Webontwikkeling 3

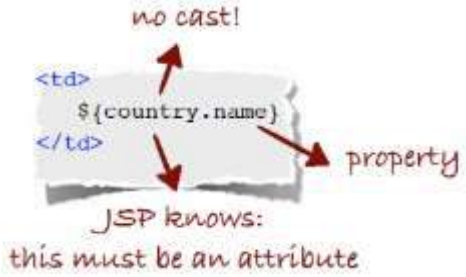
Samenvatting

Inhoud

1. JSPNoScriptlets

- MVC: seperates business logic from UI
- Scriptless page: a JSP page without Java (in order for designers to not have to know coding)

1.1 JSP 2.0: Expression Language (EL)

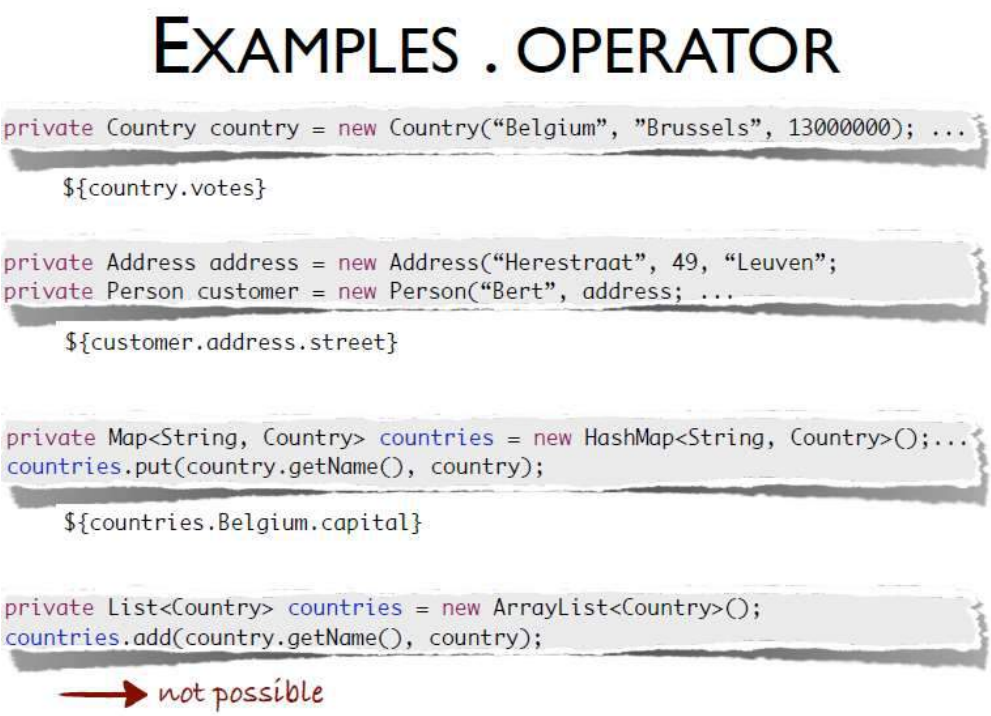


Instead of:

```
<td>
  <%= ((Country) request.getAttribute("country")).getName()%>
</td>
```

1.1.1 Point operator

- **Syntax**:
  `${attribute.property}`
  `${attribute.property.propertyOfProperty}`
- Only for **JavaBean** or **Map**
- Examples:



Intermezzo: what is a JavaBean?

JavaBean is a **standard:**

1. All **properties are private** (getters and setters are used)
2. It has a **public no-argument constructor (= default constructor)**
   note: default constructor exists implicitly in Java if there are no other constructors with parameters
3. It **implements Serializable**

Example:

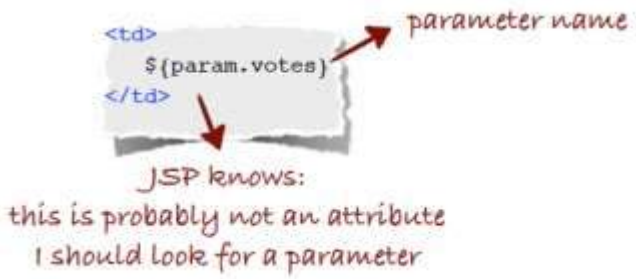
1.1.2 [] operator

- **Syntax:**
  `${attribute["property"]}`
  `${attribute[0]}`
- For **JavaBean**, **Map**, **Array**, **List**,…
- Examples:

1.1.3 Parameters

- Reading parameters is similar, use **keyword** "**param**":

```
http://localhost:8080/Tourism/Controller?votes=3
```



1.1.4 Checking for scriptlets

- Does my work still contain scriptlets?
- **Disable scriptlets** in web.xml and **watch for errors:**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
                       java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

   <jsp-config>
      <jsp-property-group>
         <url-pattern>*.jsp</url-pattern>
         <scripting-invalid>true</scripting-invalid>
      </jsp-property-group>
   </jsp-config>

   <welcome-file-list>
      <welcome-file>index.html</welcome-file>
   </welcome-file-list>
</web-app>
```

1.2 JSP actions

- **Predefined XML tags** (components of tag library)
- **Functions** like include, foreach,…
- **2 categories**: standard and custom

1.2.1 Standard actions

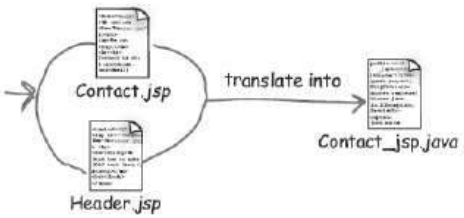
- Defined **in JSP itself**
- Begin with prefix: **jsp:**
- Examples:

```
<jsp:include page="header.jsp"/>
```

```
<jsp:include page="header.jsp">
   <jsp:param name="title"value="Countries"/>
</jsp:include>
```

```
<jsp:plugin type="applet" code="Molecule.class" codebase="/html">
  <jsp:params>
    <jsp:param name="molecule" value="molecules/benzene.mol" />
  </jsp:params>
  <jsp:fallback>
    <p>Unable to load applet</p>
  </jsp:fallback>
</jsp:plugin>
```

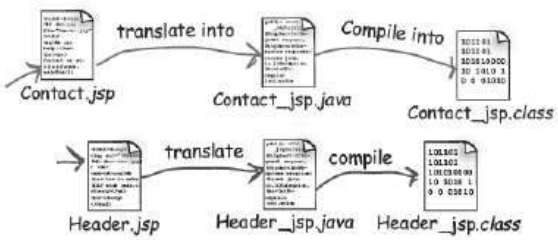
```
<%@include file="header.jsp" %>
```



inserts SOURCE of "Header.jsp" at translation time

```
<jsp:include page="header.jsp"/>
```

inserts RESPONSE of "Header.jsp" at runtime



1.2.1 Custom actions

- **User-defined** JSP language element
- Not defined in JSP itself → needs to be **included**
- Example: **JSP Standard Tag Library (JSTL)**
  - Bundles recurring functionality:
  - **Iteration**

```
<c:forEach var="country" items="${countries}">
    <tr>
        <td>${country.name}</td>
        ...
    </tr>
</c:forEach>
```

- **Choice**

```
<c:if test="${country.numberInhabitants > 1000000}">
        ...
</c:if>
```

```
<c:choose>
        <c:when test="${country.numberInhabitants > 1000000}">
                ...
        </c:when>
        <c:when test="${country.numberInhabitants > 50000000}">
                ...
        </c:when>
        <c:otherwise>
                ...
        </c:otherwise>
</c:choose>
```

- **How to add:**
  - Add JSTL JAR in WEB_INF/lib
  - & add line to JSP page:

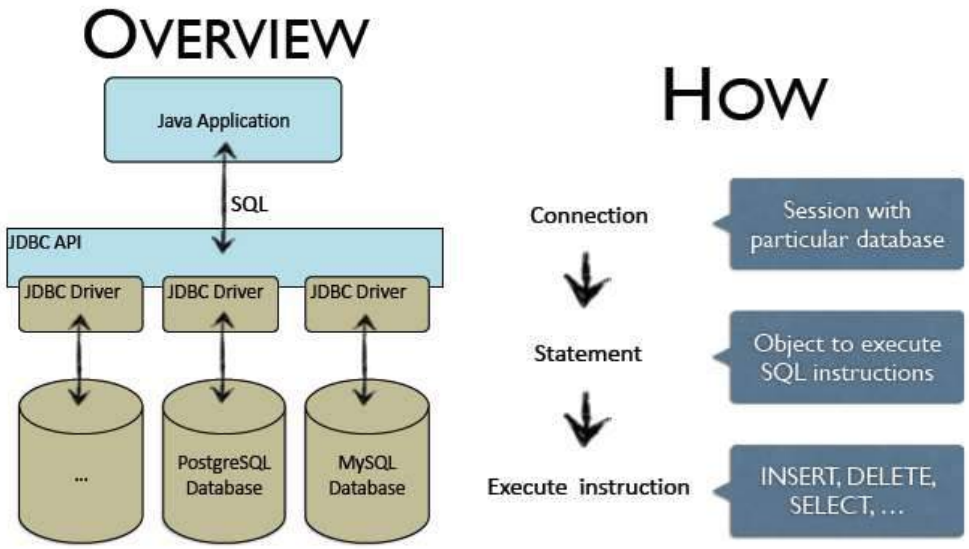
```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

Self-made actions? To be continued…

2. JDBC

2.1 JDBC

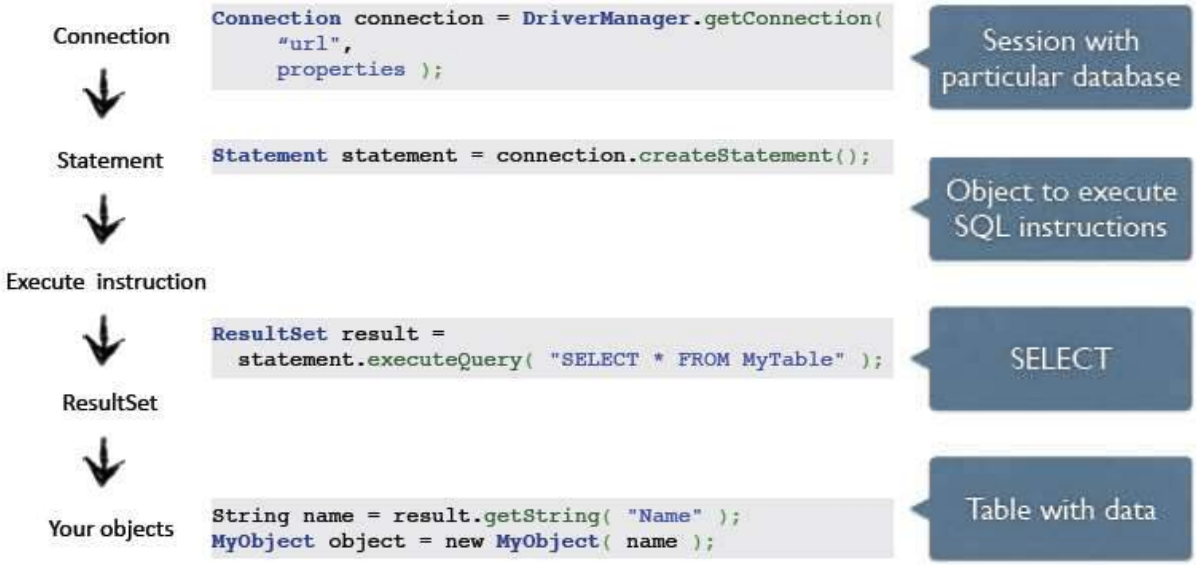
- **J**ava **D**ata**B**ase **C**onnectivity
- Java API → set klassen die je kan gebruiken
- Zorgt ervoor dat een Java applicatie kan **communiceren** met een **database** via **SQL**
- **"Write once, use anywhere"**: communicatie mogelijk met relationele databases (PostgreSQL, JavaDB, MySQL,…), spreadsheets,…



Action/query instructions

Action: INSERT, UPDATE, DELETE,…	**Query: SELECT**
`statement.execute("query");` Returns nothing	`statement.executeQuery("query ")` Returns: part of the data (ResultSet)
`statement.executeUpdate("query");` Returns: number of affected rows	

Query



Action

PostgreSQL driver

- http://jdbc.postgresql.org/download.html
- Eclipse: add jar in folder WEB-INF/lib

Connection data

- **URL**
  - **driver**: <vendor>://<server>:<port>/<dbname>
  - **jdbc**: postgresql://gegevensbanken.khleuven.be:51617/2TX?
- **Properties**
  - **user**: <LDAP userid>
  - **password**: <password>
  - **ssl**: true
  - **sslfactory**: org.postgresql.ssl.NonValidatingFactory

Voorbeeld:

```java
public class TestDB {
    public static void main(String[] args) throws SQLException {
        Properties properties = new Properties();
        String url = "jdbc:postgresql://gegevensbanken.khleuven.be:51314/webontwerp";
        properties.setProperty("user", <userid>);
        properties.setProperty("password", <password>);
        properties.setProperty("ssl", "true");
        properties.setProperty("sslfactory", "org.postgresql.ssl.NonValidatingFactory");

        Class.forName("org.postgresql.Driver");
        Connection connection = DriverManager.getConnection(url,properties);
        Statement statement = connection.createStatement();
        ResultSet result = statement.executeQuery( "SELECT * FROM test_u0082726.country" );

        while(result.next()){
            String name = result.getString("name");
            String capital = result.getString("capital");
            int numberOfVotes = Integer.parseInt(result.getString("votes"));
            int numberOfInhabitants = Integer.parseInt(result.getString("inhabitants"));

            Country country= new Country(name, numberOfInhabitants,capital, numberOfVotes);
            System.out.println(country);
        }
    }
```

JDBC API

for web-application

access to elements of ResultSet

access to field values in the rows

2.2 Design

COUNTRYREPOSITORYSQL.JAVA

```java
public class CountryRepositorySql {
    private Statement statement;

    public CountryRepositorySql() {
        //create statement object
    }

    public void add(Country country){
        //use statement object to execute an insert query
    }

    public List<Country> getAll(){
        //use statement object to execute a select query
        //convert result to list of countries
        //return list of countries;
    }
}
```

```
public CountryRepositorySql() {
      Properties properties = new Properties();
      String url = "jdbc:postgresql://gegevensbanken.khleuven.be:51415/webontwerp";
      properties.setProperty("user", "u0015529");
      properties.setProperty("password", "XXX");
      properties.setProperty("ssl", "true");
      properties.setProperty("sslfactory", "org.postgresql.ssl.NonValidatingFactory");
      Connection connection;
      try {
         Class.forName("org.postgresql.Driver");
         connection = DriverManager.getConnection(url, properties);
         statement = connection.createStatement();
      } catch (SQLException e) {
         throw new DbException(e.getMessage(), e);
      } catch (ClassNotFoundException e) {
         throw new DbException(e.getMessage(), e);
      }
   }

public void add(Country country){
   if(country == null){
      throw new DbException("Nothing to add.");
   }
   String sql = "INSERT INTO greetje.country (name, capital, inhabitants, votes)"
         + "VALUES ('"
         + country.getName() + "', '" + country.getCapital() + "', "
         + country.getNumberInhabitants() + ", "+ country.getVotes() + ")";
   try {
      statement.executeUpdate(sql);
   } catch (SQLException e) {
      throw new DbException(e);
   }
}
```

WHERE DOES COUNTRYREPOSITORYSQL BELONG?

- [x] Model
- [ ] View
- [ ] Controller

WHO USES COUNTRYREPOSITORYSQL?

- [ ] countryOverview.jsp
- [ ] countryForm.jsp
- [ ] Controller.java
- [ ] Country.java
- [x] CountryService.java
- [ ] CountryRepositoryInMemory.java

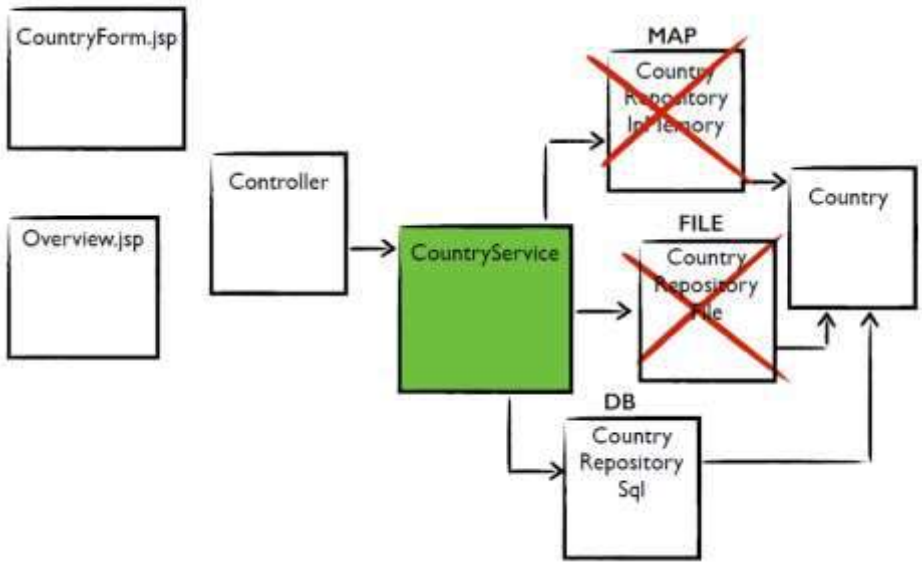
COUNTRYSERVICE.JAVA

```
public class CountryService {
   private CountryRepositorySql db = new CountryRepositorySql();

   public void addCountry(Country country){
      db.add(country);
   }

   public List<Country> getCountries(){
      return db.getAll();
   }
```

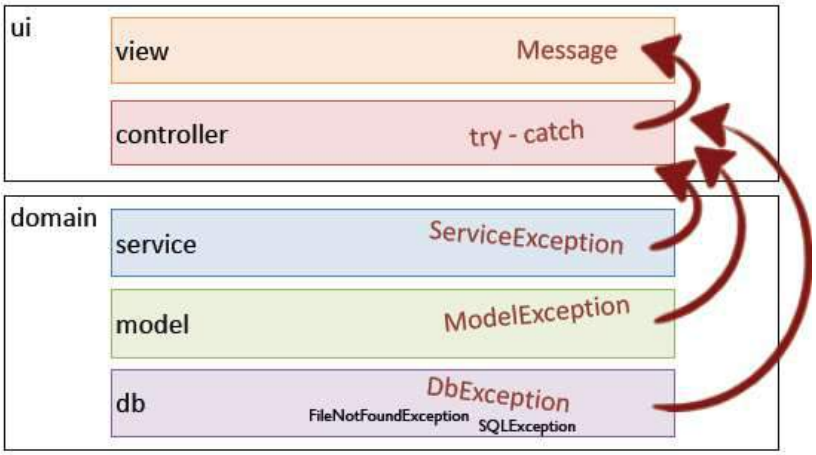
- order of addition does not matter
- we want no doubles
- we want to use an identifier to lookup elements
- **data are stored permanently**

Het maakt dus niet uit als je je model wijzigt, de servlet zelf hoeft dit niet te weten:

FACADE + STRATEGY



EXCEPTION HANDLING



CUSTOM EXCEPTIONS

Tells me where it went wrong

Unchecked

```
public class DbException extends RuntimeException {

    public DbException() {
        super();
    }

    public DbException(String message, Throwable exception) {
        super(message, exception);
    }

    public DbException(String message) {
        super(message);
    }

    public DbException(Throwable exception) {
        super(exception);
    }
}
```

you can pass the original exception

EXAMPLE

COUNTRYREPOSITORYSQL.JAVA IN PACKAGE DB

```
public void add(Country country){
     if(country == null){
        throw new DbException("Nothing to add.");
     }
     String sql = "INSERT …";
     try {
        statement.execute(sql);
     } catch (SQLException e) {
        throw new DbException(e.getMessage, e);
     }
  }
```

pass the original exception

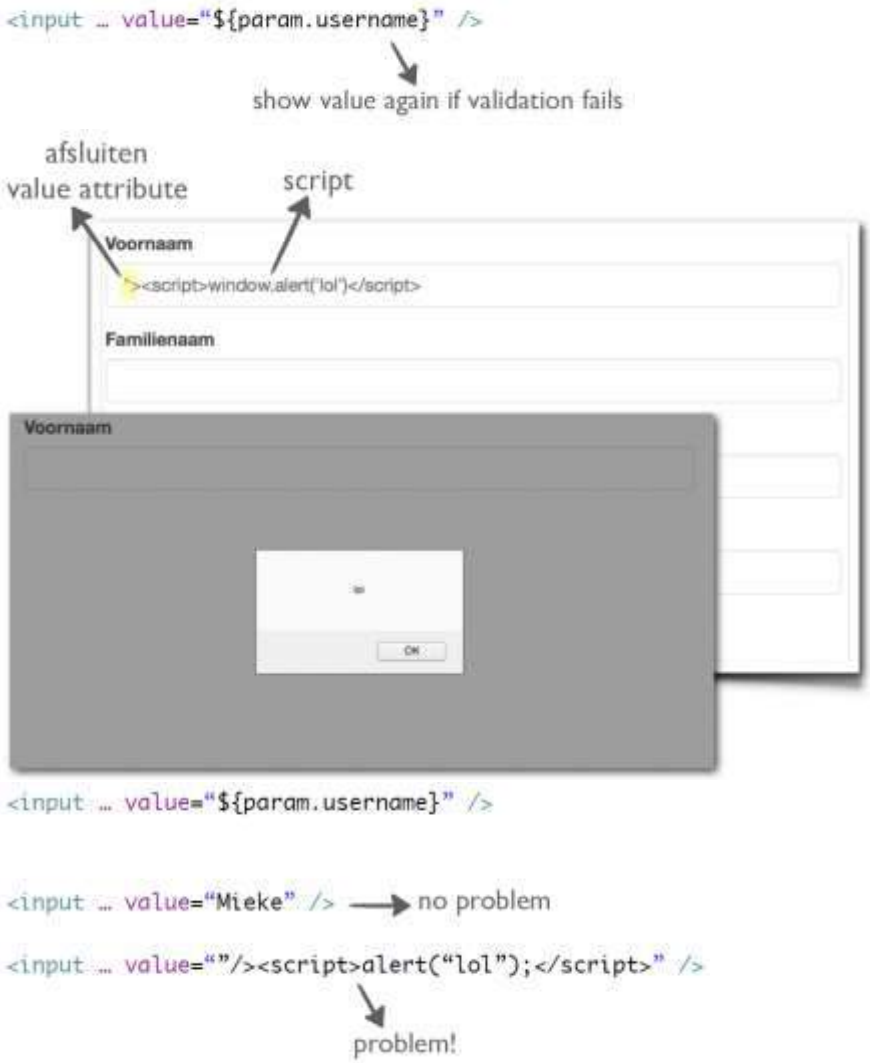
3. XSS

3.1 What is XSS

- XSS = **Cross Site Scripting**
- **Security risk**
- Hacker injects **malicious code into safe website**
- Often through script in the browser

3.1 Problem

- Scripts often placed through a form
- When reloaded (e.g. after validation error), browser parses previous value and executes script
- Example:



3.2 Solution

- **Cleanup input**
- Convert HTML Symbols to **HTML entities**
- **Browser shows entities as symbols but doesn't interpret them**
- **In JSP:** `<c:out value="">` → escapes HTML/XML tags

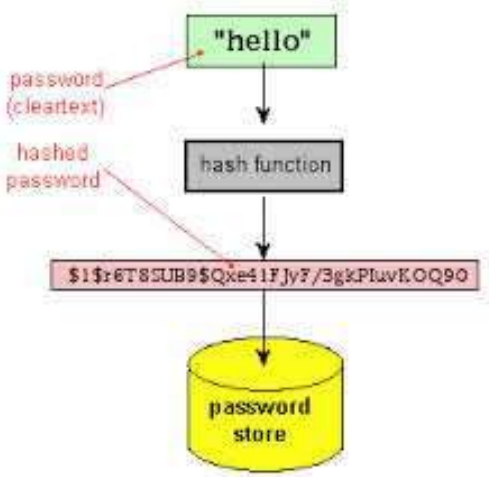
4. Hashing

4.1 What is hashing

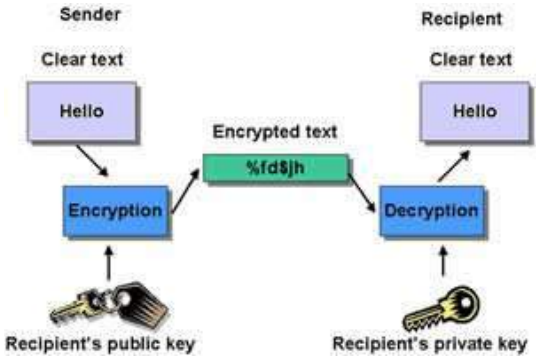
- **Never put passwords in plain text in a DB**
- Use a **hash function: encrypt data**
- Examples
  - MD5
    - Unsafe
    - Rainbow tables
  - SHA-1
    - Safe for today's encryption
  - SHA-256
    - Safe for today and the future

4.2 Hashing – Encryption – Encoding

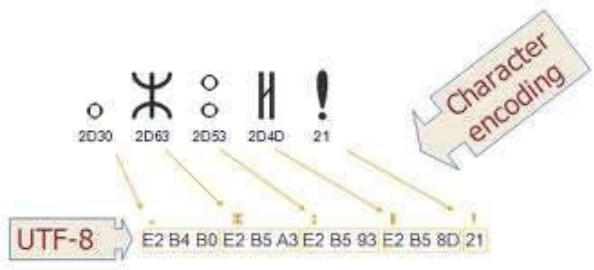
- **Hashing**
  - One-way
  - Using a salt



- **Encryption**
  - Reversible
  - Using a key



- **Encoding**
  - Reversible
  - Integrety instead of security



4.3 SHA-1

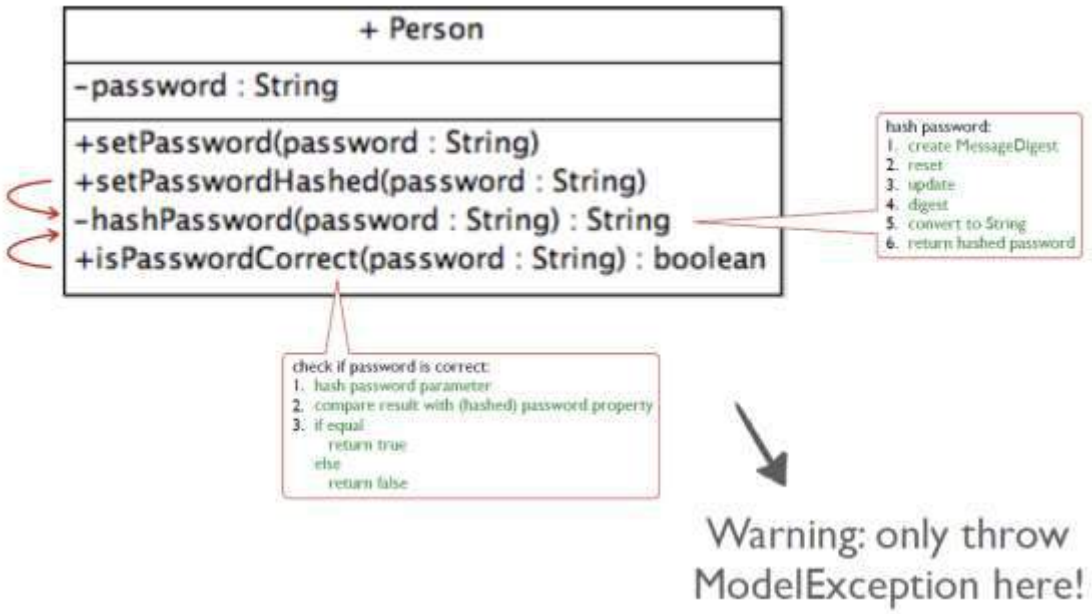
- **String => hexadecimal number**
- **40 digits long:** fixed length of 40 in DB
- Example:
  `sha1(banana) => 250e77f12a5ab6972a0895d290c4792f0a326ea8`

4.4 MessageDigest

- Hashing in **Java**
- Part of Java.Security
- Methods:
  - `MessageDigest.getInstance(algorithm)`
  - `update(stringToEncrypt) //encrypt`
  - `digest():byte[] //finalize (padding,…)`
  - `reset() //for further use`
- Code example:

```
import java.io.UnsupportedEncodingException;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class HashingExample {

   public static void main(String arg[]) throws Exception {
       System.out.println(sha1("banana"));
     }

   private static String sha1(String password) throws
        NoSuchAlgorithmException, UnsupportedEncodingException {
      MessageDigest crypt = MessageDigest.getInstance("SHA-1");
      crypt.reset();
      crypt.update(password.getBytes("UTF-8"));

      byte[] digest = crypt.digest();
      return new BigInteger(1, digest).toString(16);
}
```

- UML example:



4.5 Salting

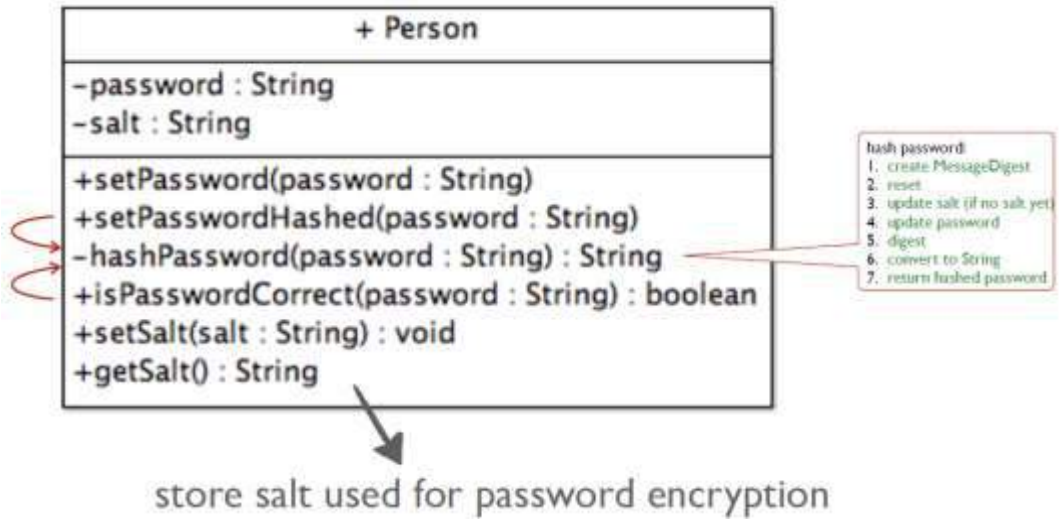
- Appending or prepending a **random string before hashing**
- String is called **salt**
- Save both salt and hash in the DB
- **Order of hashing and salting** important
  - Choose your own order
  - Hacker doesn't know the order
  - Therefore, saving in DB is no problem
  - Hashing/salting multiple times is good for security

4.6 SecureRandom

- Salting in **Java**
- Part of Java.Security
- Methods:
  - `new SecureRandom();`
  - `Random.generateSeed(20) //generate seed bytes (= salt)`
- Code example:

```
public class EncryptionExample {

   public static void main(String arg[]) throws Exception {
       System.out.println(sha1("banana"));
     }

   private static String sha1(String password) throws
        NoSuchAlgorithmException, UnsupportedEncodingException {
      MessageDigest crypt = MessageDigest.getInstance("SHA-1");
      crypt.reset();
      // salt
      SecureRandom random = new SecureRandom();
      byte seed[] = random.generateSeed(20);
      crypt.update(seed);
      crypt.update(password.getBytes("UTF-8"));

      return new BigInteger(1, crypt.digest()).toString(16);
   }
}
```

- UML example:



5. SQL Injection

5.1 What is SQL-injection

- Code injection in data-driven applications
- Insert **malicious SQL** statement in form field
- **Get sensitive data** or **modify database**
- Example:
  - SELECT statement to get user data
  - Contains WHERE statement
  - Hacker inserts: `' OR 1=1 OR '1' = '1`
  - **This is always true**
  - Returns **all user data**

5.2 Solution: prepared statements

- **Placeholders** instead of values
  - Parse statement once
  - Call multiple times with other parameter value

- Advantages:
  - **Faster**
  - **Cleaner syntax**
  - Query structure is fixed, so **safe**
- Example:

REFACTOR USERDB

CONSTRUCTOR

```
public class UserDB {
   private Connection connection;
   private PreparedStatement statement;

   public UserDB() {
      Properties properties = new Properties();
      ...
      try {
         Class.forName("org.postgresql.Driver");
         connection = DriverManager.getConnection(url, properties);
         statement = connection.createStatement();
      } catch (Exception e) {
         throw new DbException(e.getMessage(), e);
      }
   }
```

ADD

```
public void add(Person person) {
   if (person == null) {
      throw new DbException("Nothing to add.");
   }
   String sql = "INSERT INTO person (name, email, password)"
         + " VALUES (?,?,?)";

   try {
      statement = connection.prepareStatement(sql);
      statement.setString(1, person.getName());
      statement.setString(2, person.getEmail());
      statement.setString(3, person.getPassword());
      statement.execute();
   } catch (SQLException e) {
      throw new DbException(e);
   }
}
```

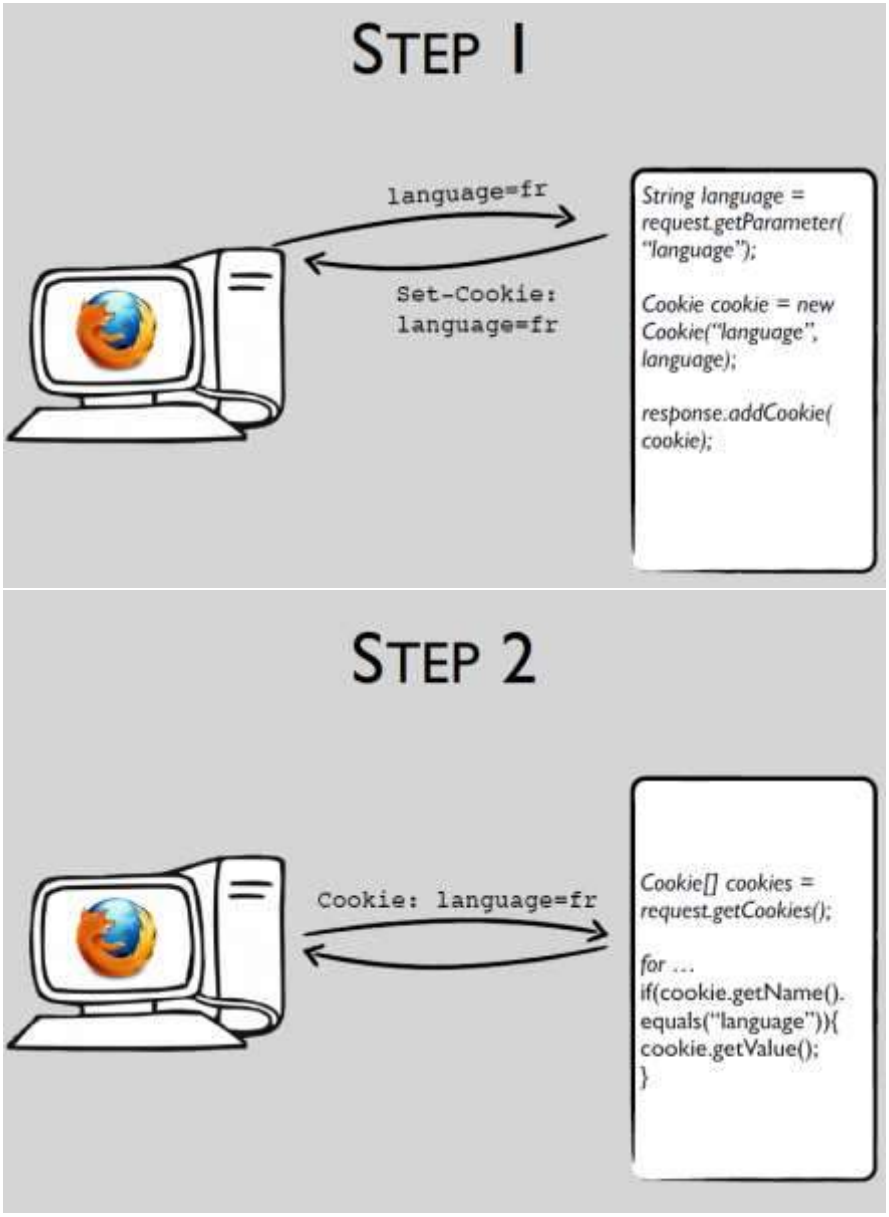
GET

```
public Person get(String email) {
   Person person;
   try {
      String sql =
        "SELECT * FROM person WHERE email = ?";
      statement = connection.prepareStatement(sql);
      statement.setString(1, email);
      ResultSet result = statement.executeQuery();
      result.next();
      String name = result.getString("name");
      String password = result.getString("password");
      person = new Person(name, email, password);
   } catch (SQLException e) {
      throw new DbException(e.getMessage(), e);
   }
   return person;
}
```

6. Cookies

6.1 What are cookies

- **Name-value** pairs, e.g. `language = english`
- **Why?**
  - To save **temporary data**
  - Info about **one client**
  - For **one domain/path**
- **Where?**
  - Made by a '**script**' on the **server**
  - Saved by the **client** in its **browser**
- **How?**
  1. Server makes a cookie
  2. Server gives cookie to the client in the **response header**
  3. Browser sends the cookie within each **request header**
- **Example**



6.2 Duration

- Length of a cookie can be set
- Until **expiration time**
  - `cookie.setMaxAge(24*60*60) // 24 hours`
  - `cookie.setMaxAge(0) // immediately`
- Until **closing the browser**
  - `cookie.setMaxAge(-1)`

6.3 Using cookies

- In browser
- Can be **removed or blocked**
- **Unsafe** → google "edit http cookies"
- Expiration time **can be set**
- Only use when sending **few/small amounts of data** (network load!)
- Example: language choice

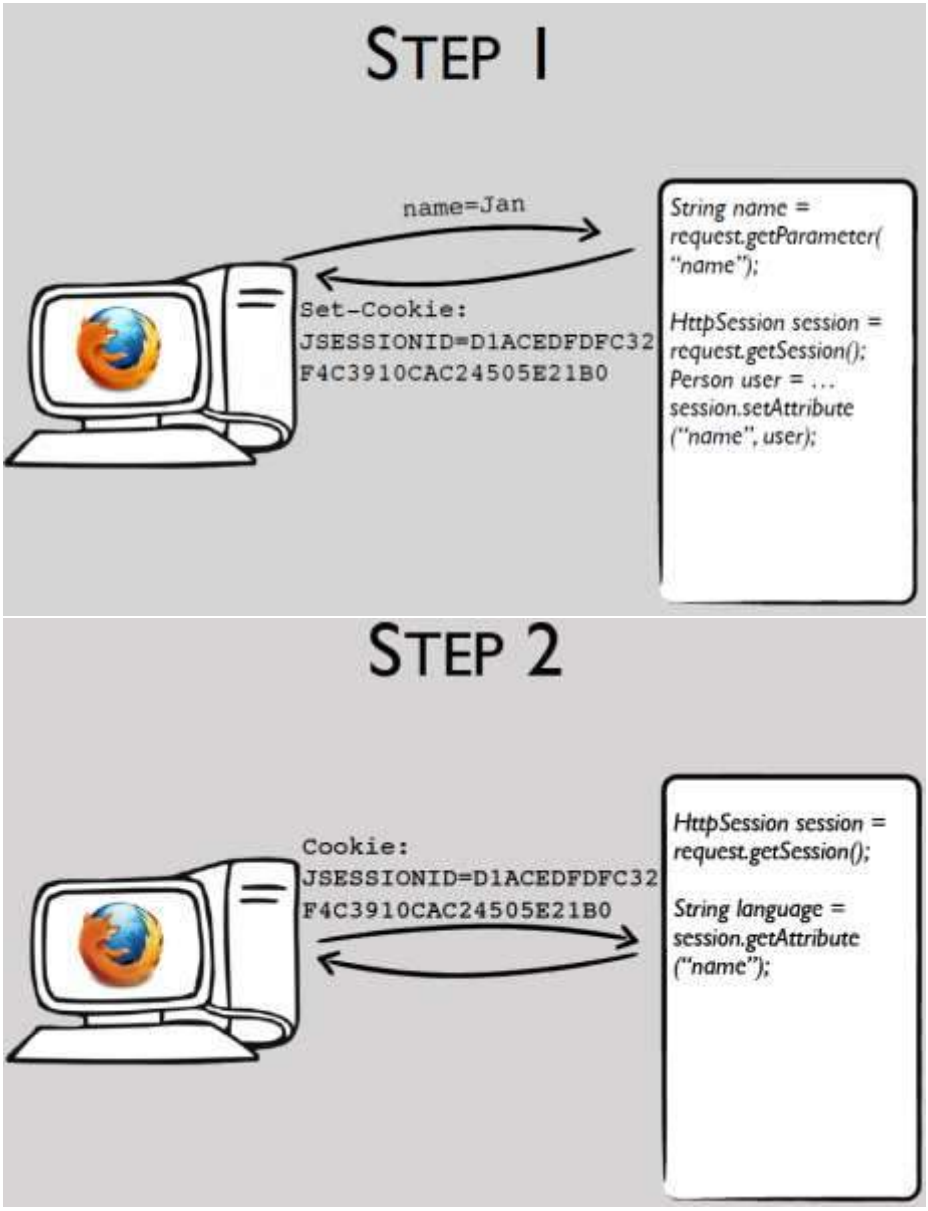
6.4 Important!

- After each creation or setter call:
  - `setPath("/");`
  - `response.addCookie(cookie);`

7. Sessions

7.1 What are sessions

- **Name-value** pairs e.g. `language = english`
- **Why**
  - To save **temporary data**
  - Info about **one client**
- **Where**
  - Made by a '**script**' on the **server**
  - Saved by **server**
  - ID saved by **the client** in its **browser**
- **How**
  1. Server makes a **session object**
  2. ... and **stores the information** in the session
  3. … and gives the **id** of the session to the client in the **response header**
  4. Brower sends the id within each **request header**
  5. Server uses the id to **find** the corresponding session
  6. Server gets the **information** from the session
- **Example**



7.2 Creating and modifying a session

- **`request.getSession()`**
  - Returns existing session OR
  - Creates a new one if no session exists
- **`request.getSession(false)`**
  - Returns existing session OR
  - Returns null if no session exists
- **`session.setAttribute(name, object)`**
  - Sets attribute with given name and object
  - If name already exists, object is replaced
- **`session.getAttribute(name)`**
  - returns object with given name

7.3 Duration

- until **destroyed**
  - `session.invalidate()`

- **until timeout**
  - `session.setMaxInactiveInterval(seconds)`
  - timeout may be defined in web.xml
  - or can be server default

7.4 Using sessions

- On server
- **Safer** than cookies
  - Only sessionID accessible, not data
  - Risk: session hijacking
- Can be **invalidated**
- **More data** possible
- **Point operator** can be used to retrieve attribute properties ("sessionScope" is an implicit object -> sessionScope['variable'])
- Example: shopping cart

7.5 Disabled cookies

- Session ID can still be sent if cookies are disabled
- Check if cookies are disabled:

```
<!DOCTYPE html>
<html>
<head>
   <title>Check whether cookies are enabled yes/no</title>
   <script>
      function myFunction() {
         document.getElementById("demo").innerHTML =
         "Cookies enabled is " + navigator.cookieEnabled;
      }
   </script>
</head>
<body>
   <p>Are cookies enabled in your browser?</p>
   <button onclick="myFunction()">Try it</button>
   <p id="demo"></p>
</body>
</html>
```

- URL rewriting with `<c:url>`
  - Encodes URL
  - Adds session id at the end of URL
  - Only when cookies are disabled

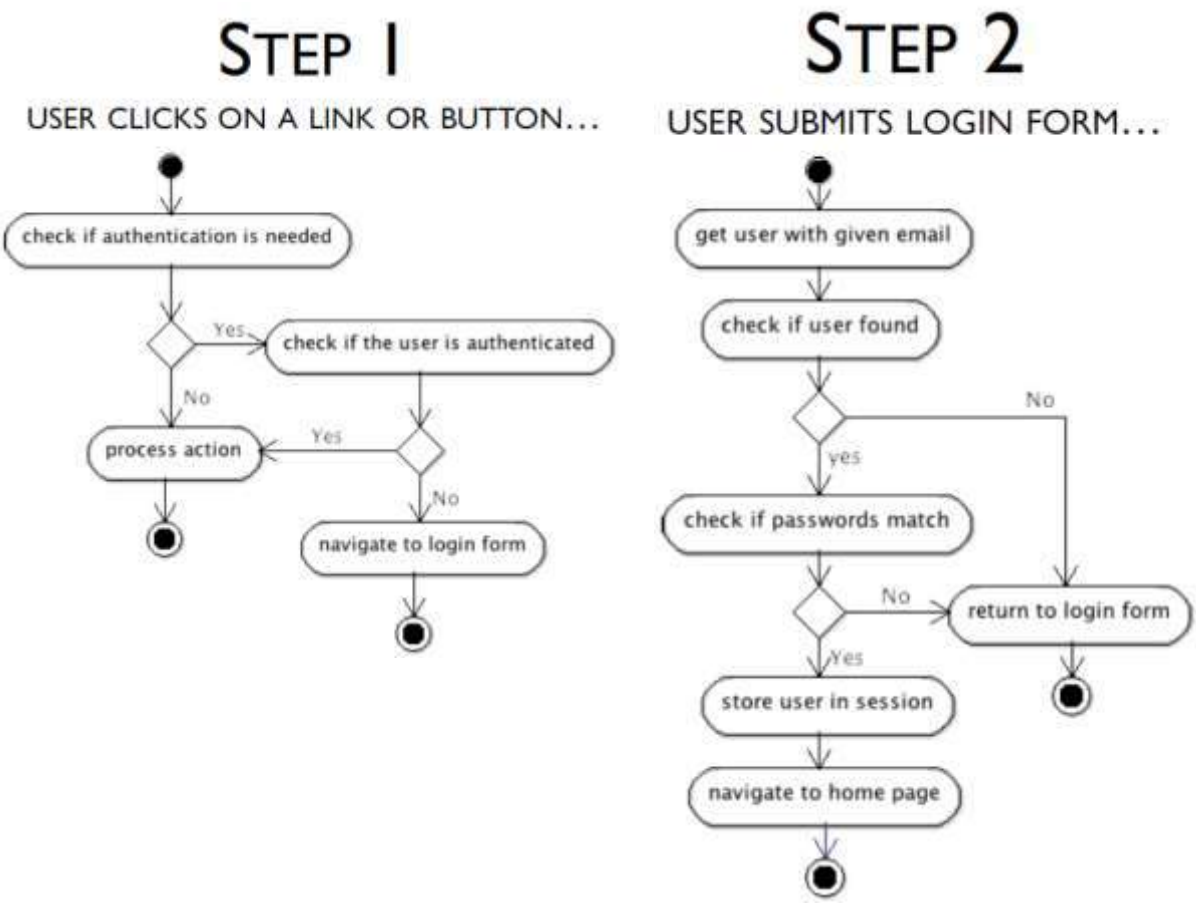
7.6 Alternative: hidden form fields

- Alternative to sessions and cookies

- Field is hidden, thus **not shown to user**
- Parameter is **sent with request** when **form is submitted**
- **Unsafe:** value could be changed with element inspector

8. Authentication

Authentication: check whether a person that logs in can be authenticated (has the right credentials).



For implementation, see slides.

9. Authorization

- **Authorization**
  - check whether a person has the right role (permission)
  - … to access particular pages of the webapp
  - an administrator is authorized to do more than a normal user
- **Example:** slides 6-10:

  Role.java

```
public enum Role {
    ADMINISTRATOR,
    CUSTOMER
}
```

Person.java

```
public class Person {
    private String userId;
    private String password;
    private String salt;
    private String firstName;
    private String lastName;
    private Role role;

    public Person(String userId, String password, String firstName, String lastName, Role role)
{
        setUserId(userId);
        setPassword(password);
        setFirstName(firstName);
        setLastName(lastName);
        setRole(role);
    }

    public Role getRole() {
        return role;
    }

    public void setRole(Role role) {
        this.role = role;
    }
```

Controller.java

```
@WebServlet("/Controller")
public class Controller extends HttpServlet {

…
    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        String destination = "index.jsp";
        …
        try {
        …
        else if (action.equals("addProduct")) {
            destination=addProduct(request,response);
        }
        …
        catch (NotAuthorizedException exc) {
            List<String> errors = new ArrayList<String>();
            errors.add(exc.getMessage());
            request.setAttribute("errors", errors);
            destination="index.jsp";
        }
        request.setAttribute("action", action);
        RequestDispatcher view = request.getRequestDispatcher(destination);
        view.forward(request, response);
    }
```

```
    private String addProduct (HttpServletRequest request, HttpServletResponse response) {
        String destination = "";
        if (isFromUserWithRole(request, Role.ADMINISTRATOR)) {
            destination = "addProduct.jsp";
        }
        else {
            throw new NotAuthorizedException("U heeft niet voldoende rechten om deze pagina te bekijken ...");
        }
        return destination;
    }

    private boolean isFromUserWithRole (HttpServletRequest request, Role role) {
        Person person = (Person) request.getSession().getAttribute("user");
        if (person != null && person.getRole().equals(role)) {
            return true;
        }
        return false;
    }

    …
}
```

NotAuthorizedException.java

```
public class NotAuthorizedException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public NotAuthorizedException (String message) {
        super(message);
    }

}
```

10. Configuration

10.1 Initialize

10.1.1 The configuration file

- **Never hardcode properties** of the site (like DB properties)
- Store properties in **configuration file** (web.xml)
- Read properties when app starts
- Web.xml:

```
javaee/web-app_2_5.xsd"
    version="2.5">
    <welcome-file-list>
        <welcome-file>Controller</welcome-file>
    </welcome-file-list>
    <context-param>
        <param-name>url</param-name>
        <param-value>
            jdbc:postgresql://gegevensbanken.khleuven.be:51415/webontwerp
        </param-value>
    </context-param>
    <context-param>
        <param-name>user</param-name>
        <param-value>daily.build</param-value>
    </context-param>
    <context-param>
        <param-name>password</param-name>
        <param-value>irooZiNgae0daiba</param-value>
    </context-param>
    <context-param>
        <param-name>ssl</param-name>
        <param-value>true</param-value>
    </context-param>
    <context-param>
        <param-name>sslfactory</param-name>
        <param-value>org.postgresql.ssl.NonValidatingFactory</param-value>
    </context-param>
</web-app>
```

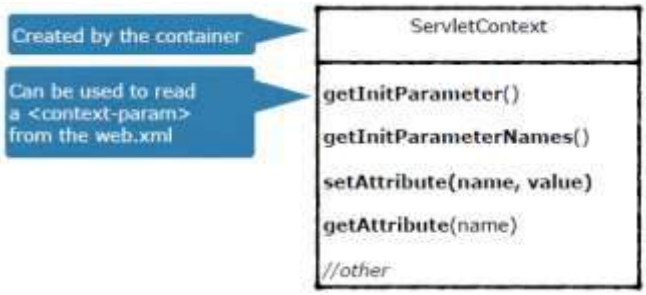
STORE

WEB.XML

how can you access them?

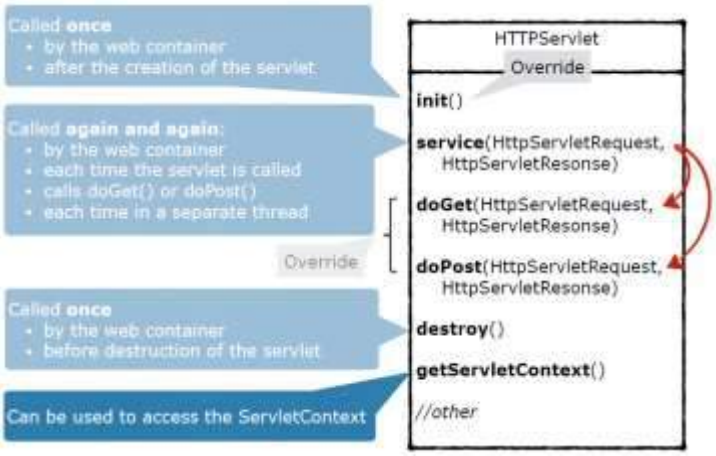
10.1.2 ServletContext

- When app is deployed, container creates 1 **ServletContext** object
- Contains **meta-information about app**
- Has **methods to communicate with container**
- Context parameters of web.xml are **automatically stored** in this object
- Get the object with **HTTPSevlet's getServletContext() method**



10.1.3 Reading properties

- HTTPServlet has method **init()**, called immediately after the creation of the servlet
- When can **override init()** to get the context parameters
- Why in init() method and not in **constructor**: we have to wait for ServletContext object
  - 1. Container creates servlet
  - 2. Container creates ServletConfig and ServletContext
  - 3. Container calls init()



- Example:
  Step 1:

The service class has to be initialized with the properties in the web.xml

called by the web container, after the creation of the servlet

```
@WebServlet("/Controller")
public class Controller extends HttpServlet {
	private ShopService service;

	@Override
	public void init() throws ServletException {

		super.init();

		// TODO get ServletContext object

		// TODO ask ServletContext for properties from web.xml

		// TODO create facade object (service) with these properties

	}
```

Step 2

```
private ShopService service;

@Override
public void init() throws ServletException {

     super.init();

     ServletContext context = getServletContext();

     // TODO ask ServletContext for properties from web.xml

     // TODO create facade object (service) with these properties

}
```

ask for the ServletContext

Step 3

```
private ShopService service;

@Override
public void init() throws ServletException {

     super.init();

     ServletContext context = getServletContext();

     Properties properties = new Properties();
     properties.setProperty("user", context.getInitParameter("user"));
     properties.setProperty("password", context.getInitParameter("password"));
     properties.setProperty("ssl", context.getInitParameter("ssl"));
     properties.setProperty("sslfactory", context.getInitParameter("sslfactory"));
     properties.setProperty("url", context.getInitParameter("url"));

     // TODO create facade object (service) with these properties

}
```

read parameters from web.xml

Step 4

```
private ShopService service;

@Override
public void init() throws ServletException {

     super.init();

     ServletContext context = getServletContext();

     Properties properties = new Properties();
     properties.setProperty("user", context.getInitParameter("user"));
     properties.setProperty("password", context.getInitParameter("password"));
     properties.setProperty("ssl", context.getInitParameter("ssl"));
     properties.setProperty("sslfactory", context.getInitParameter("sslfactory"));
     properties.setProperty("url", context.getInitParameter("url"));

     service = new ShopService(properties);

}
```

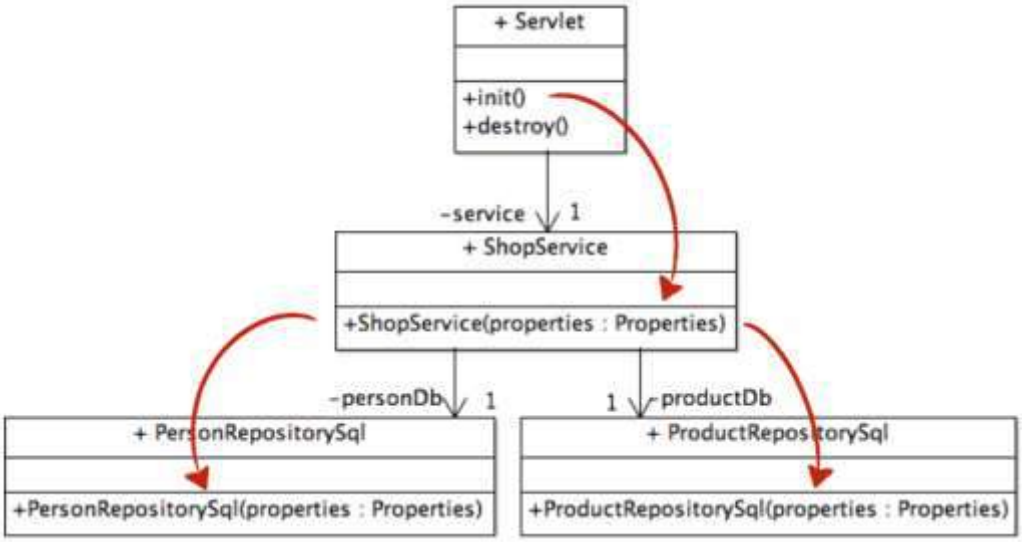
create facade object with properties

Or better:

```
private ShopService service;

@Override
public void init() throws ServletException {

    super.init();

    ServletContext context = getServletContext();

    Properties properties = new Properties();
    Enumeration<String> parameterNames = context.getInitParameterNames();
    while (parameterNames.hasMoreElements()){
       String propertyName = parameterNames.nextElement();
       properties.setProperty(propertyName, context.getInitParameter(propertyName));
    }

    service = new ShopService (properties);

}
```

more flexible

Class diagram



10.2 Finalize

- When the **app goes down**, the method **destroy()** (from HTTPServlet) is called
- Used to close down connections

10.2.1 Application scope

- Example: **1 connection per application** with **100 simultaneous users**:
  - There's **1 connection**
  - **Not much time is lost** creating connections
  - There are **100** statements per connections
  - This is **not scalable**, threads will wait for each other
  - OK for **desktop app,** not for a web app

Servlet:

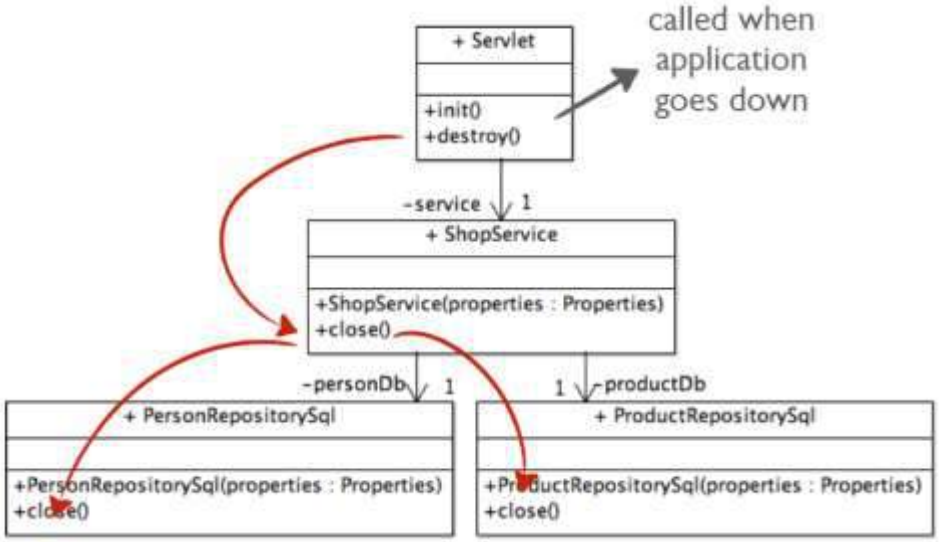
```
@Override
public void destroy() {

      service.close();

      super.destroy();

}
```

Database class:

```
public class PersonRepositorySql {
   private Connection connection;
   private PreparedStatement statement;

   ...

   private void closeConnection() {
      try {
         statement.close();
         connection.close();
      } catch (SQLException e) {
         throw new DbException(e.getMessage(), e);
      }
   }
}
```

in your database class create method to close connection with db

UML:



10.2.2 Request scope

- Example: **1 connection per method** (thread) with **100 simultaneous users**:
  - There's **100 connections**
  - **A lot of time is lost** creating connections
  - There is **1** statement per connections
  - This is **scalable**
  - Better for **web app,** although **performance issue**
  - Solution: **connection pooling**

DATABASE CLASS

open connection in each method

```
public String get(String language) {
   try {
      String url = getProperties().getProperty("url");
      connection = DriverManager.getConnection(url, getProperties());

      String sql = "SELECT * FROM u0082726.greeting WHERE code = ?";
      statement = connection.prepareStatement(sql);

      ResultSet result = statement.executeQuery();
      result.next();
      return result.getString("greeting");
   } catch (SQLException e) {
      throw new DbException(e.getMessage(), e);
   } finally {
      closeConnection();
   }
}
```

close connection after each method

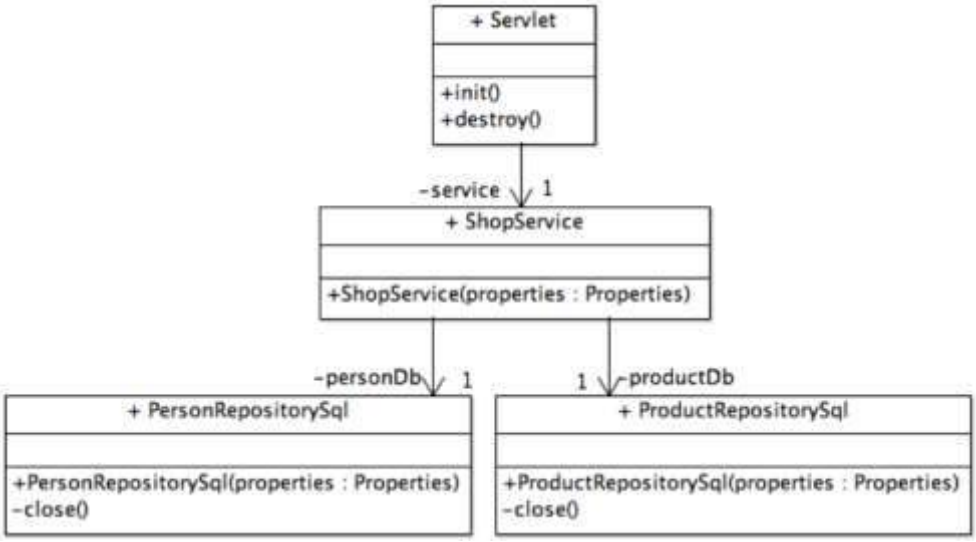
REQUEST SCOPE

TODO

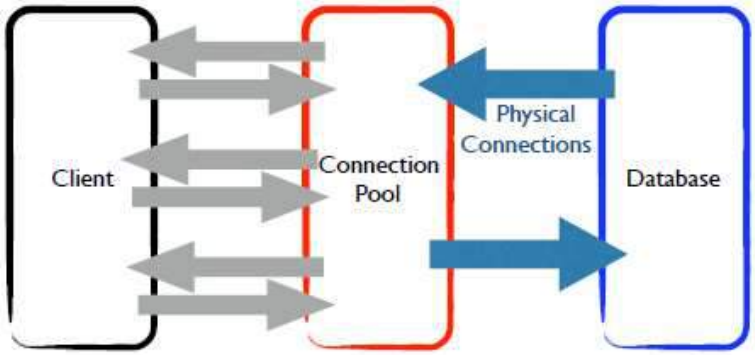
```
public class PersonRepositorySql {
   private Connection connection;
   private PreparedStatement statement;
   private Properties properties;

   public PersonRepositorySql(Properties properties) {
      try {
         Class.forName("org.postgresql.Driver");
         setProperties(properties);
      } catch (Exception e) {
         throw new DbException(e.getMessage(), e);
      }
   }

   ...

   private Properties getProperties() {
      return properties;
   }

   private void setProperties(Properties properties) {
      this.properties = properties;
   }
}
```

store properties as instance variables

UML:



Connection pooling:



10.3 Error Page

JSP:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page isErrorPage="true"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Something wrong</title>
<link rel="stylesheet" href="css/sample.css">
</head>
<body>
   <main>
   <article>
      <h1>Oh dear !</h1>
      <p>You caused a ${pageContext.exception} on the server!</p>
      <p>
         <a href="Controller">Home</a>
      </p>
   </article>
   </main>
</body>
</html>
```

WEB.XML:

```
<error-page>
  <exception-type>java.lang.Throwable</exception-type>
  <location>/error.jsp</location>
</error-page>
<context-param>
  <param-name>url</param-name>
```

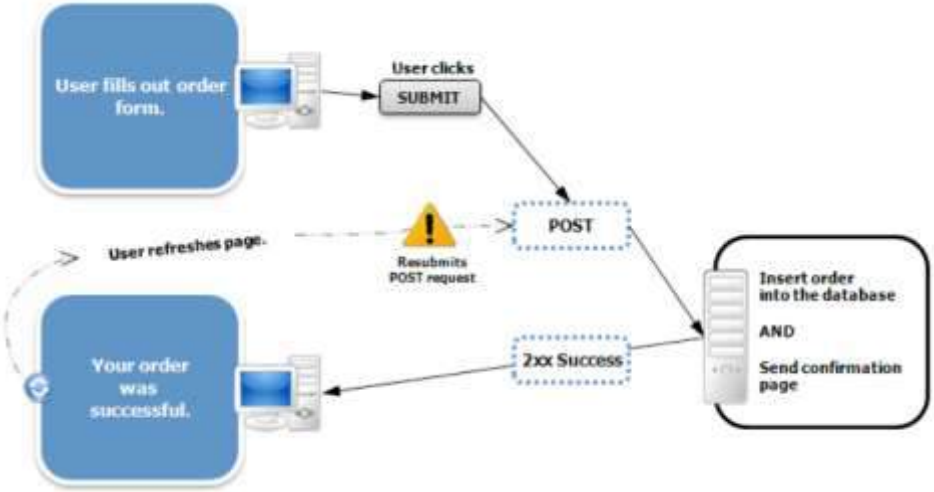
In case of an exception navigate to error.jsp

Remark: 2 ways of exception handling:

- Validation
  - User did something wrong
  - Catch exceptions from model
  - Show message **on the same page**
- Other
  - Something unexpected went wrong
  - **Show error page**

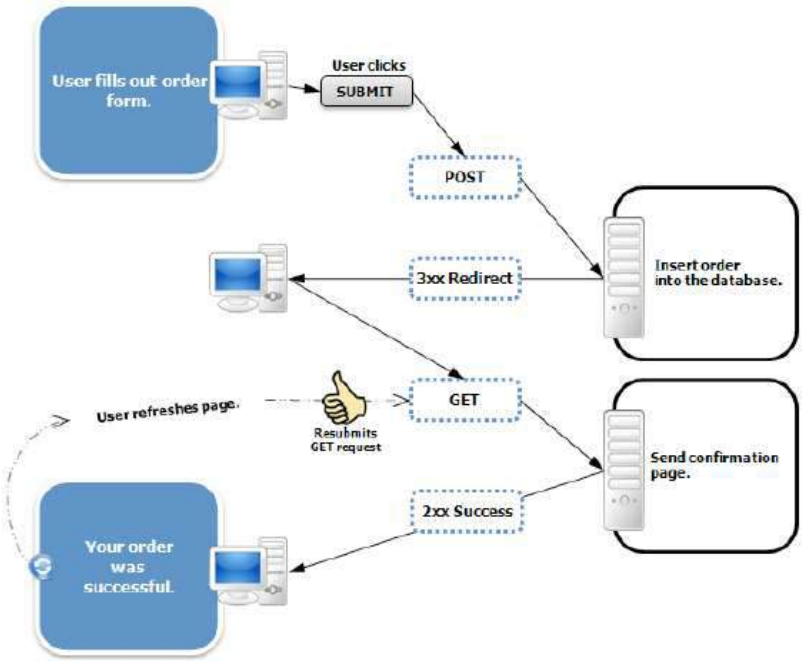
11. Post/Redirect/Get

- **Problem**: user refreshes page after submitting a form with POST (i.e. order confirmation, login,…). Refreshing the page can resend the request, possibly having unwanted effects (i.e. double orders,…)



- **Solution**: Post Redirect Get (PRG). Don't forward the request with `forward()`, but redirect the response using `sendRedirect()`.

```
protected void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
   String action = request.getParameter("action");
   if (action.equals("login")) {
      String userId = request.getParameter("userid");
      String password = request.getParameter("password");
      if (userId.equals("Elke") && password.equals("t")) {
         response.sendRedirect("loggedin.html");
      }
      else {
         request.getRequestDispatcher("index.html").forward(request, response);
         }
      }
   }
```



- **Differences:**

forward()	*sendRedirect()*
Method is executed on the server side	Method is executed on the client side
It can be used within server	It can be used within and outside the server
Information in the request is still available	Information in the request is lost

- **Remark**
  Before PRG, we could forward inside the `processRequest` method. Now we have to move this functionality to the helper methods, as this varies from case to case.

```
public class Controller extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse
         response) throws ServletException, IOException {
      String destination = "index.jsp";
      String action = request.getParameter("action");

      if (action == null) {
         destination = "index.jsp";
      }
      else if (action.equals("overview")) {
         destination = getOverview(request, response);
      }
      else if (action.equals("signUp")) {
         destination = "signUp.jsp";
      }
      else if (action.equals("confirmSignup")) {
         destination=confirmSignup(request,response);
      }
      else if (…) {
        …
      }
      request.setAttribute("action", action);
      RequestDispatcher view = request.getRequestDispatcher(destination);
      view.forward(request, response);
}
```

WARNING

Move to helper methods !

12. Front Controller

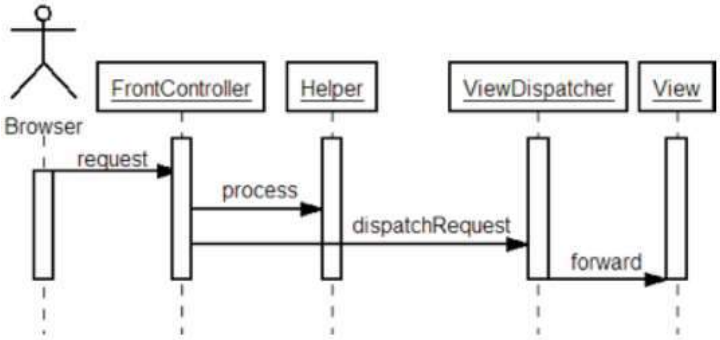
- Single point of entry
  - **Handles requests**
  - **Delegates** (business processing, choice of view,…)
- Use **handlers:**

Single point of Entry

```
public class Controller extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse
         response)
         throws ServletException, IOException {

         String action = request.getParameter("action");
         RequestHandler handler = null;

         if (action.equals("overview")) {
             handler = new PersonOverviewHandler(service);
         } else if (action.equals("signUp")) {
             handler = new SignUpHandler();
         } else if (action.equals("confirmSignup")) {
             handler = new ConfirmSignupHandler(service);
         } else if (action.equals("…")) {
             handler = new … Handler();
         }
         String destination = handler.handleRequest(request, response);
         RequestDispatcher view = request.getRequestDispatcher(destination);
         view.forward(request, response);
      }

   }
```

Use helpers to delegate to model

Use dispatcher to navigate to proper view



- **Problem**: the if/else structure – or switch/case – is not good OO design. There are a few options to solve this problem:

12.1 Option 1

- **ControllerFactory** with **HashMap**<String, RequestHandler> for handlers
- **+** isolated
- **-** violation of OCP
- Example:

```
protected void processRequest(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
   try {
      String action = request.getParameter("action");
      RequestHandler handler =
                        handlerFactory.getHandler(action, service);
      String destination = handler.handleRequest(request, response);
      RequestDispatcher view = request.getRequestDispatcher(destination);
      view.forward(request, response);
   } catch (Exception e) {
      throw new ServletException(e.getMessage(), e);
   }
}
```

```
public class HandlerFactory {
    private Map<String, RequestHandler> handlers = new HashMap<>();

    public HandlerFactory(ShopService service) {
        handlers.put("overview", new PersonOverviewHandler(service));
        handlers.put("signUp", new SignUpHandler());
        handlers.put("confirmSignup", new ConfirmSignupHandler(service));
        …;
    }

    public RequestHandler getHandler(String key) {
        return handlers.get(key);
    }
}
```

12.2 Option 2

- **ControllerFactory** using **Reflection**
- **+** OCP OK
- **-** name action should match handler name → tight coupling view-controller
- Example:

```
public class HandlerFactory {

   private RequestHandler getHandler(String handlerName, ShopService model)
         throws ServiceException {
      RequestHandler handler = null;
      try {
         Class handlerClass = Class.forName("controller."+ handlerName);
         Object handlerObject = handlerClass.newInstance();
         handler = (RequestHandler) handlerObject;
         handler.setModel(model);
      } catch (ClassNotFoundException e) {
         throw new ServiceException(e);
      }

      return handler;
   }
}
```

12.3 Option 3

- **handlerr.xml** in combination with **Factory**
- **+** OCP OK
- **+** name action doesn't have to match handler name
- **-** a lot of XML
- <u>Example:</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <entry key="userOverview">controller.handler.UserOverviewHandler</entry>
    <entry key="confirmSignup">controller.handler.ConfirmSignupHandler</entry>
    <entry key="signUp">controller.handler.SignUpHandler</entry>
     …
</properties>
```

```
private HandlerFactory handlerFactory;

public void init() throws ServletException {

    super.init();
    …
    try {
        ShopService service = new ShopService(…);

        InputStream input = context.getResourceAsStream("/WEB-INF/handler.xml");
        Properties properties = new Properties();
        properties.loadFromXML(input);

        handlerFactory = new HandlerFactory(properties, service);
    } catch (Exception ex) {
        …
    }
}

public class HandlerFactory {
    private Map<String, RequestHandler> handlers = new HashMap<>();

    public HandlerFactory(Properties handlers, ShopService model) {
      for(Object key : handlers.keySet()) {
        RequestHandler handler = null;
        String handlerName = controllers.getProperty((String) key);
        try {
            Class<?> handlerClass = Class.forName(handlerName);
            Object handlerObject = handlerClass.newInstance();
            handler = (RequestHandler) handlerObject;
        } catch (ClassNotFoundException e) {
            …
        }
        handler.setModel(model);
        handlers.put(key, handler);
    }

    public RequestHandler getHandler(String key) {
        return handlers.get(key);
    }
}
```

12.4 Option 4

- **Annotations**
- **+** OCP OK
- **+** name action doesn't have to match handler name
- **+** no configuration file
- Example:

```
@RequestMapping(action="specialties")
public class GetSpecialtiesHandler extends RequestHandler {
  …
}
```

13. Maven

13.1 Build tool

- **Do not depend on your IDE**

- When working in a team, each developer should be able to choose which IDE to use
- Some files may mess with other team members' workspace (.project, .settings, etc)

- **Build & install** can be done
  - By the IDE (e.g. Eclipse: *Run As | Run on Server…*)
  - Manually
  - **By a build tool**
    - Can be started from IDE
    - But is **executed independent from IDE**
    - Can also be used on a server environment, etc.
    - Examples: Ant, Maven, Gradle (Groovy), sbt (Scala), Rake (Ruby), GNU Make, etc.

13.2 Maven

- **Maven Build Script**
  - **Pom.xml**
  - Contains all necessary project dependencies
  - *Dependency injection*
  - Example:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
           http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.khl.demo</groupId>
    <artifactId>hello</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</project>
```

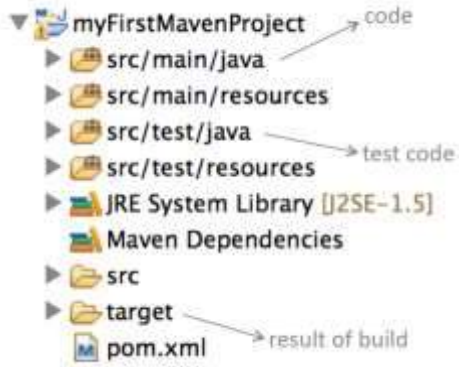
- **Maven Build**
  - Central concept in Maven: **Build lifecycle**
  - A build lifecycle is made up of **phases** (steps of the build process)

Lifecycle Phase
validate
initialize
generate-sources
process-sources
generate-resources
process-resources
compile
process-classes
generate-test-sources
process-test-sources
test-compile
process-test-classes
test
prepare-package
package
pre-integration-test
integration-test
pre-integration-test
verify
install
deploy

- **Convention over configuration**
  - You do not have to write anything if you…

- don't need anything extra
- use the project structure Maven expects

- **Maven project structure**

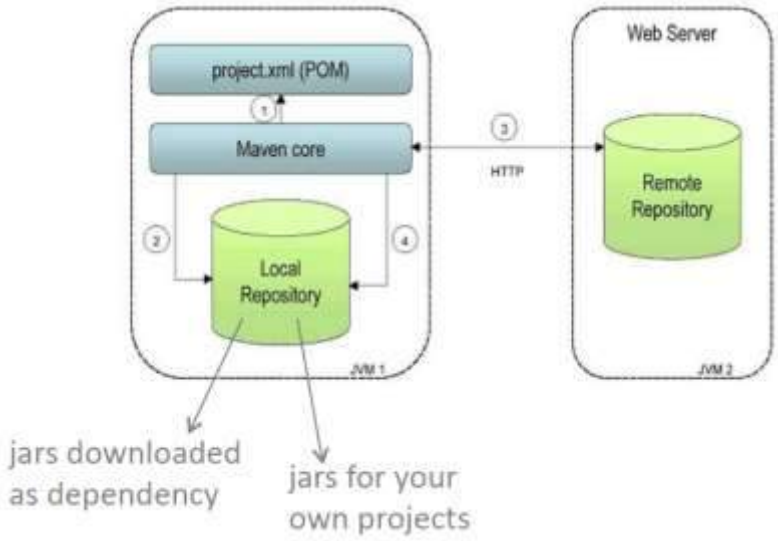


- Example: `maven package`
  - Build script is executed
  - Phase *package* will be executed
  - But previous phases are executed first
  - → **Maven always executes all phases before requested phase**
- **Dependency management**
  - Manage libraries needed for compiling/testing/running
  - I.e. Junit, JSTL, PostgreSQL, …
  - Extra JAR that is needed is added as a dependency in POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
           http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.khl.demo</groupId>
    <artifactId>hello</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <junit.version>4.11</junit.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

→ Jar is:
- downloaded and
- stored in local repository

- **Repositories**
  - **Directory with all the necessary (or more) JAR files**
  - **3 Types**
    - **Local**: on your computer
    - **Central**: provided by Maven community
    - **Remote**: custom (e.g. company server)


project.xml (POM)
1
Maven core
2
Local Repository
4
JVM 1
3
HTTP
Web Server
Remote Repository
JVM 2
jars downloaded as dependency
jars for your own projects