

INTRO TO JSP AND SERVLETS

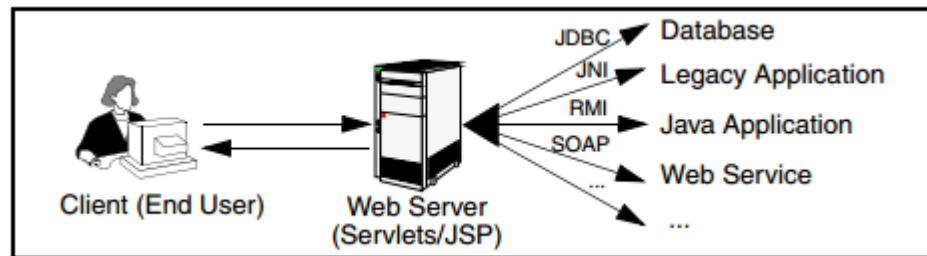
JAVA SERVLET

Java Servlets

- “A servlet is a Java™ technology-based Web component, managed by a **container**, that generates dynamic content.
- Like other Java-based components, Servlets are **platform-independent** Java classes that are compiled to **platform-neutral byte code** that can be loaded dynamically into and run by a java-enabled web server.
- Java Servlet specification defines an API for communication between the Web/application server and application program running in the server
 - E.g., methods to get parameter values from Web forms, and to send HTML text back to client

Servlets

- **Servlets are Java programs that run on Web or application servers,** acting as middle layer between requests coming from Web browser (HTTP) and application (database) on the HTTP server



Java Servlets

❑ What is a Servlet Container?

❑ “The servlet container is a part of a Web server or application server that provides the network services over which requests and responses are sent, decodes MIME-based requests, and formats MIME-based responses”

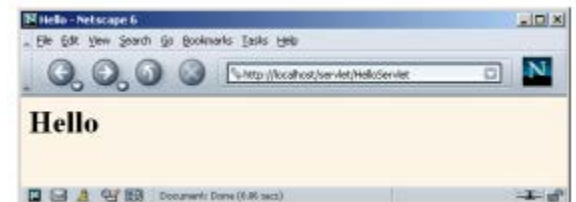
❑ A servlet container also contains and manages servlets through their lifecycle.”

❑ Java Servlet Containers

- ▶ Apache Tomcat
- ▶ BEA WebLogic Server
- ▶ IBM WebSphere Application Server
- ▶ JBoss Application Server
- ▶ Oracle Application Server
- ▶ Glassfish

Java Servlet Example: HelloServlet.java

- ❑ import java.io.*;
- ❑ import javax.servlet.*;
- ❑ import javax.servlet.http.*;
- ❑ public class **HelloServlet** extends **HttpServlet** {
 - ❑ public void **doGet**(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
 - ❑ response.setContentType("text/html");
 - ❑ PrintWriter out = response.getWriter();
 - ❑ String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
 - ❑ \"Transitional//EN\">\n";
 - ❑ out.println(docType +
 - ▶ "<HTML>\n" +
 - ▶ "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
 - ▶ "<BODY BGCOLOR=\"#FDF5E6\">\n" +
 - ▶ "<H1>Hello</H1>\n" +
 - ▶ "</BODY></HTML>");
 - ❑ }
 - ❑ }



Example Servlet Code: request and response OBJECTS

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PersonQueryServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE> Query Result</TITLE></HEAD>");
        out.println("<BODY>");
        ..... BODY OF SERVLET (next slide) ...
        out.println("</BODY>");
        out.close();
    }
}
```

Example Servlet Code: request and response OBJECTS

```
String persontype = request.getParameter("persontype");
String number = request.getParameter("name");
if(persontype.equals("student")) {
    ... code to find students with the specified name ...
    ... using JDBC to communicate with the database ..
    out.println("<table BORDER COLS=3>");
    out.println(" <tr> <td>ID</td> <td>Name: </td>" + " <td>Department</td> </tr>");
    for(... each result ...){
        ... retrieve ID, name and dept name
        ... into variables ID, name and deptname
        out.println("<tr> <td>" + ID + "</td>" + "<td>" + name + "</td>" + "<td>" + deptname
            + "</td></tr>");
    };
    out.println("</table>");
}
else {
    ... as above, but for instructors ...
}
```


Servlet: Typical Sequence of Events

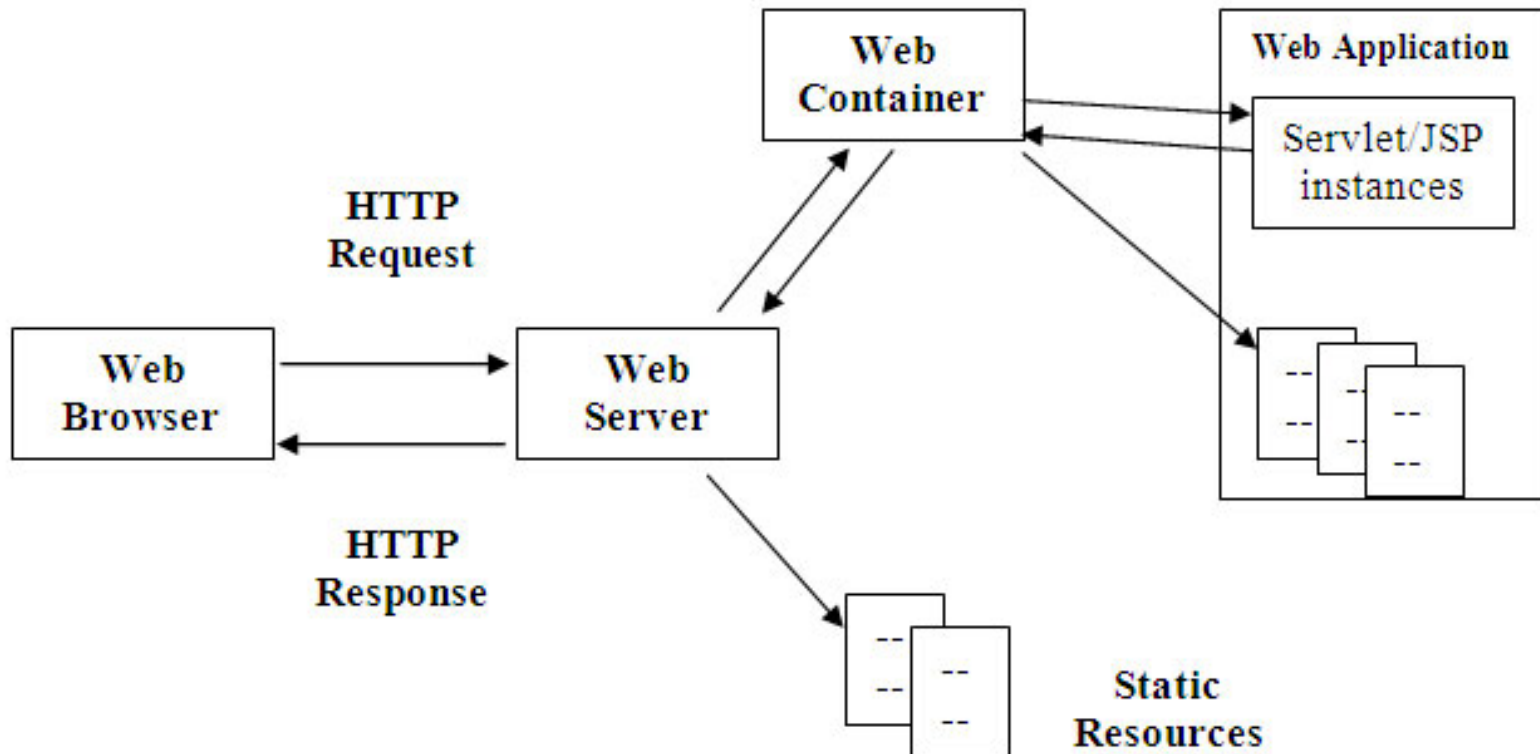
1. A **client** (e.g., a Web browser) accesses a Web server and makes an HTTP request.
2. The request is received by the Web server and handed off to the **servlet container**. The servlet container can be running :
 - in the same process as the host web server,
 - in a different process on the same host, or
 - on a different host from the web server for which it processes requests.
3. The servlet container determines **which servlet to invoke** based on the configuration of its servlets, and calls it with objects representing the request and response.
 - Web.xml configuration file (URLs to Java Servlet code)
4. The servlet uses the **request object** to find out who the remote user is, what HTTP POST parameters may have been sent as part of this request, and other relevant data.
 - The servlet performs whatever logic it was programmed with, and generates data to send back to the client. It sends this data back to the client via the **response object**.
5. Once the servlet has finished processing the request, the servlet container ensures that the response is properly flushed, and returns control back to the host Web server.

Servlet Sessions

- Servlet API supports handling of sessions
 - Sets a **cookie** on first interaction with browser, and uses it to identify session on further interactions
- To check if session is already active:
 - `if (request.getSession(false) == true)`
 - ▶ .. then existing session
 - ▶ else .. redirect to authentication page
 - authentication page
 - ▶ check login/password
 - ▶ `request.getSession(true)`: creates new session
- Store/retrieve attribute value pairs for a particular session
 - `session.setAttribute("userid", userid)`
 - `session.getAttribute("userid")`

Web Servers

- Outputs and receives HTML



JSP

SERVER-SIDE SCRIPTING

Server-Side Scripting

- Server-side scripting simplifies the task of connecting a database to the Web
 - Define an HTML document with **embedded executable** code/SQL queries.
 - Input values from HTML forms can be used directly in the embedded code(SQL queries).
 - When the document is requested, the Web server executes the embedded code to generate the actual HTML document.

- Numerous server-side scripting languages
 - JSP, PHP
 - General purpose scripting languages: VBScript, Perl, Python

JSP

- JSP is HTML pages with Java code embedded inside of them
 - a JSP document is just another way of writing a servlet.
 - JSP pages get translated into servlets, the servlets get compiled, and it is the servlets that run at request time.

- **JSP And Servlets are essentially the same thing**
 - JSP is focused on simplifying the creation and maintenance of the HTML.
 - Servlets are best at invoking the business logic and performing complicated operations.
 - A quick rule of thumb is that servlets are best for tasks oriented toward **processing**, whereas JSP is best for tasks oriented toward **presentation**

- “You can think of **servlets** as Java code with HTML inside”
- “you can think of **JSP** as HTML with Java code inside”

JSP Programming

- To make JSP page more interactive we can read and process the data entered in a an HTML form.
 - **request.getParameter method**
 - ▶ Takes the name of the HTML field name as parameter
 - ▶ Returns the value entered by the user
 - ▶ `String authorName = request.getParameter ("AuthorName");`

Java Server Pages (JSP)

- A JSP page with embedded Java code

```
<html>
```

```
<head> <title> Hello </title> </head>
```

```
<body>
```

```
<% if (request.getParameter("name") == null)
```

```
    { out.println("Hello World"); }
```

```
else
```

```
    { out.println("Hello, " + request.getParameter("name")); }
```

```
%>
```

```
</body>
```

```
</html>
```

- JSP is compiled into Java + Servlets

JSP Programming

□ JSP Elements

- All of these elements are identified by their own tags

□ **Declarations**

- ▶ These tags allows you to define variables and methods
- ▶ **<%!**
 - `private int count = 0;`
 - `private void increments () { count ++;}`
- ▶ **%>**

□ **Expressions**

- ▶ We can access variables and methods
- ▶ `<p> The value of Count is : <%= count %> </p>`

□ **Scriptlets**

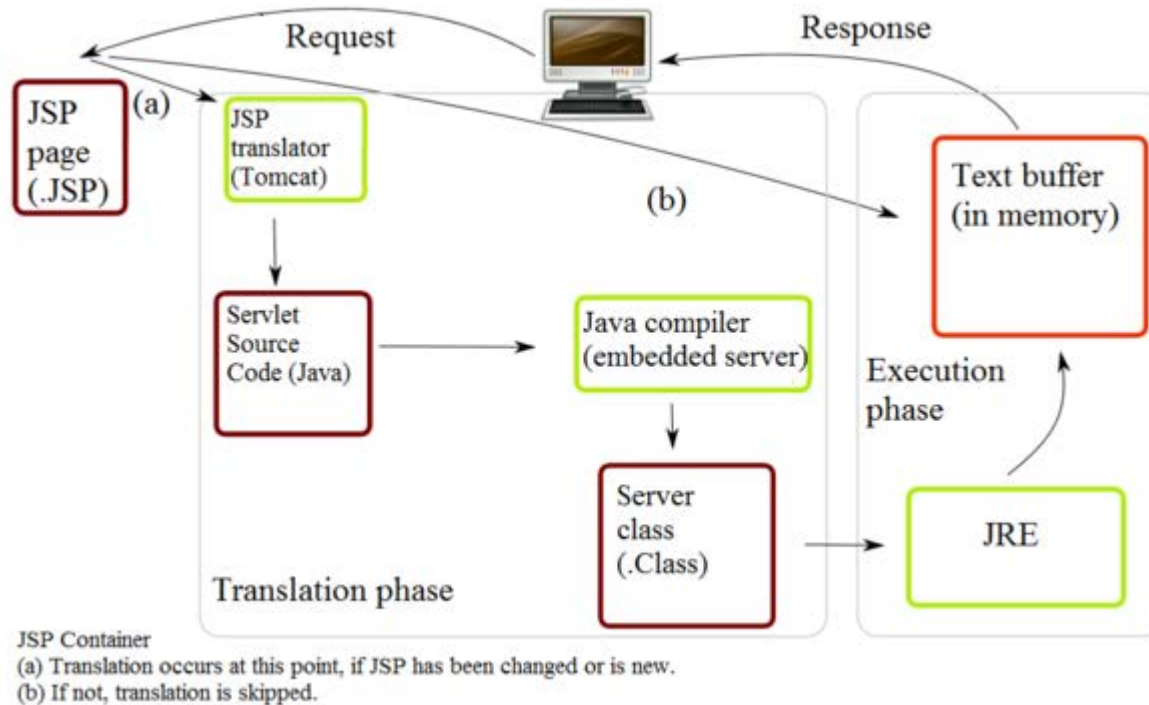
- ▶ Blocks of Java code
- ▶ `<% out.println ("Hello Students"); %>` //out is already defined in javax.servlet.jsp.JspWriter
- ▶ **<%**
 - `out.println("The counter =" + count +
);`
 - `incrementCount();`
- ▶ **%>**

□ **Directives**

- ▶ Instructions on how to process a JSP program (Importing packages)
- ▶ `<%@ page import="java.util.*,java.sql.*" %>`

Running JSP Program

- JSP Servlets Engine
 - JSP/Servlet engine dynamically compiles the JSP source code into Servlet if a current compiled Servlet does not exist



STATIC JSP DEMO

Sample web.xml File

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <display-name>HelloWorld Application</display-name>
  <description>
    This is a simple web application with a source code organization
    based on the recommendations of the Application Developer's Guide.
  </description>

  <servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>examples.Hello</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>

</web-app>
```

- ❑ In the preceding web.xml deployment descriptor file:
 - ❑ **<servlet>** XML element declares the HelloServlet,
 - ▶ the examples.Hello Java class implements the servlet,
 - ❑ **<servlet-mapping>** XML element specifies the /hello URL pattern that invokes the servlet in a browser.
 - ▶ This URL pattern is used in the index.html file.



Java Source of the Hello.java Servlet

- package examples;

- import java.io.IOException; import java.io.PrintWriter;
- import javax.servlet.ServletException; import javax.servlet.http.HttpServlet;
- import javax.servlet.http.HttpServletRequest;
- import javax.servlet.http.HttpServletResponse;

- **public final class Hello extends HttpServlet {**

- public void doGet(HttpServletRequest request, HttpServletResponse response)
- throws IOException, ServletException {
- response.setContentType("text/html");
- PrintWriter writer = response.getWriter();
- writer.println("<html>");
- writer.println("<head>");
- writer.println("<title>Sample Application Servlet Page</title>");
- writer.println("</head>");
- writer.println("<body bgcolor=white>");

- writer.println("<h1>Sample Application Servlet</h1>");

- writer.println("This is the output of a servlet that is part of");
- writer.println("the Hello, World application.");

- writer.println("</body>");
- writer.println("</html>");
- }
- }



JSP Source for the hello.jsp JSP

- ❑ `<html>`
- ❑ `<head>`
- ❑ `<title>Sample Application JSP Page</title>`
- ❑ `</head>`
 - `<h1>Sample Application JSP Page</h1>`
 - `<p>This is the output of a JSP page that is part of the HelloWorld application.</p>`
- ❑ `<%= new String("Hello!") %>`
- ❑ `</body>`
- ❑ `</html>`

The hello.jsp includes the following simple JSP directive:

```
<%= new String("Hello!") %>
```

This JSP directive simply prints out a message to the client (browser): Hello!

Sample Default index.html File

- `<html>`
- `<head>`
- `<title>Sample "Hello, World" Application</title>`
- `</head>`
- `<body>`
 - `<h1>Sample "Hello, World" Application</h1>`
 - `<p>This is the home page for the HelloWorld Web application. </p>`
 - `<p>To prove that they work, you can execute either of the following links:`
 - ▶ ``
 - ▶ `To a JSP page.`
 - ▶ `To a servlet.`
 - ▶ ``
- `</body>`
- `</html>`

PHP

PHP Scripting Environment

- PHP is widely used for Web server scripting
- Extensive libraries including for database access using ODBC

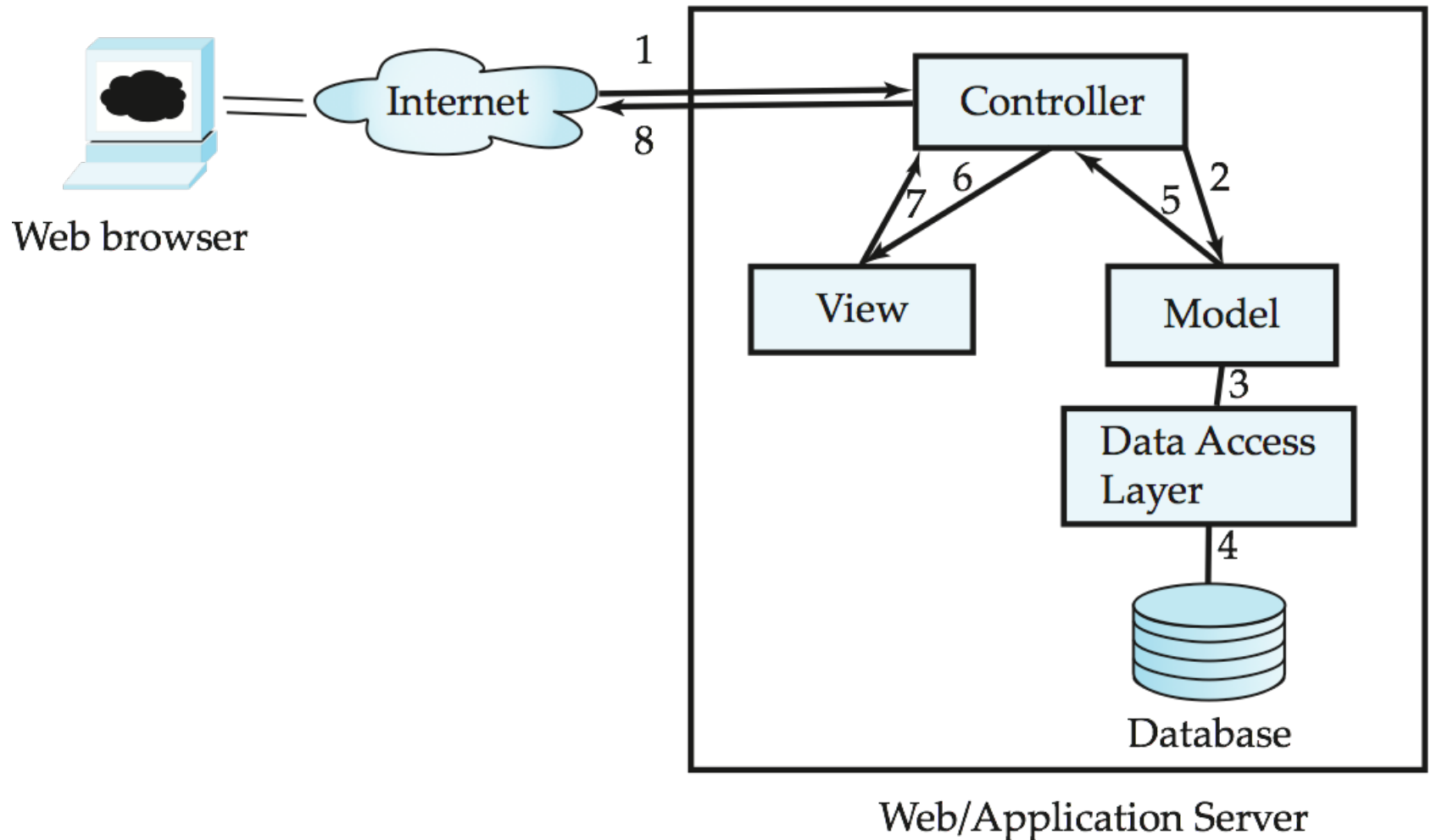
```
<html>
  <head> <title> Hello </title> </head>
  <body>
    <?php
    if (!isset($_REQUEST['name']))
      { echo "Hello World"; }
    else
      { echo "Hello, " + $_REQUEST['name']; }
    ?>
  </body>
</html>
```

Application Architectures

Application Architectures

- Application layers
 - Presentation or user interface
 - ▶ **model-view-controller (MVC)** architecture
 - **model**: business logic
 - **view**: presentation of data, depends on display device
 - **controller**: receives events, executes actions, and returns a view to the user
 - **business-logic** layer
 - ▶ provides high level view of data and actions on data
 - often using an object data model
 - ▶ hides details of data storage schema
 - **data access** layer
 - ▶ interfaces between business logic layer and the underlying database
 - ▶ provides mapping from object model of business layer to relational model of database

Application Architecture



Business Logic Layer

- Provides abstractions of entities
 - e.g. students, instructors, courses, etc
- Enforces **business rules** for carrying out actions
 - E.g. student can enroll in a class only if she has completed prerequisites, and has paid her tuition fees
- Supports **workflows** which define how a task involving multiple participants is to be carried out
 - E.g. how to process application by a student applying to a university
 - Sequence of steps to carry out task
 - Error handling
 - ▶ e.g. what to do if recommendation letters not received on time
 - Workflows discussed in Section 26.2

Object-Relational Mapping

- Allows application code to be written on top of object-oriented data model, while storing data in a traditional relational database
 - alternative: implement object-oriented or object-relational database to store object model
- Schema designer has to provide a mapping between object data and relational schema
 - e.g. Java class *Student* mapped to relation *student*, with corresponding mapping of attributes
 - An object can map to multiple tuples in multiple relations
- Application opens a session, which connects to the database
- Objects can be created and saved to the database using `session.save(object)`
 - mapping used to create appropriate tuples in the database
- Query can be run to retrieve objects satisfying specified predicates

Object-Relational Mapping and Hibernate (Cont.)

- The **Hibernate** object-relational mapping system is widely used
 - public domain system, runs on a variety of database systems
 - supports a query language that can express complex queries involving joins
 - ▶ translates queries into SQL queries
 - allows relationships to be mapped to sets associated with objects
 - ▶ e.g. courses taken by a student can be a set in Student object
- The **Entity Data Model** developed by Microsoft
 - provides an entity-relationship model directly to application
 - maps data between entity data model and underlying storage, which can be relational
 - Entity SQL language operates directly on Entity Data Model

NEXT

JAVA NETWORK DEMO1

java.net.* package

Download source code from the code folder on blackboard