Intro to our Projects and JDBC (review)

The Pizza Project: the provided example

See Pizza Project Description Pizza shop for dormitory: students can order pizza, choosing toppings and size. Also admin actions like adding a topping.

Three versions we'll study:

pizza1: client-server/JDBC <---now available, linked to class web page!

pizza2: client-server with transactions and JPA (which itself uses JDBC), using Spring Boot

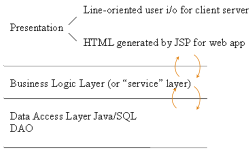
pizza3: web app with JSP and Spring Boot, using JDBC directly again

Music Project: for you to do

Music Website for a Band: music1, (music2), music3. (music2 not actually implemented)

- The band has multiple CDs to sell, so a CD is a Product. Each CD has multiple Tracks, each with a song
- To track visitors, user registration is required before track samples can be played (downloaded) for free.
- CDs can be bought, and this is handled by the usual idea of a "shoppng cart", Cart for short, which can hold one or more CDs.
- A visitor does not need to be registered to have a Cart, only to "check out", that is, actually buy the CDs, or download samples.
- **For images of pages of the website, see Chap. 22 of Murach.**

The Layers again…



- These are "call-down layers": code in presentation calls methods in service code, and then they call methods in the DAO. That call returns from the DAO to the service layer, and then returns to the presentation code.
- This important picture corresponds to the figure on p. 17 of Murach, and our icon.

Domain Objects

- ✓The domain objects hold and carry the data that describes what the application is doing.
- Thus a pizza stop app has objects describing pizza orders, toppings, etc.
- A music store has objects describing guitars, drums, etc.
- (These are our two major examples in this course)
- One PizzaOrder object may reference several Topping objects, making up a little object graph.
- Domain Objects are Java objects, so of course have Java classes to define them.
- We put these classes together in one Java package named domain

How can a PizzaOrder object reference multiple PizzaTopping objects?

We have a field (instance variable) of PizzaOrder of type Set<PizzaTopping> or List<PizzaTopping>.

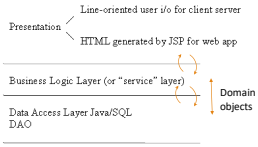
The actual objects for this Set or List (say a HashSet object or LinkedList object) contain multiple references to the associated PizzaTopping objects.

See PizzaOrder.java and PizzaTopping.java in the now available pizza1 project (zip linked to class web page).

Domain Objects: their life cyles

- A PizzaOrder object might be created in the DAO layer from database data, then returned as a method return value to the service layer
- Or it might be created in the service layer for a brand new order, and sent down to the DAO layer to be sent to the database. In this case it will be a method argument value.
- So we see that domain objects can cross layer boundaries.
- But not all boundaries: for reasons we will study, it is not good practice to let them enter the presentation layer.
- We can add them to the layers diagram…

The Layers, showing domain objects



- Domain objects are designed for describing the data that the service layer needs to work with to do the actions of the application.
- The DAO layer, being the servant of the service layer, accepts them to drive inserts/updates to the database, or creates them from database data as needed by the service layer.

Idea of "impedence mismatch" between objects and database tables

- In Java, we use graphs of objects. In the database, we use tables, a different way to hold data. Different enough to be an impediment sometimes, hence the "mismatch."
- How can we hold object data in the database?
- Simple case: no problem. For simple Employee objects (name, age, address, …) and employees table, each Java object has one row in the table.
- More complicated case: A Book has multiple Chapters, say a List<Chapter>. We need a book table and a chapter table, with columns name, bookid, pageno, etc. Bookid should be a foreign key to id in the book table (or name it bookid in both tables).
- Our case: a PizzaOrder object has a Set<PizzaTopping> to describe its toppings. One pizza can have many toppings. Also, one topping can be used in several different pizzas. What database tables are needed?

Database modeling for pizza orders

- Since one pizza can have multiple toppings, and a topping can be used in multiple pizzas, this could be handled is an N-N relationship, and that way would need a relationship table in the database as well.
- However, the toppings are also managed on their own in this app. You can add a topping so that future pizzas can use that new topping, and delete old toppings due to lack of supplies or demand.
- So it turns out there are really two concepts of "topping" here: a topping on the menu, which can be deleted, and a topping serving already in use for a pizza order, and no longer deletable.

The pizza_toppings and menu_toppings tables

So we will hava a menu_toppings table to list the toppings on the menu.

And a pizza_toppings table to express what toppings are on a particular pizza order. Each pizza order can have multiple toppings (or topping servings if you prefer).

Note that table names are in all-lowercase with underscores separating words: that will be our convention, but it's also common in practice. Table names are caseless under standard SQL. MySql supports case in table names, at least on Linux, but for portability its best to not depend on that.

This allows a pizza order in the database to own its pizza toppings for its own lifetime, and avoids the N-N relationship in the model. There is a one-to-many relationship from pizza_orders to pizza_toppings.

The DAO layer for pizza orders

- For this situation, the DAO layer is expected to be able to query the database for all the information needed to construct a certain PizzaOrder object.
- It has to query the database tables using JDBC.
- In the other direction, the DAO is passed a PizzaOrder object and told to save it in the database. The DAO has to do inserts on the appropriate tables. We'll look at the details later.
- Obviously we need to look at JDBC…

JDBC

- Portable across DB's
- i.e., JDBC has a portable API, specified in the JDK
- Covers up difference between DB's, Why necessary?

Note that SQL92 Entry Level is claimed by all major DB's

- Standard covers tables, view, select, insert, delete, etc.
- But not users, catalog tables

We will use Oracle, MySQL, and H2 (embedded database)

Compared to PHP: PHP has a portable API for database access, but no good implementation of it for Oracle

Oracle Setup

- One database at a site (on dbs3.cs.umb.edu).
- Each user gets a schema (a container/namespace that holds tables, etc.), separate from other users.
- User eoneil can have an orders table

  eoneil.orders long-form table name including schema name

  schema table
- and also user joe can have an orders table

  joe.orders
- User eoneil can refer to this table as simply "orders", and similarly joe can too, and they are each accessing different tables.

MySql

- no automatic schema per user (this is not required by SQL standard)
- ...so I'll set up a database for each user
  - user joe would get database joedb
  - you get your own db!
  - our mysql server is on pe07.cs.umb.edu
- We'll be able to access mysql or Oracle with JDBC from anywhere on the Internet
- For this, we'll need to use tunnels through the cs.umb.edu firewall. See DatabaseSetup.

H2

- Not for permanent data, just holds data in memory (as we use it), answers SQL from it, or in a file
- Great for testing: very fast: runs in JVM of our app
- No permanent server of its own running, like we have for Oracle and MySql. Instead, runs in a thread of our app process.
- Two ways to run it, using a file, or using memory, used only for JUnit testing.
- Although it supports multiple users, we'll just use one user "test", password "", for the current user.

In all three databases, once you're logged in, you have access to a similar environment: can create tables, insert rows, use queries, etc.

- We can use the same createdb.sql to create a set of tables for a project for all three.

Using MySQL: Murach Intro (Chap. 11)

- Note the discussion on pg. 349 of how version 5.5 of MySQL has overcome previous deficiencies to become a "real" database, offering relational integrity (FKs that work) and transactions by default. We are using version 5.7 on pe07.
- MySQL still has some problems with case-sensitive identifiers (table names, column names, etc.). By the SQL standard, all identifiers are caseless, but by default on Linux and other UNIX (but not MacOSX) systems, MySQL table and database names are case-sensitive. I have overridden this bad behavior on pe07 by setting lower_case_names = 1 in /etc/mysql/my.cnf. If you have a Linux development system with your own mysql installed (not required for this course), set this in mysql/bin/my.ini.
- Also, by default setup (not overridden), Mysql uses caseless compare in selecting data. For example select name from users where name = 'bob' will retrieve 'Bob' too.

Chap. 11 Notes, continued.

Section on "Two ways to interact":

Command line tool: this requires a login on the server, i.e., topcat.cs.umb.edu, which you have or will soon have. We'll use this approach there.

MySQL GUI tool: this can be used from another system, but it works for MySQL only. We are planning to use eclipse database support, which works with all our databases. See hw1.

p. 362 How to create a database—I'll do this for you. UNIX user joe will get database joedb.

p. 364: How to create a table: this shows useful SQL examples. Continue to the end of the chapter looking at SQL examples.

Chap. 12 JDBC Notes

- read this to p. 387 for now.
- Database drivers: Mysql, Oracle and H2 drivers are in $cs636/jdbc, where $cs636 stands for www.cs.umb.edu/cs636, the class's web home page.
- p. 381: how to connect. We use the newer method for JdbcCheckup.
- p. 383: how to do a SQL select statement, queries fill a ResultSet object with rows.
- Note that from these examples it looks like mysql is case-sensitive for table and column names, whereas standard SQL has caseless identifiers. However, with the proper setup, you can use any case with mysql, just like Oracle and other DBs following the SQL standard. Our installation of mysql on pe07 is set to use caseless identifiers, but still uses caseless compare on string column values.

Chap. 12 Notes, continued

Pg. 383 alternative ways we can do these queries:

select userid from user where email = 'jsmith@gmail.com';

Select Userid from User whERE eMAIL = 'jsmith@gmail.com';

p. 385: how to get values out of rows in the ResultSet

Note: This coverage ignores the exception handling needed to actually code with JDBC. Need full examples, such as JdbcCheckup.java.

Also uses this code uses double for money variable, not great.

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!)

Using JDBC: Java Tutorial

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

- BLOBs
- Prepared statements
- 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

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 16700 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

Using a jar file to provide classes to a program (i.e. 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 it in the current directory "."

Bigger program: Java uses the classpath, a list of directories to search

- OK, the simple program uses a default classpath of just ".".
- Java looks for the needed class in each directory of the classpath in turn

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

We'll look at the command line method here.

Building JDBCCheckup.java, revisited

First compile JdbcCheckup.java. No special classpath needs to be set up for this:

```
javac JdbcCheckup.java
```

Now we have JdbcCheckup.class in the current dir.

Use java to run it. We need to add the driver jar file to the classpath:

```
java -classpath driver.jar:. JdbcCheckup   (UNIX/Linux)
```

(change ':' to ';' for Windows)

This classpath has two directories, driver.jar and ".".

The whole jar file is treated like a directory: it is a compressed directory (with subdirectories). Java decompresses it on the fly.

Driver.jar: ojdbc6.jar or mysql-connector-java-xxx-bin.jar or h2.jar

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

```
pe07$ java -classpath 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 its 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.
```

In JdbcCheckup.java

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".

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

Then get a connection from the driver:

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

JDBCCheckup.java, continued

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 tutorial: http://www.tutorialspoint.com/java/java_exceptions.htm

Another tutorial: Java Tutorial: Exceptions

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