JDBC

OCP 11

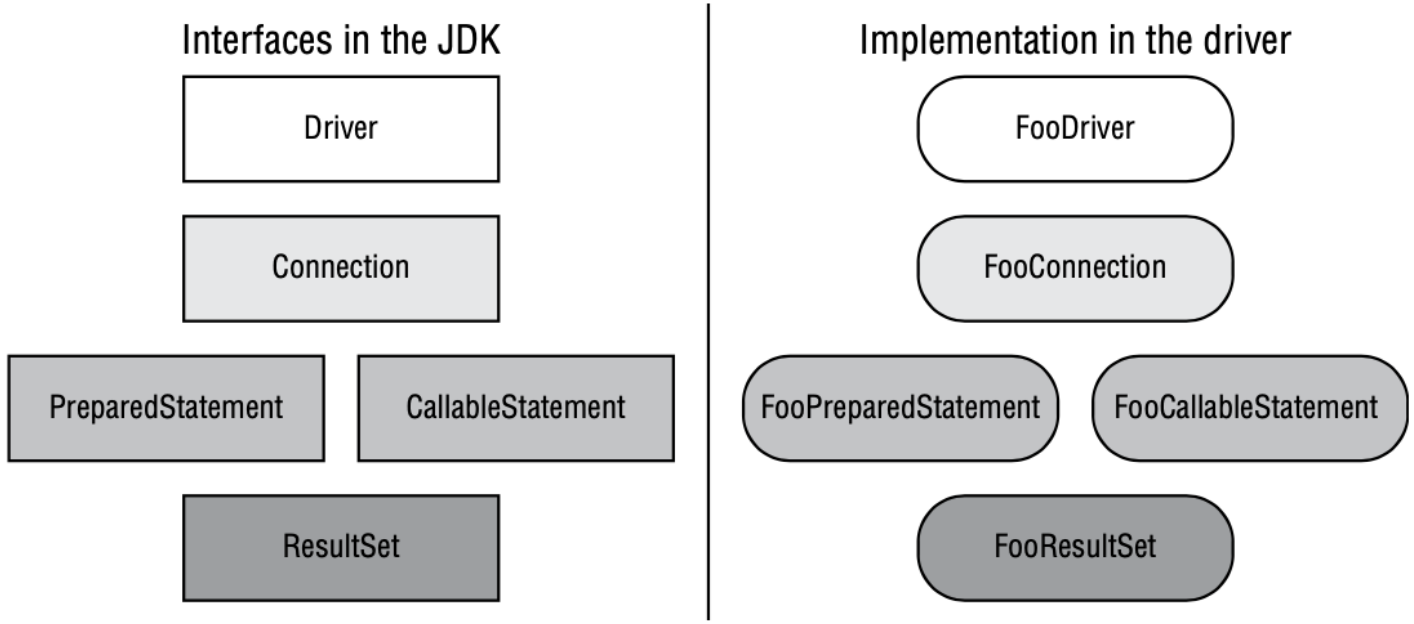
Alina Puscasu

For exam

- Connect to db with JDBC urls and DriverManager (not DataSource)
- PreparedStatement for CRUD operations
- PreparedStatement and CallableStatement for db operations
- No SQL related questions for exam

JDBC 5 interfaces – JDK (java.sql package)

FIGURE 21.2 Key JDBC interfaces



- `Driver`: Establishes a connection to the database
- `Connection`: Sends commands to a database
- `PreparedStatement`: Executes a SQL query
- `CallableStatement`: Executes commands stored in the database
- `ResultSet`: Reads results of a query

JDBC Interfaces

- No need to know the concrete classes from jdbc drivers
- Each db comes with a different jar that needs to be added to the classpath
- These implementations know how to communicate to the db
- For exam, you need yo know just these interfaces

When working with modules

UPDATE MODULE-INFO FILE WITH:
REQUIRES JAVA.SQL

JDBC CLASSES ARE IN THE JAVA.SQL MODULE, JAVA.SQL PACKAGE

JDBC example

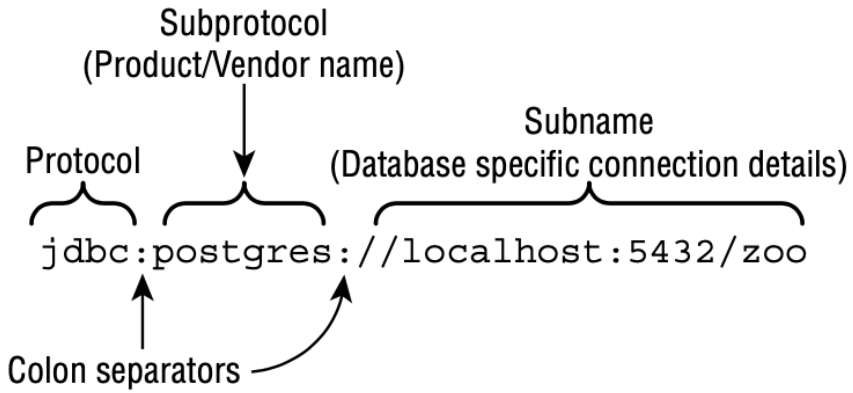
- Concrete classes from derby driver will be used, no need to know them

```
public class MyFirstDatabaseConnection {
   public static void main(String[] args) throws SQLException {
      String url = "jdbc:derby:zoo";
      try (Connection conn = DriverManager.getConnection(url);
         PreparedStatement ps = conn.prepareStatement(
            "SELECT name FROM animal");
         ResultSet rs = ps.executeQuery()) {
         while (rs.next())
            System.out.println(rs.getString(1));
      }
   }
}
```

Connecting to db

- Build JDBC Url

FIGURE 21.3 The JDBC URL format




```
jdbc:postgresql://localhost/zoo
jdbc:oracle:thin:@123.123.123.123:1521:zoo
jdbc:mysql://localhost:3306
jdbc:mysql://localhost:3306/zoo?profileSQL=true
```

Connecting to db

- Build JDBC Url
  - What is wrong with these?

```
jdbc:postgresql://local/zoo
jdbc:mysql://123456/zoo
jdbc;oracle;thin;/localhost/zoo
```

Connecting to db

- **Getting DB Connection**
  - 2 ways:

1. **DriverManager** - for exam but not recommended in real life

2. DataSource - not for exam but recommended in real life (more features, uses a connection pool – the same connection for operations, no need to know db passwords, just use an already set up data source by db team, just reference it) - info kept outside the app

- **getConnection(String url)** -> static method, factory pattern, you can get any impl of Connection, returns Connection

SQLException – checked exception

- For exam: to know when this is thrown, but no need to know the exception message
- At exam, if code snippets are not in a method, you should assume that checked exceptions are handled or declared

Getting DB Connection

```
java -cp "<path_to_derby>/derby.jar" TestConnect.java
```

```
import java.sql.*;
public class TestConnect {
   public static void main(String[] args) throws SQLException {
      Connection conn =
         DriverManager.getConnection("jdbc:derby:zoo");
      System.out.println(conn);
   }
}
```

```
org.apache.derby.impl.jdbc.EmbedConnection40@1372082959
(XID = 156), (SESSIONID = 1), (DATABASE = zoo), (DRDAID = null)
```

Getting DB Connection

```
import java.sql.*;
public class TestExternal {
   public static void main(String[] args) throws SQLException {
      Connection conn = DriverManager.getConnection(
         "jdbc:postgresql://localhost:5432/ocp-book",
         "username",
         "Password20182");
       System.out.println(conn);
    }
}
```

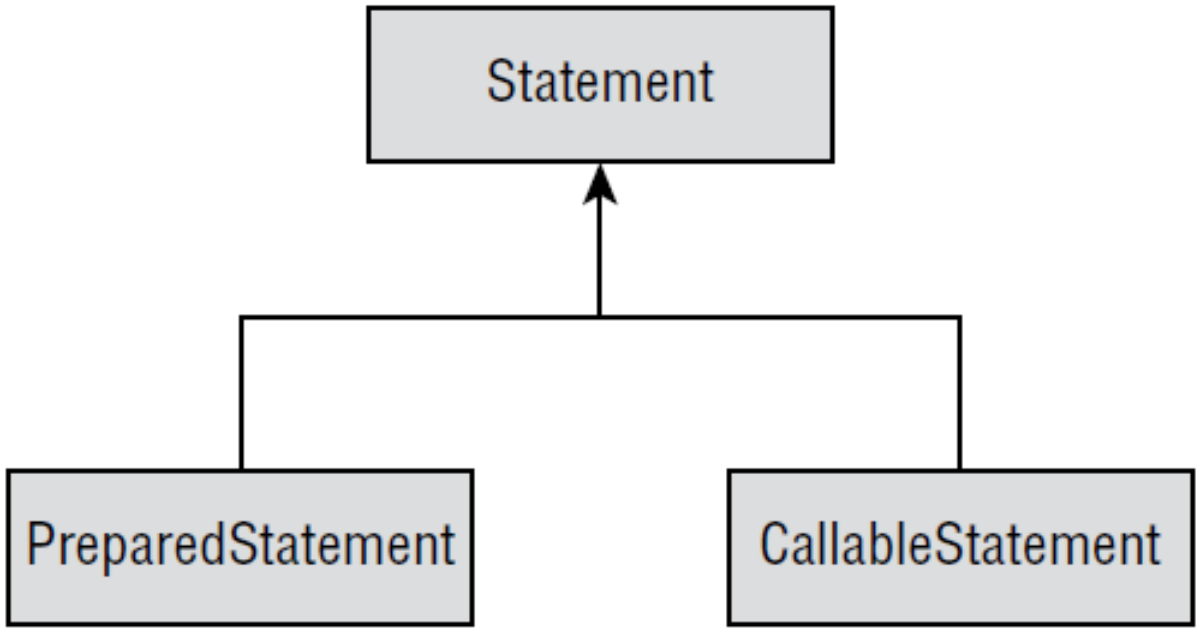
```
org.postgresql.jdbc4.Jdbc4Connection@eed1f14
```

Getting DB Connection

- DriverManager (Java class) looks in the JAR files to see if it can handle the JDBC url
  - If so,it creates Connection using that JAR
  - If not => SQLException

PreparedStatement

FIGURE 21.4 Types of statements



Statement vs PreparedStatement

- Statement does not take any param, it executes whatever query
- PreparedStatement recommended for:

- **Performance:** In most programs, you run similar queries multiple times. A `PreparedStatement` figures out a plan to run the SQL well and remembers it.
- **Security:** As you will see in Chapter 22, "Security," you are protected against an attack called SQL injection when using a `PreparedStatement` correctly.
- **Readability:** It's nice not to have to deal with string concatenation in building a query string with lots of parameters.
- **Future use:** Even if your query is being run only once or doesn't have any parameters, you should still use a `PreparedStatement`. That way future editors of the code won't add a variable and have to remember to change to `PreparedStatement` then.

Obtaining PreparedStatement

- This does not executes it, it just obtains it

```
try (PreparedStatement ps = conn.prepareStatement(
   "SELECT * FROM exhibits")) {
   // work with ps
}
```

- SQL string not passed -> compilation errors

```
try (var ps = conn.prepareStatement()) { // DOES NOT COMPILE
}
```

Executing PreparedStatement

- executeUpdate – with INSERT, UPDATE, DELETE

```
10: var insertSql = "INSERT INTO exhibits VALUES(10, 'Deer', 3)";
11: var updateSql = "UPDATE exhibits SET name = '' " +
12:    "WHERE name = 'None'";
13: var deleteSql = "DELETE FROM exhibits WHERE id = 10";
14:
15: try (var ps = conn.prepareStatement(insertSql)) {
16:    int result = ps.executeUpdate();
17:    System.out.println(result); // 1
18: }
19:
20: try (var ps = conn.prepareStatement(updateSql)) {
21:    int result = ps.executeUpdate();
22:    System.out.println(result); // 0
23: }
24:
25: try (var ps = conn.prepareStatement(deleteSql)) {
26:    int result = ps.executeUpdate();
27:    System.out.println(result); // 1
28: }
```

Executing PreparedStatement

- executeQuery – with SELECT

```
30: var sql = "SELECT * FROM exhibits";
31: try (var ps = conn.prepareStatement(sql);
32:    ResultSet rs = ps.executeQuery() ) {
33:
34:    // work with rs
35: }
```

Executing PreparedStatement

- execute – works with ALL CRUD operations
  - Returns boolean: true for SELECT, false for other operation

```
boolean isResultSet = ps.execute();
if (isResultSet) {
   try (ResultSet rs = ps.getResultSet()) {


       System.out.println("ran a query");
    }
 } else {
    int result = ps.getUpdateCount();
    System.out.println("ran an update");
 }
```

Wrong methods called for statements -> SQLExceptions

```
var sql = "SELECT * FROM names";
try (var conn = DriverManager.getConnection("jdbc:derby:zoo");
   var ps = conn.prepareStatement(sql)) {

   var result = ps.executeUpdate();
}
```

PreparedStatement methods for executing query

TABLE 21.2 SQL runnable by the execute method

Method	DELETE	INSERT	SELECT	UPDATE
`ps.execute()`	Yes	Yes	Yes	Yes
`ps.executeQuery()`	No	No	Yes	No
`ps.executeUpdate()`	Yes	Yes	No	Yes

PreparedStatement methods for executing query

TABLE 21.3 Return types of execute methods

Method	Return type	What is returned for SELECT	What is returned for DELETE/INSERT/UPDATE
`ps.execute()`	`boolean`	`true`	`false`
`ps.executeQuery()`	`ResultSet`	The rows and columns returned	n/a
`ps.executeUpdate()`	`int`	n/a	Number of rows added/ changed/removed

PreparedStatement and bind variables

```
public static void register(Connection conn) throws SQLException {
   var sql = "INSERT INTO names VALUES(6, 1, 'Edith')";

   try (var ps = conn.prepareStatement(sql)) {
      ps.executeUpdate();
   }
}
```

```
14: public static void register(Connection conn, int key,
15:     int type, String name) throws SQLException {
16:
17:    String sql = "INSERT INTO names VALUES(?, ?, ?)";
18:
19:    try (PreparedStatement ps = conn.prepareStatement(sql)) {
20:       ps.setInt(1, key);
21:       ps.setString(3, name);
22:       ps.setInt(2, type);
23:       ps.executeUpdate();
24:    }
25: }
```

PreparedStatement and bind variables

- Bind variables can be set using only indexes of column in any order
  - Only rule: to be set before executing the query
- They start counting from 1 not 0
  - JDBC counts from 1

PreparedStatement and bind variables

- What happens here ?

In both cases, SQLException with messages varying from driver to driver

```
var sql = "INSERT INTO names VALUES(?, ?)";
try (var ps = conn.prepareStatement(sql)) {
   ps.setInt(1, key);
   ps.setInt(2, type);
   ps.setString(3, name);
   ps.executeUpdate();
}
```

```
var sql = "INSERT INTO names VALUES(?, ?, ?)";
try (var ps = conn.prepareStatement(sql)) {
   ps.setInt(1, key);
   ps.setInt(2, type);
   // missing the set for parameter number 3
   ps.executeUpdate();
}
```

PreparedStatement and bind variables

TABLE 21.4 PreparedStatement methods

Method name	Parameter type	Example database type
setBoolean	Boolean	BOOLEAN
setDouble	Double	DOUBLE
setInt	Int	INTEGER
setLong	Long	BIGINT
setObject	Object	Any type
setString	String	CHAR, VARCHAR

setObject() works with any type. Autoboxing happens. Best to use the most specific type for setting.

```
String sql = "INSERT INTO names VALUES(?, ?, ?)";
try (PreparedStatement ps = conn.prepareStatement(sql)) {
   ps.setObject(1, key);
   ps.setObject(2, type);
   ps.setObject(3, name);
   ps.executeUpdate();
}
```

SQLException

Compile vs. Runtime Error When Executing

The following code is incorrect. Do you see why?

```
ps.setObject(1, key);
ps.setObject(2, type);
ps.setObject(3, name);
ps.executeUpdate(sql);  // INCORRECT
```

The problem is that the last line passes a SQL statement. With a `PreparedStatement`, we pass the SQL in when creating the object.

More interesting is that this does not result in a compiler error. Remember that `PreparedStatement` extends `Statement`. The `Statement` interface does accept a SQL statement at the time of execution, so the code compiles. Running this code gives `SQLException`. The text varies by database.

Updating multiple times – set only what is different

```
var sql = "INSERT INTO names VALUES(?, ?, ?)";

try (var ps = conn.prepareStatement(sql)) {

    ps.setInt(1, 20);
    ps.setInt(2, 1);
    ps.setString(3, "Ester");
    ps.executeUpdate();

    ps.setInt(1, 21);
    ps.setString(3, "Elias");
    ps.executeUpdate();
}
```

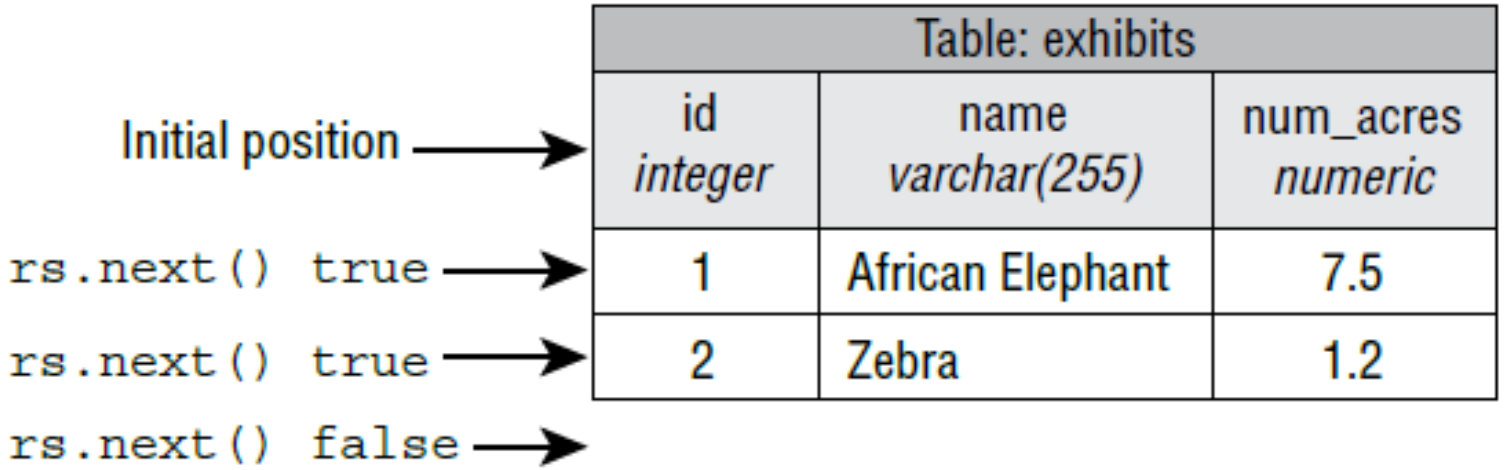
ResultSet

- On the exam, assume SQL and column names are correct if no other mention
- Columns counted from 1, not 0 – both in PreparedStatement and ResultSet
- You can access the column by name or index, but preffer column names

```
20: String sql = "SELECT id, name FROM exhibits";
21: Map<Integer, String> idToNameMap = new HashMap<>();
22:
23: try (var ps = conn.prepareStatement(sql);
24:    ResultSet rs = ps.executeQuery()) {
25:
26:    while (rs.next()) {
27:       int id = rs.getInt(1);
28:       String name = rs.getString(2);
29:       idToNameMap.put(id, name);
30:    }
31:    System.out.println(idToNameMap);
32: }
```

ResultSet cursor

FIGURE 21.5 The `ResultSet` cursor



ResultSet and SQLException

- If rs does not have a record (rs.next() is false) and you call get method -> SQLException;
- If you call rs.getInt("unexistentColumn"/ unexistentIndex)->SQLException
- If you want a single row from the result – use if instead of while

```
var sql = "SELECT count(*) FROM exhibits";

try (var ps = conn.prepareStatement(sql);
   var rs = ps.executeQuery()) {

   if (rs.next()) {
      int count = rs.getInt(1);
      System.out.println(count);
   }
}
```

Spot the issues!

```
var sql = "SELECT count(*) FROM exhibits";

try (var ps = conn.prepareStatement(sql);
   var rs = ps.executeQuery()) {

   rs.getInt(1); // SQLException
}
```

```
var sql = "SELECT * FROM exhibits where name='Not in table'";

try (var ps = conn.prepareStatement(sql);
   var rs = ps.executeQuery()) {

   rs.next();
   rs.getInt(1); // SQLException
}
```

Spot the issues!

```
var sql = "SELECT count(*) FROM exhibits";

try (var ps = conn.prepareStatement(sql);
   var rs = ps.executeQuery()) {

   if (rs.next())
      rs.getInt(0); // SQLException
}
```

```
var sql = "SELECT name FROM exhibits";

try (var ps = conn.prepareStatement(sql);
   var rs = ps.executeQuery()) {

   if (rs.next())
   rs.getInt("badColumn"); // SQLException
}
```

ResultSet – to remember :)

- Always use an `if` statement or `while` loop when calling `rs.next()`.
- Column indexes begin with 1.

ResultSet get methods

- No tricky questions at exam for get methods name (like using methods that does not exist)
  - No getChar method, there are: getFloat, getByte (but not for exam)

TABLE 21.5 ResultSet get methods

Method name	Return type
getBoolean	boolean
getDouble	double
getInt	int
getLong	long
getObject	Object
getString	String

ResultSet - getObject() returns any type; for primitive it uses the wrapper classes

```
16: var sql = "SELECT id, name FROM exhibits";
17: try (var ps = conn.prepareStatement(sql);
18:    var rs = ps.executeQuery()) {
19:
20:    while (rs.next()) {
21:       Object idField = rs.getObject("id");
22:       Object nameField = rs.getObject("name");
23:       if (idField instanceof Integer) {
24:          int id = (Integer) idField;
25:          System.out.println(id);
26:       }
27:       if (nameField instanceof String) {
28:          String name = (String) nameField;
29:          System.out.println(name);
30:       }
31:    }
32: }
```

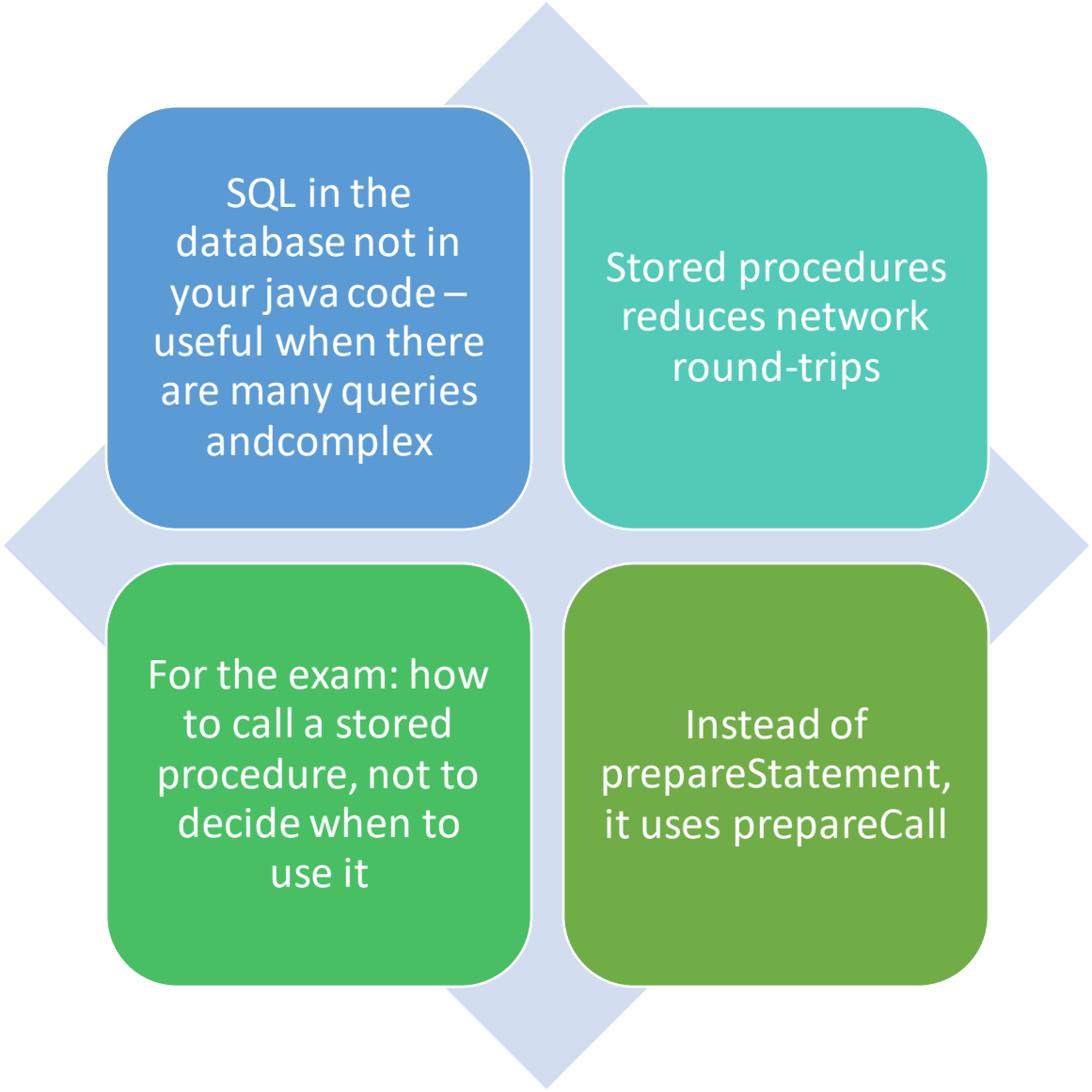
Lines 21 and 22 get the column as whatever type of `Object` is most appropriate. Lines 23–26 show you how to confirm that the type is `Integer` before casting and unboxing it

Bind variables

- Until now, PreparedStatement and ResultSet both in try with resources
- When bind variables are used, they need to be set in between so try with resources for both of them cannot be used
- -> solution is to nest try with resources for ResultSet in another try with res for PreparedStatement

```
30: var sql = "SELECT id FROM exhibits WHERE name = ?";
31:
32: try (var ps = conn.prepareStatement(sql)) {
33:    ps.setString(1, "Zebra");
34:
35:    try (var rs = ps.executeQuery()) {
36:       while (rs.next()) {
37:          int id = rs.getInt("id");
38:          System.out.println(id);
39:       }
40:    }
41: }
```

CallableStatement (use stored procedures when queries are complex)



Types of stored procedures

TABLE 21.6 Sample stored procedures

Name	Parameter name	Parameter type	Description
`read_e_names()`	n/a	n/a	Returns all rows in the names table that have a name beginning with an E
`read_names_by_letter()`	`prefix`	`IN`	Returns all rows in the names table that have a name beginning with the specified parameter
`magic_number()`	`Num`	`OUT`	Returns the number 42
`double_number()`	`Num`	`INOUT`	Multiplies the parameter by two and returns that number

Calling procedures without params

```
12: String sql = "{call read_e_names()}";
13: try (CallableStatement cs = conn.prepareCall(sql);
14:     ResultSet rs = cs.executeQuery()) {
15:
16:     while (rs.next()) {
17:         System.out.println(rs.getString(3));
18:     }
19: }
```

Calling procedures with IN param

```
25: var sql = "{call read_names_by_letter(?)}";
26: try (var cs = conn.prepareCall(sql)) {
27:     cs.setString("prefix", "Z");
28:
29:     try (var rs = cs.executeQuery()) {
30:         while (rs.next()) {
31:             System.out.println(rs.getString(3));
32:         }
33:     }
34: }
```

- For **PreparedStatement**, bind variables are given values with **setString(index, "value") only**, for **CallableStatement we can also use column name:**

```
cs.setString(1, "Z");
cs.setString("prefix", "Z");
```

Calling procedures with OUT param

- **?=** specifies an output is expected but is **optional since we register the OUT param**
- It does not return a ResultSet but a value, so we call **execute()** at 43, not executeQuery
- **registerOutParameter**() needs to be called for each OUT or INOUT param

```
40: var sql = "{?= call magic_number(?) }";
41: try (var cs = conn.prepareCall(sql)) {
42:     cs.registerOutParameter(1, Types.INTEGER);
43:     cs.execute();
44:     System.out.println(cs.getInt("num"));
45: }
```

To complicate things a little :)

Database-Specific Behavior

Some databases are lenient about certain things this chapter says are required. For example, some databases allow you to omit the following:

- Braces ({})
- Bind variable (?) if it is an OUT parameter
- Call to `registerOutParameter()`

For the exam, you need to answer according to the full requirements, which are described in this book. For example, you should answer exam questions as if braces are required.

Calling procedures with INOUT param

Same param for both input and output

```
50: var sql = "{call double_number(?)}";
51: try (var cs = conn.prepareCall(sql)) {
52:     cs.setInt(1, 8);
53:     cs.registerOutParameter(1, Types.INTEGER);
54:     cs.execute();
55:     System.out.println(cs.getInt("num"));
56: }
```

Stored procedures param types

TABLE 21.7 Stored procedure parameter types

	IN	OUT	INOUT
Used for input	Yes	No	Yes
Used for output	No	Yes	Yes
Must set parameter value	Yes	No	Yes
Must call `registerOutParameter()`	No	Yes	Yes
Can include ?=	No	Yes	Yes

Closing resources when working with JDBC

Otherwise -> resource leak

Close: Connection, PreparedStatement, CallableStatement, ResultSet

Order of closing is important -> avoid exceptions and leaks

Not all resources need to be closed

- Connection closed will close the rest of them, since it created them
- PreparedStatement/CallableStatement will also close the ResultSet
- If you want to close all of them, close first ResultSet, then Statement and in the end Connection

Closing resources when working with JDBC

Do not create resources before try with resources, exceptions can be thrown before try and resources will not be closed:

```
43:     var conn = DriverManager.getConnection(url);
44:     var ps = conn.prepareStatement(sql);
45:     var rs = ps.executeQuery();
46:
47:     try (conn; ps; rs) {
48:         while (rs.next())
49:             System.out.println(rs.getString(1));
50:     }
51: }
```

Closing ResultSet

JDBC automatically closes ResultSet when another query is executed from the same Statement (either PreparedStatement/CallableStatement) - line 28 closes rs1

How many resources closed here?

```
14: var url = "jdbc:derby:zoo";
15: var sql = "SELECT count(*) FROM names where id = ?";
16: try (var conn = DriverManager.getConnection(url);
17:    var ps = conn.prepareStatement(sql)) {
18:
19:    ps.setInt(1, 1);
20:
21:    var rs1 = ps.executeQuery();
22:    while (rs1.next()) {
23:       System.out.println("Count: " + rs1.getInt(1));
24:    }
25:
26:    ps.setInt(1, 2);
27:
28:    var rs2 = ps.executeQuery();
29:    while (rs2.next()) {
30:       System.out.println("Count: " + rs2.getInt(1));
31:    }
32:    rs2.close();
33: }
```

Catch SQLException

Any JDBC method can throw checked SQL exception

```
var sql = "SELECT not_a_column FROM names";
var url = "jdbc:derby:zoo";
try (var conn = DriverManager.getConnection(url);
   var ps = conn.prepareStatement(sql);
   var rs = ps.executeQuery()) {

   while (rs.next())
      System.out.println(rs.getString(1));
} catch (SQLException e) {
   System.out.println(e.getMessage());
   System.out.println(e.getSQLState());
   System.out.println(e.getErrorCode());
}
```

```
Column 'NOT_A_COLUMN' is either not in any table ...
42X04
30000
```

Each of these methods gives you a different piece of information. The `getMessage()` method returns a human-readable message as to what went wrong. We've only included the beginning of it here. The `getSQLState()` method returns a code as to what went wrong. You can Google the name of your database and the SQL state to get more information about the error. By comparison, `getErrorCode()` is a database-specific code. On this database, it doesn't do anything.

Review

1. Which interfaces or classes are in a database-specific JAR file? (Choose all that apply.)
   - **A.** `Driver`
   - **B.** `Driver`'s implementation
   - **C.** `DriverManager`
   - **D.** `DriverManager`'s implementation
   - **E.** `PreparedStatement`
   - **F.** `PreparedStatement`'s implementation

Review

1. Which interfaces or classes are in a database-specific JAR file? (Choose all that apply.)
   - **A.** `Driver`
   - **B.** `Driver`'s implementation
   - **C.** `DriverManager`
   - **D.** `DriverManager`'s implementation
   - **E.** `PreparedStatement`
   - **F.** `PreparedStatement`'s implementation

Driver, DriverManager and PreparedStatement interfaces are in the JDK; Concrete class of DriverManager also in the JDK

Review

2. Which are required parts of a JDBC URL? (Choose all that apply.)
   - **A.** Connection parameters
   - **B.** IP address of database
   - **C.** `jdbc`
   - **D.** Password
   - **E.** Port
   - **F.** Vendor-specific string

Review

2. Which are required parts of a JDBC URL? (Choose all that apply.)
   - **A.** Connection parameters
   - **B.** IP address of database
   - **C.** `jdbc`
   - **D.** Password
   - **E.** Port
   - **F.** Vendor-specific string

3 parts: **jdbc**, **subprotocol** (vendor/product name), **subname** (vendor specific string)

Review

3. Which of the following is a valid JDBC URL?

   A. `jdbc:sybase:localhost:1234/db`

   B. `jdbc::sybase::localhost::/db`

   C. `jdbc::sybase:localhost::1234/db`

   D. `sybase:localhost:1234/db`

   E. `sybase::localhost::/db`

   F. `sybase::localhost::1234/db`

Review

3. Which of the following is a valid JDBC URL?

   A. `jdbc:sybase:localhost:1234/db`

   B. `jdbc::sybase::localhost::/db`

   C. `jdbc::sybase:localhost::1234/db`

   D. `sybase:localhost:1234/db`

   E. `sybase::localhost::/db`

   F. `sybase::localhost::1234/db`

Single colons to sepparate; jdbc is required;

Review

A.
```
ps.setString(0, "snow");
```
B.
```
ps.setString(1, "snow");
```
C.
```
ps.setString("environment", "snow");
```
D.
```
ps.setString(1, "snow"); ps.setString(1, "snow");
```
E.
```
ps.setString(1, "snow"); ps.setString(2, "snow");
```
F.
```
ps.setString("environment", "snow");
ps.setString("environment", "snow");
```

4. Which of the options can fill in the blank to make the code compile and run without error? (Choose all that apply.)

```
var sql = "UPDATE habitat WHERE environment = ?";
try (var ps = conn.prepareStatement(sql)) {

   _____

   ps.executeUpdate();
}
```

Review

4. Which of the options can fill in the blank to make the code compile and run without error? (Choose all that apply.)

```
var sql = "UPDATE habitat WHERE environment = ?";
try (var ps = conn.prepareStatement(sql)) {

   __________

   ps.executeUpdate();
}
```

A. `ps.setString(0, "snow");`
B. `ps.setString(1, "snow");`
C. `ps.setString("environment", "snow");`
D. `ps.setString(1, "snow"); ps.setString(1, "snow");`
E. `ps.setString(1, "snow"); ps.setString(2, "snow");`
F.
```
ps.setString("environment", "snow");
ps.setString("environment", "snow");
```

Setting param on PreparedStatements can be done only using indexes for columns, not the names of the column; Indexes start with 1 not 0; You can overwrite setting params;

Review

5. Suppose that you have a table named `animal` with two rows. What is the result of the following code?

```
 6:  var conn = new Connection(url, userName, password);
 7:  var ps = conn.prepareStatement(
 8:     "SELECT  count(*) FROM animal");
 9:  var rs = ps.executeQuery();
10: if (rs.next()) System.out.println(rs.getInt(1));
```

A. `0`

B. `2`

C. There is a compiler error on line 6.

D. There is a compiler error on line 10.

E. There is a compiler error on another line.

F. A runtime exception is thrown.

Review

. Suppose that you have a table named `animal` with two rows. What is the result of the following code?

```
 6:  var conn = new Connection(url, userName, password);
 7:  var ps = conn.prepareStatement(
 8:     "SELECT  count(*) FROM animal");
 9:  var rs = ps.executeQuery();
10: if (rs.next()) System.out.println(rs.getInt(1));
```

A. `0`
B. `2`
C. There is a compiler error on line 6.
D. There is a compiler error on line 10.
E. There is a compiler error on another line.
F. A runtime exception is thrown.

Connection does not have a constructor – created with static method from DriverManager; if the Connection were created correctly, the answer would be '2'

Review

6. Which of the options can fill in the blanks in order to make the code compile?

```
boolean bool = ps._________();
int num = ps._____________();
ResultSet rs = ps._________();
```

A. `execute, executeQuery, executeUpdate`

B. `execute, executeUpdate, executeQuery`

C. `executeQuery, execute, executeUpdate`

D. `executeQuery, executeUpdate, execute`

E. `executeUpdate, execute, executeQuery`

F. `executeUpdate, executeQuery, execute`

Review

6. Which of the options can fill in the blanks in order to make the code compile?

```
boolean bool = ps._____();
int num = ps._____();
ResultSet rs = ps._____();
```

A. `execute, executeQuery, executeUpdate`
B. `execute, executeUpdate, executeQuery`
C. `executeQuery, execute, executeUpdate`
D. `executeQuery, executeUpdate, execute`
E. `executeUpdate, execute, executeQuery`
F. `executeUpdate, executeQuery, execute`

Review

8. Suppose that the table `animal` has five rows and the following SQL statement updates all of them. What is the result of this code?

```
public static void main(String[] args) throws SQLException {
    var sql = "UPDATE names SET name = 'Animal'";
    try (var conn = DriverManager.getConnection("jdbc:derby:zoo");
       var ps = conn.prepareStatement(sql)) {

       var result = ps.executeUpdate();
       System.out.println(result);
    }
 }
```

A. `0`

B. `1`

C. `5`

D. The code does not compile.

E. A `SQLException` is thrown.

F. A different exception is thrown.

Review

8. Suppose that the table `animal` has five rows and the following SQL statement updates all of them. What is the result of this code?

```
public static void main(String[] args) throws SQLException {
    var sql = "UPDATE names SET name = 'Animal'";
    try (var conn = DriverManager.getConnection("jdbc:derby:zoo");
       var ps = conn.prepareStatement(sql)) {

       var result = ps.executeUpdate();
       System.out.println(result);
    }
 }
```

A. `0`

B. `1`

C. `5`

D. The code does not compile.

E. A `SQLException` is thrown.

F. A different exception is thrown.

Review

A. Line 18 is missing braces.
B. Line 18 is missing a ?.
C. Line 19 is not allowed to use `var`.
D. Line 20 does not compile.
E. Line 22 does not compile.
F. Something else is wrong with the code.
G. None of the above. This code is correct.

9. Suppose `learn()` is a stored procedure that takes one `IN` parameter. What is wrong with the following code? (Choose all that apply.)

```
18: var sql = "call learn()";
19: try (var cs = conn.prepareCall(sql)) {
20:    cs.setString(1, "java");
21:    try (var rs = cs.executeQuery()) {
22:       while (rs.next()) {
23:          System.out.println(rs.getString(3));
24:       }
25:    }
26: }
```

Review

9. Suppose `learn()` is a stored procedure that takes one `IN` parameter. What is wrong with the following code? (Choose all that apply.)

```
18: var sql = "call learn()";
19: try (var cs = conn.prepareCall(sql)) {
20:    cs.setString(1, "java");
21:    try (var rs = cs.executeQuery()) {
22:       while (rs.next()) {
23:          System.out.println(rs.getString(3));
24:       }
25:    }
26: }
```

A. Line 18 is missing braces.

B. Line 18 is missing a ?.

C. Line 19 is not allowed to use `var`.

D. Line 20 does not compile.

E. Line 22 does not compile.

F. Something else is wrong with the code.

G. None of the above. This code is correct.

Review

10. Suppose that the table `enrichment` has three rows with the animals `bat`, `rat`, and `snake`. How many lines does this code print?

```
var sql = "SELECT toy FROM enrichment WHERE animal = ?";
try (var ps = conn.prepareStatement(sql)) {
   ps.setString(1, "bat");

   try (var rs = ps.executeQuery(sql)) {
      while (rs.next())
         System.out.println(rs.getString(1));
   }
}
```

A. 0

B. 1

C. 3

D. The code does not compile.

E. A `SQLException` is thrown.

F. A different exception is thrown.

Review

10. Suppose that the table `enrichment` has three rows with the animals `bat`, `rat`, and `snake`. How many lines does this code print?

```
var sql = "SELECT toy FROM enrichment WHERE animal = ?";
try (var ps = conn.prepareStatement(sql)) {
   ps.setString(1, "bat");

   try (var rs = ps.executeQuery(sql)) {
      while (rs.next())
         System.out.println(rs.getString(1));
   }
}
```

A. 0
B. 1
C. 3
D. The code does not compile.
E. A `SQLException` is thrown.
F. A different exception is thrown.

Since we are using executeQuery(sql) -> it uses it from Statement (parent of PreparedSTatement) and at compile time is ok, but since we are using prepareStatement with bind variable (Statement cannot have bind variab) at runtime it will not recognize this method from PreparedStatement

Review

11. Suppose that the table `food` has five rows and this SQL statement updates all of them. What is the result of this code?

```
public static void main(String[] args) {
   var sql = "UPDATE food SET amount = amount + 1";
   try (var conn = DriverManager.getConnection("jdbc:derby:zoo");
      var ps = conn.prepareStatement(sql)) {

      var result = ps.executeUpdate();
      System.out.println(result);
   }
}
```

A. `0`

B. `1`

C. `5`

D. The code does not compile.

E. A `SQLException` is thrown.

F. A different exception is thrown.

Review

11. Suppose that the table `food` has five rows and this SQL statement updates all of them. What is the result of this code?

```
public static void main(String[] args) {
   var sql = "UPDATE food SET amount = amount + 1";
   try (var conn = DriverManager.getConnection("jdbc:derby:zoo");
      var ps = conn.prepareStatement(sql)) {

      var result = ps.executeUpdate();
      System.out.println(result);
   }
}
```

A. `0`

B. `1`

C. `5`

D. The code does not compile.

E. A `SQLException` is thrown.

F. A different exception is thrown.

We are in a main method where SQLException is not declared or handled (all JDBC methods can throw SQLException so this should be declared/handled). Therefore, code does not compile

Review

12. Suppose we have a JDBC program that calls a stored procedure, which returns a set of results. Which is the correct order in which to close database resources for this call?

A. `Connection, ResultSet, CallableStatement`

B. `Connection, CallableStatement, ResultSet`

C. `ResultSet, Connection, CallableStatement`

D. `ResultSet, CallableStatement, Connection`

E. `CallableStatement, Connection, ResultSet`

F. `CallableStatement, ResultSet, Connection`

Review

12. Suppose we have a JDBC program that calls a stored procedure, which returns a set of results. Which is the correct order in which to close database resources for this call?

A. `Connection, ResultSet, CallableStatement`

B. `Connection, CallableStatement, ResultSet`

C. `ResultSet, Connection, CallableStatement`

D. `ResultSet, CallableStatement, Connection`

E. `CallableStatement, Connection, ResultSet`

F. `CallableStatement, ResultSet, Connection`

Review

13. Suppose that the table `counts` has five rows with the numbers 1 to 5. How many lines does this code print?

```
var sql = "SELECT num FROM counts WHERE num > ?";
try (var ps = conn.prepareStatement(sql)) {
   ps.setInt(1, 3);

   try (var rs = ps.executeQuery()) {
      while (rs.next())
         System.out.println(rs.getObject(1));
   }

   ps.setInt(1, 100);

   try (var rs = ps.executeQuery()) {
      while (rs.next())
         System.out.println(rs.getObject(1));
   }
}
```

A. 0
B. 1
C. 2
D. 4
E. The code does not compile.
F. The code throws an exception.

Review

13. Suppose that the table `counts` has five rows with the numbers 1 to 5. How many lines does this code print?

```
var sql = "SELECT num FROM counts WHERE num > ?";
try (var ps = conn.prepareStatement(sql)) {
   ps.setInt(1, 3);

   try (var rs = ps.executeQuery()) {
      while (rs.next())
         System.out.println(rs.getObject(1));
   }

   ps.setInt(1, 100);

   try (var rs = ps.executeQuery()) {
      while (rs.next())
         System.out.println(rs.getObject(1));
   }
}
```

A. 0
B. 1
C. 2
D. 4
E. The code does not compile.
F. The code throws an exception.

Review

14. Which of the following can fill in the blank correctly? (Choose all that apply.)

```
var rs = ps.executeQuery();
if (rs.next()) {
   _____________________________;
}
```

A. `String s = rs.getString(0)`
B. `String s = rs.getString(1)`
C. `String s = rs.getObject(0)`
D. `String s = rs.getObject(1)`
E. `Object s = rs.getObject(0)`
F. `Object s = rs.getObject(1)`

Review

14. Which of the following can fill in the blank correctly? (Choose all that apply.)

```
var rs = ps.executeQuery();
if (rs.next()) {
    _____________________________;
}
```

A. `String s = rs.getString(0)`
B. `String s = rs.getString(1)`
C. `String s = rs.getObject(0)`
D. `String s = rs.getObject(1)`
E. `Object s = rs.getObject(0)`
F. `Object s = rs.getObject(1)`

An Object cannot be assigned to a String without a cast

Review

15. Suppose `learn()` is a stored procedure that takes one `IN` parameter and one `OUT` parameter. What is wrong with the following code? (Choose all that apply.)

```
18: var sql = "{?= call learn(?)}";
19: try (var cs = conn.prepareCall(sql)) {
20:    cs.setInt(1, 8);
21:    cs.execute();
22:    System.out.println(cs.getInt(1));
23: }
```

A. Line 18 does not call the stored procedure properly.
B. The parameter value is not set for input.
C. The parameter is not registered for output.
D. The code does not compile.
E. Something else is wrong with the code.
F. None of the above. This code is correct.

Review

15. Suppose `learn()` is a stored procedure that takes one `IN` parameter and one `OUT` parameter. What is wrong with the following code? (Choose all that apply.)

```
18: var sql = "{?= call learn(?)}";
19: try (var cs = conn.prepareCall(sql)) {
20:    cs.setInt(1, 8);
21:    cs.execute();
22:    System.out.println(cs.getInt(1));
23: }
```

A. Line 18 does not call the stored procedure properly.

B. The parameter value is not set for input.

C. The parameter is not registered for output.

D. The code does not compile.

E. Something else is wrong with the code.

F. None of the above. This code is correct.

Review

16. Which of the following can fill in the blank? (Choose all that apply.)

```
var sql = "_____________________";
try (var ps = conn.prepareStatement(sql)) {
   ps.setObject(3, "red");
   ps.setInt(2, 8);
   ps.setString(1, "ball");
   ps.executeUpdate();
}
```

A. `{ call insert_toys(?, ?) }`

B. `{ call insert_toys(?, ?, ?) }`

C. `{ call insert_toys(?, ?, ?, ?) }`

D. `INSERT INTO toys VALUES (?, ?)`

E. `INSERT INTO toys VALUES (?, ?, ?)`

F. `INSERT INTO toys VALUES (?, ?, ?, ?)`

Review

16. Which of the following can fill in the blank? (Choose all that apply.)

```
var sql = "_____________________";
try (var ps = conn.prepareStatement(sql)) {
   ps.setObject(3, "red");
   ps.setInt(2, 8);
   ps.setString(1, "ball");
   ps.executeUpdate();
}
```

A. `{ call insert_toys(?, ?) }`

B. `{ call insert_toys(?, ?, ?) }`

C. `{ call insert_toys(?, ?, ?, ?) }`

D. `INSERT INTO toys VALUES (?, ?)`

E. `INSERT INTO toys VALUES (?, ?, ?)`

F. `INSERT INTO toys VALUES (?, ?, ?, ?)`

Review

17. Suppose that the table `counts` has five rows with the numbers 1 to 5. How many lines does this code print?

```
var sql = "SELECT num FROM counts WHERE num > ?";
try (var ps = conn.prepareStatement(sql)) {
   ps.setInt(1, 3);

   try (var rs = ps.executeQuery()) {
      while (rs.next())
         System.out.println(rs.getObject(1));
   }

   try (var rs = ps.executeQuery()) {
      while (rs.next())
         System.out.println(rs.getObject(1));
   }
}
```

A. 0
B. 1
C. 2
D. 4
E. The code does not compile.
F. The code throws an exception.

Review

17. Suppose that the table `counts` has five rows with the numbers 1 to 5. How many lines does this code print?

```
var sql = "SELECT num FROM counts WHERE num > ?";
try (var ps = conn.prepareStatement(sql)) {
   ps.setInt(1, 3);

   try (var rs = ps.executeQuery()) {
      while (rs.next())
         System.out.println(rs.getObject(1));
   }

   try (var rs = ps.executeQuery()) {
      while (rs.next())
         System.out.println(rs.getObject(1));
   }
}
```

A. 0
B. 1
C. 2
D. 4
E. The code does not compile.
F. The code throws an exception.

Review

A. `100`

B. `101`

C. The code does not compile.

D. A `SQLException` is thrown.

E. A different exception is thrown.

18. There are currently 100 rows in the table `species` before inserting a new row. What is the output of the following code?

```
String insert = "INSERT INTO species VALUES (3, 'Ant', .05)";
String select = "SELECT count(*) FROM species";
try (var ps = conn.prepareStatement(insert)) {
   ps.executeUpdate();
}
try (var ps = conn.prepareStatement(select)) {
   var rs = ps.executeQuery();
   System.out.println(rs.getInt(1));
}
```

Review

18. There are currently 100 rows in the table `species` before inserting a new row. What is the output of the following code?

```
String insert = "INSERT INTO species VALUES (3, 'Ant', .05)";
String select = "SELECT count(*) FROM species";
try (var ps = conn.prepareStatement(insert)) {
   ps.executeUpdate();
}
try (var ps = conn.prepareStatement(select)) {
   var rs = ps.executeQuery();
   System.out.println(rs.getInt(1));
}
```

A. `100`

B. `101`

C. The code does not compile.

D. A `SQLException` is thrown.

E. A different exception is thrown.

Cursor is not positioned, rs.next() missing

Review

19. Which of the options can fill in the blank to make the code compile and run without error? (Choose all that apply.)

```
var sql = "UPDATE habitat WHERE environment = ?";
try (var ps = conn.prepareCall(sql)) {

   _____

   ps.executeUpdate();
}
```

A. `ps.setString(0, "snow");`

B. `ps.setString(1, "snow");`

C. `ps.setString("environment", "snow");`

D. The code does not compile.

E. The code throws an exception at runtime.

Review

19. Which of the options can fill in the blank to make the code compile and run without error? (Choose all that apply.)

```
var sql = "UPDATE habitat WHERE environment = ?";
try (var ps = conn.prepareCall(sql)) {

   _____

   ps.executeUpdate();
}
```

A. `ps.setString(0, "snow");`

B. `ps.setString(1, "snow");`

C. `ps.setString("environment", "snow");`

D. The code does not compile.

E. The code throws an exception at runtime.

Should use prepareStatement. How it is, it creates CallableStatement, complaining then at runtime because trying to execute SQL as if it were a stored procedure

Review

20. Which of the following could be true of the following code? (Choose all that apply.)

```
var sql = "{call transform(?)}";
try (var cs = conn.prepareCall(sql)) {
   cs.registerOutParameter(1, Types.INTEGER);
   cs.execute();
   System.out.println(cs.getInt(1));
}
```

A. The stored procedure can declare an `IN` or `INOUT` parameter.

B. The stored procedure can declare an `INOUT` or `OUT` parameter.

C. The stored procedure must declare an `IN` parameter.

D. The stored procedure must declare an `INOUT` parameter.

E. The stored procedure must declare an `OUT` parameter.

Review

20. Which of the following could be true of the following code? (Choose all that apply.)

```
var sql = "{call transform(?)}";
try (var cs = conn.prepareCall(sql)) {
   cs.registerOutParameter(1, Types.INTEGER);
   cs.execute();
   System.out.println(cs.getInt(1));
}
```

A. The stored procedure can declare an `IN` or `INOUT` parameter.
B. The stored procedure can declare an `INOUT` or `OUT` parameter.
C. The stored procedure must declare an `IN` parameter.
D. The stored procedure must declare an `INOUT` parameter.
E. The stored procedure must declare an `OUT` parameter.

It calls registerOutParameter and no set -> no IN/INOUT param.

Review

21. Which is the first line containing a compiler error?

```
25: String url = "jdbc:derby:zoo";
26: try (var conn = DriverManager.getConnection(url);
27:    var ps = conn.prepareStatement();
28:    var rs = ps.executeQuery("SELECT * FROM swings")) {
29:    while (rs.next()) {
30:       System.out.println(rs.getInteger(1));
31:    }
32: }
```

A. Line 26
B. Line 27
C. Line 28
D. Line 29
E. Line 30
F. None of the above

Review

21. Which is the first line containing a compiler error?

```
25: String url = "jdbc:derby:zoo";
26: try (var conn = DriverManager.getConnection(url);
27:    var ps = conn.prepareStatement();
28:    var rs = ps.executeQuery("SELECT * FROM swings")) {
29:    while (rs.next()) {
30:       System.out.println(rs.getInteger(1));
31:    }
32: }
```

A. Line 26
B. Line 27
C. Line 28
D. Line 29
E. Line 30
F. None of the above

prepareStatement() needs SQL be passed in; also line 30 getInteger is wrong, we have getInt. First line with error is 27 though