Prepared Statements

The Prepared Statement

- The java.sql.PreparedStatement interface extends java.sql.Statement.
- Objects of type java.sql.PreparedStatement represent SQL statements that are partially compiled with binded parameters that should be replaced with specific values before the statement is executed.
- Thanks for being precompiled, using prepared statements is usually faster than using simple ones.

The Binded Parameters

- When creating a prepared statement object, we don't need to specify all values the statement uses. Instead, the statement can use a question mark (?) as a placeholder for each missing value.
- Before executing the statement we must set a value for each question mark (?).
- The questions marks are numbered left to right starting with 1.

Code Sample

- Getting a PreparedStatement object:

```
PreparedStatement preparedStatement =
        connection.prepareStatement(“UPDATE customers SET fee=? WHERE type=?”);
```

By calling the prepareStatement() method we can get a PreparedStatement object. Instead of each unknown value we write a question mark (?). As a result for creating this PreparedStatement object, the SQL statement we sent to the preparedStatement method is sent to the Database and partially compiled.

Code Sample

- Setting the values:

```
preparedStatement.setDouble(1, 2.4);
preparedStatement.setInt(2, 4);
```

It is a MUST to set the values instead of each one of the questions marks before executing the statement. The PreparedStatement interface includes various methods that allow setting values instead of each one of the question marks. Various setXxx() methods allow setting various types of values (setInt, setDouble, setString, setLong etc..). These methods get two parameters. The first is the index number of the question mark we want to replace with a value and the second is the value itself. Once a specific question mark was replaced with a value, each concurrent execution of the prepared statement will use that value.

Code Sample

- Executing the statement:

  preparedStatement.execute();

  Executing a prepared statement is possible via three different methods:

  boolean execute()

  The SQL statement is simply executed.

  ResultSet executeQuery()

  This version is used for quering the database. It returns a ResultSet object. You can get a scrollable and updatable ResultSet object by calling:

  PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)

  int executeUpdate()

  This version is used to execute SQL statement of type INSERT, UPDATE, DELETE or a simple SQL statement.

Java Types & JDBC Types Mapping

- When calling any of the setXXX methods in order to switch a question mark with a value it is our responsibility to choose the setXXX method that is compatible with the type expected by the database.
- Many databases know how to handle a case in which we weren't accurate in our types mapping (e.g. We called setInt instead of setLong) and convert the int value they receive to long.
- Be accurate in our types mapping will increase our code portability.

Java Types & JDBC Types Mapping

- The following partial table describes how the Java types and the JDBC types are mapped with each other.

String	CHAR, VARCHAR or LONGVARCHAR
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	BINARY, VARBINARY or LONGBINARY
java.sql.Date	DATE

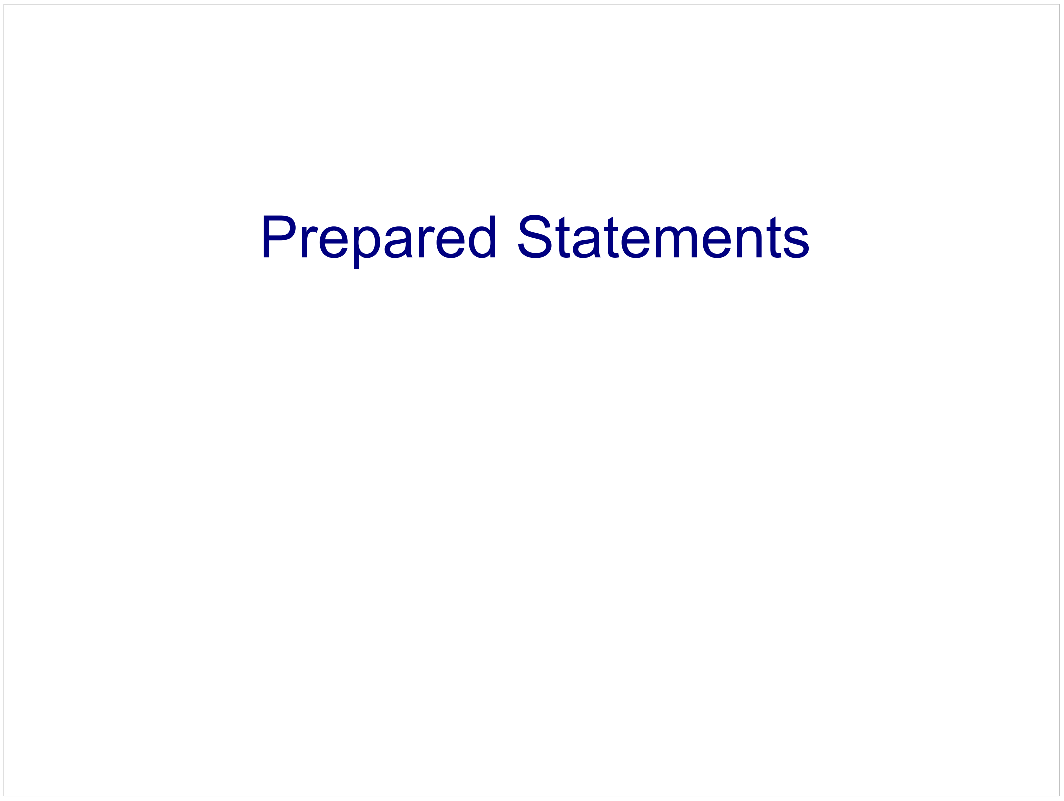
Batch Update

- Using the PreparedStatement object it is possible to create a batch of statements.
- The addBatch() method should be called on the prepared statement object we use each time we replace the question marks with a new set of values.
- Executing the batch of statements represented by the prepared statement object will be done by calling the executeBatch() method.

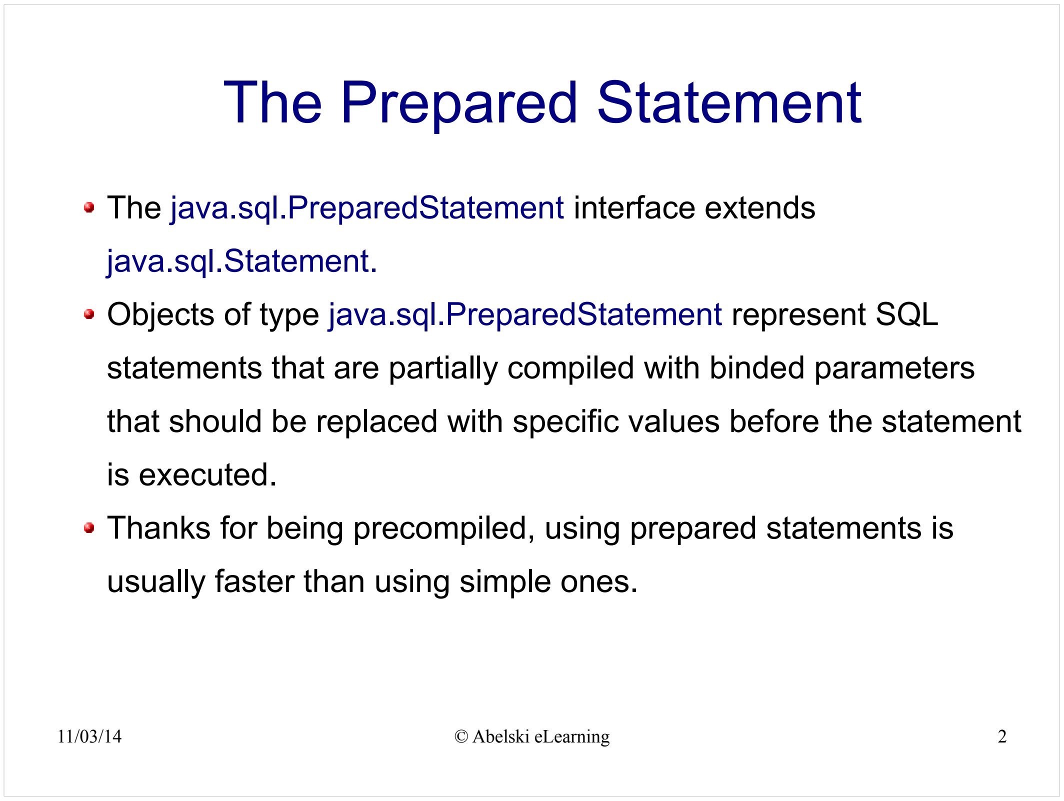
Batch Update Code Sample

```
...
PreparedStatement prepstate =
        connection.prepareStatement("UPDATE customers SET fee=? WHERE type=?");
prepstate.setDouble(1, 4.2);
prepstate.setInt(2, 4);
prepstate.addBatch();
prepstate.setDouble(1, 3.2);
prepstate.setInt(2, 5);
prepstate.addBatch();
prepstate.setDouble(1, 1.2);
prepstate.setInt(2, 9);
prepstate.addBatch();
int[] vec = prepstate.executeBatch();
...
```

Calling the `executeBatch()` method will cause the statement to be executed 3 times. Each time with another set of values replacing the question marks.



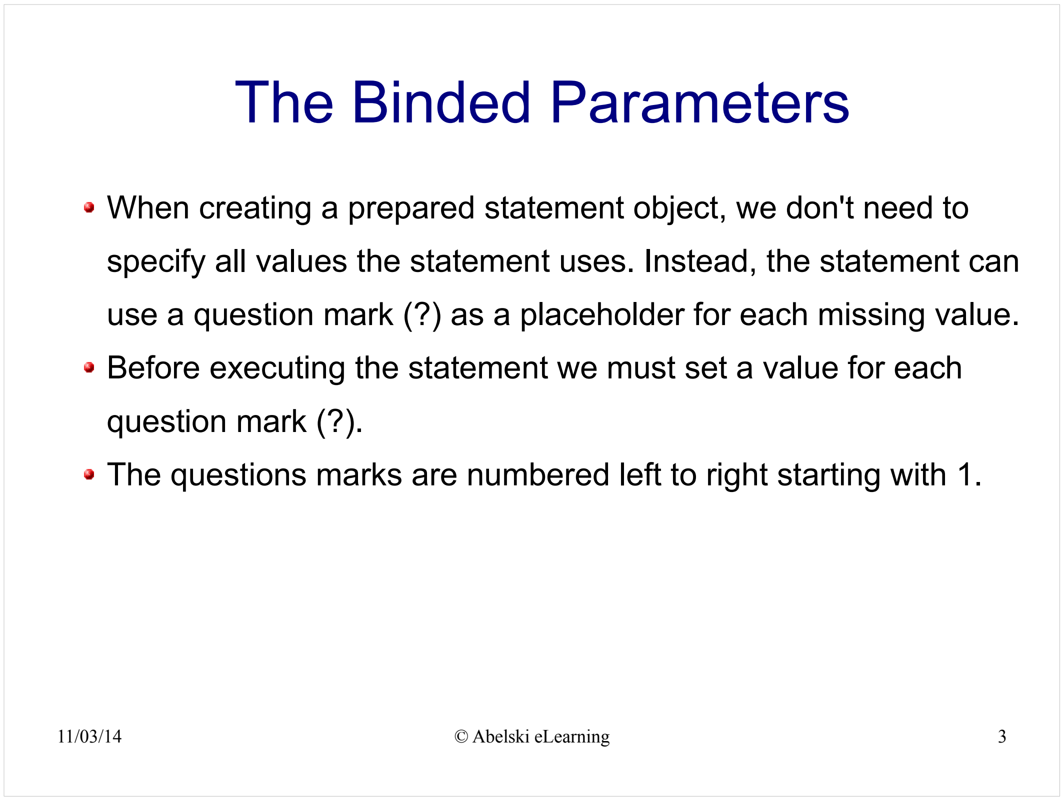
Prepared Statements



The Prepared Statement

- The java.sql.PreparedStatement interface extends java.sql.Statement.
- Objects of type java.sql.PreparedStatement represent SQL statements that are partially compiled with binded parameters that should be replaced with specific values before the statement is executed.
- Thanks for being precompiled, using prepared statements is usually faster than using simple ones.

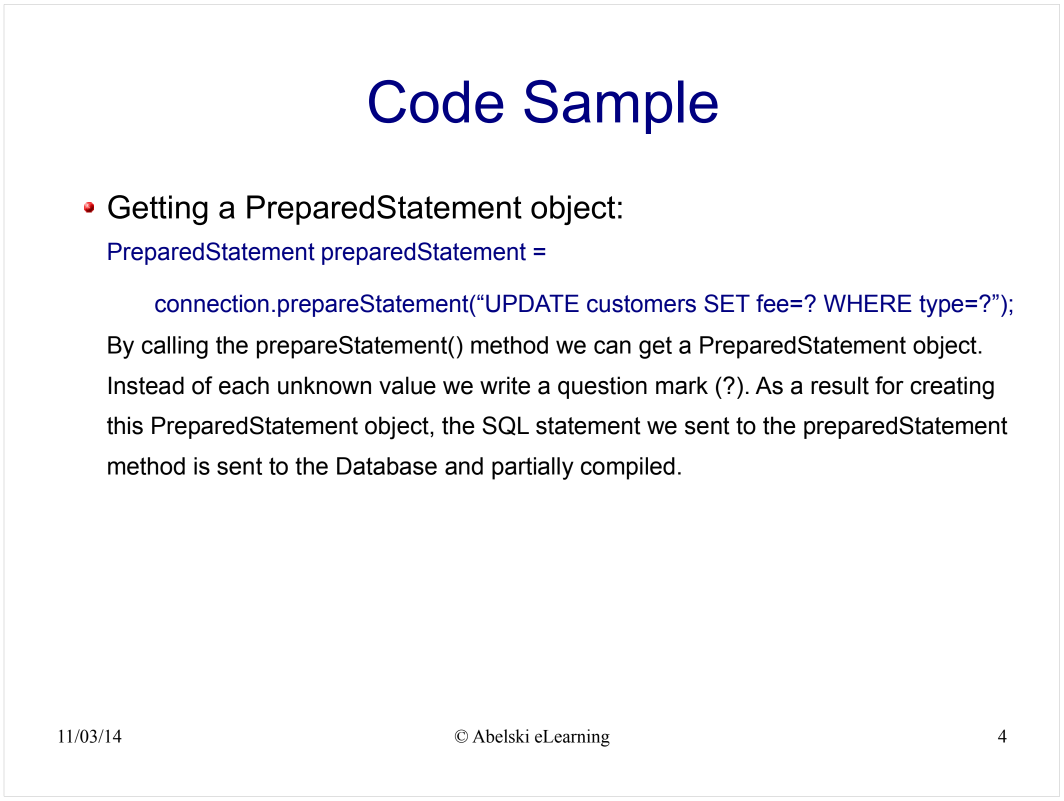
11/03/14 © Abelski eLearning 2



The Binded Parameters

- When creating a prepared statement object, we don't need to specify all values the statement uses. Instead, the statement can use a question mark (?) as a placeholder for each missing value.
- Before executing the statement we must set a value for each question mark (?).
- The questions marks are numbered left to right starting with 1.

11/03/14 © Abelski eLearning 3



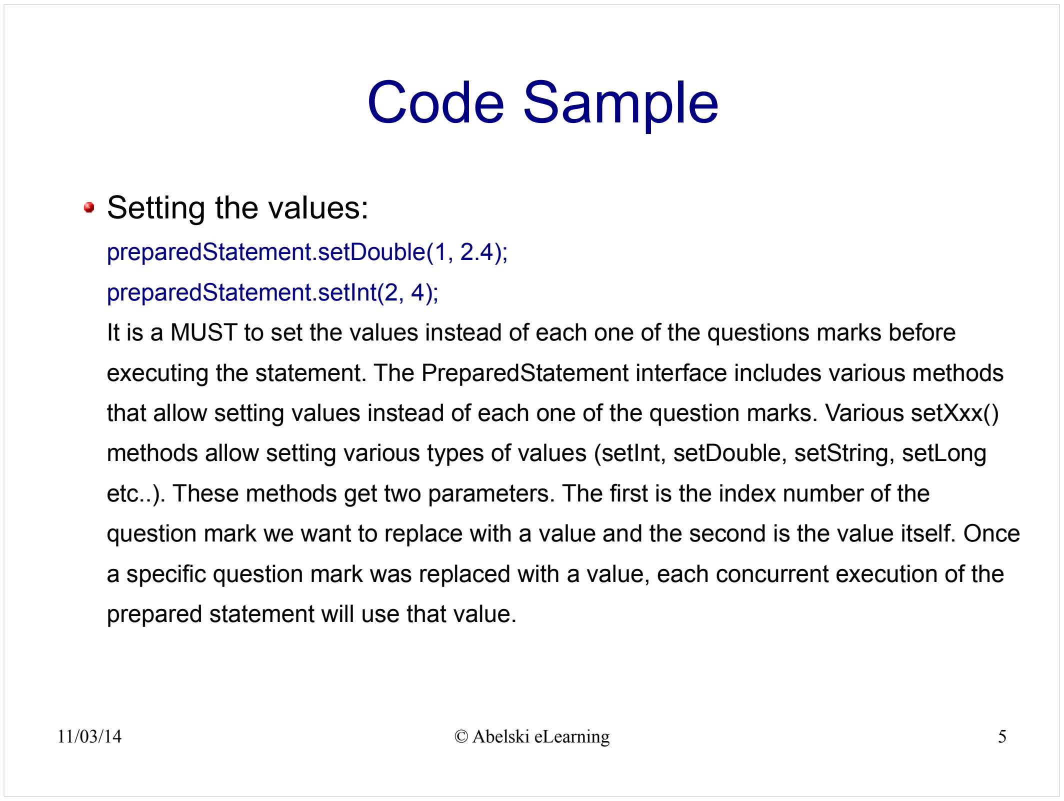
Code Sample

- Getting a PreparedStatement object:

```
PreparedStatement preparedStatement =
    connection.prepareStatement("UPDATE customers SET fee=? WHERE type=?");
```

By calling the prepareStatement() method we can get a PreparedStatement object. Instead of each unknown value we write a question mark (?). As a result for creating this PreparedStatement object, the SQL statement we sent to the preparedStatement method is sent to the Database and partially compiled.

11/03/14 © Abelski eLearning 4



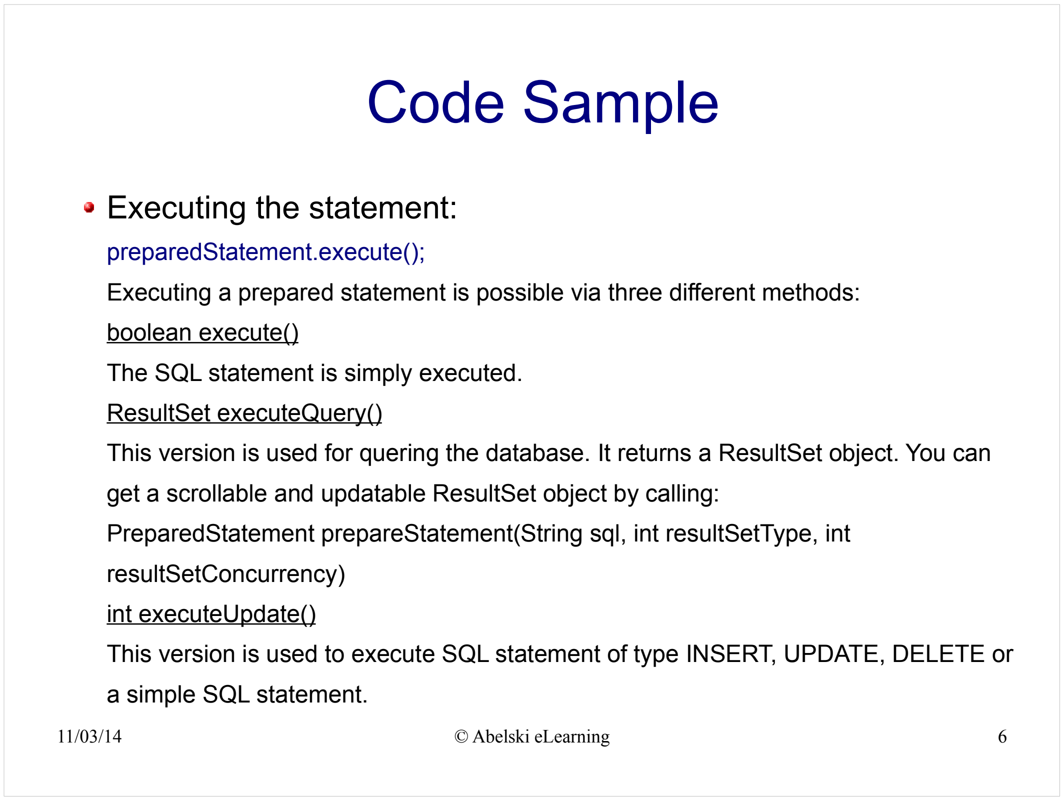
Code Sample

- Setting the values:

```
preparedStatement.setDouble(1, 2.4);
preparedStatement.setInt(2, 4);
```

It is a MUST to set the values instead of each one of the questions marks before executing the statement. The PreparedStatement interface includes various methods that allow setting values instead of each one of the question marks. Various setXxx() methods allow setting various types of values (setInt, setDouble, setString, setLong etc..). These methods get two parameters. The first is the index number of the question mark we want to replace with a value and the second is the value itself. Once a specific question mark was replaced with a value, each concurrent execution of the prepared statement will use that value.

11/03/14 © Abelski eLearning 5



Code Sample

- Executing the statement:

  preparedStatement.execute();

  Executing a prepared statement is possible via three different methods:

  boolean execute()

  The SQL statement is simply executed.

  ResultSet executeQuery()

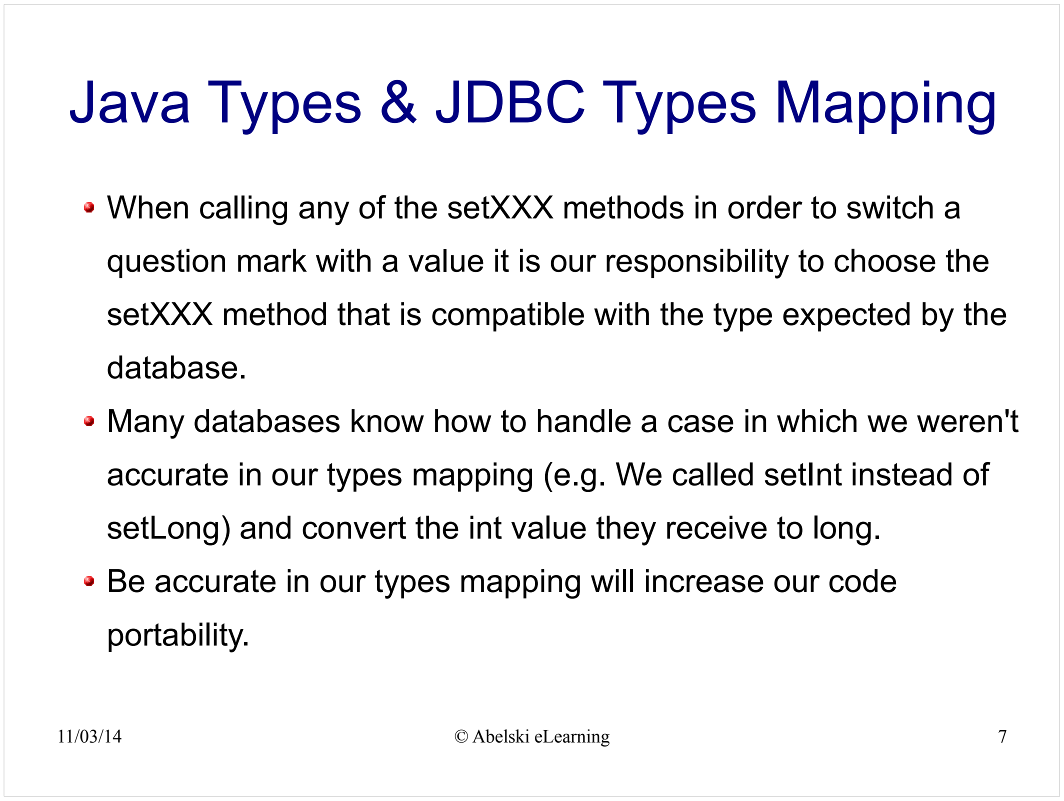
  This version is used for quering the database. It returns a ResultSet object. You can get a scrollable and updatable ResultSet object by calling:

  PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)

  int executeUpdate()

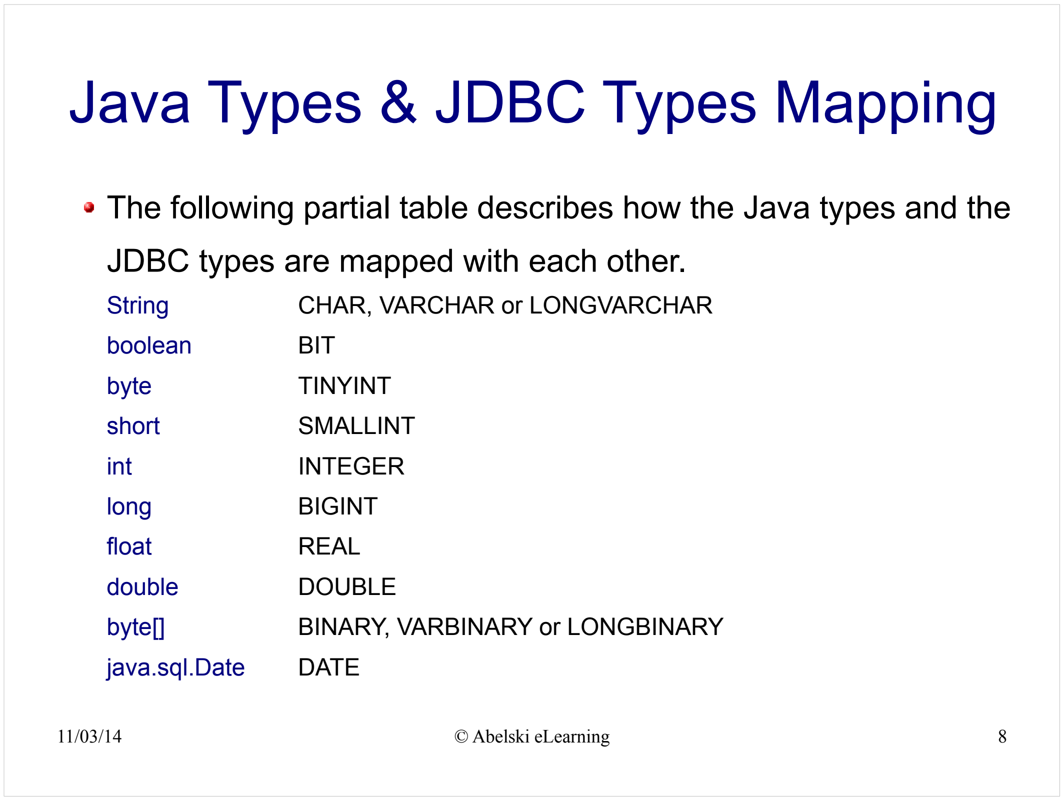
  This version is used to execute SQL statement of type INSERT, UPDATE, DELETE or a simple SQL statement.

11/03/14 © Abelski eLearning 6



Java Types & JDBC Types Mapping

- When calling any of the setXXX methods in order to switch a question mark with a value it is our responsibility to choose the setXXX method that is compatible with the type expected by the database.
- Many databases know how to handle a case in which we weren't accurate in our types mapping (e.g. We called setInt instead of setLong) and convert the int value they receive to long.
- Be accurate in our types mapping will increase our code portability.

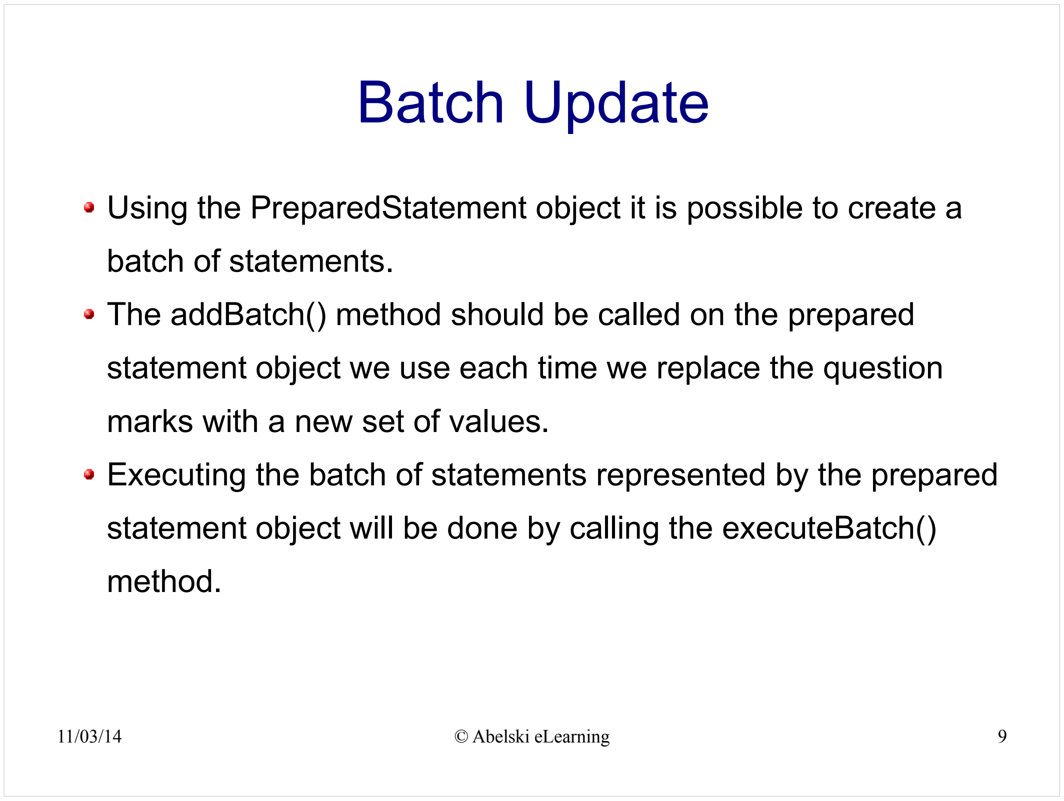


Java Types & JDBC Types Mapping

- The following partial table describes how the Java types and the JDBC types are mapped with each other.

String	CHAR, VARCHAR or LONGVARCHAR
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	BINARY, VARBINARY or LONGBINARY
java.sql.Date	DATE

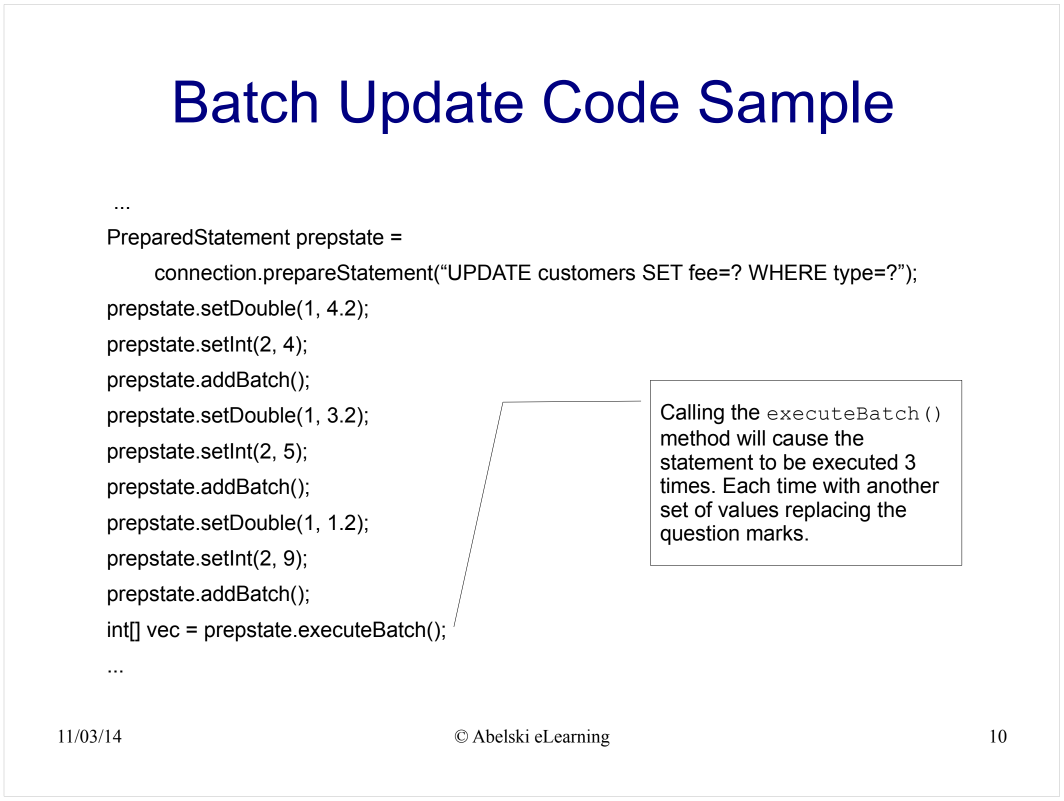
You can find the complete mapping table at
http://java.sun.com/javase/6/docs/technotes/guides/jdbc/



Batch Update

- Using the PreparedStatement object it is possible to create a batch of statements.
- The addBatch() method should be called on the prepared statement object we use each time we replace the question marks with a new set of values.
- Executing the batch of statements represented by the prepared statement object will be done by calling the executeBatch() method.

11/03/14 © Abelski eLearning 9



Batch Update Code Sample

```
 ...
PreparedStatement prepstate =
      connection.prepareStatement(“UPDATE customers SET fee=? WHERE type=?”);
prepstate.setDouble(1, 4.2);
prepstate.setInt(2, 4);
prepstate.addBatch();
prepstate.setDouble(1, 3.2);
prepstate.setInt(2, 5);
prepstate.addBatch();
prepstate.setDouble(1, 1.2);
prepstate.setInt(2, 9);
prepstate.addBatch();
int[] vec = prepstate.executeBatch();
...
```

Calling the `executeBatch()` method will cause the statement to be executed 3 times. Each time with another set of values replacing the question marks.

11/03/14 © Abelski eLearning 10

The prepared statement object used in this sample will be executed three times. The executeBatch() method will return an array of three values representing each one of the values that were returned in each one of the three times the prepared statement was executed.