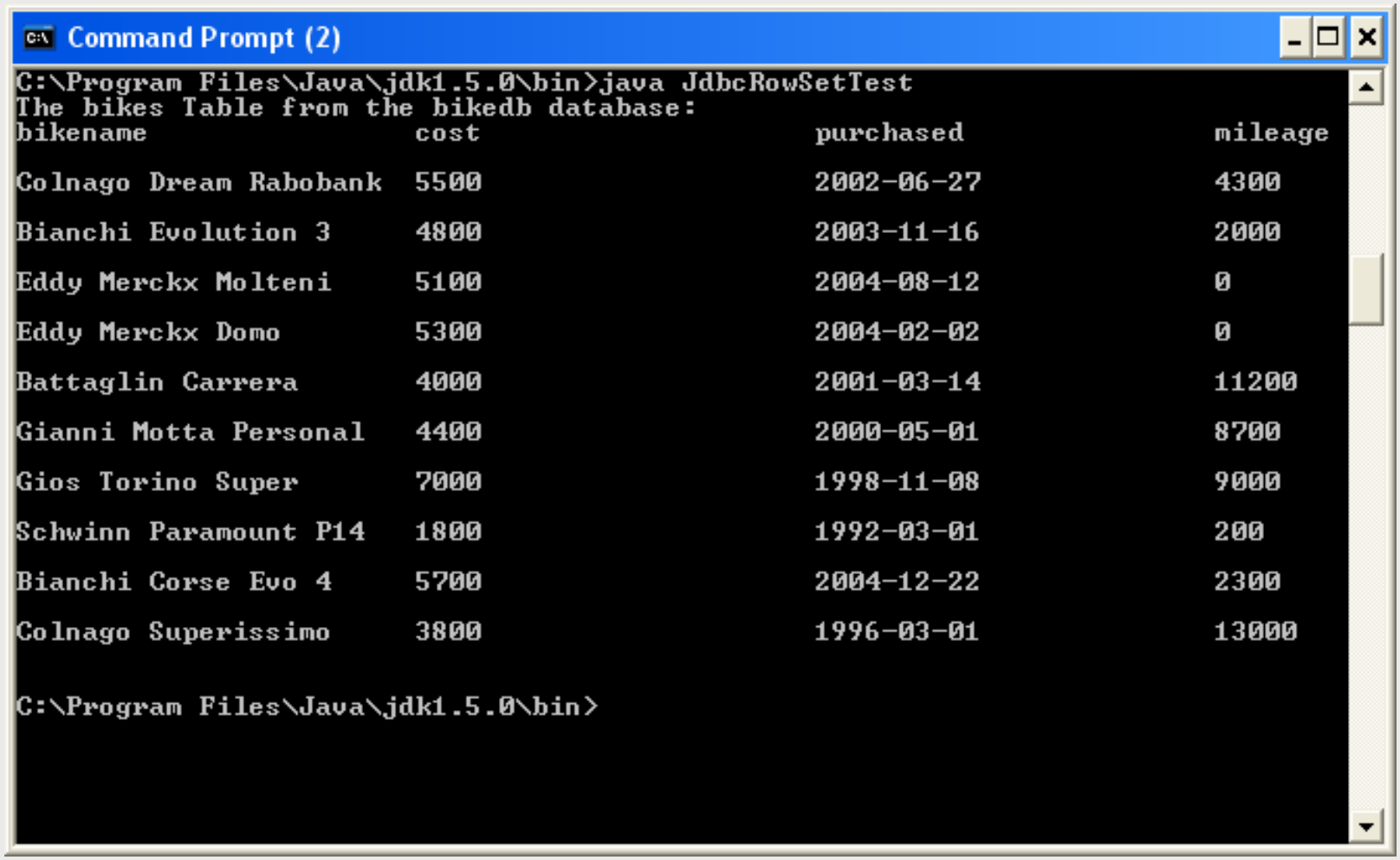
SQL command to be executed.

Class: JdbcRowSetTest – page 3

```
        // display each row
        while ( rowSet.next() )   {
              for ( int i = 1; i <= numberOfColumns; i++ )
              System.out.printf( "%-12s\t", rowSet.getObject( i ) );
           System.out.println();
        } // end while
      } // end try
      catch ( SQLException sqlException )  {
            sqlException.printStackTrace();
        System.exit( 1 );
      } // end catch
      catch ( ClassNotFoundException classNotFound )  {
              classNotFound.printStackTrace();
        System.exit( 1 );
      } // end catch
    } // end DisplayBikes constructor

    // launch the application
    public static void main( String args[] )  {
    {      JdbcRowSetTest window = new JdbcRowSetTest();
    } // end main
  } // end class JdbcRowSetTest
```


Execution of JdbcRowSetTest



Display of default query results from JdbcRowSetTest application