Coding Interview on Database - JDBC (java.sql)

Connecting to a Database

This example uses the JDBC-ODBC bridge to connect to a database called "mydatabase".

```
try {

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

String url = "jdbc:odbc:mydatabase";

Connection con = DriverManager.getConnection(

url, "login", "password");

} catch (ClassNotFoundException e) {

} catch (SQLException e) {

}
```

Creating a Table

This example creates a table called "mytable" with three columns: COL_A which holds strings, COL_B which holds integers, and COL_C which holds floating point numbers.

```
try {

Statement stmt = con.createStatement();

stmt.executeUpdate("CREATE TABLE mytable (

COL_A VARCHAR(100), COL_B INTEGER, COL_C FLOAT)");

} catch (SQLException e) {

}
```

Entering a New Row into a Table

This example enters a row containing a string, an integer, and a floating point number into the table called "mytable".

```
try {

Statement stmt = connection.createStatement();
```

```
stmt.executeUpdate("INSERT INTO mytable

VALUES ('Patrick Chan', 123, 1.23)");

connection.close();

} catch (SQLException e) {

}
```

Getting All Rows from a Table

This example retrieves all the rows from a table called "mytable". A row in "mytable" consists of a string, integer, and floating point number.

```
try {

Statement stmt = connection.createStatement();

// Get data using colunm names.

ResultSet rs = stmt.executeQuery(

"SELECT * FROM mytable");

while (rs.next()) {

String s = rs.getString("COL_A");

int i = rs.getInt("COL_B");

float f = rs.getFloat("COL_C");

process(s, i, f);

}

// Get data using colunm numbers.

rs = stmt.executeQuery(

"SELECT * FROM mytable");

while (rs.next()) {

String s = rs.getString(1);
```

```
int i = rs.getInt(2);

float f = rs.getFloat(3);

process(s, i, f);

}

} catch (SQLException e) {

}
```

Getting Particular Rows from a Table

This example retrieves all rows from a table called "mytable" whose column COL_A equals ``Patrick Chan". A row in "mytable" consists of a string, integer, and floating point number.

```
try {

Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(

"SELECT * FROM mytable WHERE COL_A = 'Patrick Chan'");

rs.next();

String s = rs.getString("COL_A");

int i = rs.getInt("COL_B");

float f = rs.getFloat("COL_C");

process(s, i, f);

} catch (SQLException e) {

}
```

Updating a Row of Data in a Table

This example updates a row in a table called ``mytable". In particular, for all rows whose column COL_B equals 123, column COL_A is set to "John Doe".

```
try {
```

```
Statement stmt = connection.createStatement();

int numUpdated = stmt.executeUpdate(

"UPDATE mytable SET COL_A = 'John Doe'

WHERE COL_B = 123");

connection.close();

} catch (SQLException e) {

}
```

Using a Prepared Statement

A prepared statement should be used in cases where a particular SQL statement is used frequently. The prepared statement is more expensive to set up but executes faster than a statement. This example demonstrates a prepared statement for getting all rows from a table called "mytable" whose column COL_A equals "Patrick Chan". This example also demonstrates a prepared statement for updating data in the table. In particular, for all rows whose column COL_B equals 123, column COL_A is set to "John Doe".

```
try {

// Retrieving rows from the database.

PreparedStatement stmt = connection.prepareStatement(

"SELECT * FROM mytable WHERE COL_A = ?");

int columm = 1;

stmt.setString(columm, "Patrick Chan");

ResultSet rs = stmt.executeQuery();

// Updating the database.

stmt = connection.prepareStatement(

"UPDATE mytable SET COL_A = ? WHERE COL_B = ?");

columm = 1;

stmt.setString(columm, "John Doe");
```

```
columnm = 2;

stmt.setInt(columnm, 123);

int numUpdated = stmt.executeUpdate();

} catch (SQLException e) {

}
```


Handling Events with an Anonymous Class

If an event handler is specific to a component (that is, not shared by other components), there is no need to declare a class to handle the event. The event handler can be implemented using an anonymous inner class. This example demonstrates an anonymous inner class to handle key events for a component.

```
component.addKeyListener(new KeyAdapter() {

public void keyPressed(KeyEvent evt) {

}

});
```

Handling Action Events

Action events are fired by subclasses of AbstractButton and includes buttons, checkboxes, and menus.

```
AbstractButton button = new JButton(quot;OK");

button.addActionListener(new MyActionListener());

public class MyActionListener

implements ActionListener {

public void actionPerformed(ActionEvent evt) {

// Determine which abstract

// button fired the event.
```

```
AbstractButton button =

(AbstractButton)evt.getSource();

}

}
```

Handling Key Presses

You can get the key that was pressed either as a key character (which is a Unicode character) or as a key code (a special value representing a particular key on the keyboard).

```
component.addKeyListener(new MyKeyListener());

public class MyKeyListener extends KeyAdapter {

public void keyPressed(KeyEvent evt) {

// Check for key characters.

if (evt.getKeyChar() == 'a') {

process(evt.getKeyChar());

}

// Check for key codes.

if (evt.getKeyCode() == KeyEvent.VK_HOME) {

process(evt.getKeyCode());

}

}

}
```

Handling Mouse Clicks

```
component.addMouseListener(

new MyMouseListener());
```

```
public class MyMouseListener
extends MouseAdapter {
public void mouseClicked(MouseEvent evt) {
if ((evt.getModifiers() &
InputEvent.BUTTON1_MASK) != 0) {
processLeft(evt.getPoint());
}
if ((evt.getModifiers() &
InputEvent.BUTTON2_MASK) != 0) {
processMiddle(evt.getPoint());
}
if ((evt.getModifiers() &
InputEvent.BUTTON3_MASK) != 0) {
processRight(evt.getPoint());
}
}
}
```

Handling Mouse Motion

```
component.addMouseMotionListener(
new MyMouseMotionListener());
public class MyMouseMotionListener
extends MouseMotionAdapter {
public void mouseMoved(MouseEvent evt) {
```

```
// Process current position of cursor
// while all mouse buttons are up.
process(evt.getPoint());
}
public void mouseDragged(MouseEvent evt) {
// Process current position of cursor
// while mouse button is pressed.
process(evt.getPoint());
}
}
```

Detecting Double and Triple Clicks

```
component.addMouseListener(
new MyMouseListener());
public class MyMouseListener extends MouseAdapter {
public void mouseClicked(MouseEvent evt) {
if (evt.getClickCount() == 3) {
// triple-click
} else if (evt.getClickCount() == 2) {
// double-click
}
}
}
```

Handling Focus Changes

```
component.addFocusListener(
new MyFocusListener());
public class MyFocusListener
extends FocusAdapter {
public void focusGained(FocusEvent evt) {
// The component gained the focus.
}
public void focusLost(FocusEvent evt) {
// The component lost the focus.
}
}
```

Files, Streams, I/O (java.io)

Constructing a Path

On Windows, this example creates the path \blash a\blash b. On Unix, the path would be /a/b.

```
String path = File.separator +
"a" + File.separator + "b";
```

Reading Text from Standard Input

```
try {
BufferedReader in = new BufferedReader(
new InputStreamReader(System.in));
```

```
String str = "";
while (str != null) {
System.out.print("> prompt ");
str = in.readLine();
process(str);
}
} catch (IOException e) {
}
```

Reading Text from a File

```
try {
BufferedReader in = new BufferedReader(
new FileReader("infilename"));
String str;
while ((str = in.readLine()) != null) {
process(str);
}
in.close();
} catch (IOException e) {
}
```

Writing to a File

If the file does not already exist, it is automatically created.

```
try {
BufferedWriter out = new BufferedWriter(
```

```
new FileWriter("outfilename"));

out.write("aString");

out.close();

} catch (IOException e) {

}
```

Creating a Directory

```
(new File("directoryName")).mkdir();
```

Appending to a File

```
try {

BufferedWriter out = new BufferedWriter(

new FileWriter("filename", true));

out.write("aString");

out.close();

} catch (IOException e) {

}
```

Deleting a File

```
(new File("filename")).delete();
```

Deleting a Directory

```
(new File("directoryName")).delete();
```

Creating a Temporary File

```
try {

// Create temp file.

File temp = File.createTempFile(
```

```
"pattern", ".suffix");
// Delete temp file when program exits.
temp.deleteOnExit();
// Write to temp file
BufferedWriter out = new BufferedWriter(
new FileWriter(temp));
out.write("aString");
out.close();
} catch (IOException e) {
}
```

Using a Random Access File

```
try {
File f = new File("filename");
RandomAccessFile raf =
new RandomAccessFile(f, "rw");
// Read a character.
char ch = raf.readChar();
// Seek to end of file.
raf.seek(f.length());
// Append to the end.
raf.writeChars("aString");
raf.close();
} catch (IOException e) {
```

```
}
```

Serializing an Object

The object to be serialized must implement java.io.Serializable.

```
try {
ObjectOutput out = new ObjectOutputStream(
new FileOutputStream("filename.ser"));
out.writeObject(object);
out.close();
} catch (IOException e) {
}
```

Deserializing an Object

This example deserializes a java.awt.Button object.

```
try {
ObjectInputStream in = new ObjectInputStream(
new FileInputStream("filename.ser"));
AnObject object = (AnObject) in.readObject();
in.close();
} catch (ClassNotFoundException e) {
} catch (IOException e) {
}
```

Traversing a Directory

```
public static void traverse(File f) {
process(f);
```

```
if (f.isDirectory()) {
String[] children = f.list();
for (int i=0; i<children.length; i++) {
traverse(new File(f, children[i]));
}
}
}
```

Reading UTF-8 Encoded Data

```
try {
BufferedReader in = new BufferedReader(
new InputStreamReader(new FileInputStream(
"infilename"), "UTF8"));
String str = in.readLine();
} catch (UnsupportedEncodingException e) {
} catch (IOException e) {
}
```

Writing UTF-8 Encoded Data

```
try {
Writer out = new BufferedWriter(
new OutputStreamWriter(new FileOutputStream(
"outfilename"), "UTF8"));
out.write(aString);
out.close();
```

```
} catch (UnsupportedEncodingException e) {
} catch (IOException e) {
}
```

Reading ISO Latin-1 Encoded Data

```
try {
BufferedReader in = new BufferedReader(
new InputStreamReader(new FileInputStream(
"infilename"), "8859_1"));
String str = in.readLine();
} catch (UnsupportedEncodingException e) {
} catch (IOException e) {
}
```

Writing ISO Latin-1 Encoded Data

```
try {
Writer out = new BufferedWriter(
new OutputStreamWriter(new FileOutputStream(
"outfilename"), "8859_1"));
out.write(aString);
out.close();
} catch (UnsupportedEncodingException e) {
} catch (IOException e) {
}
```

Networking (java.net)

Creating a URL

```
try {
// With components.
URL url = new URL("http","hostname", 80, "index.html");
// With a single string.
url = new URL(
"http://hostname:80/index.html");
} catch (MalformedURLException e) {
}
```

Parsing a URL

```
try {
URL url = new URL("http://hostname:80/index.html#_top_");
String protocol = url.getProtocol();// http
String host = url.getHost();// hostname
int port = url.getPort();// 80
String file = url.getFile(); // index.html
String ref = url.getRef();// _top_
} catch (MalformedURLException e) {
}
```

Reading Text from a URL

```
try {
```

```
URL url = new URL("http://hostname:80/index.html");
BufferedReader in = new BufferedReader(
new InputStreamReader(url.openStream()));
String str;
while ((str = in.readLine()) != null) {
process(str);
}
in.close();
} catch (MalformedURLException e) {
} catch (IOException e) {
}
```

Resolving a Hostname

Creating a Client Socket

```
try {
InetAddress addr = InetAddress.getByName("java.sun.com");
int port = 80;
Socket sock = new Socket(addr, port);
} catch (IOException e) {
}
```

Creating a Server Socket

```
try {
int port = 2000;
ServerSocket srv = new ServerSocket(port);
```

```
// Wait for connection from client.
Socket socket = srv.accept();
} catch (IOException e) {
}
```

Reading Text from a Socket

```
try {
BufferedReader rd = new BufferedReader(
new InputStreamReader(socket.getInputStream()));
String str;
while ((str = rd.readLine()) != null) {
process(str);
}
rd.close();
} catch (IOException e) {
}
```

Writing Text to a Socket

```
try {
BufferedWriter wr = new BufferedWriter(
new OutputStreamWriter(socket.getOutputStream()));
wr.write("aString");
wr.flush();
} catch (IOException e) {
}
```

Sending a Datagram

```
public static void send(InetAddress dst,
int port, byte[] outbuf, int len) {
try {
DatagramPacket request = new DatagramPacket(
outbuf, len, dst, port);
DatagramSocket socket = new DatagramSocket();
socket.send(request);
} catch (SocketException e) {
} catch (IOException e) {
}
}
```

Receiving a Datagram

```
try {
byte[] inbuf = new byte[256]; // default size
DatagramSocket socket = new DatagramSocket();
// Wait for packet
DatagramPacket packet = new DatagramPacket(
inbuf, inbuf.length);
socket.receive(packet);
// Data is now in inbuf
int numBytesReceived = packet.getLength();
} catch (SocketException e) {
```

```
} catch (IOException e) {
}
```

Joining a Multicast Group

```
public void join(String groupName, int port) {
try {
MulticastSocket msocket = new MulticastSocket(port);
group = InetAddress.getByName(groupName);
msocket.joinGroup(group);
} catch (IOException e) {
}
}
```

Receiving from a Multicast Group

```
public void read(MulticastSocket msocket,
byte[] inbuf) {
try {
DatagramPacket packet = new DatagramPacket(
inbuf, inbuf.length);
// Wait for packet
msocket.receive(packet);
// Data is now in inbuf
int numBytesReceived = packet.getLength();
} catch (IOException e) {
}
```

}

Sending to a Multicast Group

```
byte[] outbuf = new byte[1024];
int port = 1234;
try {
DatagramSocket socket = new DatagramSocket();
InetAddress groupAddr = InetAddress.getByName(
"228.1.2.3");
DatagramPacket packet = new DatagramPacket(
outbuf, outbuf.length, groupAddr, port);
socket.send(packet);
} catch (SocketException e) {
} catch (IOException e) {
}
```

Defining and Exporting a Remote Object

```
1. Define the remote interface.
import java.rmi.*;
public interface RObject extends Remote {
void aMethod() throws RemoteException;
}
```

Looking Up a Remote Object and Invoking a Method

```
try {
// Look up remote object
```

```
RObject robj = (RObject) Naming.lookup(

"//localhost/RObjectServer");

// Invoke method on remote object

robj.aMethod();

} catch (MalformedURLException e) {

} catch (UnknownHostException e) {

} catch (NotBoundException e) {

} catch (RemoteException e) {

}
```

Passing Parameters to a Remote Method

Arguments to remote methods must be primitive, serializable, or Remote. This example demonstrates the declaration and use of all three parameter types.

1. Define the remote interface.

```
import java.rmi.*;

public interface RObject extends Remote {

// This parameter is primitive.

void primitiveArg(int num) throws RemoteException;

// This parameter implements Serializable.

void byValueArg(Integer num) throws RemoteException;

// This parameter implements Remote.

void byRefArg(ArgObject arg) throws RemoteException;

}

public interface ArgObject extends Remote {

int aMethod() throws RemoteException;
```

```
}
```

2. Define the remote object implementation.

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class RObjectImpl extends UnicastRemoteObject implements RObject {
public RObjectImpl() throws RemoteException {
super();
}
public void primitiveArg(int num) throws RemoteException {
}
public void byValueArg(Integer num) throws RemoteException {
}
public void byRefArg(ArgObject arg) throws RemoteException {
}
}
```

3. Compile the remote object implementation.

```
> javac RObject.java RObjectImpl.java
```

4. Generate the skeletons and stubs.

```
> rmic RObjectImpl
```

5. Create an instance of RObjectImpl and bind it to the RMI Registry.

```
try {
RObject robj = new RObjectImpl();
Naming.rebind("//localhost/RObjectServer", robj);
```

```
} catch (MalformedURLException e) {

} catch (UnknownHostException e) {

} catch (RemoteException e) {

}
```

6. Look Up the Remote object and pass the parameters.

```
try {

// Look up the remote object

RObject robj = (RObject) Naming.lookup("//localhost/RObjectServer");

// Pass a primitive value as argument

robj.primitiveArg(1998);

// Pass a serializable object as argument

robj.byValueArg(new Integer(9));

// Pass a Remote object as argument

robj.byRefArg(new ArgObjectImpl());

} catch (MalformedURLException e) {

} catch (UnknownHostException e) {

} catch (NotBoundException e) {

} catch (RemoteException e) {

}
```

Returning Values from a Remote Method

Return values from remote methods must be primitive, serializable, or Remote. This example demonstrates the declaration and use of all three return types. 1. Define the remote interface.

```
import java.rmi.*;
```

```
public interface RObject extends Remote {
// This return value is primitive.
int primitiveRet() throws RemoteException;
// This return value implements Serializable.
Integer byValueRet() throws RemoteException;
// This return value implements Remote.
ArgObject byRefRet() throws RemoteException;
}
public interface ArgObject extends Remote {
int aMethod() throws RemoteException;
}
```

2. Define the remote object implementation.

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class RObjectImpl extends UnicastRemoteObject
implementsRObject {
public RObjectImpl() throws RemoteException {
super();
}
public int primitiveRet() throws RemoteException {
return 3000;
}
public Integer byValueRet() throws RemoteException {
```

```
return new Integer(2000);

}

public ArgObject byRefRet() throws RemoteException {

return new ArgObjectImpl();

}

}
```

3. Compile the remote object implementation.

```
> javac RObject.java RObjectImpl.java
```

4. Generate the skeletons and stubs.

```
> rmic RObjectImpl
```

5. Create an instance of RObjectImpl and bind it to the RMI Registry.

```
try {

RObject robj = new RObjectImpl();

Naming.rebind("//localhost/RObjectServer", robj);

} catch (MalformedURLException e) {

} catch (UnknownHostException e) {

} catch (RemoteException e) {

}
```

6. Look Up the Remote object, invoke the methods, and receive the return values.

```
try {

// Look up the remote object

RObject robj = (RObject) Naming.lookup(

"//localhost/RObjectServer");
```

```
// Receive the primitive value as return value
int r1 = robj.primitiveRet();
// Receive the serializable object as return value
Integer r2 = robj.byValueRet();
// Receive the Remote Object as return value
ArgObject aobj =robj.byRefRet();
} catch (MalformedURLException e) {
} catch (UnknownHostException e) {
} catch (NotBoundException e) {
} catch (RemoteException e) {
}
```

Throwing an Exception from a Remote Method

1. Define the remote interface.

```
import java.rmi.*;
public interface RObject extends Remote {
void aMethod() throws RemoteException;
}
```

2. Define the remote object implementation.

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class RObjectImpl extends
UnicastRemoteObject implements RObject {
public RObjectImpl() throws RemoteException {
```

```
super();
}
public void aMethod() throws RemoteException {
// The actual exception must be wrapped in
// a RemoteException
throw new RemoteException(
"message", new FileNotFoundException("message"));
}
}
```

3. Compile the remote object implementation.

```
> javac RObject.java RObjectImpl.java
```

4. Generate the skeletons and stubs.

```
> rmic RObjectImpl
```

5. Create an instance of RObjectImpl and bind it to the RMI Registry.

```
try {
RObject robj = new RObjectImpl();
Naming.rebind("//localhost/RObjectServer", robj);
} catch (MalformedURLException e) {
} catch (UnknownHostException e) {
} catch (RemoteException e) {
}
```

6. Look up the Remote object, invoke the method, and catch the exception.

```
try {
```

```
// Look up the remote object.
RObject robj = (RObject) Naming.lookup(
"//localhost/RObjectServer");
// Invoke the method.
robj.aMethod();
} catch (MalformedURLException e) {
} catch (UnknownHostException e) {
} catch (NotBoundException e) {
} catch (RemoteException e) {
// Get the actual exception that was thrown.
Throwable realException = e.detail;
}
```

Strings (java.lang)

Constructing a String

If you are constructing a string with several appends, it may be more efficient to construct it using a StringBuffer and then convert it to an immutable String object.

```
StringBuffer buf = new StringBuffer("Initial Text");
// Modify
int index = 1;
buf.insert(index, "abc");
buf.append("def");
// Convert to string
String s = buf.toString();
```

Getting a Substring from a String

```
int start = 1;

int end = 4;

String substr = "aString".substring(start, end);// Str
```

Searching a String

```
String string = "aString";

// First occurrence.

int index = string.indexOf('S');// 1

// Last occurrence.

index = string.lastIndexOf('i');// 4

// Not found.

index = string.lastIndexOf('z');// -1
```

Replacing Characters in a String

```
// Replace all occurrences of 'a' with 'o'

String newString = string.replace('a', 'o');
```

Replacing Substrings in a String

```
static String replace(String str,

String pattern, String replace) {

int s = 0;

int e = 0;

StringBuffer result = new StringBuffer();

while ((e = str.indexOf(pattern, s)) >= 0) {

result.append(str.substring(s, e));
```

```
result.append(replace);

s = e+pattern.length();

}

result.append(str.substring(s));

return result.toString();

}
```

Converting a String to Upper or Lower Case

```
// Convert to upper case

String upper = string.toUpperCase();

// Convert to lower case

String lower = string.toLowerCase();
```

Converting a String to a Number

```
int i = Integer.parseInt("123");

long l = Long.parseLong("123");

float f = Float.parseFloat("123.4");

double d = Double.parseDouble("123.4e10");
```

Converting Unicode to UTF-8

```
try {

String string = "\u5639\u563b";

byte[] utf8 = string.getBytes("UTF8");

} catch (UnsupportedEncodingException e) {

}
```

Converting UTF-8 to Unicode

```
public static String toUnicode(byte[] utf8buf) {
try {
return new String(utf8buf, "UTF8");
} catch (UnsupportedEncodingException e) {
}
return null;
}
```

Determining a Character's Unicode Block

```
char ch = '\u5639';
Character.UnicodeBlock block =
Character.UnicodeBlock.of(ch);
```

Breaking a String into Words

```
String aString = "word1 word2 word3";
StringTokenizer parser =
new StringTokenizer(aString);
while (parser.hasMoreTokens()) {
processWord(parser.nextToken());
}
```