CS209

Computer system design and application

Stéphane Faroult
faroult@sustc.edu.cn

Zhao Yao zhaoy6@sustc.edu.cn

There is another way for getting JDBC connections, which is using a DataSource object. It abstracts the database (it uses a directory saying what is the servername and database name and database type for a given "service"), and in a big company it allows moving databases between servers transparently for users.

Database Access

using a DataSource object

Configured by system administrator

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup(DBName);
Connection con = ds.getConnection(username, password);
```

Switch from development to production

The other JDBC pakage

```
import javax.sql.*;
```

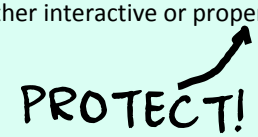
- Data sources
- Connection pools
- Distributed transactions

DataSources and other advanced features used mostly in very big companies (connection pools are mostly useful when you have thousands of users, and distributed transactions are required to coordinate work across several databases) are inside an additional package called javax.sql (regular JDBC is java.sql)

Security Issues

DON'T hardcode username and password

either interactive or property file



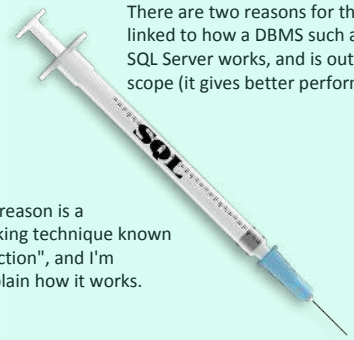
Usernames and passwords should never be hardcoded in programs, because it's a common policy to force passwords to change every 3 or 6 months. Users should be prompted for their credentials or, if the program is run at 2am, read from a file. But then you should make sure that only the file owner can read it!

Security Issues

DON'T append user entry to statements

use PreparedStatements

It's very tempting to use "+" to build an SQL statement. The rule is that whenever there is a constant in the SQL command that was input by a user, it should be passed as a parameter to a PreparedStatement, not simply added.



There are two reasons for this rule. One is linked to how a DBMS such as Oracle or SQL Server works, and is outside our scope (it gives better performance).

The second reason is a famous hacking technique known as "SQL injection", and I'm going to explain how it works.

```
query = "select memberid from members where username = '"
      + enteredUsername
      + "' and password = '" + enteredPassword . "'";
```

```
select memberid from members
where username = 'Mickey'
  and password = '3#(ag89zhost]=89'
```

Most web applications store into a table of members the login name, and the crypted password. When users log in, the table is searched for a row containing their name and the encryption of the password they have entered is compared to the value stored. If the query above returns no rows, the user is rejected, otherwise s/he enters the site. When a developer builds the query as coded above, the type of expected input is what is in red under it.

```
query = "select memberid from members where username = '"
      + enteredUsername
      + "' and password = '" + enteredPassword . "'";
```

```
select memberid from members
where username = 'Pete'
  and password = 'hack' or 'hack'='hack'
```

False True

But if someone enters any name, let's say `Pete`, and something such as **`hack' or 'hack'='hack`** as password, when this is added to the statement it changes it. Because of the priorities rules between and and or, it will become a condition that is always false or a condition that is always true, and the query will always return rows, thus granting access to the site.

Two solutions

- Escape quotes in input
- Use variables

To protect yourself against such an attack, you can apply a function that escapes quotes to make sure that the quotes are understood as belonging to the constant (not as belonging to the SQL command), or to use variables in a `PreparedStatement`. The second is better because as said above it may make a performance difference with some database management systems.

IMPORTANT

MINIMIZE database accesses

DON'T do in Java what SQL does better

Remember that if Object Oriented programming is mostly about exchanging messages between objects, talking to a remote database is completely different, and remember that database management systems process data better and faster than you can do – they were built for that. It's all the more important that most Java developers have it completely wrong, partly because of available tools.

Other Performance Related Issues

- Transactions
- Batching

There are other important points. To simplify, by default every time you change something in the database JDBC requires the DBMS to write something to a file to make sure it's not lost (it's called "autocommit"). It's very bad for massive operations, and it's very bad for linked operations that should all succeed or all fail. You can also ask the database to perform several operations in one go (this is called "batching"), or adjust how much data is retrieved from the database in one return trip. All this is quite important when an application is heavily used.

JPA

Java Persistence API

Because Java developers are assumed, the poor things, to be unable to code SQL, tools have been developed to allow them to focus on "business functions", and here things become quite worrisome ...
JDBC allows you to write some good applications; you cannot say that it's very difficult to use, but it requires you to know some SQL, a language far less easy than it looks. Like everything, it takes time to learn. Tools have been designed to let anybody write database applications. Unfortunately anybody wrote database applications, and the result was often not pretty.

Persistence:

an important part of a software product

Time spent searching/storing objects must be as short as possible

Focus on business functions

Everybody acknowledges that persistence is very important, and that you cannot write a serious enterprise application without a database as backend. Given that many developers were writing terrible queries that were taking hours, the idea was to say "we'll generate database accesses for you and developers will no longer have to worry about the hard technical stuff but will able to focus on business functions". Which usually translates as "we'll be able to hire cheap beginners to do the job instead of expensive experienced people" (but you won't hear it).

Enterprise Java Bean (EJB)

IBM 1997

Sun microsystems Java JEE

INCREA$E PRODUCTIVITY !
FOCU$ ON BU$INE$$$!

Manage the business layer
(including persistence)

It all started with IBM, and it was incorporated later by Sun (then owner of Java) into an "Enterprise Edition" of Java for big companies.

How can we persist objects in a more advanced way than serialization?

The big idea of the 1990s

Object-Oriented databases FAIL (OODBMS)

The big idea of the 1990s was the "Object Oriented Database", a kind of super-serialization. It never took off. Companies didn't want to take the risk of moving their data from relational databases that had been around for 10 to 15 years to untried products.

Learning to live with SQL databases

ORACLE

Microsoft SQL Server

MySQL

Big database vendors paid lip service to object orientation, and Java developers had to look for bridges to the relational world.

New solutions?

Object Relational Mapping

One of the early products was Hibernate, which maps Java objects to database tables.

HIBERNATE

Works well in simple cases.

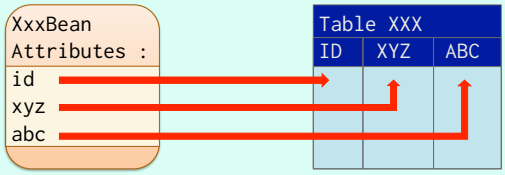
Started early 2000s

Simple to use

Compatible with multiple database systems

Object Relational Mapping

The idea is to have objects that correspond to tables, or tables that correspond to objects.



Object Relational Mapping

JPA (2006) inspired by Hibernate – JEE or JSE

Hibernate implements JPA

Relational Databases

As any successful idea, it turned into a kind of standard for accessing relational databases without writing dirty SQL.

JDO (2002) – relic from OODBMS – works with NoSQL

Object Relational Mapping

Layer above JDBC

No more SQL, no more ResultSet

Improved connection

Caching

PRODUCTIVITY!

Object Relational Mapping

No more SQL, no more ResultSet

SQL can be very powerful and efficient

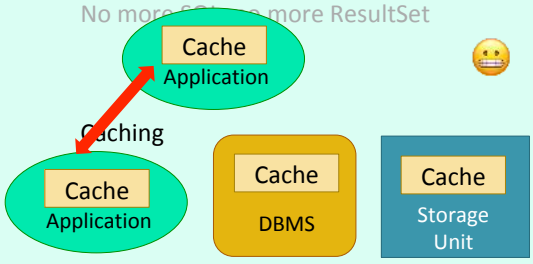
Caching

In practice, you do very efficiently in SQL operations that are very hard in Java (I'm not sure that iterating over Collections is a marked improvement over iterating over a ResultSet). In practice, people often by-pass ORM for what is critical.

Object Relational Mapping

Caching (comes from a French word meaning "to hide") can also be a problem. When you change data, you have to make sure that everybody is in synch and it can be very hard.

No more SQL, no more ResultSet



Object Relational Mapping

Entities

entity = lightweight persistence domain object

Java class that typically represents a table in a relational database, instances correspond to rows

They didn't understand the relational model!!

ORMs start with the premise "one class = one table, one object = one row" that is completely wrong. The relational logic is different.

Object Relational Mapping
Entities

Annotated with the `javax.persistence.Entity` annotation

`public` or `protected`, no-argument constructor

The class must not be declared `final`

No methods or persistent instance variables must be declared `final`

Object Relational Mapping

Each entity must have a unique object identifier (persistent identifier)

```
@Entity
public class Student {
   @Id private int studentId;
   private String name;
   private Date   birthDate;

   public int getId() { return StudentId; }
   public void setId(int id) { this.studentId = id; }
   . . .
}
```

Primary Key

Object Relational Mapping

Simple id – single field/property

```
@Id int studentId;
```

Compound id – multiple fields/properties

```
@Id String name;
@Id Date   birthDate;
@Id String birthPlace;
```

A key, which uniquely identifies an object is a row, can be one value or a combination of several values (each part may appear several times, but a combination appears only once)

Object Relational Mapping

Assumes that names in the class and names in the database match.

If not it can be set in annotations:

```
@Entity(name = "SUSTC_STUDENTS")
public class Student{ …… }
@Id   @Column(name = "STUDENT_ID", nullable = false)
private String studentId;

@Column(name = "FULL_NAME" nullable = true, length = 30)
private String name;
```

I won't do it in examples that follow to keep everything short …

Object Relational Mapping
Relationships

@OneToOne ← Inheritance (or design error)

Relationships describe how many rows in one table match how many rows in another table. OneToOne is typical inheritance. To use an example I have used several times, you can have a table of GeoPoints with a column that says what type of points this is, and a table of Cities with the reference to the GeoPoint plus City information. One row in one table matches one row in the other table, and only one.

Object Relational Mapping
Relationships

@OneToOne
@OneToMany
@ManyToOne

Those two mirror each other. You'll say that one section of a course is taught by one instructor, but that one instructor can teach many sections (example coming soon)

Object Relational Mapping
Relationships

@OneToOne
@OneToMany
@ManyToOne
@ManyToMany

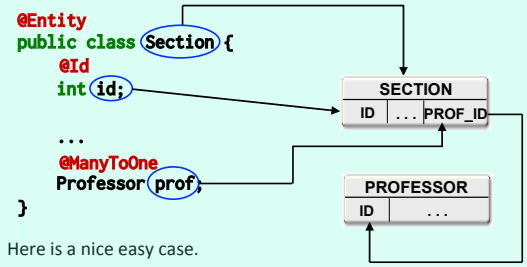
As a student, you take many courses and there are many students enrolled in each course.

Object Relational Mapping
Relation Attributes

```
@ManyToMany(
   cascade = {CascadeType.PERSIST, CascadeType.MERGE},
   fetch = FetchType.EAGER)
```

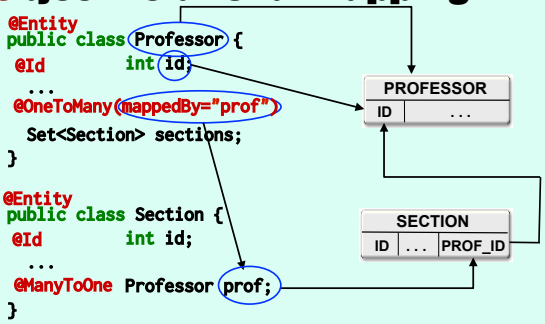
Relation attributes control what happens when you save and query objects related to each other. A tactic frequently used is a "lazy fetch", in which you fetch related data only when needed. Popular and bad for databases, because it generates zillions of queries to fetch what you might have returned in one operation (at the expense of using more memory)

Object Relational Mapping



Here is a nice easy case.

Object Relational Mapping



A bit uglier.

Object Relational Mapping

```
@Entity
public class Student {
  ...
  @JoinTable(
       name="STUDENT_SECTION",
    joinColumns=@JoinColumn(
       name="STUDENT_ID",referencedColumnName="STUDENT_ID"),
    inverseJoinColumns=@JoinColumn(
       name="SECTION_ID", referencesColumnName="SECTION_ID")
  Collection<Section> sections;
}
@Entity
public class Section {
  ...
  @ManyToMany(mappedBy="sections")
  Collection<Student> students;
}
```

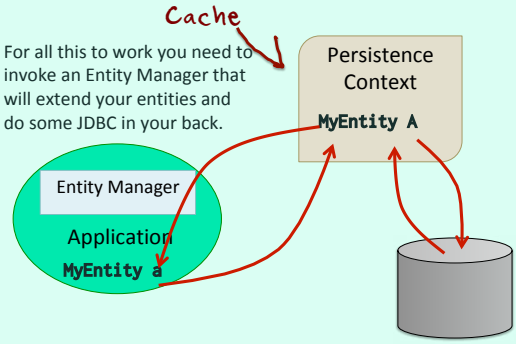
Frankly, between this and the equivalent SQL join, I prefer SQL any time ... This generates a very BASIC SQL query.

Object Relational Mapping

Supports inheritance and polymorphism

I have mentioned earlier (@OneToOne) how you implement inheritance in a relational database. Obviously it's not too hard to translate to the Java world.

Object Relational Mapping



Object Relational Mapping

- EntityManager API operations:
  - `persist()` - Insert the state of an entity into the db
  - `remove()` - Delete the entity state from the db
  - `refresh()` - Reload the entity state from the db
  - `merge()` - Synchronize the state of detached entity with the pc
  - `find()` - Execute a simple PK query
  - `createQuery()` - Create query instance using dynamic JP QL
  - `createNamedQuery()` - Create instance for a predefined query
  - `createNativeQuery()` - Create instance for an SQL query
  - `contains()` - Determine if entity is managed by pc
  - `flush()` - Force synchronization of pc to database

No SQL, no JDBC! Happy?

Object Relational Mapping

The sad truth is that as the set operations of SQL and the Object-by-Object approach of Object Oriented programming are two very different ways of solving problems, which both have their strengths and weaknesses, trying to match one to the other only works well in very simple cases. The incompability is known as "impedance mismatch'. All the Hibernate developers I have known had to write SQL queries (run through Hibernate).

Works well in simple cases.

Drives people crazy in complicated cases.

Flickr : Jeff Sandquist

Object Relational Mapping

Create, **r**ead, **u**pdate and **d**elete

Basic assumption

What you see is what is stored

The basic ORM assumption that tables match objects is usually wrong. You need to be cleverer, usually. One way is to create "views" in the database, which are stored queries that may return something looking like your objects.

Way out: views in the database

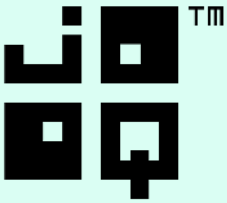
but update may be tricky

Object Relational Mapping

Database folks **hate** Object-Relational Mapping

Object-centric

Some ORM generate SQL code to create tables that match your objects. This can only generate a bad database, because objects should suit one application while the database should accommodate all cases. It's hard to design a good database. If you modify your objects, it may become difficult to change the database later; and another application may have very different requirements. ORMs can be useful, but don't expect miracles.



A more recent product, JOOQ, takes the opposite approach. It's not sure that it gives better results, for the reverse reasons. Instead of having bad tables, you may get bad objects.

"Database First"

Creates classes from DBMS metadata

Standard SQL? They claim to generate "Standard SQL" usable everywhere, which often means underusing a particular DBMS.

Writing code is one thing ...

PROJECTS REQUIRE MORE

Documentation

Organizing Code/Build/Integration

Testing

Deployment

In general, and with any language, a project is more than writing one program and there is a whole eco-system around it. This will be a brief overview, mostly to provide ideas and topics to search on the web.

Everybody wants documentation.

Everybody hates writing it.

You have already been told about **javadoc** and asked to use it for your assignments, no need to say more about it. Great tool.

Organizing Code

The single assignment23.java file may be good enough fo the university, but not for a company project where projects are very big things that require a lot of organization and methods.

Side note: if you are interested in software projects, there is a famous book called "The Mythical Man-Month" by Fred Brooks. It's based on the experience on a project 50 years ago but, except for one outdated chapter, it's still quite relevant today.

MyCode.java

```
public class MyCode {
```

Lots of methods

```
static void main() {
}
```

The "one big file with lots of methods" approach cannot work with several people working on the same project. You have to break everything into smaller classes with their own methods, calling methods from other classes.

MyClass1.java

MyClass2.java

MyClass3.java

MyCode.java

```
public class MyCode {
 MyClass1 mc1;
 ...
 static void main() {
 }
}
```

No main() function

public class

Only one class contains a public static void main() methods. Other classes are public classes with constructors and methods, but no main().

It's interesting to see what javac generates when applied to a single .java file. It won't always be a single .class file.

MyCode.java

javac

```
class OtherClass {
}

public class MyCode {
 MyClass1 mc;
 ...
 class InternalClass {
  ...
 }

 static void main() {
 }
}
```

OtherClass$1.class

If you are using another class, the JVM loader expects to find the corresponding .class somewhere

MyClass1.class?

MyCode$InternalClass.class

MyCode.class

Classes outside the main one, as well as internal classes, will be turned into separate .class files. In fact (surprise) you have one .class per class.

The loader will look in what is supplied with the Java runtime environment, by default it will look into the current directory or any directory or .jar file (more about .jar files later) that you specify.

Looks for `.class` files in the current directory (folder)

Or in directories specified in **CLASSPATH** or with `-cp`

directory1:*directory2*:...:...

directory1;*directory2*;...;...

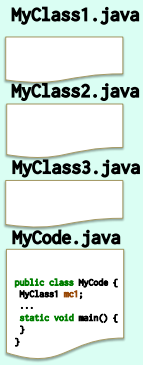
Or .jar files

Special directories:

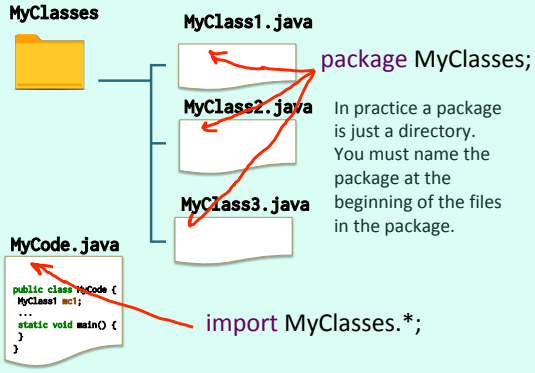
- `.` Current
- `..` Parent

Packages

"Packages", as the name suggests, as for storing related software classes together.



You want to use packages especially when you feel that the classes could be reused in a different software project.



In practice a package is just a directory. You must name the package at the beginning of the files in the package.

Package naming

There is always with naming the problem of name conflict: giving the same name as something that already exists.

reversed web site name

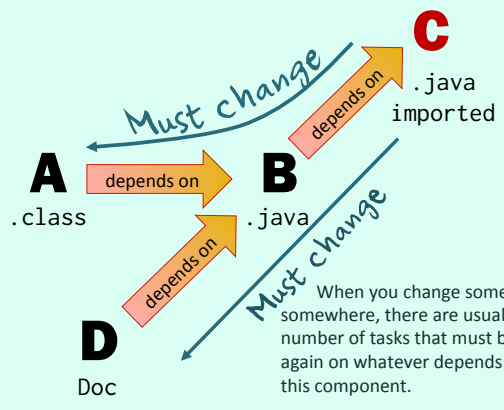
To avoid name conflict, companies build package directories that look like their reversed website name, followed by whatever they want. By giving the full package name, you know exactly what you are talking about.

Once again, running javac isn't the only thing you have to do. Before you compile the code, you must often extract it from a repository (more about this in the next lecture). If javac is successful, you need to generate the doc, we have seen it. You need to run (automated) tests, you need to prepare software for deployment... A lot of small, often boring tasks, depending on each other, that developers have tried to automate as much as possible.

Build tools

Huge project = Many components

Dependencies



When you change something somewhere, there are usually a number of tasks that must be run again on whatever depends on this component.

Based on file modification time

MAKE

The grand-dad of build tools is **make**, still very much in use and very popular among C developers. You can use it with Java as well!

Java-era build tools

But other build tools appeared that were more specifically targeted at Java development. As they were created when XML was all the rage, they usually rely heavily on XML configuration files saying what depends on what and what to do when something changes.

```
<?xml version="1.0"?>
<project name="Hello" default="compile">
    <target name="clean" description="remove intermediate files">
        <delete dir="classes"/>
    </target>
    <target name="clobber" depends="clean" description="remove all artifact files">
        <delete file="hello.jar"/>
    </target>
    <target name="compile" description="compile the Java source code to class files">
        <mkdir dir="classes"/>
        <javac srcdir="." destdir="classes"/>
    </target>
    <target name="jar" depends="compile" description="create a Jar file for the application">
        <jar destfile="hello.jar">
            <fileset dir="classes" includes="**/*.class"/>
            <manifest>
                <attribute name="Main-Class" value="HelloProgram"/>
            </manifest>
        </jar>
    </target>
</project>
```

Example build.xml *coming from wikipedia*

Describes dependencies and tasks:

- Get source code out of source control
- Compile
- Run test
- Copy file
- *and so forth*

All common tasks correspond to an XML tags, with the detail provided as XML attributes.

Another famous Java build tool is Maven, also an Apache project, that we'll see in some detail next time.

maven