UNIVERSITÉ Concordia UNIVERSITY

CENTRE FOR CONTINUING EDUCATION

DESKTOP APPLICATION DEVELOPMENT WITH JAVA – CEJV569

Lecture #4

Binary I/O

JDBC

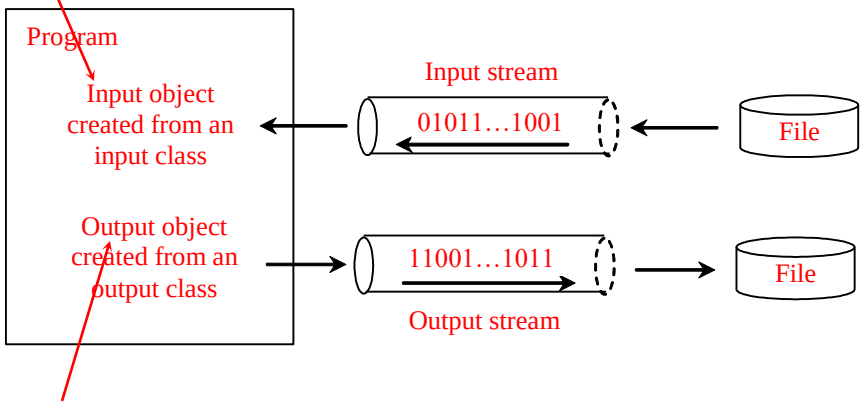
Motivations

- Data stored in a text file is represented in human-readable form.
- Data stored in a binary file is represented in binary form.
  - You cannot read binary files.
- They are designed to be read by programs.
  - For example, Java source programs are stored in text files and can be read by a text editor, but Java classes are stored in binary files and are read by the JVM.
- The advantage of binary files is that they are more efficient to process than text files.

How is I/O Handled in Java?

A File object encapsulates the properties of a file or a path, but does not contain the methods for reading/writing data from/to a file. In order to perform I/O, you need to create objects using appropriate Java I/O classes.

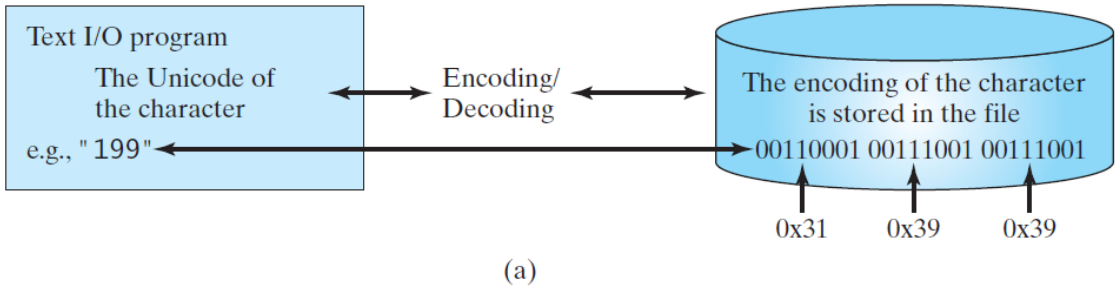
```
Scanner input = new Scanner(new File("temp.txt"));
System.out.println(input.nextLine());
```



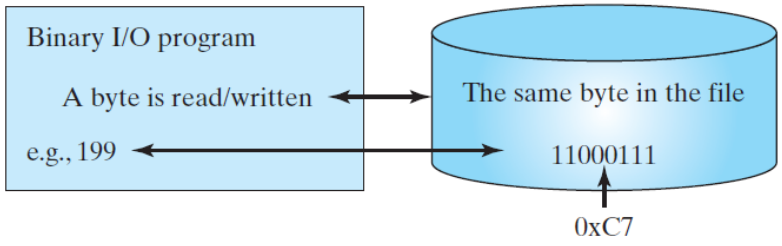

```
PrintWriter output = new PrintWriter("temp.txt");
output.println("Java 101");
output.close();
```

Binary I/O

- Text I/O requires encoding and decoding.
- The JVM converts a Unicode to a file specific encoding when writing a character and coverts a file specific encoding to a Unicode when reading a character.
- Binary I/O does not require conversions.
- When you write a byte to a file, the original byte is copied into the file.
- When you read a byte from a file, the exact byte in the file is returned.



(a)



(b)

FileInputStream/FileOutputStream

- FileInputStream/FileOutputStream associates a binary input/output stream with an external file.
- All the methods in FileInputStream/FileOuptputStream are inherited from its superclasses.

FileInputStream

To construct a FileInputStream, use the following constructors:

public FileInputStream(String filename)

public FileInputStream(File file)

A java.io.FileNotFoundException would occur if you attempt to create a FileInputStream with a nonexistent file.

FileOutputStream

- To construct a FileOutputStream, use the following constructors:

```
public FileOutputStream(String filename)
public FileOutputStream(File file)
public FileOutputStream(String filename, boolean append)
public FileOutputStream(File file, boolean append)
```

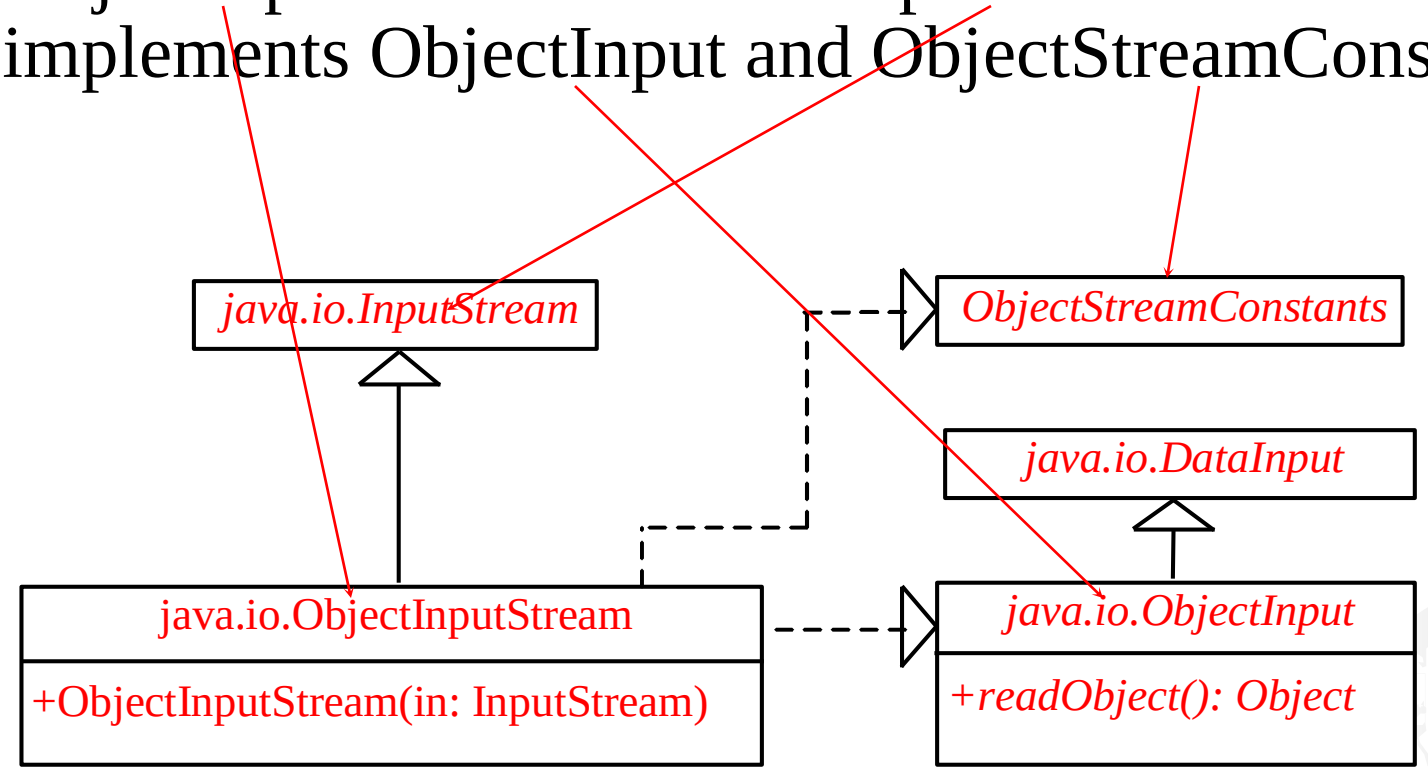
- If the file does not exist, a new file would be created. If the file already exists, the first two constructors would delete the current contents in the file.
- To retain the current content and append new data into the file, use the last two constructors by passing true to the append parameter.

Checking End of File

TIP: If you keep reading data at the end of a stream, an EOFException would occur. So how do you check the end of a file? You can use input.available() to check it. input.available() == 0 indicates that it is the end of a file.

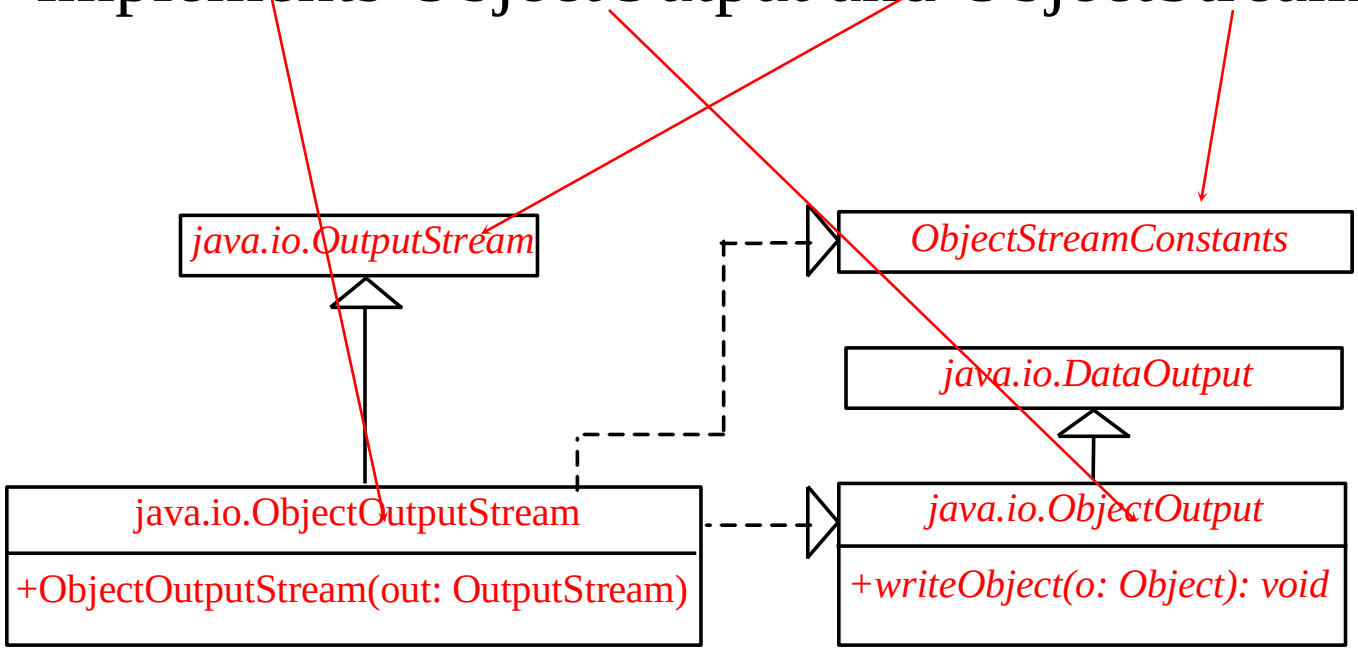
ObjectInputStream

ObjectInputStream extends InputStream and implements ObjectInput and ObjectStreamConstants.



ObjectOutputStream

ObjectOutputStream extends OutputStream and implements ObjectOutput and ObjectStreamConstants.



Using Object Streams

You may wrap an ObjectInputStream/ObjectOutputStream on any InputStream/OutputStream using the following constructors:

```
// Create an ObjectInputStream
public ObjectInputStream(InputStream in)

// Create an ObjectOutputStream
public ObjectOutputStream(OutputStream out)
```

ObjectOutputStream

```
import java.io.*;

public class TestObjectOutputStream {
  public static void main(String[] args) throws IOException {
    try ( // Create an output stream for file object.dat
      ObjectOutputStream output =
        new ObjectOutputStream(new FileOutputStream("object.dat"));
    ) {
      // Write a string, double value, and object to the file
      output.writeUTF("John");
      output.writeDouble(85.5);
      output.writeObject(new java.util.Date());
    }
  }
}
```

ObjectInputStream

```
import java.io.*;

public class TestObjectInputStream {
  public static void main(String[] args)
    throws ClassNotFoundException, IOException {
    try ( // Create an input stream for file object.dat
      ObjectInputStream input =
        new ObjectInputStream(new FileInputStream("object.dat"));
    ) {
      // Read a string, double value, and object from the file
      String name = input.readUTF();
      double score = input.readDouble();
      java.util.Date date = (java.util.Date)(input.readObject());
      System.out.println(name + " " + score + " " + date);
    }
  }
}
```

The `Serializable` Interface

Not all objects can be written to an output stream. Objects that can be written to an object stream is said to be **serializable.** A serializable object is an instance of the java.io.Serializable interface. So the class of a serializable object must implement Serializable.

The Serializable interface is a marker interface. It has no methods, so you don't need to add additional code in your class that implements Serializable.

Implementing this interface enables the Java serialization mechanism to automate the process of storing the objects and arrays.

The `transient` Keyword

- If an object is an instance of Serializable, but it contains non-serializable instance data fields, can the object be serialized?
- The answer is no.
- To enable the object to be serialized, you can use the ***transient*** keyword to mark these data fields to tell the JVM to ignore these fields when writing the object to an object stream.

The `transient` Keyword, cont.

Consider the following class:

```
public class Foo implements java.io.Serializable {
  private int v1;
  private transient A v3 = new A();
}
class A { } // A is not serializable
```

- When an object of the Foo class is serialized, variable v3 is not serialized because it is marked transient.
- If v3 were not marked transient, a java.io.NotSerializableException would occur.

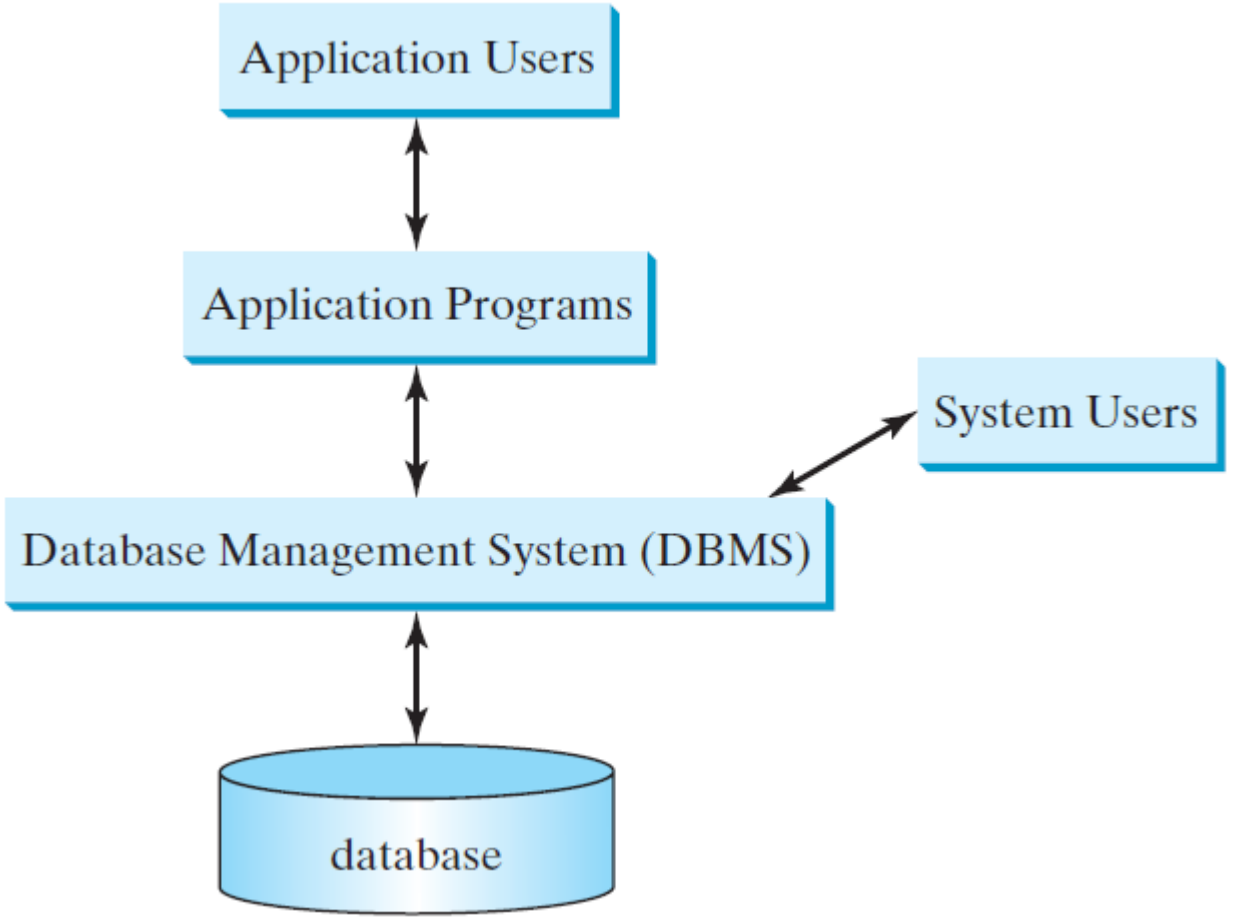
Serializing Arrays

- An array is serializable if all its elements are serializable.
- So an entire array can be saved using writeObject into a file and later restored using readObject.

Exercise 17

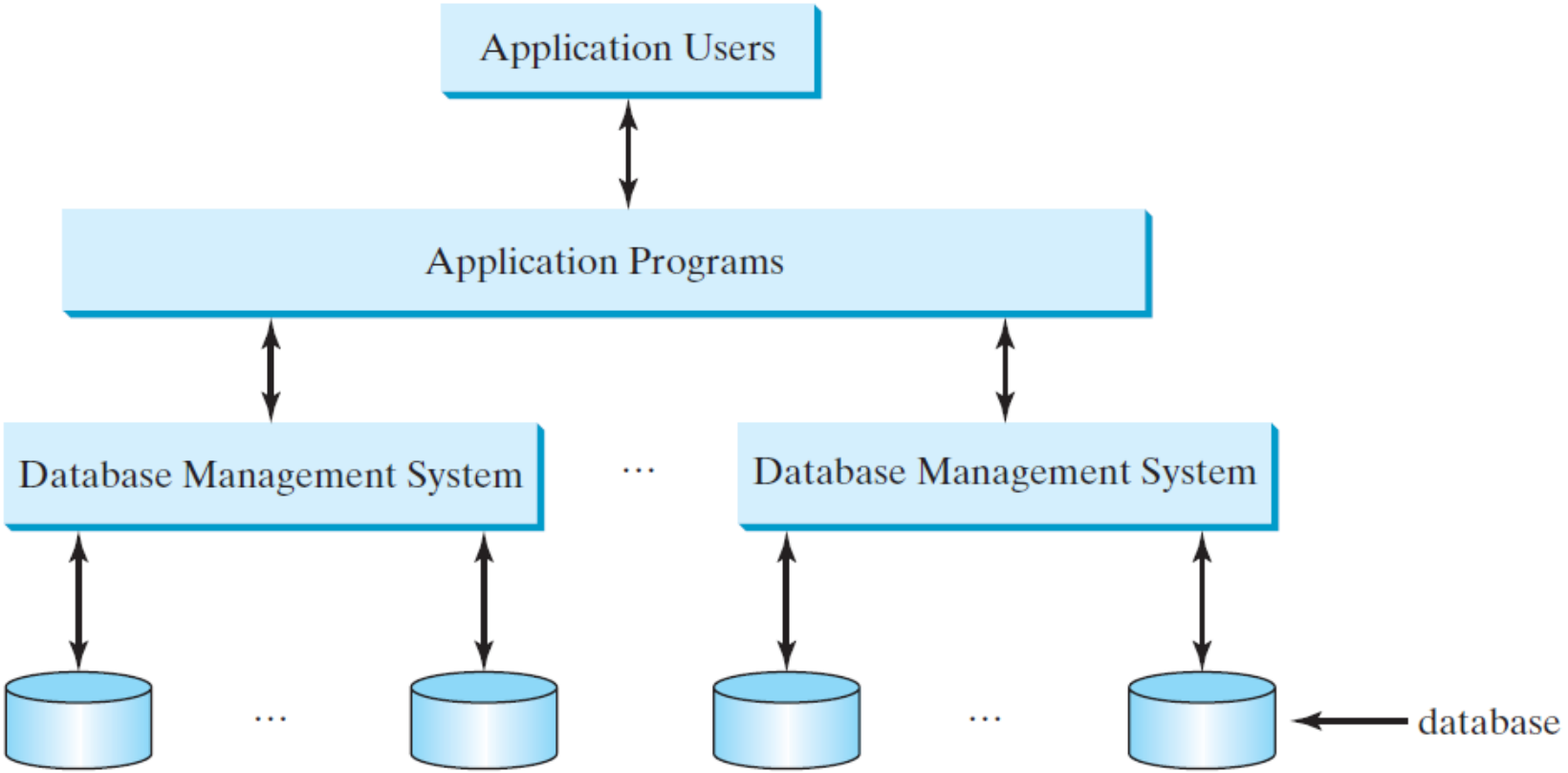
- Create a Binary Data File
- Store objects and Arrays in a file

What is a Database System?



e.g., Access, MySQL, Oracle, and MS SQL Server

Database Application Systems



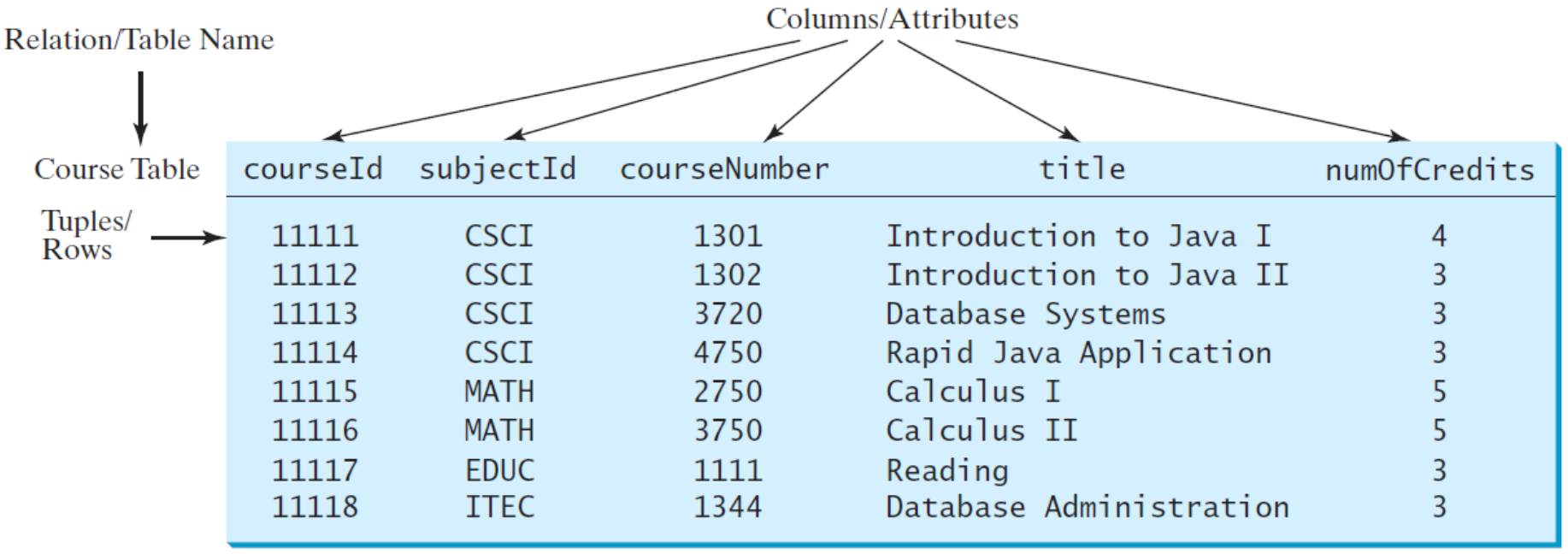
Relational Structure

- A relational database consists of a set of relations.
- A relation has two things in one: a *schema* and an *instance* of the schema.
- The schema defines the relation and an instance is the content of the relation at a given time.

Relational Structure

- An instance of a relation is nothing more than a table with rows and named columns.
- For convenience with no confusion, we refer instances of relations as just *relations* or *tables*.

Course Table



courseId	subjectId	courseNumber	title	numOfCredits
11111	CSCI	1301	Introduction to Java I	4
11112	CSCI	1302	Introduction to Java II	3
11113	CSCI	3720	Database Systems	3
11114	CSCI	4750	Rapid Java Application	3
11115	MATH	2750	Calculus I	5
11116	MATH	3750	Calculus II	5
11117	EDUC	1111	Reading	3
11118	ITEC	1344	Database Administration	3

Student Table

Student Table

ssn	firstName	mi	lastName	phone	birthDate	street	zipCode	deptID
444111110	Jacob	R	Smith	9129219434	1985-04-09	99 Kingston Street	31435	BIOL
444111111	John	K	Stevenson	9129219434	null	100 Main Street	31411	BIOL
444111112	George	K	Smith	9129213454	1974-10-10	1200 Abercorn St.	31419	CS
444111113	Frank	E	Jones	9125919434	1970-09-09	100 Main Street	31411	BIOL
444111114	Jean	K	Smith	9129219434	1970-02-09	100 Main Street	31411	CHEM
444111115	Josh	R	Woo	7075989434	1970-02-09	555 Franklin St.	31411	CHEM
444111116	Josh	R	Smith	9129219434	1973-02-09	100 Main Street	31411	BIOL
444111117	Joy	P	Kennedy	9129229434	1974-03-19	103 Bay Street	31412	CS
444111118	Toni	R	Peterson	9129229434	1964-04-29	103 Bay Street	31412	MATH
444111119	Patrick	R	Stoneman	9129229434	1969-04-29	101 Washington St.	31435	MATH
444111120	Rick	R	Carter	9125919434	1986-04-09	19 West Ford St.	31411	BIOL

Enrollment Table

Enrollment Table

ssn	courseId	dateRegistered	grade
444111110	11111	2004-03-19	A
444111110	11112	2004-03-19	B
444111110	11113	2004-03-19	C
444111111	11111	2004-03-19	D
444111111	11112	2004-03-19	F
444111111	11113	2004-03-19	A
444111112	11114	2004-03-19	B
444111112	11115	2004-03-19	C
444111112	11116	2004-03-19	D
444111113	11111	2004-03-19	A
444111113	11113	2004-03-19	A
444111114	11115	2004-03-19	B
444111115	11115	2004-03-19	F
444111115	11116	2004-03-19	F
444111116	11111	2004-03-19	D
444111117	11111	2004-03-19	D
444111118	11111	2004-03-19	A
444111118	11112	2004-03-19	D
444111118	11113	2004-03-19	B

SQL

- To access or write applications for database systems, you need to use the Structured Query Language (SQL).
- SQL is the universal language for accessing relational database systems.
- Application programs may allow users to access database without directly using SQL, but these applications themselves must use SQL to access the database.

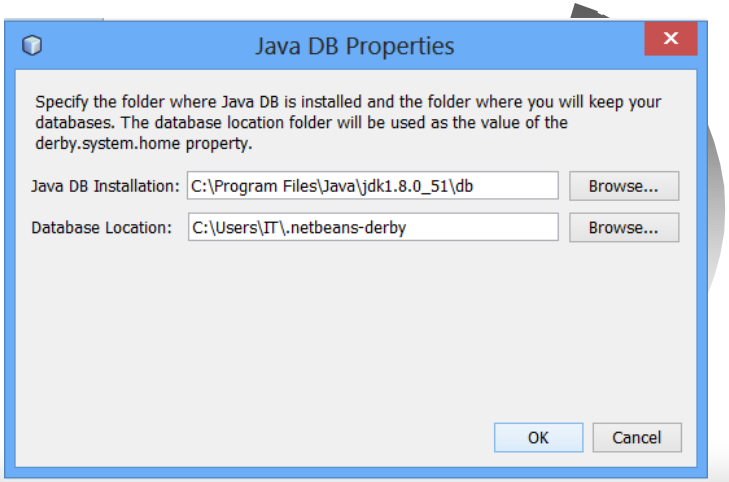
Using Derby in NetBeans

- Derby can be configured and used with the tools provided by NetBeans
- The following instructions come from:
  - http://netbeans.org/kb/docs/ide/java-db.html
- Every installation of Java includes the Derby DBMS
- On a Windows machine they are found in a folder called db where the SDK is installed such as
  - C:\Program Files\Java\jdk1.8.0\db

Registering the Database in NetBeans IDE

1. In the Services window, right-click the Java DB Database node and choose Properties to open the Java DB Settings dialog box.
2. For the Java DB Installation text field, enter the path to the Java DB root directory (javadb) that you specified in the previous step.
3. For Database Location, use the default location if a location is already provided.

   Click OK



Starting the Server and Creating a Database

- Java DB Database menu options are displayed when you right-click the Java DB node in the Services window.
- Contextual menu items allow you to start and stop the database server, create a new database instance, as well as register database servers in the IDE
- To start the database server:
  - In the Services window, right-click the Java DB node and choose Start Server.
  - Right-click the Java DB node and choose Create Database to open the Create Java DB Database dialog.
  - Type **fishDB** for the Database Name.
  - Type **fish** for the User Name and Password.
  - Click OK.

Connecting to the Database

- In the Services window of the IDE you can perform the following common tasks on database structures, Create, Read, Update, and Delete (CRUD):
  - creating, deleting, modifying tables
  - populating tables with data
  - viewing tabular data
  - executing SQL statements and queries

Using the SQL Editor:

- In the Service window right-click the fishDB on the Tables node
  - Chose Connect
  - Choose Execute Command.
- A blank canvas opens in the SQL Editor in the main window.

Exercise 18

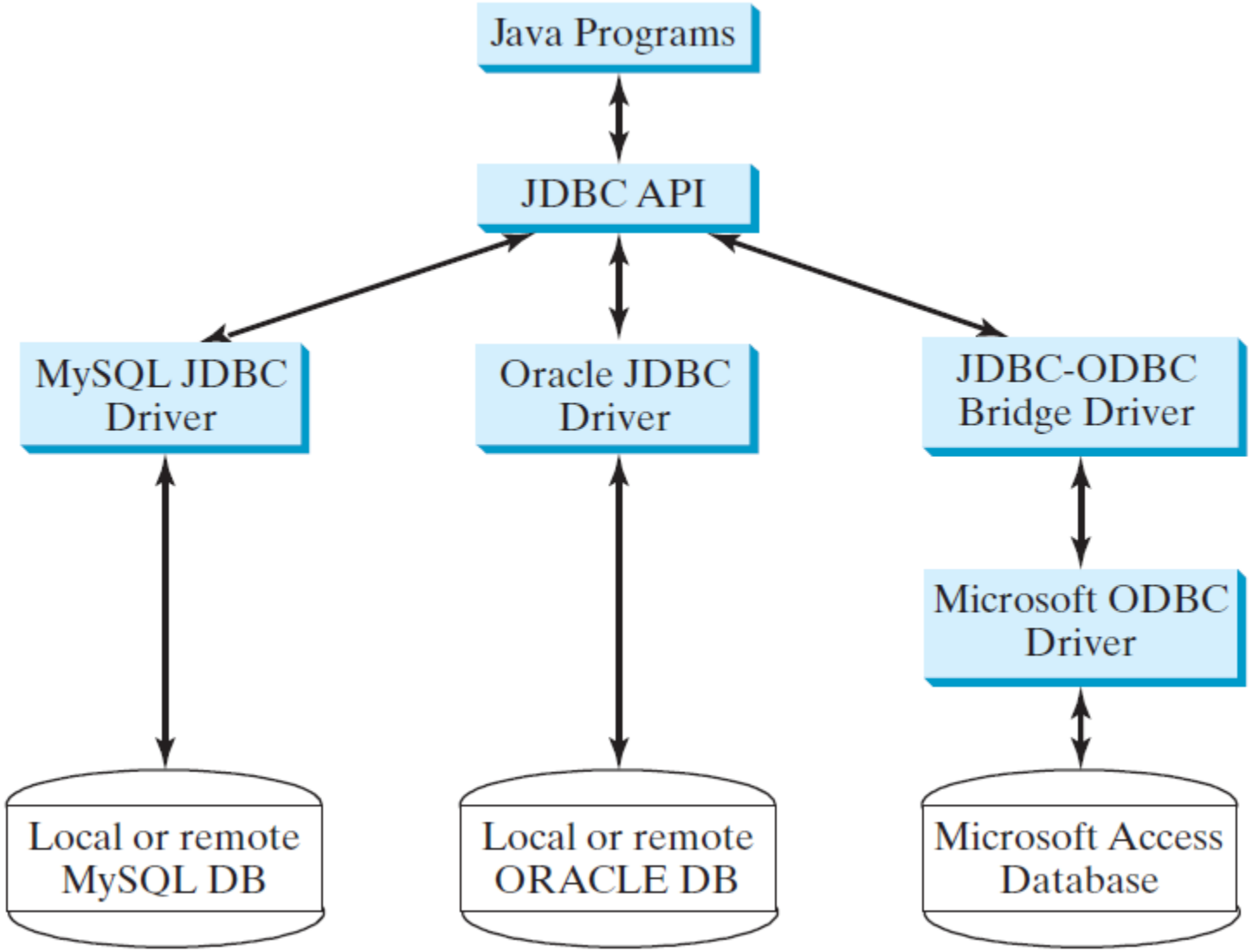
- Creating School database

What Does JDBC Do?

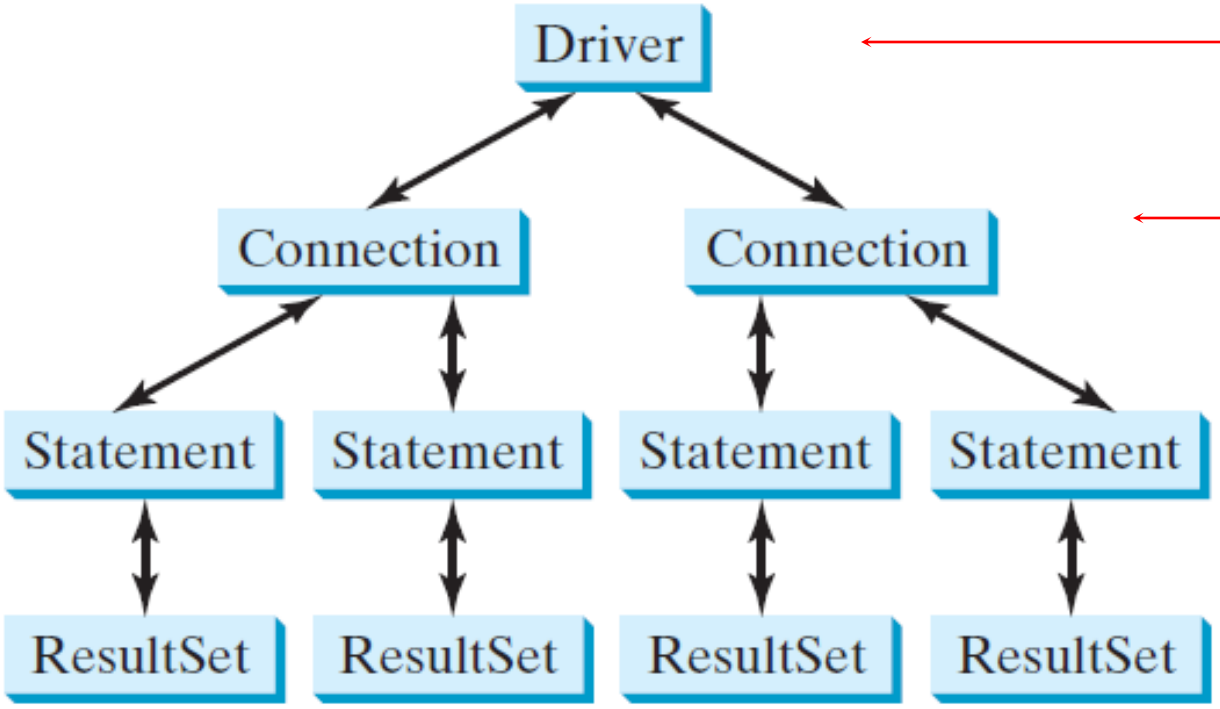
- Makes it possible to do these three things:

1. establish a connection with a database
2. execute SQL statements
3. process the results from the statements

The Architecture of JDBC



The JDBC Interfaces



Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Database Driver

- JDBC defines a standard API for database access
- It does not know the specific language of any database
- JDBC drivers must be available to the code to translate between JDBC and the particular database

Establish a connection – the database parameters

- Using Derby on a local server
  - `String url = "jdbc:derby://localhost:1527/FishDB";`
  - `String user = “fish";`
  - `String password = “fish";`
- Using MySQL on a remote server
  - `String url = "jdbc:mysql://url.to.db:3306/db.name";`
  - `String user = “root";`
  - `String password = “rootPass";`

Connecting to a Database

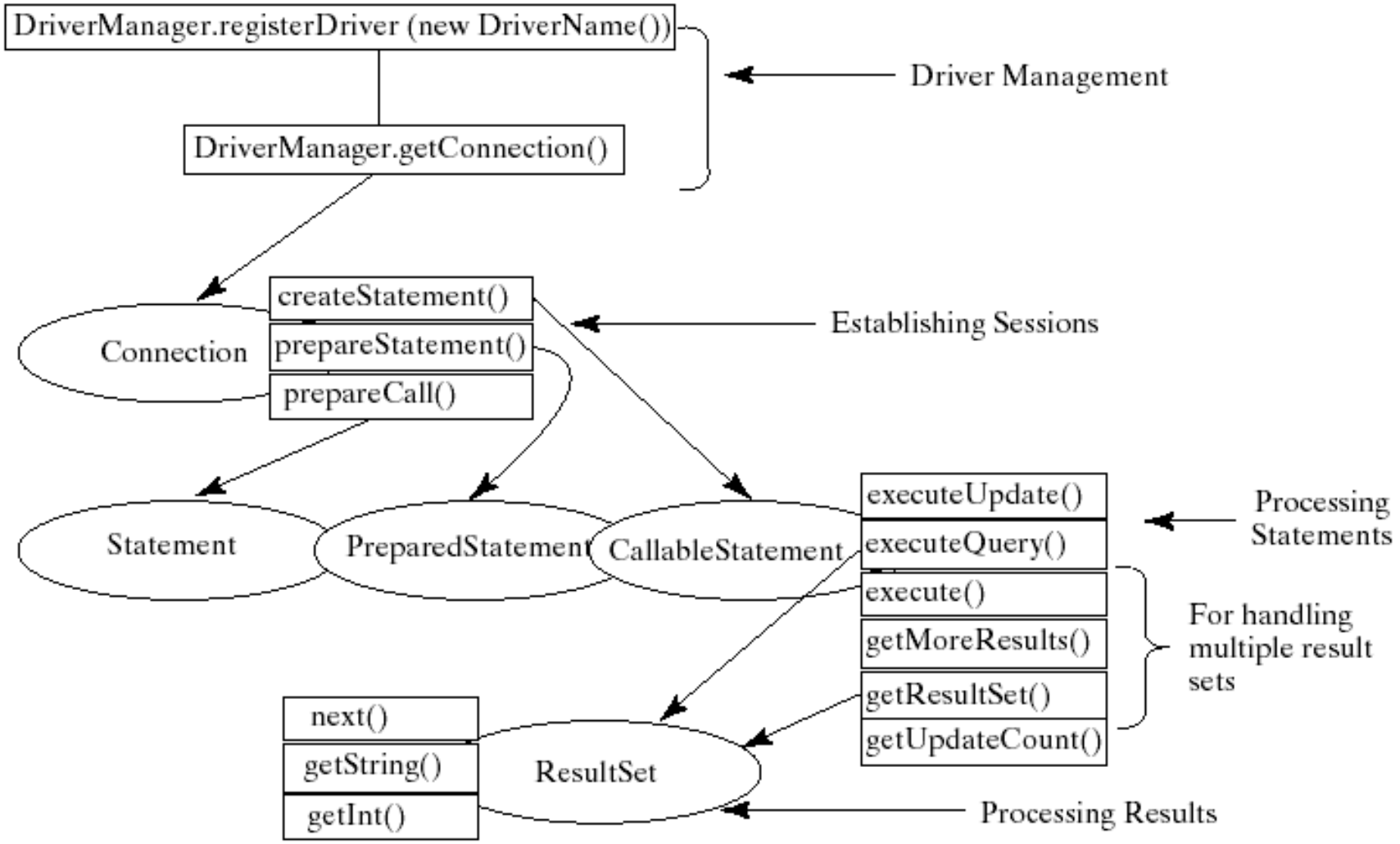
```
try (
   Connection connection = DriverManager.getConnection(url, user, password);
   // Code to read or write goes here
) {


   // Code to process results of read or write go here



} catch (SQLException sqlex) {
   // Code to handle the exception
}
```

Processing Statements Diagram



Statements

- A statement is responsible for transmitting SQL to the database
- Statement objects are created by the connection object
- Throws an exception if the SQL is invalid
- Two types of statements
  - PreparedStatement
    - Secure
  - Statement
    - Insecure

Send SQL statements

```
String query = "SELECT * FROM FISH";
try (
    Connection connection =
        DriverManager.getConnection(url, user, password);
    PreparedStatement pStatement =
        connection.prepareStatement(query);
    ResultSet resultSet = pStatement.executeQuery())
) {
    // Code to process results of read or write go here
} catch (SQLException sqlex) {
    // Code to handle the exception
}
```

ResultSet

- The result of a statement that does not throw an exception is stored in a ResultSet object
- Contains an array of objects for every record returned by the statement
- If many records are returned it automatically pages through the result
- May contain zero arrays of records if query returns nothing

ResultSet

- **`prepareStatement`** method of a **`Connection`** object creates a **`PreparedStatement`** object
- **`executeQuery`** method of the **`Statement`** object executes a SELECT statement that returns a **`ResultSet`** object
- By default, the **`prepareStatement`** method creates a forward-only, read-only result set
- In most cases this is all we need

How to move the cursor through a result set

- When you create a result set, the cursor is positioned before the first record.
- The first time you call the **`next()`** method:
  - If records are found then will move to the first record in the result set and return true
  - If no records are found then false is returned
- **`next()`** will return false when there are no more records to process
- All of the methods throw an exception of the SQLException type.

Forward only, Read only

- Queries whose result is transferred to a data structure coded by the programmer

```
PreparedStatement pStatement =
    connection.prepareStatement("SELECT * FROM FISH");
ResultSet rs = pStatement.executeQuery();
while(rs.next())
{
    // code that works with each record
}
```

Processing records – creating objects

```
ArrayList<FishData> rows = new ArrayList<>();
String query = "SELECT * FROM FISH";
try (
      Connection connection = DriverManager.getConnection(url, user, password);
      PreparedStatement pStatement = connection.prepareStatement(query);
      ResultSet resultSet = pStatement.executeQuery()) {
   while (resultSet.next()) {
      rows.add(new FishData(
        resultSet.getLong("ID"),          resultSet.getString("COMMONNAME"),
        resultSet.getString("LATIN"),     resultSet.getString("PH"),
        resultSet.getString("KH"),        resultSet.getString("TEMP"),
        resultSet.getString("FISHSIZE"), resultSet.getString("SPECIESORIGIN"),
        resultSet.getString("TANKSIZE"), resultSet.getString("STOCKING"),
        resultSet.getString("DIET")));
   }
} catch (SQLException sqlex) {
   sqlex.printStackTrace();
}
```

Parameterized Query

- **String Concatenation: Bad**

```
String loanType = getLoanType();
PreparedStatement pStatement = conn.prepareStatement(
   "select banks from loan where loan_type=" + loanType);
```

- **Paramerterized: Good**

```
String loanQuery = "select banks from loan where loan_type = ?";

String loanType = getLoanType();

PreparedStatement pStatement= conn.prepareStatement(loanQuery);

pStatement.setString(1,loanType);
```

How to work with prepared statements

- Prepared statements are cached on the server for reuse.
- Server only has to check the syntax and prepare an execution plan once for each SQL statement.
- Improves the efficiency of the database operations.
- To specify a parameter for a prepared statement, type a question mark (?) in the SQL statement.

How to work with prepared statements

- To supply values for the parameters in a prepared statement, use the set methods of the PreparedStatement interface.
- To execute a SELECT statement, use the executeQuery method.
- To execute an INSERT , UPDATE, or DELETE statement, use the executeUpdate method.

Security with PreparedStatements

- Parameter values in a PreparedStatement may not contain SQL syntax
- Placing SQL code into a statement is called SQL Injection
- Most common exploit for compromising databases
- PreparedStatement will reject parameters with SQL

How to use a prepared statement

To return a result set

```
String preparedSQL = "SELECT Code, Description, Price "
    + "FROM Product WHERE Code = ?";
PreparedStatement ps =
    connection.prepareStatement(preparedSQL);
ps.setString("Code", productCode);
ResultSet product = ps.executeQuery();
```

How to use a prepared statement

To modify data

```
String preparedSQL = "UPDATE Product SET "
                   + "   Description = ?, "
                   + "   Price = ?"
                   + "WHERE Code = ?";
PreparedStatement ps =
     connection.prepareStatement(preparedSQL);
ps.setString(1, product.getDescription());
ps.setDouble(2, product.getPrice());
ps.setString(3, product.getCode());
int records = ps.executeUpdate();
// executeUpdate returns the number of records
// affected by the update
// This should be tested to determine if the
// the update was successful
```

How to use a prepared statement

To insert a record

```
String preparedQuery =
       "INSERT INTO Product (Code,
        Description, Price) "
    + "VALUES (?, ?, ?)";
PreparedStatement ps =
    connection.prepareStatement(preparedQuery);
ps.setString(1, product.getCode());
ps.setString(2, product.getDescription());
ps.setDouble(3, product.getPrice());
int records = ps.executeUpdate();
```

How to use a prepared statement

To delete a record

```
String preparedQuery = "DELETE FROM Product "
                     + "WHERE Code = ?";
PreparedStatement ps =
    connection.prepareStatement(preparedQuery);
ps.setString(1, productCode);
int records = ps.executeUpdate();
```

The get methods of a ResultSet object can be used to return…

- All eight primitive types
- Some objects such as dates and times
- *BLOB objects* (*Binary Large Objects*) and *CLOB objects* (*Character Large Objects*)

Selected methods of a ResultSet object that return data from a result set

Method	Description
`getString(int ColumnIndex)`	Returns a String from the specified column number.
`getString(String ColumnName)`	Returns a String from the specified column name.
`getDouble(int ColumnIndex)`	Returns a double value from the specified column number.
`getDouble(String ColumnName)`	Returns a double value from the specified column name.

How to retrieve data from a result set

- The getXXX methods can be used to return all eight primitive types. Examples:
  - getInt
  - getLong
- The getXXX methods can also be used to return strings, dates, and times. Examples:
  - getString
  - getDate
  - getTime
  - getTimestamp

How to use the getMetaData method to create a ResultSetMetaData object

```
ResultSetMetaData metaData = resultSet.getMetaData();
```

How to work with metadata

- Information about the definition of a result set – like the number and names of its columns – is known as *metadata*.
- To access the metadata for a result set, you can return a ResultSetMetaData object from a ResultSet object and then use its methods.

Methods of a ResultSetMetaData object for working with metadata

Method	Description
`getColumnCount()`	Returns the number of columns in this RecordSet object as an int type.
`getColumnName(intColumn)`	Returns the name of this column as a String object.
`getColumnLabel(intColumn)`	Returns the label of this column as a String object.
`getColumnType(intColumn)`	Returns an int type that represents the SQL data type that's used to store the data in this column.
`getColumnTypeName(intColumn)`	Returns a String object that identifies the SQL data type that's used to store the data in this

A method that returns the column names of a result set

```
public ArrayList<String>
   getColumnNames(ResultSet results) throws SQLException {
    ArrayList<String> columnNames = new ArrayList();
    ResultSetMetaData metaData = results.getMetaData();
    int columnCount = metaData.getColumnCount();
    for (int i = 1; i <= columnCount; i++)
        columnNames.add(metaData.getColumnName(i));
    return columnNames;
}
```

How SQL data types map to Java data types

SQL data type	Java data type
VARCHAR, LONGVARCHAR	String
BIT	boolean
TINYBIT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
DOUBLE	double
VARBINARY, LONGVARBINARY	byte[]
NUMERIC	java.math.BigDecimal
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
DATETIME	java.sql.Timestamp

Exercise 19

- Open Exercise 19
- Spend some time to understand the structure of this project
- Complete the class FishDAOImpl.java
- Modify the class SimpleJDBCConsole.java to test all of your new methods.