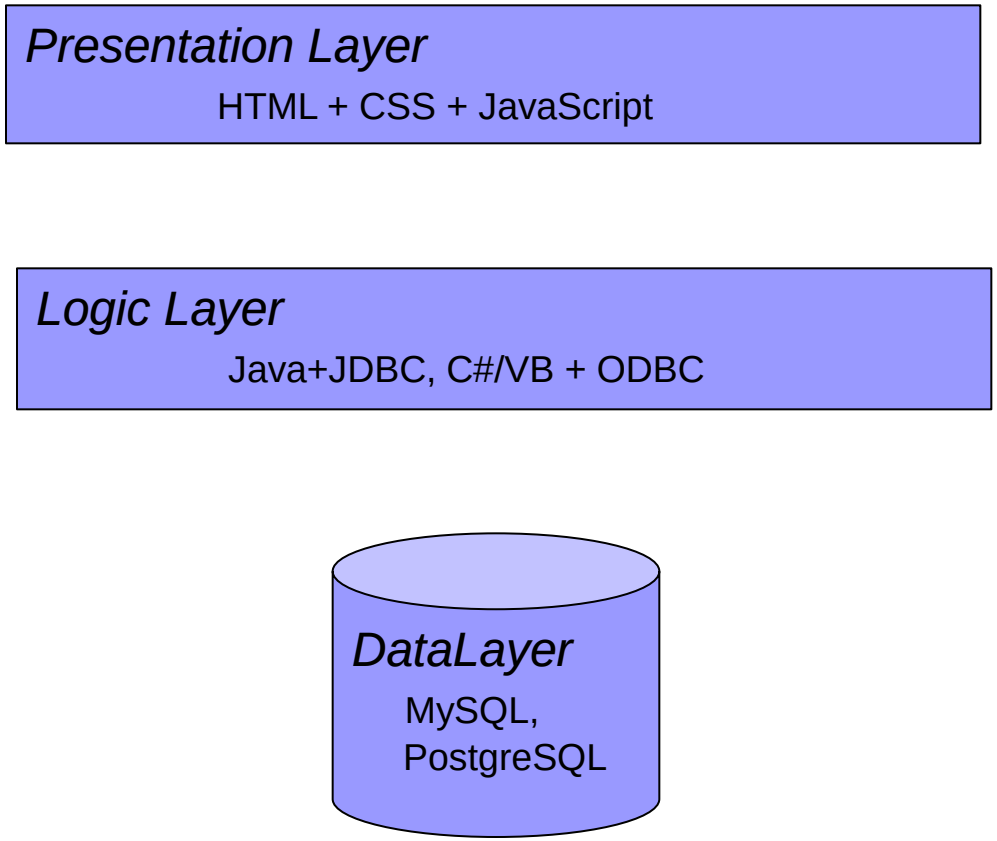
CSC309: Introduction to Web Programming

Lecture 7

Wael Aboulsaadat

3 Tier Architecture



Typical Java/JDBC

```
String  crs_code, semester;
……..…
String query = "SELECT  T.StudId  FROM  Transcript T" +
                        "WHERE  T.CrsCode = ?  AND  T.Semester = ?";

ps.setString(1, crs_code);     // set value of first in parameter
ps.setString(2, semester);     // set value of second in parameter

ResultSet  res = ps.executeQuery ( );
                                // Creates a result set object, res
                                // Executes the query
                                // Stores the result set produced by execution in res

while  ( res.next ( ) )  {                  // advance the cursor
      j = res.getInt ("StudId");            // fetch output int-value
      …process output value…
}
```

So, what's the problem?

Problems with embedding SQL In Java

- Difficult to maintain
- Intimate relation between 2 different technologies: Relations vs. OOP
  - Programmers ❤ objects and only objects

This is called the Object Relational Mapping Problem or (ORM)

When is it acceptable ?

- Small number of tables…
- Small development team..
- Small website…

Recall: Why is database good?

- A collection of programs that enable:
  - Defining (describing the structure),
  - Populating by data (Constructing),
  - Manipulating (querying, updating),
  - Preserving consistency,
  - Protecting from misuse,
  - Recovering from failure, and
  - Concurrent using

  of a database.

Solutions to ORM?

1) Stored Procedures

Stored procedures or functions

- Stored routines (procedures and functions)
  - A stored procedure is a set of SQL statements that can be stored in the server.
  - With stored procedures, clients don't need to keep reissuing the individual statements. Instead a simple call to the stored procedure will perform the same functions.
  - Stored routines can provide improved performance because less information exchanged between the server and the client.

Create Stored Routines

```
CREATE PROCEDURE sp_name ([proc_parameter[,...]])
    routine_body

CREATE  FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    routine_body
```

Parameters

- Each parameter is an IN parameter by default. You can also explicitly specify a parameter to be OUT or INOUT parameter if you define a PROCEDURE (not FUNCTION).
  - An IN parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns.
  - An OUT parameter passes a value from the procedure back to the caller. Its initial value is NULL within the procedure, and its value is visible to the caller when the procedure returns.
  - An INOUT parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.
- Note
  - Specifying a parameter as IN, OUT, or INOUT is valid only for a PROCEDURE. For FUNCTION, parameters are all IN.

Stored Procedure Example

```
> delimiter //

> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

> delimiter ;

> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

> SELECT @a;
+------+
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

Typical Stored Procedure

```
create or replace function retrievePassword(varchar)
returns varchar as '

declare
 my_email varchar;
 my_password varchar;
 output_error varchar;

begin
 my_email := $1;

 select into output_error "120";

 select into my_password password
 from userinformation
 where email = my_email;

 --return error is given username does not exist
 if(my_password is null) then
 return (output_error);
 end if;
 --else return password for given username
 return (my_password);
end;
'language 'plpgsql';
```

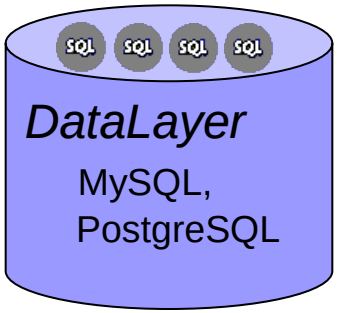
3 Tier Architecture + SP

Presentation Layer

HTML + CSS + JavaScript

Logic Layer

Java+JDBC, C#/VB + ODBC



Typical Java/JDBC and Using Stored Procedure

```
CallableStatement cs;

// Call a function with one IN parameter; the function returns a VARCHAR
 cs = connection.prepareCall("{? = call retrievePassword (?)}");

// Register the type of the return value
 cs.registerOutParameter(1, Types.VARCHAR);

 // Set the value for the IN parameter
  cs.setString(2, "a string");

 // Execute and retrieve the returned value
 cs.execute();
 retValue = cs.getString(1);
```

Typical Java/JDBC and Using Stored Procedure – cont'd

```
CallableStatement cs;
    try {
      // Call a procedure with no parameters
        cs = connection.prepareCall("{call myproc}");
        cs.execute();

      // Call a procedure with one IN parameter
        cs = connection.prepareCall("{call myprocin(?)}");

        // Set the value for the IN parameter
        cs.setString(1, "a string");

        // Execute the stored procedure
        cs.execute();

      // Call a procedure with one OUT parameter
        cs = connection.prepareCall("{call myprocout(?)}");

        // Register the type of the OUT parameter
        cs.registerOutParameter(1, Types.VARCHAR);

        // Execute the stored procedure and retrieve the OUT value
        cs.execute();
        String outParam = cs.getString(1);     // OUT parameter
```

```
      // Call a procedure with one IN/OUT parameter
        cs = connection.prepareCall("{call myprocinout(?)}");

        // Register the type of the IN/OUT parameter
        cs.registerOutParameter(1, Types.VARCHAR);

        // Set the value for the IN/OUT parameter
        cs.setString(1, "a string");

        // Execute the stored procedure and retrieve the IN/OUT
        // value
        cs.execute();
        outParam = cs.getString(1);           // OUT parameter
    } catch (SQLException e) {
    }
```

Pros of Stored Procedures

- Removed SQL from Java code
- DB group develop SQL independently

Cons of Stored Procedures

- Often interpreted, rarely compiled
- Danger of putting logic in database

Have we really solved the problem with SP?

```
CallableStatement cs;

// Call a function with one IN parameter; the function returns a VARCHAR
 cs = connection.prepareCall("{? = call retrievePassword (?)}");

// Register the type of the return value
 cs.registerOutParameter(1, Types.VARCHAR);

 // Set the value for the IN parameter
  cs.setString(2, "a string");

 // Execute and retrieve the returned value
 cs.execute();
 retValue = cs.getString(1);
```

So, what's the problem now ?

Have we really solved the problem with SP?

- Almost, but not yet
  - We will need very simple template engine…
  - Why?

```
CallableStatement cs;

// Call a function with one IN parameter; the function returns a VARCHAR
 cs = connection.prepareCall("{? = call retrievePassword (?)}");

// Register the type of the return value
 cs.registerOutParameter(1, Types.VARCHAR);

 // Set the value for the IN parameter
  cs.setString(2, "a string");

 // Execute and retrieve the returned value
cs.execute();
retValue = cs.getString(1);
```

Solutions to ORM?

1) Stored Procedures

2) Use an ORM library…

- very popular approach: 55 ORM libraries

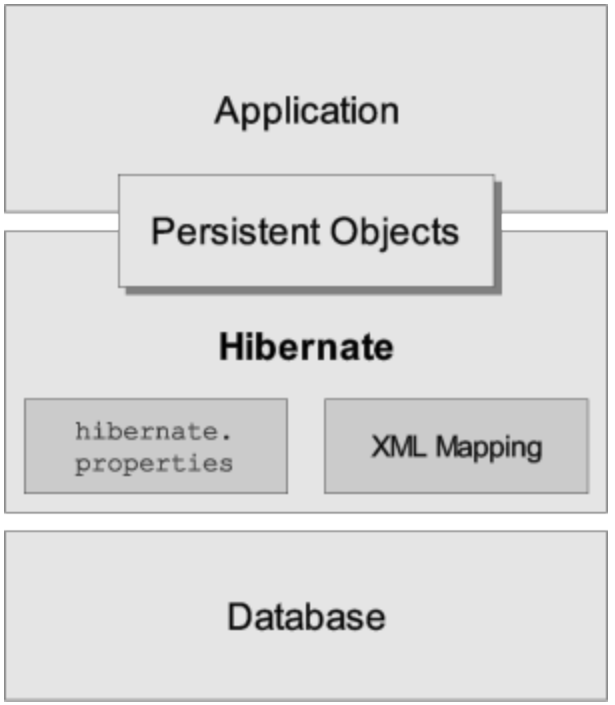
http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software

Hibernate as an example ORM

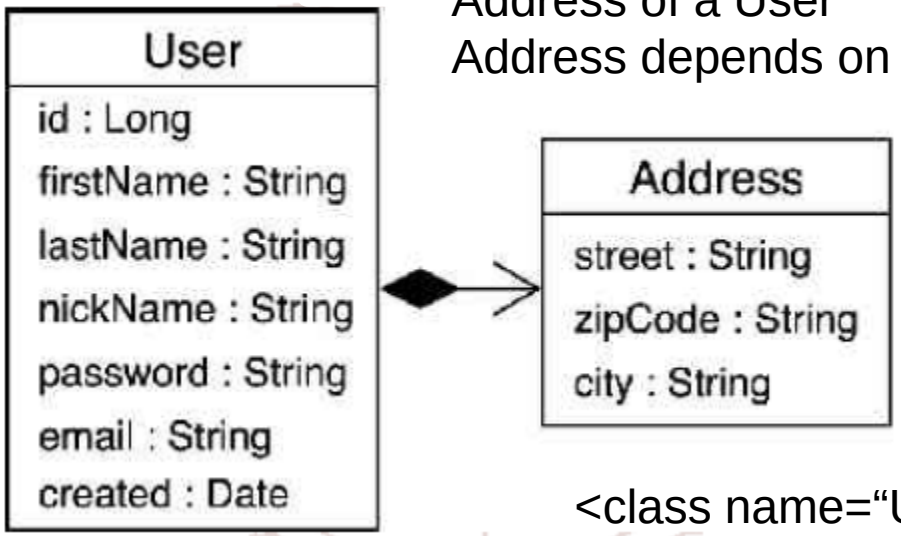
- Provides object/relational persistence
- Query service, the ability to query data persistence using …
  - Native SQL
  - HQL – Hibernate's own portable SQL extension.
- "Hibernate's goal is to relieve the developer from 95 percent of common data persistence related programming tasks, compared to manual coding with SQL and the JDBC API."

Hibernate (cont.)

- High level overview of Hibernate (middleware)
- Hibernate uses the database and configuration properties to provide persistence services (and objects) to the application.



Hibernate examples



Address of a User
Address depends on User

```
<class name="User" table="USER">
…
    <component name="address">
      <property name="street" column="STREET"/>
      <property name="zipCode" column="ZIPCODE"/>
      <property name="city" column="CITY"/>
    </component>
</class>
```

Hibernate examples

```
public void updateItem(AuctionItem item) throws … {
        getSession().update(item);
}
```

```
Session session = sf.openSession();
Transaction tx = session.beginTransaction();
session.update(item);
tx.commit();
session.close();
```

Hibernate examples

```
public class BookDAOHibernateImpl implements BookDAO
{    private HibernateTemplate template;
    public BookDAOHibernateImpl(HibernateTemplate
template){
        this.template = template;
    }
    public Book findBook(String isbn)
    {
        return (Book) hibernateTemplate.get(isbn);
    }
}
```

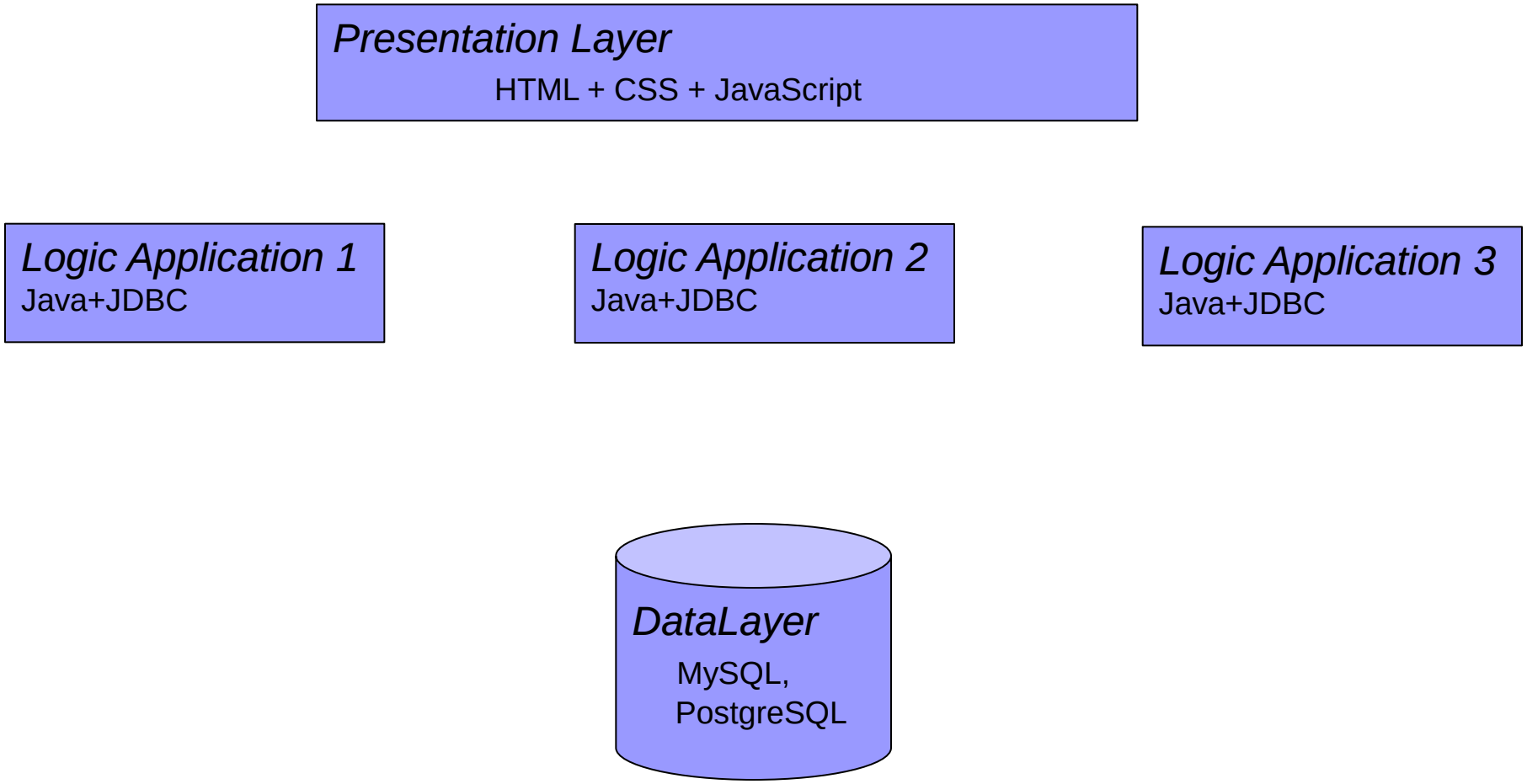
Pros and Cons of Hibernate (ORM libraries)

- SQL is responsibility of developers
- Complex mapping file
- Didn’t introduce new classes

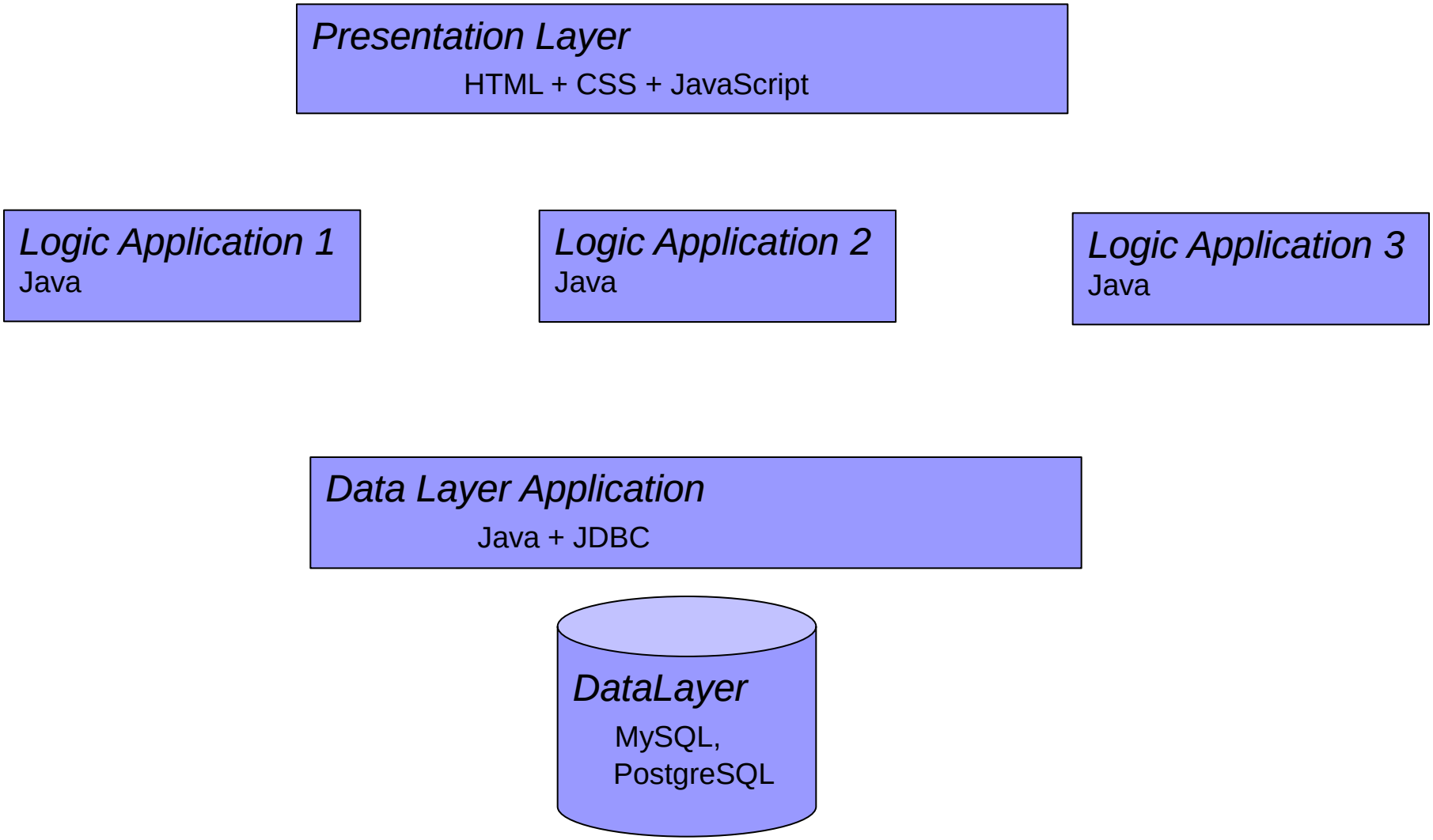
N Tier Architecture

- As a website grow, the application logic grows, what to do?
  - Divide and conquer
  - 1 team per subsystem
  - 1 server per subsystem

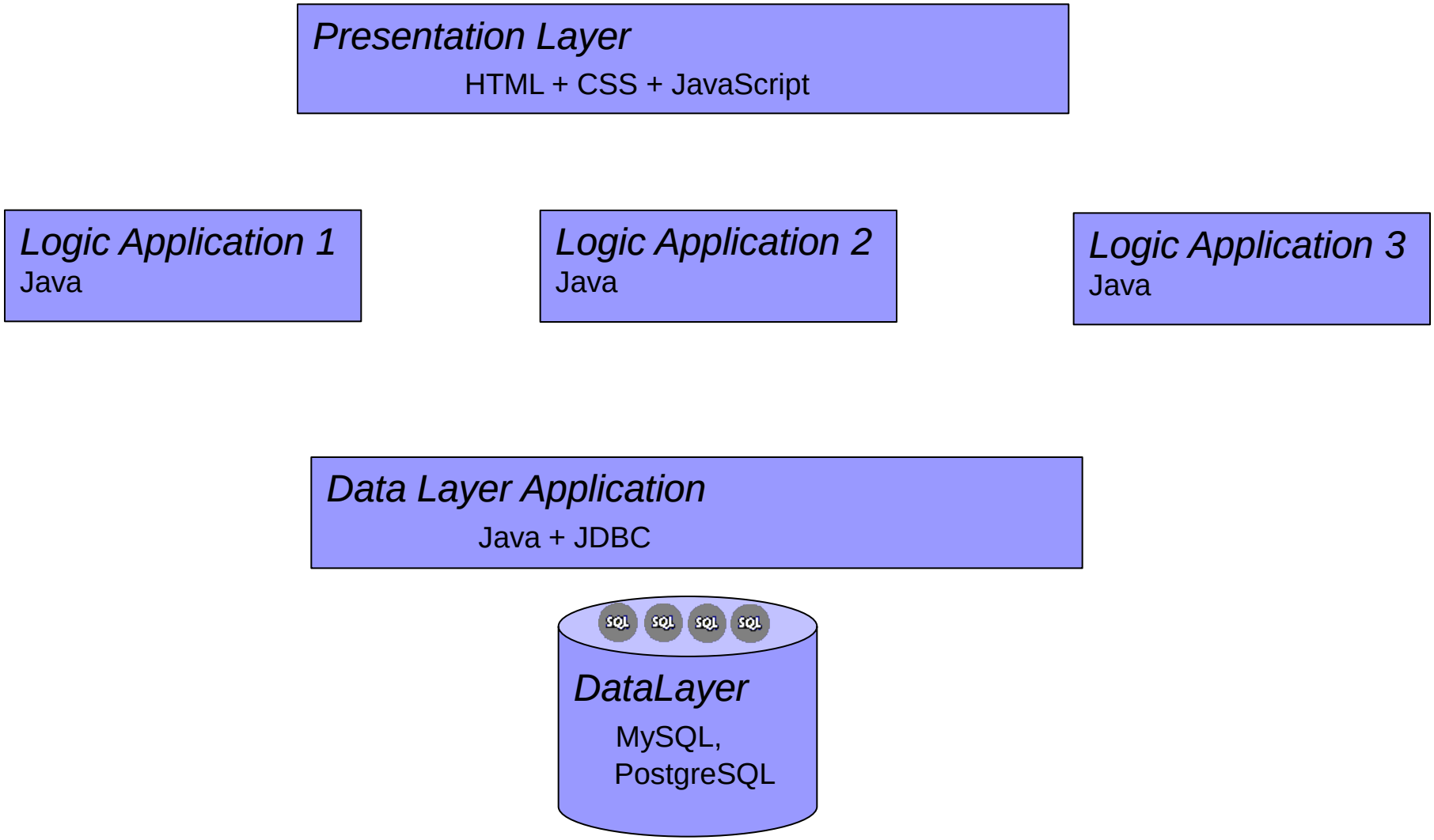
N Tier Architecture



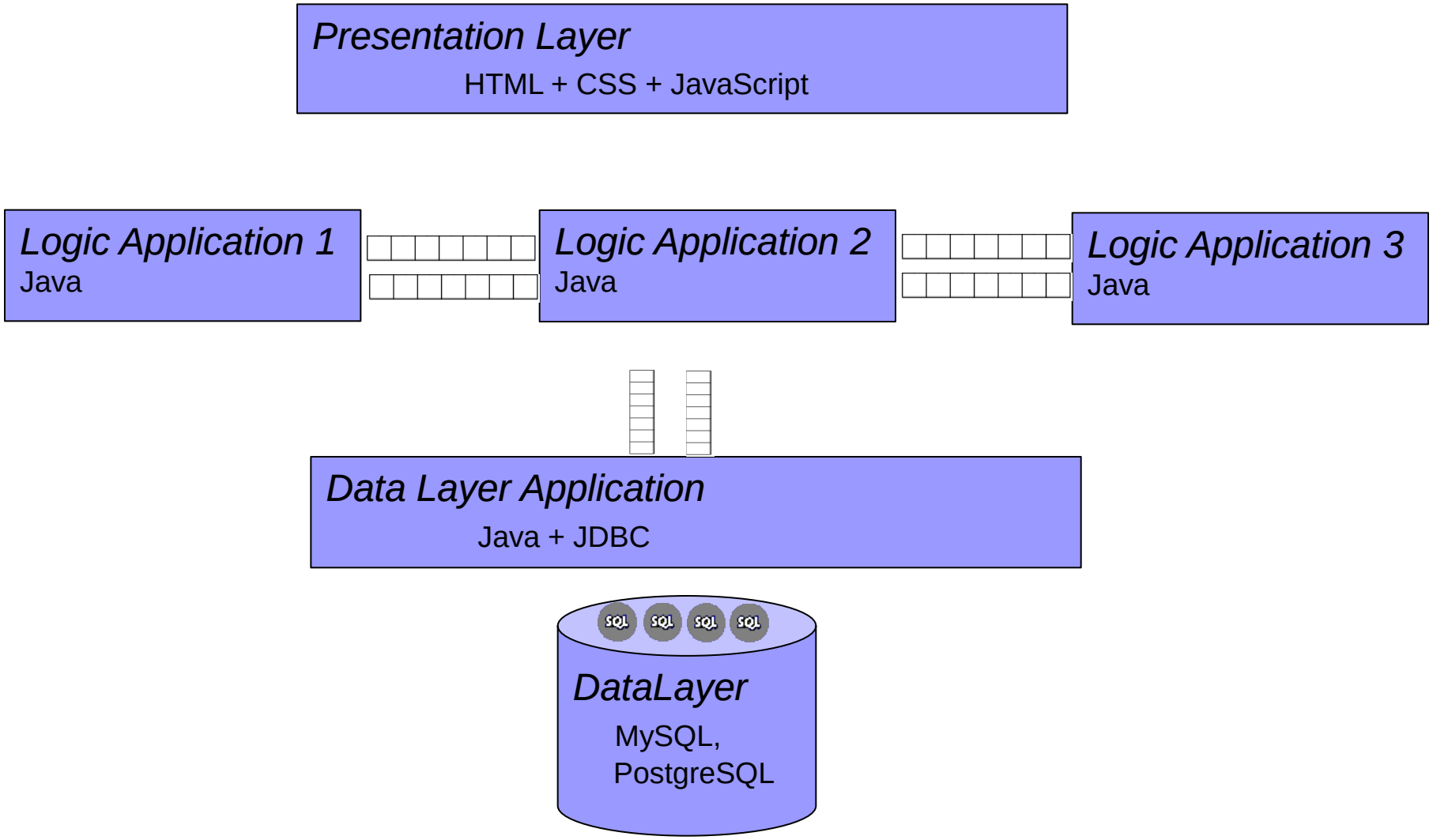
N Tier Architecture



N Tier Architecture + SP



N Tier Architecture + SP + MQ



Pros of using N + SP + MQ

- N: can replace an application without affecting other code
- Using DataLayer Application: Database is hidden behind a Java application
- Using SP: isolated SQL from Java code
- Using MQ: most scalable

Cons of using N + SP + MQ

- How are we going to integrate XML with Java objects and also with the Database?
  - 3 different technologies: XML, Java and DB!!

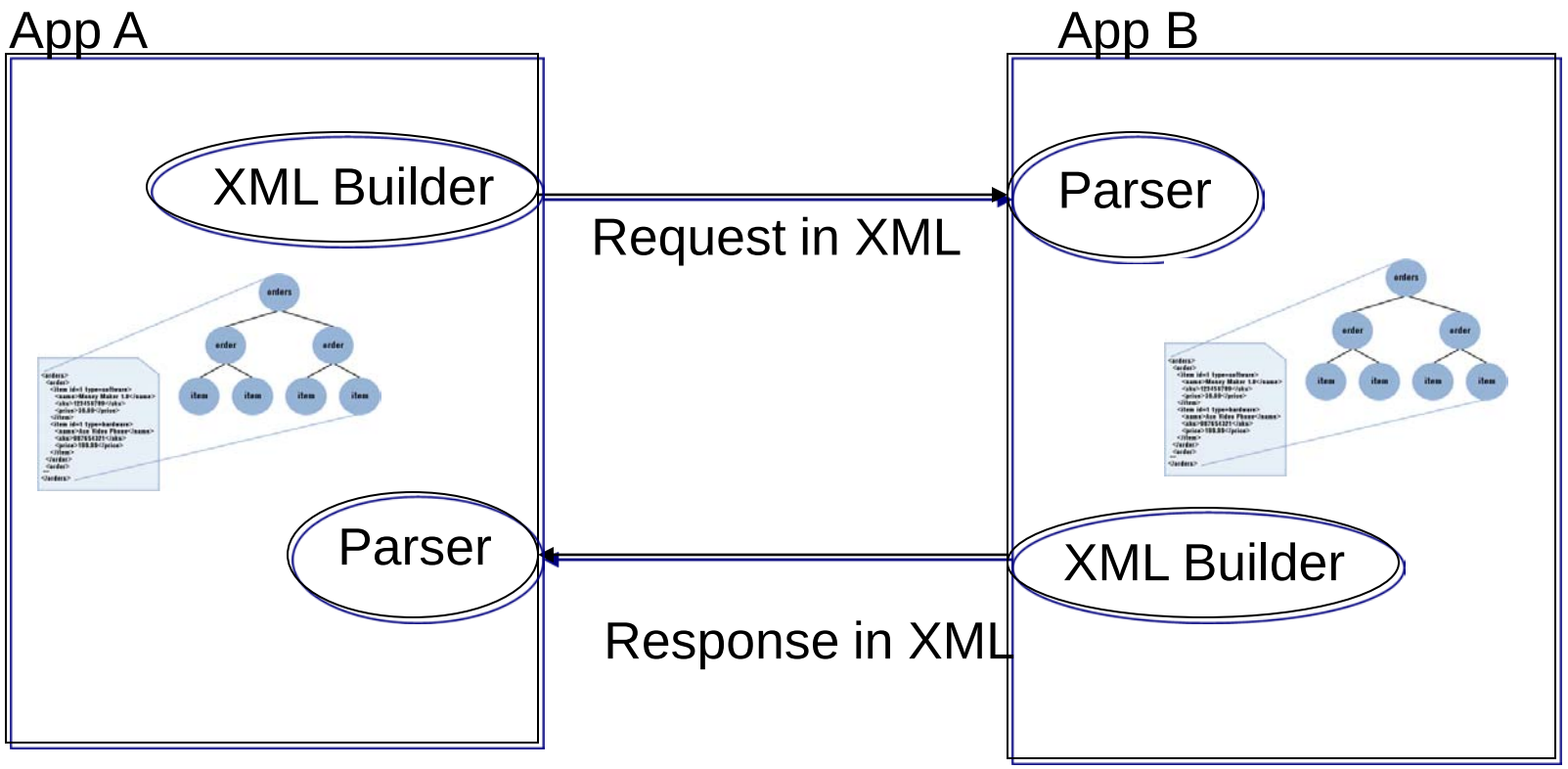
This is called the XML-binding Problem (or simply XML-binding)

XML Binding

- Binding Process
  - Details about how the XML is to be mapped
- Marshalling
- Un-marshalling

Solutions to the XML binding Problem?

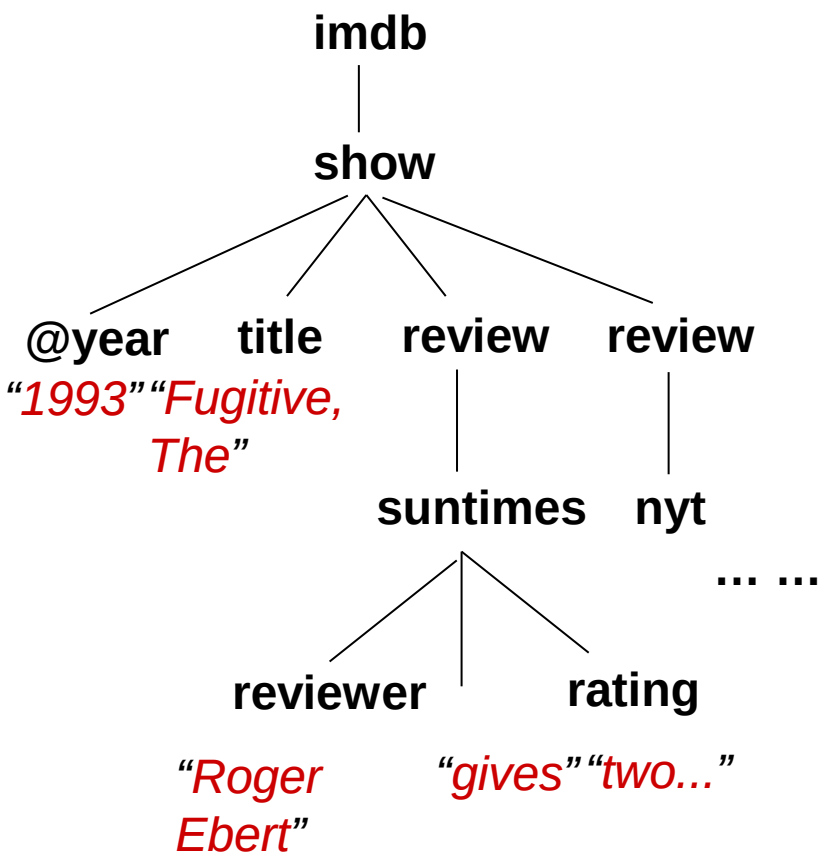
1) Use SAX/DOM parser



So, what's the problem now ?

Solutions to the XML binding Problem?

1) Use SAX/DOM parser



What does this code fragment do?

```
var x=getElementsByTagName('title')
for (i=0;i<x.length;i++) {
    document.write(x[i].childNodes[0].
    nodeValue)
}
```

Solutions to the XML binding Problem?

1) Use SAX/DOM parser
   - Typical Code
   - Pros & Cons ?

Solutions to the XML binding Problem?

1) Use SAX/DOM parser

2) Use a XML-binding library

Performance, small sets	Performance, medium sets	Performance, large sets	Subjective ease-of-use ranking
1. JiBX - 12.9 (29)	1. JiBX - 13.0 (32)	1. Javolution - 12.4 (21)	1. JAXB 2.0 - 10
2. Javolution - 14.4 (48)	2. Javolution - 13.3 (26)	2. JiBX - 14.2 (34)	2. JAXB - 9.5
3. Relaxer - 17.2 (61)	3. XMLSpy $ - 17.4 (69)	3. XMLSpy $ - 16.6 (66)	3. Liquid $ - 9.5
4. Zeus ☒ - 22.6 (84)	4. XMLBeans - 19.8 (86)	4. XMLBeans - 16.8 (60)	4. JiBX - 9
5. JAXB - 23.5 (116)	5. JAXB - 20.6 (92)	5. JAXB 2.0 - 18.0 (86)	5. Relaxer - 8.5
6. XMLBeans - 24.6 (125)	6. Zeus ☒ - 21.0 (98)	6. EMF (Eclipse) - 20.2 (89)	6. XmlIO - 8.5
7. JaxMe - 24.8 (95)	7. EMF (Eclipse) - 21.8 (113)	7. JAXB - 22.8 (90)	7. JSX $ - 8.5
8. EMF (Eclipse) - 26.7 (148)	8. Liquid $ - 22.3 (112)	8. Liquid $ - 23.1 (116)	8. JaxMe - 8.5
9. XMLSpy $ - 28.0 (101)	9. JAXB 2.0 - 23.8 (103)	9. Zeus ☒ - 24.0 (119)	9. Zeus ☒ - 8
10. XStream - 29.5 (128)	10. XStream - 29.8 (152)	10. Castor - 27.9 (132)	10. XMLBeans - 8
11. JAXB 2.0 - 29.8 (117)	11. Castor - 30.9 (157)	11. XStream - 35.6 (176)	11. XStream - 7
12. C24 $ - 29.8 (144)	12. Relaxer - 33.1 (121)	12. Relaxer - 37.0 (125)	12. Skaringa - 7
13. Liquid $ - 30.2 (125)	13. JaxMe - 35.0 (120)	13. JSX $ - 40.3 (163)	13. Castor - 6
14. XmlIO - 37.1 (144)	14. C24 $ - 37.6 (159)	14. Skaringa - 42.6 (192)	14. JXM ☒ - 6
15. Beck ☒ - 40.2 (166)	15. XmlIO - 39.2 (174)	15. Glue $ - 43.7 (174)	15. C24 $ - 6
16. Castor - 42.7 (191)	16. Skaringa - 40.6 (193)	16. C24 $ - 45.7 (170)	16. XMLSpy $ - 5
17. Skaringa - 44.1 (182)	17. JSX $ - 41.1 (149)	17. XmlIO - 46.2 (185)	17. Glue $ - 5
18. JSX $ - 47.1 (148)	18. Beck ☒ - 46.5 (203)	18. JaxMe - 53.8 (130)	18. EMF (Eclipse) - 5
19. Betwixt - 48.5 (192)	19. Glue $ - 61.0 (186)	19. Beck ☒ - 54.0 (210)	19. Betwixt - 4.5
20. JXM ☒ - 70.3 (196)	20. JXM ☒ - 68.1 (234)	20. JXM ☒ - 67.9 (233)	20. Javolution - 4
21. Glue $ - 94.9 (232)	21. Betwixt - 68.5 (193)	21. Betwixt - 77.5 (201)	21. Beck ☒ - 3

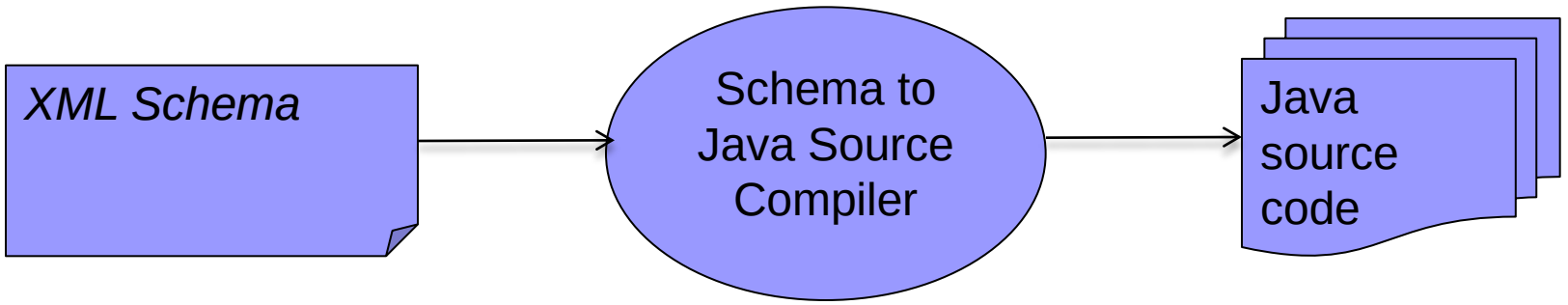
The $ refers to commercial libraries that require purchasing.
The ☒ refers to libraries that appear to be inactive.

https://bindmark.dev.java.net/old-index.html

JAXB: an XML-binding library

- Like SAX and DOM in that the application developer does not have to parse the XML
- Unlike SAX and DOM it is tied to a particular document schema
- Fast like SAX
- In memory representation like DOM but without all the general tree manipulation facilities

JAXB: an XML-binding library



Example XML

```
<?xml version="1.0" encoding="utf-8"?>
<itemList
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:noNamespaceSchemaLocation="itemList.xsd">
  <item>
    <name>pen</name>
    <quantity>5</quantity>
  </item>
  <item>
    <name>eraser</name>
    <quantity>7</quantity>
  </item>
  <item>
    <name>stapler</name>
    <quantity>2</quantity>
  </item>
</itemList>
```

Example XSD

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd
='http://www.w3.org/2001/XMLSchema'>

 <xsd:element name="itemList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="item"
   minOccurs="0" maxOccurs="3"/>
    </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
```

```
<xsd:element name="item">
   <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="name"/>
      <xsd:element ref="quantity"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

 <xsd:element name="name"
type="xsd:string"/>
 <xsd:element name="quantity"
type="xsd:short"/>
</xsd:schema>
```

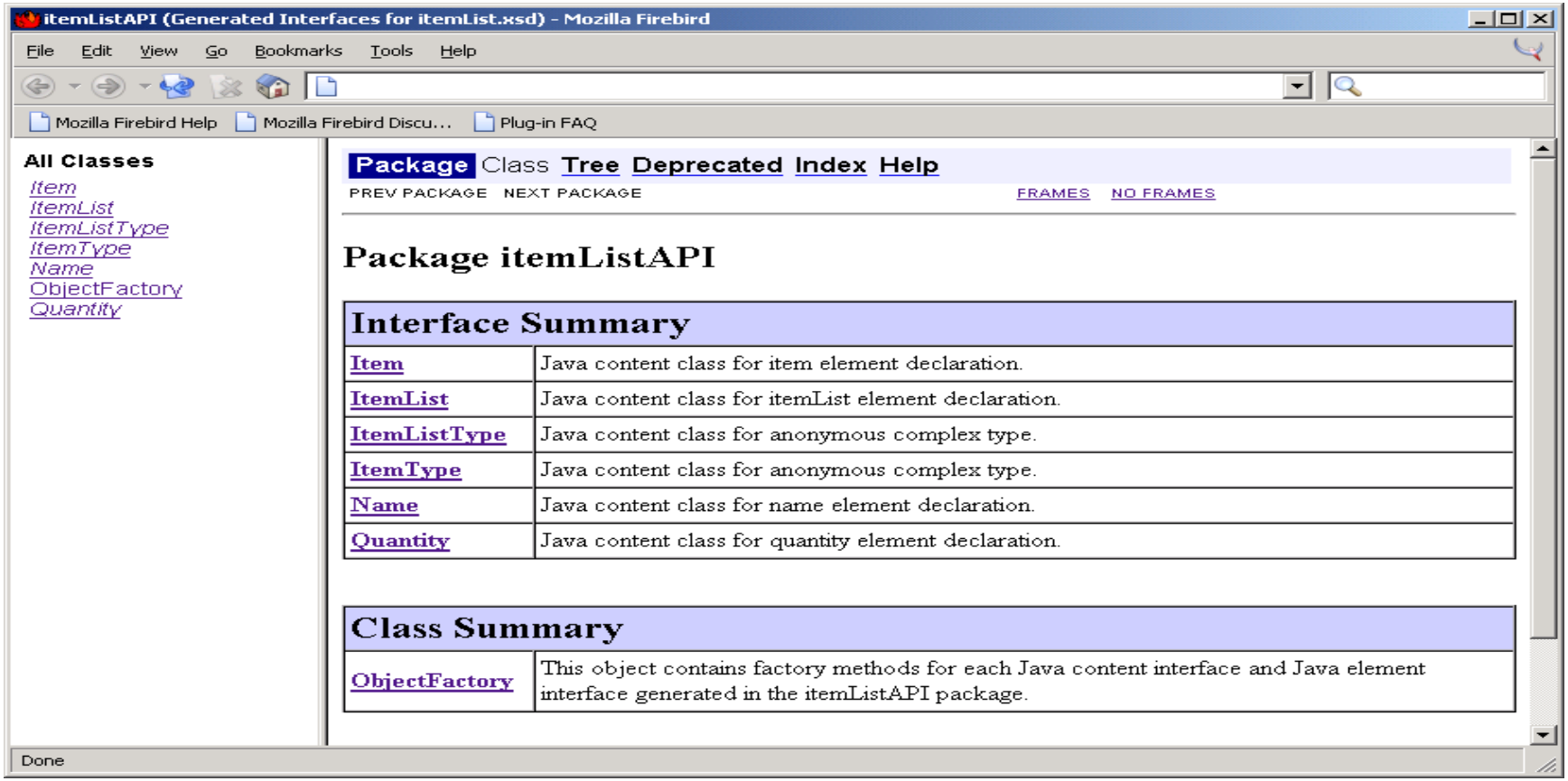
JAXB: Running xjc compiler

C:\java\jdk1.6.0_07\bin>xjc itemList.xsd

parsing a schema...
compiling a schema...
generated\impl\ItemImpl.java
generated\impl\ItemListImpl.java
generated\impl\ItemListTypeImpl.java
generated\impl\ItemTypeImpl.java
generated\impl\NameImpl.java
generated\impl\QuantityImpl.java
generated\Item.java
generated\ItemList.java
generated\ItemListType.java
generated\ItemType.java
generated\Name.java
generated\ObjectFactory.java
generated\Quantity.java
generated\bgm.ser
generated\jaxb.properties

Next, write Java Code That uses the NEW api

JAXB: Viewing the docs



JAXB: writing the client

```
import java.io.File;
import java.io.IOException;
import java.math.BigDecimal;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.UnmarshalException;
import javax.xml.bind.Unmarshaller;
import javax.xml.bind.ValidationEvent;
import javax.xml.bind.util.ValidationEventCollector;

// import java content classes generated by binding compiler
import itemListAPI.*;
```

```
public class Main {


public static void main( String[] args ) {
        try {
            // create a JAXBContext capable of handling classes generated into
            // the itemListAPI package
            JAXBContext jc = JAXBContext.newInstance( "itemListAPI" );

            // create an Unmarshaller
            Unmarshaller unmarshaller = jc.createUnmarshaller();

            // enable validation
            unmarshaller.setValidating( true );
            ItemList itemList = (ItemList) unmarshaller.unmarshal(
                                            new File( "itemList.xml" ) );
            java.util.List list = itemList.getItem();
            System.out.println("The length of the list is " + list.size());
```

Tip: how to generate Schema from XML?

- Use a library to do that!
  - E.g. http://blog.dotkam.com/2008/05/28/generate-xsd-from-xml/

Solutions to the XML binding Problem?

1) Use SAX/DOM parser

2) Use a XML-binding library

3) Work with native XML only: XQuery, XPath, etc…

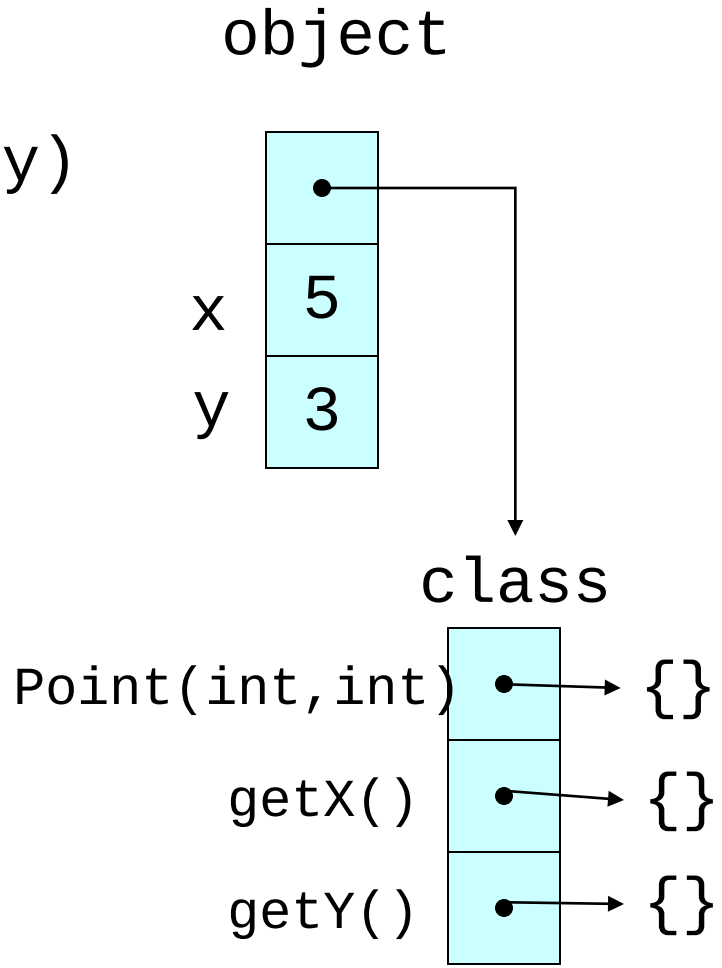
Reflection
(useful for assignment…)

Background

- Turing's great insight: programs are just another kind of data
  - Source code is text
  - Manipulate it line by line, or by parsing expressions
- Compiled programs are data, too
  - Integers and strings are bytes in memory that you interpret a certain way
  - Instructions in methods are just bytes too
- No reason why a program can't inspect itself

How objects work

```
class Point {
  public Point(int x, int y)
  {
    x = 5; Y = 10;
  }
  public int getX() {
     return x;
  }
  public int getY() {
     return y;
  }
  protected int x, y;
}
```



The class `Class`

- Instances of the class `Class` store information about classes
  - Class name
  - Inheritance
  - Interfaces implemented
  - Methods, members, etc.

- Can look up instances:
  - By name
  - From an object

http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Class.html

Showing a type

```
public static void showType(PrintStream out,
                            String className)
      throws ClassNotFoundException
{
  Class thisClass = Class.forName(className);
  String flavour  = thisClass.isInterface() ? "interface"
                                            : "class";
  out.println(flavour + " " + className);
  Class parentClass = thisClass.getSuperclass();
  if (parentClass != null) {
    out.println(" extends " + parentClass.getName());
  }
  Class[] interfaces = thisClass.getInterfaces();
  for (Class interf : interfaces) {
    out.println("  implements " + interf.getName());
  }
}
```

Output for type example

```
class java.lang.Object

class java.util.HashMap
 extends java.util.AbstractMap
  implements java.util.Map
  implements java.lang.Cloneable
  implements java.io.Serializable

class Point
 extends java.lang.Object
```

Examining class contents

```
public static void showContents(PrintStream out,
                                boolean hideObject,
                                String name)
        throws ClassNotFoundException {

  Class cls = Class.forName(name);
  out.println(name);
  showMembers(out, hideObject, name + " fields",
                  cls.getFields());
  showMembers(out, hideObject, name + " constructors",
                  cls.getConstructors());
  showMembers(out, hideObject, name + " methods",
                  cls.getMethods());
}
```

Examining class contents

```
public static void showMembers(PrintStream out,
                               boolean hideObject,
                               String title,
                               Member[] members) {
  out.println("  " + title);
  for (Member mem : members) {
    if (mem.getDeclaringClass() == Object.class) {
      if (hideObject) {
        continue;
      }
    }
    out.println("\t" + mem);
  }
}
```

Point *(somewhat edited)*

```
  Point fields
  Point constructors
    public Point(java.lang.String,int,int)

  public Point(int,int)
  Point methods
   public java.lang.String Point.toString()
   public java.lang.String Point.getName()
   public void Point.setName(java.lang.String)
   public int Point.getX()
   public void Point.setX(int)
   public int Point.getY()
   public void Point.setY(int)
```

Getting at members

- How to access members of a specific object?
  - Without making raw pointers into memory part of the language
    - (Raw pointers are a rich source of errors in C/C++)

- Introduce a class `Field`
  - Encapsulates access to a particular field of instances of a class
  - Knows "where the field is" in objects of that class
  - Use its `get()` and `set()` methods to inspect and modify the object

Examining fields

```
public static void main(String[] args) {
  PublicPoint p
   = new PublicPoint("center", 3, 3);

  showField(System.out, p, "fName");
  showField(System.out, p, "fX");
  showField(System.out, p, "fY");
  showField(System.out, p, "fZ");
}
```

```
public static void showField(PrintStream out,
                             Object obj, String fieldName) {
    try {
        Class cls    = obj.getClass();
        Field field  = cls.getField(fieldName);
        Object value = field.get(obj);
        out.println(fieldName + ": " + value);
    }
    catch (NoSuchFieldException e) {
        System.err.println(e);
    }
    catch (IllegalAccessException e) {
        System.err.println(e);
    }
}
```

Output

- `fName: center`
- `fX: 3`
- `fY: 3`
- `java.lang.NoSuchFieldException: fZ`

```
public static void showMethods(
                    PrintStream out, Object obj)
              throws NoSuchMethodException,
                     IllegalAccessException,
                     InvocationTargetException {

  Class cls = obj.getClass();
  out.println(cls.getName());
  for (Method meth : cls.getMethods()) {
    if (meth.getDeclaringClass() == cls) {
      showMethod(out, meth);
    }
  }
}
```

Calling methods

1. Look up a method based on its signature: the name and list of parameter types

2. Specify signature as a comma-separated list of `Class` objects
   - Specifies the types of arguments
   - Special values for types like int and boolean

3. Call the method, passing in parameters and capturing return value

```
Class mysClass = Class.forName("Mystery");
Object o = mysClass.newInstance();

Method m = mysClass.getMethod("euclidean",
  Double.TYPE, Double.TYPE);
double result = (Double) m.invoke(o,
 new Double(5.0), 12.0);

Method m2 = mysClass.getMethod("play",
  Class.forName("java.lang.String"),
  String.class);

m2.invoke(o, "Che", "Karl");
```

Key points

- There is no magic
  - A class is just a data structure
  - A method is just a data structure, too
- It just happens to contain bytes that look like instructions for the interpreter
- The call stack is another data structure
  - With libraries to give you access to it at runtime
- Many programming tools make use of reflection to create template engines (that's why it is useful in the assignment)