JDBC

Using JDBC: need a "driver"

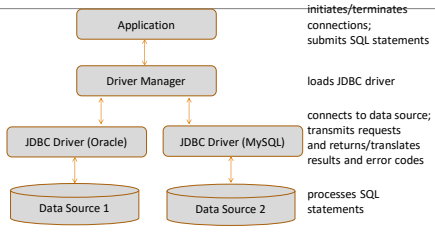
The JDBC API is part of standard Java library or "JDK", so no special imports or jar files are needed to use it.

But it needs a "driver" for the DB, a jar file

- we need an Oracle driver, ojdbc6.jar for Java 6+ (There is an ojdbc7.jar now, but it has the same code, just compiled with Java 7, no advantage, and a definite disadvantage if you are stuck with Java 6 yourself.) We are assuming Java 8 or higher for our work. See DevelopmentSetup.html.
- also one for mysql longname.jar, and h2.jar

All three of these work on UNIX/Linux, Mac, and Windows (Java portability!)

JDBC: Architecture



Using JDBC: Java Tutorial

JDBC features we don't need, at least for now:

- BLOBs
- Scrollable result sets
- "Metadata" API

Next: JDBCCheckup.java (it's in the jdbc directory, $cs636/jdbc/JDBCCheckup.java)

Using JDBC (covered in cs430/630)

3 steps to submit a database query:

1. Load the JDBC driver (not needed explicitly in JDBC 4.0). Each database vendor offers this jar file on their website. This jar file needs to be on the Java classpath.
2. Connect to the data source
3. Execute SQL statements

Building JDBCCheckup.java

First compile JdbcCheckup.java.

```
javac JdbcCheckup.java
```

Now we have JdbcCheckup.class in the current dir.

Use java to run it as follows. We need to add the driver jar file to the classpath to give the program access to the driver software:

```
java -classpath driver.jar;. JdbcCheckup    (Windows)
```

(change ';' to ':' for UNIX/Linux)

Driver.jar: ojdbc6.jar or mysql-connector-java-xxx-bin.jar or h2.jar. These are all in the jdbc subdirectory of our class website, in filesys at /data/htdocs/cs636/jdbc.

Running JdbcCheckup to Oracle from pe07, i.e., from inside the firewall

```
pe07$ java -cp ojdbc6.jar:. JdbcCheckup
Please enter information to test connection to the database
Using Oracle (o), MySql (m) or HSQLDB (h)? O
user: xxxxxx
password: xxxxxx
host: dbs3.cs.umb.edu      ← we can use dbs3's real name here*
port (often 1521): 1521
sid (site id): dbs3
using connection string: jdbc:oracle:thin:@localhost:1521:dbs3
Connecting to the database...connected.

Hello World!
Your JDBC installation is correct.
```

* whereas we'll need to enter `localhost` here when we use a tunnel from outside the firewall

JdbcCheckup.java: getting a Connection

In the program, we find out what database the user wants to connect to, and their username and password (for Oracle or mysql)

For Oracle:
- the server host is "dbs3.cs.umb.edu"
- port 1521
- sid = "dbs3"

These are used in the "connection string" or "database url"

connStr = "jdbc:oracle:thin:@dbs3.cs.umb.edu:1521/dbs3". // for use inside the firewall

The code ends up with strings connStr, user, and password.

Then get a connection from the driver:

```
Connection conn =
   DriverManager.getConnection(connStr, user, password);
```


JDBCCheckup.java: use of Connection

Create a statement using the Connection object:

```
Statement stmt = conn.create statement();
```

Do DB actions using Statement –

```
stmt.execute("drop table welcome") ;
stmt.execute("create table welcome(msg char(20))");
stmt.execute("insert into welcome values ('Hello World!')");
ResultSet rest =
          stmt.excuteQuery("select * from welcome") ;
```

Display row, close connection (and its associated objects)

`conn.close()` ← important to free up TCP/IP connection into the DB

JdbcCheckup.java: Exceptions

The previous slide ignored Exceptions

JDBC coding involves messy Exception handling

We need to use **finally** as well as **try** and **catch**

Need to review Exceptions

Java Exceptions

We looked at this excellent tutorial: http://www.tutorialspoint.com/java/java_exceptions.htm

Another tutorial: Java Tutorial: Exceptions

Skip coverage on **The try-with-resources Statement**

- For this to work, we need a JDBC 4.1 driver. Until these are common, it's best to avoid using try-with-resources. For Oracle, see current 12c driver downloads for such a driver.
- All it does is slightly simplify the try-catch syntax, i.e., it has no new capabilities

Details on getting a Connection

How does the JDBC driver code get accessed in programs?

JDBC has a standard that specifies the required classes and methods in the drivers, i.e., the handoff from JDK to the driver.

To get started, the line from JdbcCheckup.java:

...DriverManager.getConnection(...)

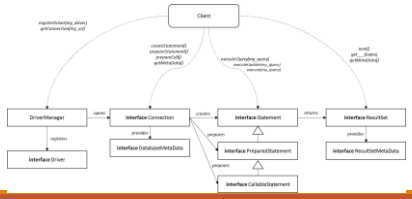
can compile because the class DriverManager, with static method getConnection, is in the JDK.

The code that executes at runtime for this call is initially code from the JDK, but it quickly calls into the driver in the jar file. See the JDK Javadoc for DriverManager for more info.
- Another API in the JDK, the DataSource interface, with implementation in the driver, can also initiate JDBC Connections. We'll use it later.
- Note: see review of Java Interfaces at the end of this slideset

JDBC types and their relationships

JDBC exposes a series of object interfaces through which drivers, connections, SQL statements and results are expressed

- DriverManager is a JDK class that is used to make a singleton object
- Other classes are implemented by the driver itself to provide the required interfaces shown



13

Exceptions in Java code

- Exceptions are widely used in Java code to signal failure of a method to its callers.
- What's special here is that this signaling can jump back to the caller's caller or even the caller's caller's caller, not just to the immediate caller.
- Each exception is thrown by a certain method and caught in another method that has prepared to catch it by providing a catch clause in a try/catch.
- Ex. Method A has try/catch, calls B with no try/catch, calls C which throws
- Execution ends up in A's catch clause. B never sees C return to it, so not all calls return to their caller in Java.
- Working with databases involves lots of error possibilities, so lots of exception handling

14

JDBC: generates SQLExceptions

- **SQLException** (base class of all JDBC exceptions) is a checked exception (not a runtime exception)
- That means we have to program try/catch to handle it or add throws… to the method signature, to pass the problem up to the caller.
- We see that this kind of exception, the checked exception, is tracked aggressively by the compiler to force developer's attention on doing something about the error conditions.
- Another example of a checked exception is IOException.
- (vs. a runtime exception, with base class RuntimeException, that does not require "throws…" for methods that generate it, e.g., NullPointerException. It's assumed that if a null pointer was legitimate for the code, it would be checked for, so this signals a bug, not a runtime error)

15

SQLExceptions (for ex.) and code design

- In properly proceduralized code, there are several levels of methods, and the errors are usually detected at the lowest level, in code that is specialized and doesn't know what to do about the error.
- For example, in our layered system, the presentation layer calls the service layer which calls the DAO layer that does the JDBC, gets in trouble…
- Thus that lowest level method should simply throw when it finds an error, not try to catch and handle it right there in its code.
  - ✓ Alternatively, it could catch it, log it, then rethrow it.
- Either way, it needs a throws clause in the method header. From pizza1:

```
public void insertOrder(PizzaOrder order) throws SQLException
```

16

Top-level code for JDBC: set up Connection, handle various errors

- Need try/catch at top level, say in main, for getting Connection (which can throw) and closing it whether or not the database access "db-stuff" throws. This means using a finally clause:

```
try {
  conn = DriverManager.getConnection(connStr, user, password);
  try {
     do-db-stuff(conn, …);  // can throw SQLException
  } finally {  // do this whether or not the above throws:
     conn.close();  // this can throw!
  }
} catch (SQLException e) {  // including throw from close
  // throw to report problem, or if in main, can exit in error
  // Don't just leave this empty: that's a "code smell"
}
```

17

Doing "db-stuff"

- We definitely want to put the "db stuff" in its own method, as we see in JdbcCheckup.
- There we see another pattern discussed a few slides back: don't catch at lowest level, let the method throw, up to code that knows what to do
- And use finally to close locally created resources

```
void doDbStuff(Connection conn, …) throws SQLException {
  Statement stmt = conn.createStatement();
  try {
    <Db actions>  // these throw SQLException
  } finally {
     stmt.close();
  }
}
```

18

Look at JdbcCheckup code, packages

- We see special handling for dropping a table that might not be there normally, a legitimate case of suppressing an exception.
- Note that this JDBC code is basic: no transactions, no packages.
- JdbcCheckup is just meant to check for proper JDBC installation and connection to a known database.
- Although JdbcCheckup doesn't use packages, our projects will, and the JDBC drivers themselves use packages in their implementation.
  - If a source file has "package com.mycompany.service" at its beginning, this java file is expected to be located in subdirectory com/mycompany/service of the source directory tree.
  - The .class file for it is expected to be in the com/mycompany/service subdirectory of the bin tree.

Back to building JdbcCheckup (here, for Oracle)

```
javac JdbcCheckup.java
java -cp ojdbc6.jar;. JdbcCheckup      (for Windows)
                    ^ (change ';' to ':' for Mac/Linux)
```

- This specifies the classpath as the jar file as the first thing to search for a class, then ., the current directory as the second thing to search.
- Without the -cp spec, the classpath is simply ., but as soon as we put -cp in the command, we need to put . in the list to search the current directory.
  - Without it, java won't find JdbcCheckup.class.

What is a jar file?

From the Java tutorial intro:

The Java™ Archive (JAR) file format enables you to bundle multiple files into a single archive file. Typically a JAR file contains the class files and auxiliary resources associated with applets and applications.

It is the standard way in Java to distribute a set of related classes

It is a zip-compressed directory (with subdirectories)

Example: the JDBC driver of a certain database vendor, like ojdbc6.jar.

The jar command lets us examine and build jar files:

"jar tf ojdbc6.jar" shows all the class names and other files in the Oracle driver software, a huge list with 1670 lines

"jar xf ojdbc6.jar" expands all the files into a filesystem tree

"jar cf myprog.jar ." packs up everything below the current directory into a jar file

Looking in a jar file using the jar command

A jar file represents a tree of files, compressed by zip. To see the tree:

```
jar tf ojdbc6.jar
META-INF/
META-INF/MANIFEST.MF
oracle/
oracle/net/
oracle/net/ano/
oracle/net/ano/Ano.class    ← class Ano, in package oracle.net.ano
oracle/net/ano/AnoComm.class
oracle/net/ano/AnoNetInputStream.class
oracle/net/ano/AnoNetOutputStream.class
oracle/net/ano/AnoServices.class
...  1670 lines of output          <--- somewhere in here, the top-level driver class
```


Finding the top-level driver

We can't expect to find DriverManager in ojdbc6.jar: that's in the JDK, but the top-level driver probably has Driver in its name, so search the output for "Driver":

```
pe07$ jar tf ojdbc6.jar|grep Driver
META-INF/services/java.sql.Driver  ←metadata related to autoloading
oracle/jdbc/OracleDriver.class      ←This is it: OracleDriver in package oracle.jdbc
oracle/jdbc/driver/OracleDriver$1.class
oracle/jdbc/driver/OracleDriver.class
oracle/jdbc/driver/OracleDriverExtension.class
oracle/jdbc/driver/T2CDriverExtension.class
oracle/jdbc/driver/T4CDriverExtension.class
```


Using a jar file to provide classes to a program (i.e. provide a library)

Where does Java look for classes needed for a program?
- Example: Java sees `Work.process("somearg");` and needs to find the class "Work"

Simple program: Java (or javac) just looks for Work in the current directory "."

Bigger program: Java uses the classpath, a list of directory systems to search
- OK, the simple program uses a default classpath of just ".".
- Java looks for the needed class in each directory system of the classpath in turn
- A directory system may be in the file system or in a jar file

The classpath can be specified on the java and/or javac command line, or by a CLASSPATH environment variable, or config in an IDE.

➢ Maven will manage the classpath for us in our projects, i.e. get eclipse to set up the right classpath for compiles.

Directory setup for projects like pizza1 that use packages

- If a source file has "package com.mycompany.service" at its beginning, this java file is expected to be located in subdirectory com/mycompany/service of the source tree.
- The .class file for it is expected to be in the com/mycompany/service subdirectory of the bin tree.
- The convention of using Internet domain names for packages helps avoid name clashes in package names, so we can use libraries from other sites immediately.
- With Maven, the java files (the sources) are in one filesystem tree with root "src/main/java" relative to the base directory, with directory names specified by the package names.
- Similarly, the .class files are in another directory tree. With Maven, it's hidden in the target directory.

Pizza1 directories/packages

Here's the directory system for the pizza1 sources (using "tree" command in Windows), looking down from the top of the source directory tree, itself at src/main in the project's directory system.

```
C:\cs\cs636\pizza1\src\main\java>tree
Folder PATH listing
Volume serial number is 385D-9017
C:.
└───cs636
    └───pizza
        ├───config      ←for package cs636.pizza.config
        ├───dao         ←for package cs636.pizza.dao, etc.
        ├───domain
        ├───presentation
        └───service
```


Compiling projects like pizza1

- For example, in the service package we have StudentService.java, with package cs636.pizza.service. It will compile to StudentService.class in cs636/pizza/service relative to the classes root directory.
- Compiling a whole tree of Java files as in pizza1 is not easy using just the javac and java commands, as we would expect to do on pe07 where we don't have eclipse working for us.
- We need a tool that works anywhere. The current tools for this are
  - ant (the older tool, less used for new projects),
  - maven (we'll use),
  - gradle (a competitor, but considered harder to learn)
- For a three-way comparison, see https://www.baeldung.com/ant-maven-gradle

Using Maven and avoiding "jar hell"

Maven can help with dependency management, a big problem for older projects.

See https://dzone.com/articles/what-is-jar-hell: "a jar cannot express which other jars it depends on in a way that the jvm will understand. an external entity is required to identify and fulfill the dependencies. developers would have to do this manually by reading the documentation, finding the correct projects, downloading the jars and adding them to the project" (also see pic of hell)

Scenario: a project uses multiple libraries, each in a jar file. Library A requires version 8.02 or later of Library B.

We get it working. Then library A comes out with a new version requiring version 8.30 of library B. We upgrade library A and our project fails to execute.

With just 2 libraries, we find the culprit soon. But many projects use 100s of libraries…

The good news is that both maven and gradle take on this responsibility. But not ant.

Maven and dependency management

Maven runs a repository of libraries (used also by gradle)

Each library has a pom.xml expressing its direct dependencies

So when you get a library from the repository, you are also getting its dependency requirements

Your own project has a pom.xml describing its dependencies.

Maven will download what it needs, recursively.
- You can examine your project's "effective pom" right in eclipse
- There still can be failing cases, but they are much rarer

Maven runs a local repository on your system, handling all your projects on that system.
- So it's relatively cheap to have several working versions of a project.

For more info, see introduction-to-dependency-mechanism

XML Basics, for maven's pom.xml

Good XML tutorial:
https://www.tutorialspoint.com/xml/xml_quick_guide.htm

A message in XML: an XML document:

```
<?xml version="1.0"?>          Official start of XML doc.
<message>
    <to> you </you>
    <from> me </from>
    <subject> XML looks like this </subject>
</message>
```

XML holding multiple messages

```
<?xml version="1.0"?>
<messages>
   <message>
      <to> you </to>
      <from> me </from>
      <subject>XML looks like this </subject>
   </message>
   <message>
      <to> you </to>
      <from> me </from>
      <subject> XML again </subject>
   </message>
</messages>
```

31

Attributes: name = "value" in start tag.

We can have multiple child elements inside an element, like <point> here, but only one value for attribute "to".

```
<message to="you" from="me">     ←start tag
     <subject> XML Properties </subject>
     <point> self-describing </point>
     <point> standardized </point>
</message>              ← end tag
```

32

Comments, like HTML:

<!--whatever -->

Next time: pom.xml for Maven

Start with:

http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

33

Review of Java Interfaces for expressing APIs

Java provides **interfaces** as a way to specify the methods (and constants) for an API, separately from classes used to implement the API, i.e., provide code for it.

geeksforgeeks intro:

A Vehicle can be a car or a bike. Here is its interface:

```
interface Vehicle {
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
```

This means the API has those methods. To use the API, we need code for it, i.e., an implementation—

```
// implementation of Vehicle, for bicycles:
class Bicycle implements Vehicle{ … }  //code for API
```

Now we can use Vehicle and Bicycle as two related types.

34

Java Interfaces for expressing APIs

geeksforgeeks intro: A Vehicle can be a car or a bike.

```
interface Vehicle {
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
// implementation of Vehicle, for bicycles:
class Bicycle implements Vehicle{ … }  //code for API
```

But note that we can "new" only Bicycle, not Vehicle:

```
// creating an instance of Bicycle
// doing some operations
Bicycle bicycle = new Bicycle();
bicycle.changeGear(2);
bicycle.speedUp(3);
bicycle.applyBrakes(1);
```

But this code doesn't use Vehicle at all…

35

Java Interfaces for expressing APIs

```
interface Vehicle {
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
// implementation of Vehicle, for bicycles:
class Bicycle implements Vehicle{ … }  //code for API
```

Now we can use Vehicle and Bicycle as two related Java types.

```
// create an instance of Bicycle
// type a variable as Vehicle, do some operations
Vehicle bicycle = new Bicycle();
bicycle.changeGear(2); // call the API via interface
bicycle.speedUp(3);
bicycle.applyBrakes(1);
```

We see that we can call the interface methods using a variable of the interface type. Those calls execute code in Bicycle.java.

36

Java Interfaces for expressing APIs

```
interface Vehicle {
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
```

// Implementation of Vehicle, for bicycles and racecars:

```
class Bicycle implements Vehicle{ … } // code for Bicycle
class RaceCar implements Vehicle{ … } // code for RaceCar
```

Set up Vehicle variables for a Bicycle and a RaceCar--

```
Vehicle v = null;  // v not yet assoc. with an object
v = new Bicycle(); // now v refs a Bicycle object
v.changeGear(2);   // executes Bicyle.changeGear
v.speedUp(3);
v = new RaceCar(); // switch v to ref a RaceCar object
v.speedup(100);    // executes RaceCar.speedup
```

These two Vehicle objects could be in an array of Vehicle

37