

## สรุป INT303 Web Programming [Final]

### Preparing

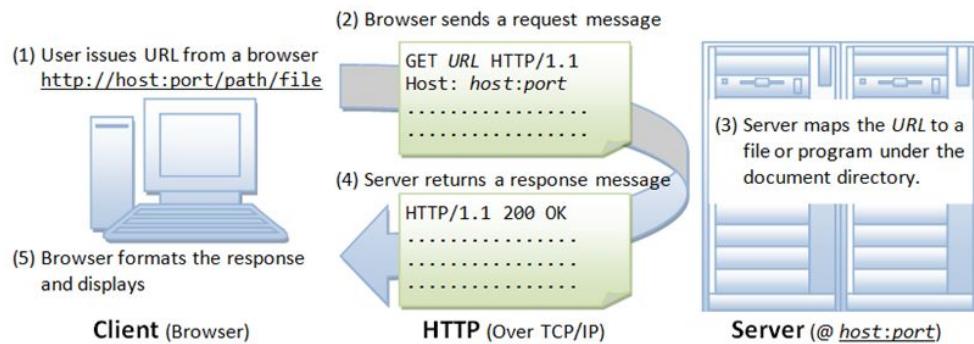
1. java : syntax, conditional, datatypes, OOP, Collection Framework
2. Web : html [form, div, table], css [inline]

### เนื้อหา Web Programming

1. Overview
2. การสร้าง Wep Application
3. Servlet
  - โครงสร้าง
  - การ GET, SET และส่งค่าไปยังที่อื่น
  - session
  - cookie
  - การดึงข้อมูลจาก DB ผ่าน JPA
4. JSP
  - โครงสร้าง
  - Form
  - JSP syntax elements, EL
5. Create and Connect DB with JPA
  - Persistence Unit
  - Entity Class
  - JPA Controller
6. Filter
7. การจัดการปัญหา ERROR

### Overview

- กระบวนการ Request มี 2 แบบหลักๆ คือ GET [Query String], POST [อญญาติ Protocol]
- กระบวนการ Response คือการตอบกลับจาก Server



## MVC Model

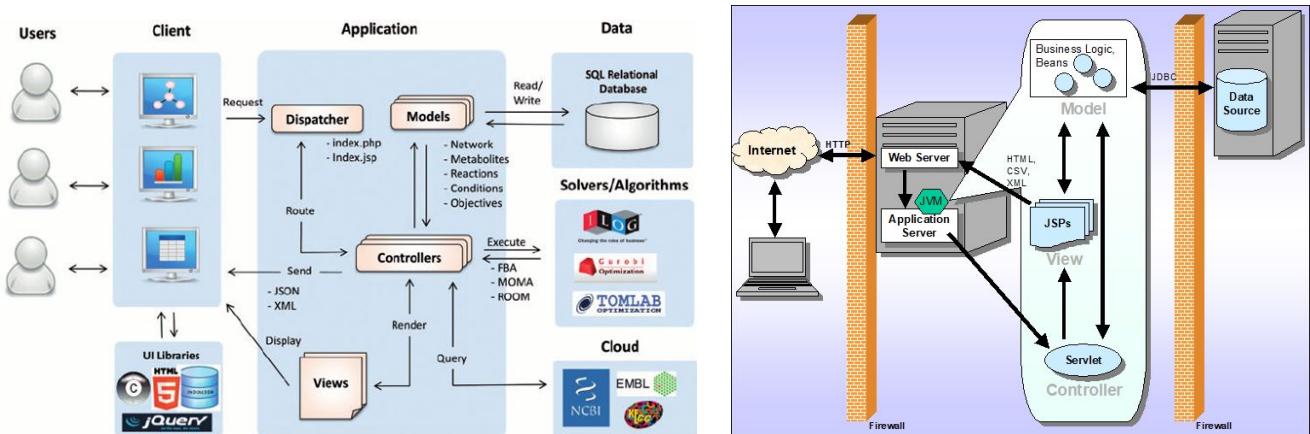
### ❖ องค์ประกอบ

- 1.] Model: Java Class ต่างๆ
- 2.] View: JSP
- 3.] Controller: Servlet

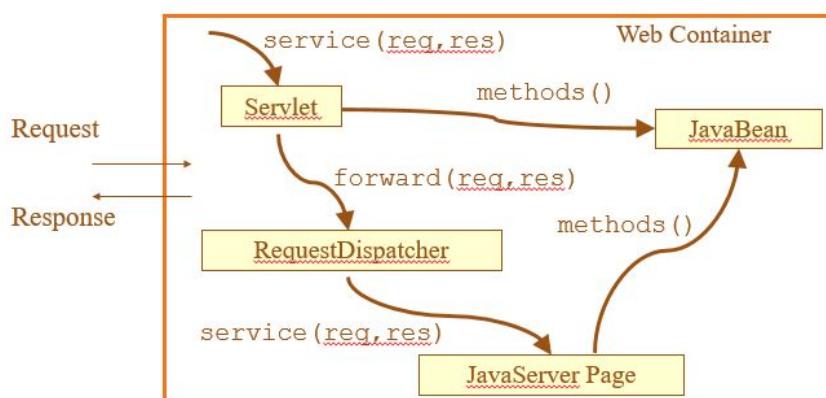
### ❖ การทำงาน

- Request เข้ามาที่ Controller [ตรวจสอบ/เช็คข้อมูลว่าครบไหม เช่น Field ต่างๆ]
- Controller จะ...
  - ส่งไปที่ View เพื่อ Display
  - ส่งไปที่ Models [หัวใจสำคัญของ WebApp] เพื่อกำหนดส่วนต่างๆ เช่นดูเงื่อนไข, เขียนลง Database
- Request----> WebServer [Container จะ Manage ว่าให้ไปเรียกส่วนไหน]
- Web Container ประกอบด้วย model, view, controller และ web.xml
- Web Container จะกำหนดการทำงานให้กับ Servlet

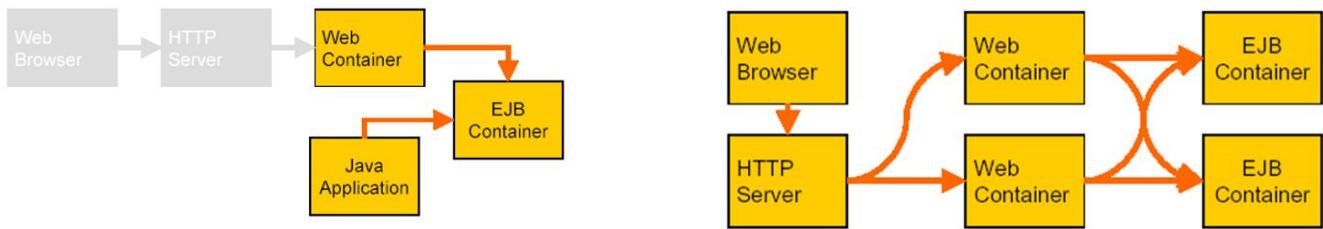
❖ ข้อดี : สามารถ reuse ได้,ลดเวลาการ development, Maintain ได้ง่ายมากขึ้น



## Web application evolution: JSP pages



# Web application evolution: EJBs



- EJBs คือ Enterprise Java Bean
  - Component based architecture
  - แยก Model ออกมาอย่างอิสระ
  - ข้อดีคือ เอา App ตัวเดียวแต่ไปใช้หลายๆ Server ได้ [แยกไปหลาย Container]  
ScaleUp : เมื่อต้องการขยายพื้นที่การรองรับ  
ScaleDown : เมื่อคนใช้น้อยก็ลด App นั้นออกจาก Server เพื่อเอา Resources ไปใช้อย่างอื่น

## Servlet

- Servlet is a standard, server-side component of a Java EE application that executes business logic on behalf of an HTTP request.
  - Servlet ถูกออกแบบมาให้เรียกผ่าน Container ได้ ซึ่ง Servlet run ที่ผู้ให้บริการ server
  - Servlet ทำงานเป็น MultiThread แยกให้ Client แต่ละคนไม่ได้มาแชร์กัน
  - Method แรกที่ Web Container จะเรียกดื้อ service method
    - โดยถ้า request มาแบบ get จะเรียก doGet
    - โดยถ้า request มาแบบ post จะเรียก doPost
  - สำหรับ netBean จัดการให้ doGet, doPost มาเรียก processRequest method

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
```

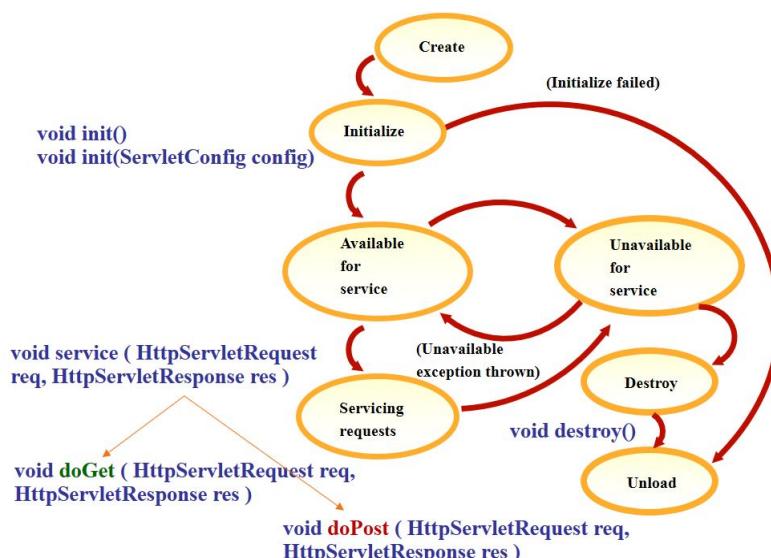
- หากลับ doGet, doPost ออกระบุกำให้เกิด 405

```
@Override  
protected void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
    processRequest(request, response);  
}
```

```
@Override  
protected void doPost(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
    processRequest(request, response);  
}
```

## Java servlet lifecycle

- Container ตรวจสอบว่ามี Servlet ใหม่ถ้าไม่มีส่ง 404
  - ถ้ามี Servlet จะดูว่า new หรือยังถ้ายัง
  - Override : init[], doGet[], doPost[], destroy[]
1. Initialization จะทำการเรียก method void init() หรือ void init(ServletConfig config)
    - a. Initialization is required before a servlet can handle requests from clients.
    - b. The init method performs one-time activities [such as loading of servlet parameters], and initializes costly resources.
    - c. There are two init methods. One takes no input parameters, and one takes a ServletConfig reference as a parameter. The init methods allow the servlet to access name-value pairs for the initialization parameters that are specific to that servlet. The ServletConfig object gives access to the ServletContext object that describes information about the servlet environment.
  2. เรียก Service method : doGet[], doPost[] หากไม่ Error จะ response กลับ
    - a. The servlet accepts client requests and send responses back via the Web server.
    - b. service, doGet, doPost [and so on] methods are called in response to clients [each HTTP request is on a different thread]. The default service method calls the doGet method whenever an HTTP GET request is sent by the Web client [usually as a result of a URL], and the doPost method whenever an HTTP POST request is sent by the Web browser client [usually as a result of an action in an HTML form].
  3. Destroy: void destroy[]
    - The Web container removes the servlet from service when the container needs to conserve memory resources, or when it itself is being shut down.
    - The container destroys and garbage collects the servlet, and calls the destroy method.
    - The destroy method should ensure that all servlet threads have completed, and should undo any initialization work that is not undone automatically by the destruction of the servlet.

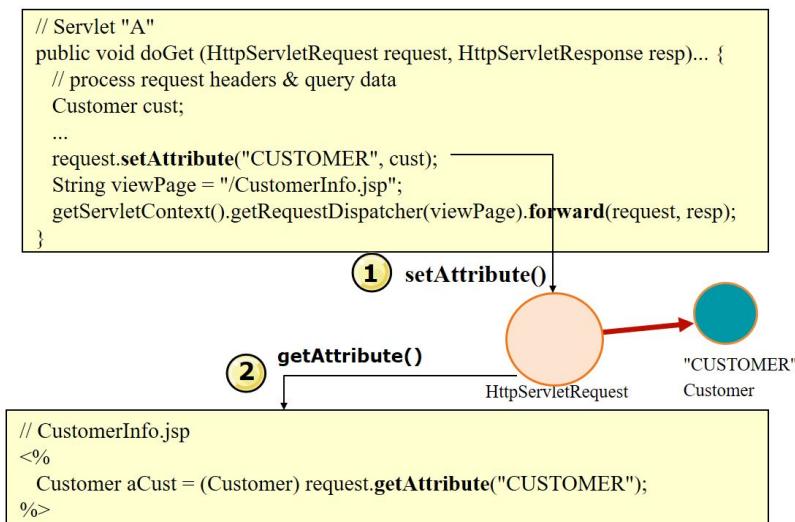


## Request dispatcher

- อุบัติให้ forward request จาก servlet อื่น หรือ include



- **forward** คือการส่งต่อไปที่อื่นแต่เราจะไม่สามารถกำหนดได้อีก [ส่วนใหญ่]
- **include** คือการที่ส่งไปแล้วและส่งกลับมาที่เรา [ส่วนใหญ่ใช้ใน jsp]
- mundrับจะรับเป็น Object ซึ่งจะใช้ JSP EL ในการค้นหา แทนการใช้ Casting

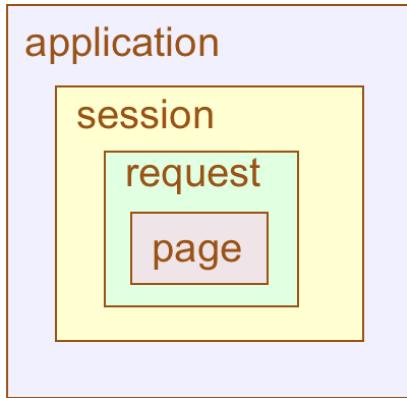


- If a reference to the RequestDispatcher is acquired from the ServletContext Path information is relative to the ServletContext
- If a reference to the RequestDispatcher is acquired from the HttpServletRequest Path information is relative to the path of the current request

## JSP : JavaServer Pages

- technology ที่อุบัติให้เราสามารถ mix ระหว่าง static HTML กับ java ได้
- เช่น server-side scripting ได้
- นามสกุล .jsp
- ประกอบด้วย 2 ส่วนคือ
  1. jsp syntax
  2. Markup tag such as HTML, XML
- JVM run ได้แต่ .class ตั้งนั้น jsp จะต้องแปลง .java ก่อน [โดย Web Container]
- เมื่อเรา jsp deploy ลง server, jsp จะถูก converted เป็น servlet [JSP Servlet] และ run JSP Servlet บน Server หรือเรียกว่า Page Compilation
- STEP : JSP source is parsed, Java servlet code is generated, JSP servlet is compiled, loaded, and run

## Scope Attributes



- Page : แซร์ช้อมูลข้ามเพจไม่ได้
- Request จะไม่สามารถส่ง redirect ได้
- Session: เป็นเหมือนกับเก็บข้อมูล
  - โดย object ของ session จะถูกสร้างที่ฝั่ง Server และจะผูกกับ browser แต่ละตัว
- Application: ทุกคนจะเห็น data เหมือนกัน [แซร์กันทั้ง App]
  - ทั้ง App ใน Server คือ ServletContext ซึ่งแม้ปัจจุบัน browser ไปแล้วก็ยังอยู่

## JSP syntax elements

- แบ่งออกเป็น 4 กลุ่ม
  1. Directive : รูปแบบคำสั่งที่สั่งให้ JSP ทำการเรียกใช้งาน Resource ที่อ้างถึง เช่น `<%@ taglib prefix='prefixOfTag' uri='uri' %>`
  2. Scripting
  3. Comments
  4. Actions : มีคุณสมบัติไว้ให้ เช่น `<% int x = 5 %>`
  - เช่น `<!-- ..... -->`
  - เช่น `<c:forEach >`

## JSP directives [page, include, taglib]

- `<%@ page language='java' %>`  
เช่น `<%@ page contentType='text/html' pageEncoding='UTF-8' %>`
- `<%@ taglib prefix='tagPrefix' uri='tagLibraryURI' %>`  
เช่น `<%@ taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core' %>`
- `<%@ include file='companyBanner.html'%>`

## JSP EL

- รูปแบบการเขียน : \${ ... }
- tag 1 อันเรียก custom tag, หลายอันรวมกันเรียก tag library [tag lib]
- ตัวอย่าง
 

```

<h2> Hello, ${user.firstName} ${user.lastName} </h2>
      
```

user จะถูกหาใน scope ต่างๆ ส่วน firstName จะมาจาก getFirstName() method
- หากหาไม่พบจะไม่แสดง ทำให้ไม่รู้ error
- ตัวอย่าง 2
 

```

<c:if test="${a<3}"> ... </c:if>
      
```

สามารถเปรียบเทียบมากกว่าหน่วยกิโลได้ โดย EL จะไป implements Comparable เอง

## Implicit Objects

- หากเรียกชื่อที่ซ้ำกัน มันจะค้นหาจาก scope ที่เล็ก → ใหญ่
- pageScope < requestScope < sessionScope < applicationScope  
จะไม่มีทางไปถึง Scope ใหญ่ได้ ดังนั้นจะต้องเรียกแบบ \${sessionScope.name}
- param, paramValues : อ้างถึง parameter
- header, headerValues : อ้างถึงข้อมูลใน Header
- cookie : อ้างถึงข้อมูลใน cookie
- initParam
- Example:  
`<%= session.getAttribute("name") %> is equivalent to ${sessionScope.name}`

## Syntax

- สามารถใช้ [ ] แทนการเรียกชื่อตัวแปรได้ในกรณีที่ชื่อตัวแปรมีปัญหา เช่น \${header.accept-language} ซึ่งเกิดปัญหาจึงเปลี่ยนเป็น \${header["Accept-language"]}
- สามารถอ้างถึงข้อมูลได้: Maps, Lists, Arrays of objects, JavaBeans properties
- ในการเรียก JavaBeans properties จะอ้างถึง method ไม่ใช่ attr

### Basic Syntax Elements

- Literals : boolean, integer, floating point, string and null
- Operators : . [] + - \* / % == != < > <= >= && || ! empty ? :
- Ex. \${empty sessionScope ? "yes" : "no"} -----> ว่างไหม  
\${param.cost \* 1.085} -----> คำนวณราคา

## JSTL tags

- JSTL = JSP Standard Tag Library
- สามารถกำหนด scope ของตัวแปรได้ เช่น <c:set var='name' scope='scope' value='expression' />
- สามารถแสดงผลค่าได้ หาก first value เป็น null  
<c:out value='expr' default='expr' escapeXml='boolean' />  
เช่น Hello <c:out value='\${user.name}' default='Guest' />!

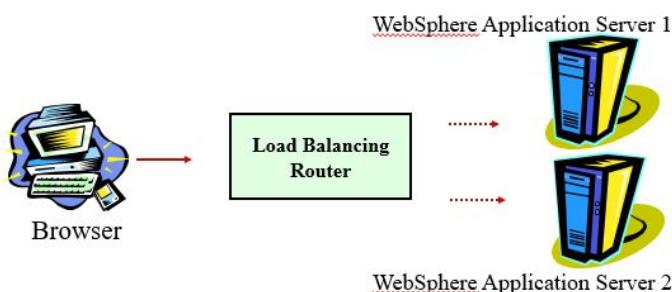
\*\*ถูก Tag อื่นๆได้ใน Part Lab

## Session management

- การจัดการ Session State ของ User แต่ละคน
- ทำให้ Browser แต่ละตัวสามารถเก็บข้อมูลข้ามเพจ request ได้
- provide a way to identify a user across more than one page request or visit to a Web site, and to store information about that user
- Multiple implementation technologies including:  
ใช้ HttpSession , HTTP Cookies , HTML Hidden Field , URL Rewriting

## The load balancing complication

- 1 App สามารถติดตั้งได้บนหลาย Container
- Load Balancing Router จะจัดการแบ่ง request มาหลายๆ Server



- Session ที่เราสร้างเก็บอยู่บน Server ถ้ามีหลาย Server อาจจะไม่สามารถเห็น Session ได้ เพราะเข้ารอบแรกอาจจะเก็บ Session อยู่บน Server1 แต่พอทำบางอย่างเสร็จไปหน้ากัดไปอาจเข้า Server ตัวที่ 2 ทำให้ไม่เห็น Session
- Your application should be **server-independent**
- วิธีการแก้
  1. เก็บ Session บน Client แทน
  2. เก็บ Session Data ไว้บน Server ที่ 3 [third-tier machine]
  3. เก็บ Session Data ไว้บน Server และมี peer-to-peer protocol ให้ Server คุยกันเอง
  4. เก็บ HttpSession Data ลง Shared Database

## Session strategies

Client Side	Server Side
<ul style="list-style-type: none"> <li>▪ Cookies</li> <li>▪ Hidden Fields</li> <li>▪ URL Rewriting</li> </ul>	<ul style="list-style-type: none"> <li>▪ HTTP Session</li> <li>▪ Content Based Routing</li> <li>▪ Store state in a database</li> <li>▪ Distributed Sessions</li> </ul>

## Cookies

- persistent = 怛າວສ
- ເກີບບນເຄຣືອງ Client
- ເປີບເກຕໂນໂລຍຂອງ Http Protocol
- Server ສ້າງແລ້ວສ່າງມາໃຫ້ Client [ໃສ່ຕຽງ Header ຂອງ response]  
Browser ຈະເກີບລົງເຄຣືອງແລະ ເປີນຄນສ່າງກລັບໃຫ້ Server ຜ່ານ Header ຂອງ request
- Cookies have a name and a value
- Cookie ມີອາຍຸ
- Proper cookie usage
  - \* ເກີບເປັນ plain text ດັ່ງນັ້ນອຍ່າເກີບຂັ້ນມູລສໍາດັບ Validation, Secure information
  - \* ຄ້າຈຳເປັນຕ້ອງເກີບຈະຕ້ອງ Encrypted Text

## Cookie API

- Creating cookies [ສ້າງ cookie]  
`Cookie [String name, String value]`
- Sending a cookie back to the browser [ສ່າງ cookie]  
`HttpServletResponse.addCookie(Cookie aCookie)`
- Retrieving cookies [ອ່ານ cookies]  
`HttpServletRequest.getCookies()`
- Retrieving a cookie's name  
`aCookie.getName()`
- Retrieving a cookie's value  
`aCookie.getValue()`
- Changing a cookie's value [ຄ້າເປັນໄວ້ value ຈະຕ້ອງ add ກລັບໄປເພື່ອໃຫ້ Client ຮັບຮູ້]  
`aCookie.setValue(String)`
- ອາຍຸ  
`setMaxAge(0)`      \*\*0 is a request for the browser to delete the cookie.

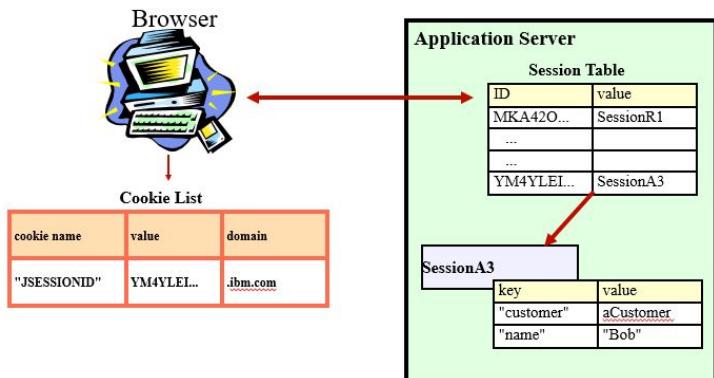
## Cookie Applicability

- have an expiration date
- `setMaxAge [int expiryInSeconds]` : ຕັວເລຂເປັນວິນາກີ ໃຫ້set ດ່າວີນ sec\*min\*hour
- ຄ້າໄມ່ set expiration date : Default ຈະເປັນ -1 [not stored persistently]  
ຈະເຮັດກະທຳກ່າວໜ້າວ່າ `transient cookie [session cookie]`  
ຄື້ວ່າ cookie ຈະຖືກສ້າງຂຶ້ນມາແຕ່ເນື້ອປັດ Browser Cookie ຈະຮ່າຍໄປ

## Session

- The session is unavailable [ไม่ได้หายแต่ใช้ไม่ได้] เมื่อ:
  1. The client browser is closed
  2. The session is explicitly invalidated [ เช่น logout, เปลี่ยน url]
  3. The session times out [หมดเวลา]
- ดูจากตาราง

### Sessions at run time



- ใช้ Session invalidation `session.invalidate ()`
- สามารถกำหนด Session ให้หมดเวลาได้ `session.setMaxInactiveInterval(int second)`
- HttpSession store as <'key', object> pairs

## Thread safety

- shared resource
- เมื่อโอกาสที่ object หนึ่งตัวสามารถทำงาน Multithread ได้ถ้ากำลังทำงานอยู่แล้วมีคนมาดึงข้อมูล ไปแต่งงานยังทำไม่เสร็จ จะทำให้คุณที่ดึงไปได้ข้อมูลที่ไม่สมบูรณ์ ดังนั้นควรป้องกันด้วย `synchronized`
- ตัวอย่าง

```
Customer cust = (Customer) session.getAttribute('customer');
synchronized (cust) {
    // work with the customer object
}
```

## Session serialization

- Objects stored in a session must be serializable:
- To share between servers in a clustered server configuration
- `public class NewObject implements java.io.Serializable {  
 ...  
}`

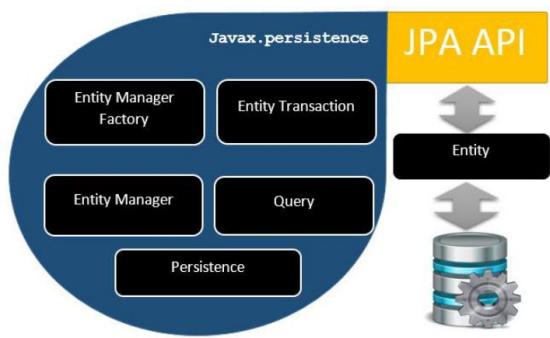
## Java Persistence API [JPA]

- Tool ที่มาช่วยในการจัดการข้อมูลในฐานข้อมูล
- ใน MVC, Model จะเป็นคนเรียกฐานข้อมูล

## Introduction to the Java Persistence API [JPA]

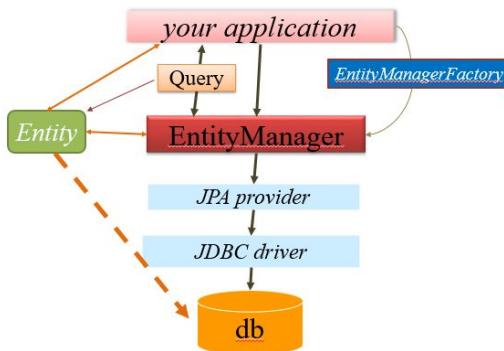
- Persistence คือ Data ที่ถูกเก็บใน Second Storage [Database]
- JPA คือเครื่องมือที่จะช่วยทำ ORM [object/relational mapping] สะดวกขึ้น
- ORM มี 2 ทางคือ แปลง Database---> Object หรือ Object---> Database
- JPA เป็น Open source
- Java ไม่ได้เป็นคนสร้าง JPA แต่เป็นคนกำหนด Spec ดังนั้นจะใช้ของที่ไหนก็ได้ ซึ่งแตกต่างกันเพียงเทคนิคเท่านั้นเช่น Hibernate, Eclipselink, Toplink, Spring Data JPA

## Class Level Architecture



- JAVA กำหนด spec ว่าต้องมี Component [Class] เหล่านี้ใน JPA
- โดย Class เหล่านี้จะไปดำเนินข้อมูลเพื่อให้ได้ Class ที่เป็น Entity [Class ที่ map กับ table] เช่น ถ้ามี Student Table ก็ต้องมี JAVA Class Student [Entity Class]

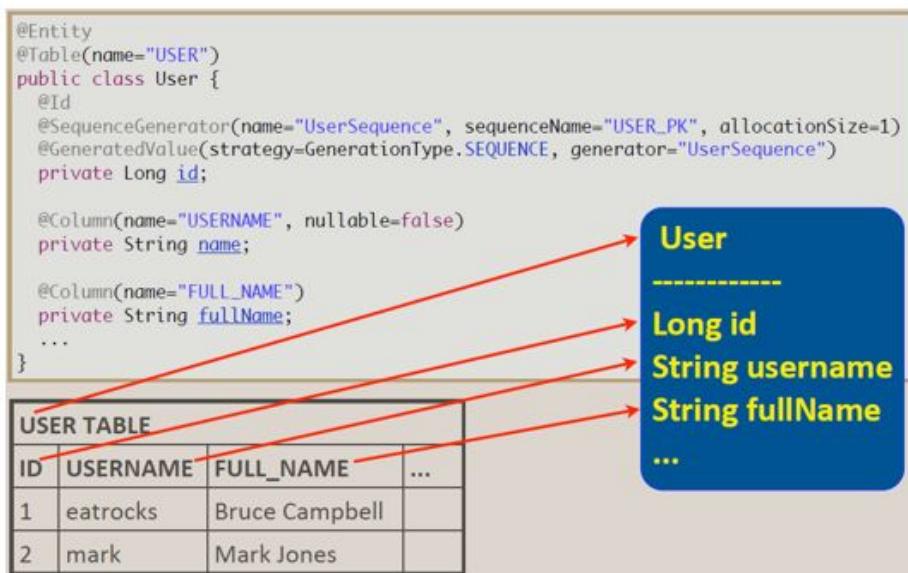
## JPA Class Architecture



- ใช้ Database ผ่าน JDBC ซึ่งปกติจะต้องเขียน Connection เองแต่เรามี `EntityManager` จะเป็นตัวช่วยจัดการและดำเนิน Entity มาให้ ทำให้เราไม่ต้องไปเขียนเอง
- สามารถส่ง query บอกบางอย่างไปให้ `EntityManager` ได้
- `EntityManagerFactory` จะสร้าง `EntityManager` ให้

## Entities

- Entities គឺ Java Class เป็น plain old java object [POJO] គឺមិនមែន getter and setter
- ត្រូវ Class មានតម្លៃ Table, 1 row [record] មានតួនាទី 1 Object
- ក្នុងការសរោះ Entity ត้องការ...
  - + annotated with the javax.persistence.Entity annotation
  - + public or protected, no-argument constructor
  - + the class, methods or persistent instance variables must not be declared final
  - + អាចត្រូវ Serializable តាមតម្លៃបាន
- Persistent Fields and Properties
  - + ដែលមែន JavaBeans-style properties [អាចអនុញ្ញាត generate getters/setters]
- Entity [POJO] Example



## Entity Manager

- EntityManager API សាមរណ creates, removes, finds, queries
- ខ្សោយ EntityManager មានការ EntityManagerFactory
  - និង EntityManagerFactory ត្រូវបានក្រឡាយពី Persistence
  - តាមរយៈ

```

EntityManagerFactory emf = Persistence.
    createEntityManagerFactory("PUnit");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
// Perform finds, execute queries,
...
// update entities, etc.
em.getTransaction().commit();
em.close();
emf.close();
  
```

- PersistentUnit [PUnit] เป็นไฟล์ xml เพื่อบอก information โดยจะต้องตั้งชื่อไฟล์ว่า **persistence.xml files** เท่านั้น
- Transaction คือถ้าทำแล้วเกิดปัญหาสามารถ Roll Back ได้หรือ Commit เพื่อยืนยัน

## Operations on Entity Objects

- persist[] Save the entity into the db บันทึก
- remove[] Delete the entity from the db ลบ
- find[] Find by primary key ค้นหา
- flush[] จัดการข้อมูลที่อยู่ใน cache [Force synchronization of persistence context]
- refresh[] Reload the entity state from the db
- merge[] Synchronize a detached entity with the persistence context
- จัดการ Query เช่น createQuery[], createNamedQuery[], createNativeQuery[]

## JPQL Introduction

- เป็น query language based on SQL [เขียน syntax คล้าย java แต่จะถูกแปลงเป็น sql]
- โดยสามารถสร้างได้ 3 แบบ
  - + EntityManager.createNamedQuery [static query]
  - + EntityManager.createQuery [dynamic query]
  - + EntityManager.createNativeQuery [native query]
- Query API:
  - getResultSet[] - เอาข้อมูลทั้งหมดมาใส่ List
  - getSingleResult[] - execute query returning single result
  - executeUpdate[] - execute bulk update or delete
  - setFirstResult[] - set the first result to retrieve
  - setMaxResults[] - set the maximum number of results to retrieve
  - setParameter[] - bind a value to a named or positional parameter
  - setFlushMode[] - apply a flush mode to the query when it gets run
- Static [Named] Queries : สร้างทึ้งไว้และนำมาใช้ได้

```

@NamedQuery(name="findAllCustomers",
            query="SELECT c FROM Customer")
Query findAllQuery =
entityManager.createNamedQuery("findAllCustomers");
List customers = findAllQuery.getResultList();

```

- สามารถเป็น Multiple Named Queries ได้

## Named Parameters

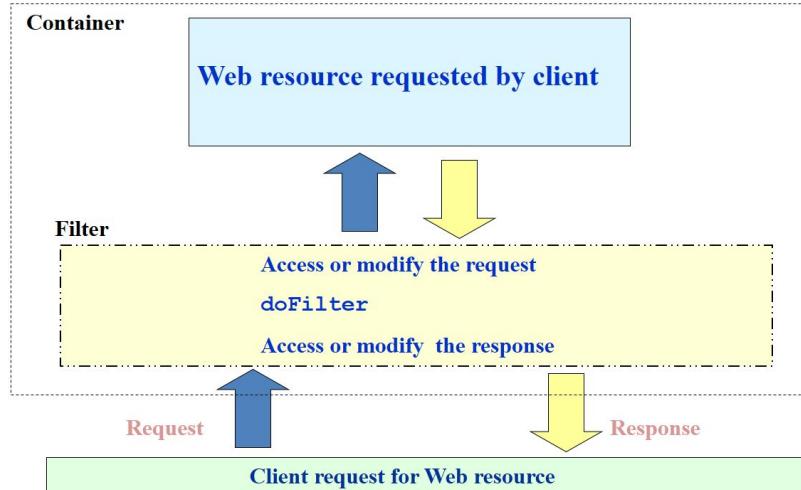
- ใน query สามารถส่ง Parameter ได้
- โดย c ต่อ Object

```
public List findWithName(String name) {  
    return em.createQuery(  
        "SELECT c FROM Customer c WHERE c.name LIKE :custName")  
        .setParameter("custName", name)  
        .getResultList();  
}
```

## Servlet Filtering

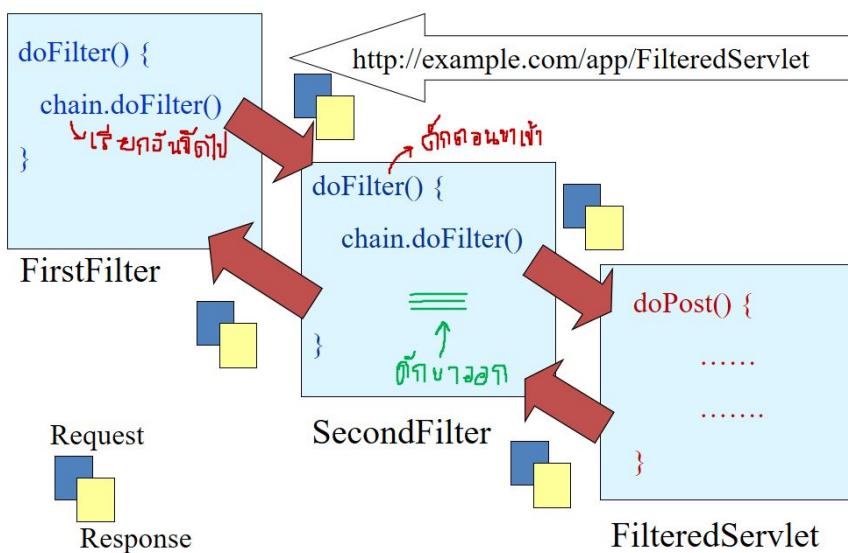
- ในส่วนที่ใช้ช้าหลายๆ Flow [reusable components]
- HTTP requests, responses, and headers
- สามารถซ้อนกันหลายชั้นได้ [chains of multiple filters]
- สามารถ config ได้ว่าให้ผ่าน Filters ตัวไหนก่อนเข้าถึง Web Resource
- Are indirectly invoked by client request for a Web resource
- เช่น ปกติในเวลาส่งเอกสารให้พนักงาน [แทน Servlet] พนักงานจะทำหน้าที่ตรวจสอบเอกสารก่อนจึงค่อยนำเข้าระบบ เราอาจจะเพิ่มคน 1 คน [Servlet Filter] เข้าไปเพื่อตรวจสอบเอกสารก่อนส่งให้พนักงาน
- กรองได้ทั้งขาเข้า และขาออก
- หน้าที่
  - Process the request
  - Process the response
  - Transfer control to the next filter or Web resource in chain [ส่งให้ Filter ตัดไป]
- Typical uses of filters
  - Authentication filters
  - Logging and auditing filters
  - Data compression filters
  - Encryption filters
  - XSLT filters to transform XML content

- Filter processing flow



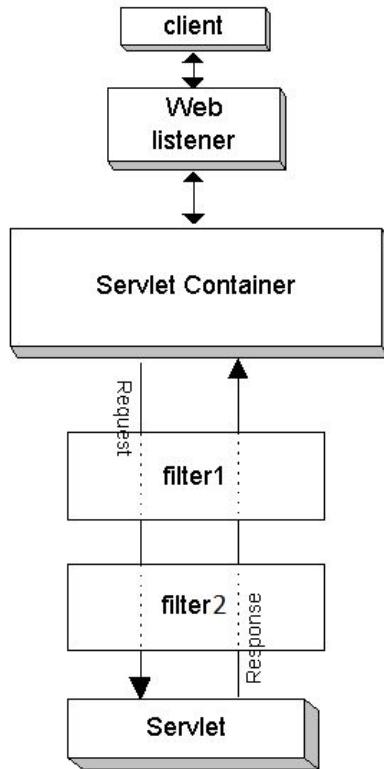
- config ใน web.xml ว่าจะให้ผ่าน Filter ไหนก่อนเข้าถึง Web Resource
- วิธีเขียน Filter
  - Access or modify the request  
doFilter
  - Access or modify the response

- Filter สามารถถูกเรียกเป็นลู กโซชได้
- FilterChain object โดยมี method ที่จะเรียกคือ doFilter()
- Class นั้นจะต้อง Implement มาจาก javax.servlet.Filter
- โดยมี Parameters 3 อัน
  - Request of type ServletRequest
  - Response of type ServletResponse
  - Chain of type FilterChain
- \*\* เกิดการ Upcasting เช่น Object x = new String[“Hello”]; → Polymorphism
- doFilter[] nested calls



- chain.doFilter[] จะดูว่ามีอันถัดไปไหม[ดูใน web.xml]

- ຕອນທີ request Servlet Container ຈະ Map ໃຫ້ເຂົ້າ Filter ກ່ອນ



- Implementing a filter
  1. สร้าง Class [New-->Other-->Web-->Filter]
  2. Override Method
    - a. init[] → เพื่อเป็นการ initialization [ถูกเรียกครั้งแรกและครั้งเดียว]
    - b. doFilter[]
    - c. destroy[]
- ໃນ FilterChain ກີ່ມີ doFilter[] method

## Configuring filter chaining

- Rules:
  - ຈະກຳ url-pattern ກ່ອນ servlet-name ແລະ ຄ່ອຍໄລ່ຈາກບນລອງລ່າງ
- ຕັວອຢ່າງ

```

<filter-mapping>
    <filter-name>FormChecker</filter-name>
    <servlet-name>Prime</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>Logger</filter-name>
    <url-mapping>/*</url-mapping>
</filter-mapping>
<filter-mapping>
    <filter-name>PrimeTrailer</filter-name>
    <servlet-name>Prime</servlet-name>
</filter-mapping>
  
```

## Dispatcher element

- ❖ New <dispatcher> element in the deployment descriptor:
  - REQUEST
    - Filter if request is directly from a client
  - FORWARD
    - Filter if request is from RequestDispatcher.forward method
  - INCLUDE
    - Filter if request is from RequestDispatcher.include method
  - ERROR
    - Filter if request is due to error redirection mechanism
- ❖ REQUEST is the default when there is no <dispatcher> element

## PROJECT

### Create Project

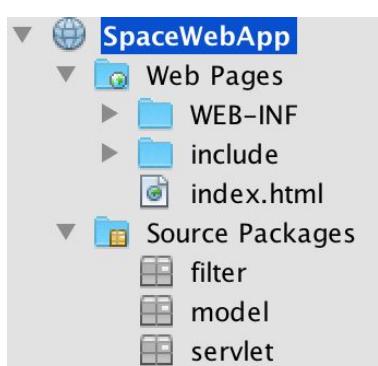
1. New Project / Java Web / Web Application

โดยตั้ง Context Path: /ชื่อที่เราอ้างถึงเวลาDeployServer

ตัวอย่าง    ← → C    ⓘ localhost:8080/Space/

2. จะได้ Project ให้คลิกขวาเพื่อ Deploy และกด Run

3. โครงสร้าง Project



- Web Pages
  - WEB-INF : เก็บ web.xml
- Source Packages
  - filter
  - model
  - servlet

**web.xml**

```

<servlet>
    <servlet-name>AddServlet</servlet-name> -----> ชื่อ
    <servlet-class> servlet.AddServlet</servlet-class> -----> Class
</servlet>

<servlet-mapping>
    <servlet-name>AddServlet</servlet-name> -----> ชื่อ
    <url-pattern>/AddServlet</url-pattern> -----> เรียก
    <url-pattern>/Add</url-pattern>
</servlet-mapping>

```

**การสร้าง Servlet [Controller]**

1. New File / Servlet ให้ตั้งชื่อ File ลงท้ายด้วย Servlet เช่น WeightConverterServlet
2. ให้ Add information ลง web.xml และตั้ง URL Pattern เป็น /WeightConverter
3. Init Param

```

<init-param>
    <param-name>name</param-name>
    <param-value>Welcome</param-value>
</init-param>

```

```

String name = getServletConfig().getInitParameter("name");
out.println(name);

```

**4. โครงสร้างไฟล์**

```

public class WeightConverterServlet extends HttpServlet {
    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // code
    }
    HttpServlet methods. Click on the + sign on the left to edit the code.
}

```

**5. แสดงผลใช้ PrintWriter**

```

PrintWriter out = response.getWriter();
out.println("<p>Hello</p>");

```

## การ GET, SET และส่งค่าไปยังที่อื่น

- GET ค่า

ตัวแปร = request.getParameter["x"] จะดึงค่ามาจาก <input name="x" > ใน view  
 \*\*รับเข้ามาเป็น String หากเก็บเป็นตัวเลขควรใช้ Integer.parseInt() หรือ .valueOf();

- SET ค่า

request.setAttribute("ที่จะส่งชื่อไป", ตัวแปร);

- ส่งค่าไปยังที่อื่น

แบบ 1: forward

getServletContext().getRequestDispatcher("/ชื่อ.jsp").forward(request, response);

อธิบาย : RequestDispatcher คือตัวที่อนุญาตให้ forward request จาก servlet อื่นมาทำงานได้โดย path จะเริ่มจาก root ซึ่งถ้าเรียกที่หน้าเดิมจะไม่เปลี่ยนเนื่องจากเป็นการทำงานผู้้ Server

แบบ 2: sendredirect

response.sendRedirect("ชื่อServlet");

อธิบาย : จะ response กลับไปหา client ซึ่งถ้าเรียกที่หน้าเดิม url ก็จะเปลี่ยนเมื่อൺกับการคลิก link ใหม่ ดังนั้น Path จะนับจากตำแหน่งที่อยู่ปัจจุบัน

เช่น AddItemToCartServlet, RemoveItemFromCartServlet, ChooseBackgroundServlet

## การดึงข้อมูลใส่ List

- สร้าง List

```
String[] subjects = request.getParameterValues('subjects');
List<String> subjectList = new ArrayList();
for [String subject : subjects] {
    subjectList.add(subject);
}
```

## การใช้ session

- สร้าง Session

```
HttpSession session = request.getSession(true); //true-ไม่มีสร้างใหม่
String textMessage = request.getParameter("message");
ObjectTypes objX = [ObjectTypes] session.getAttribute("x"); // down casting
if [objX==null] { //ถ้ายังไม่มีค่า
    //-----ในกรณีต้องการสร้าง Session-----
    objX = new ObjectTypes [];
    session.setAttribute("x", objX); //ทำให้เป็น Obj
} //set session
objX.setMessage(textMessage) //method
```

- ใช้ Session Invalidate

```
session.invalidate();
```

- ตั้งอายุ : default timeout = 30 minutes

```
session.setMaxInactiveInterval [int]
```

## การใช้ Cookie

- สร้าง Cookie

```
Cookie ck = new Cookie["ชื่อ", ค่า];
```

```
ck.setMaxAge[60*min*hour*day]; //หน่วยเป็นวินาที
```

```
response.addCookie(ck);
```

- ใช้ \${cookie.bgColor.value}

## การดึงข้อมูลจาก Database

- เขียน Annotation

```
@PersistenceUnit [unitName = "dbPU"]
```

```
EntityManagerFactory emf;
```

```
@Resource
```

```
UserTransaction utx;
```

- สร้าง Controller

```
TablenameJpaController ctrl = new TablenameJpaController[utx, emf];
```

- method ที่ใช้

1. เพิ่ม record เข้า DB:	ctrl.create[object]	[INSERT]
2. ลบ record ออกจาก DB:	ctrl.destroy[id];	[DELETE]
3. แก้ไขข้อมูลใน record:	ctrl.edit[object];	[UPDATE]
4. ดับเบิล record ใน DB:	ctrl.findName[id];	[SELECT ONE]
5. เอาข้อมูลทั้งหมด	ctrl.findNameEntities[];	[SELECT ALL]

\*\* เช่น List<Product> products = productJpaCtrl.findProductEntities();

- หากต้องใช้ try-catch ให้อยู่ใน blocked scoped นั้น

## การสร้าง JSP [View]

1. New File / JSP ซึ่งจะได้นามสกุลไฟล์เป็น .jsp

## 2. ตัวอย่างโครงสร้างไฟล์

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>SIT SPACE</title>
  </head>
  <body>
    <style>
      h1{
        text-align: center;
      }
    </style>
    <h1>Weight Converter</h1>
    <form action="ชื่อServlet" method="post">

    </form>
  </body>
</html>

```

## การใช้ FORM

- action ห้ามใส่ "/" นำหน้า เพราะมันจะไปเรียกที่ WebServer และ AppServer
- Tag ที่ใช้

<input type= "text"	name="username" required>
<input type= "password"	name="username" required>
<input type= "submit"	value="ซื้อที่แสดงบนปุ่ม">
<input type= "radio"	name="gender" value="male">
<input type= "checkbox"	name="subject" value="GEN">

## การใช้ JSP EL

- รูปแบบการเขียน : \${ ชื่อ.เมธอด } เช่น \${user.firstName}
- เช่น <h2> Hello, \${user.lastName} </h2>
- หาก EL พังลองตรวจสอบ <%@ taglib prefix='c' uri='.....' %> ussทั้งหมด
- การเช็คเงื่อนไข

แบบ 1 :

`${เงื่อนไข ? "จริง" : "เท็จ"}`  เช่น \${value == '#050E54' ? 'checked': ''}

แบบ 2 :

```

<c:if test="${message !=null}">
<p>Welcome, member!</p>
</c:if>

```

แบบ 3 :

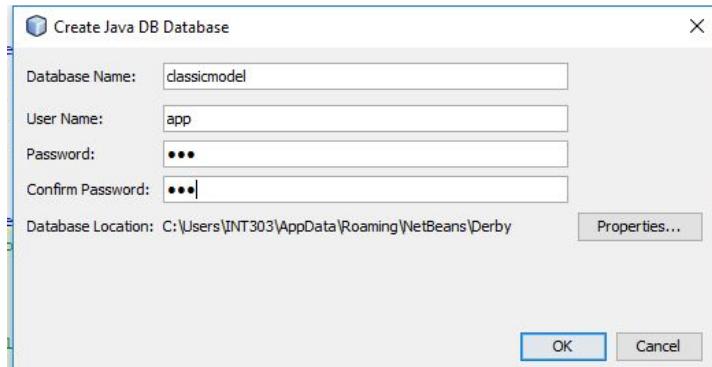
```
<c:choose>
    <c:when test='${user.role == 'member'}'>
        <p>Welcome, memberName!</p>
    </c:when>
    <c:otherwise>
        <p>Welcome, guest!</p>
    </c:otherwise>
</c:choose>
```

- การใช้ Loop

```
<c:forEach items=“${products}” var=“p” varStatus=“vs” >
<tr>
    <td>${vs.count}</td>
    <td>${p.name}</td>
</tr>
</ c:forEach >
```

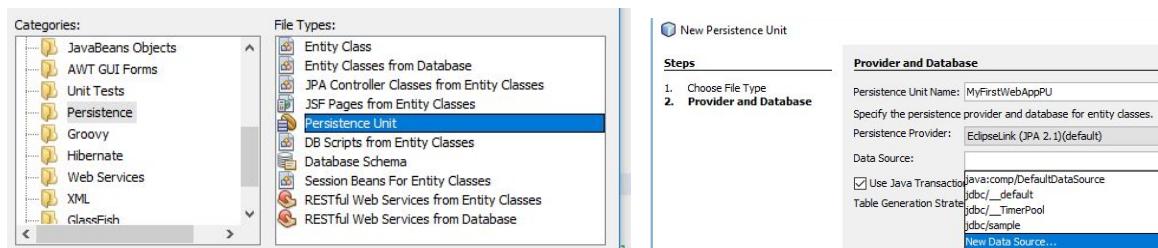
## การสร้างและเชื่อมต่อ DB ผ่าน JPA

### 1. สร้าง Java DB Database



### 2. create table

### 3. สร้าง Persistence Unit / โดย Datasource ให้สร้างใหม่และเลือก DB ที่ต้องการ Connect



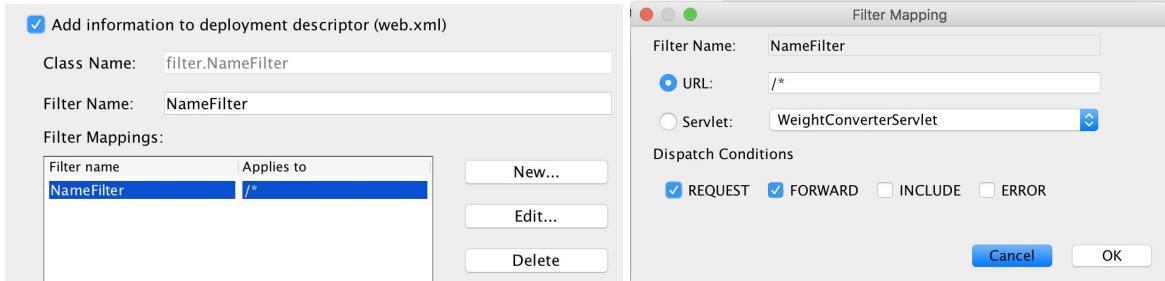
\*\* เช่น JNDI Name: jdbc/classicmodel

### 4. สร้าง Entity Class / Add ตารางที่ใช้ / Collection Type กำหนดเป็น List

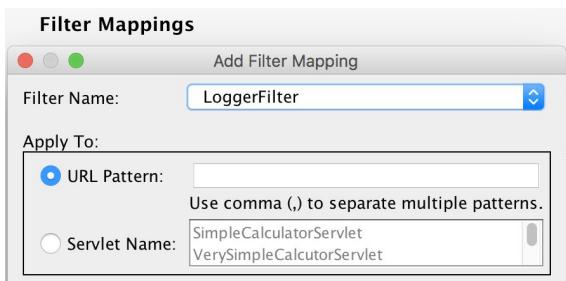
### 5. สร้าง JPA Controller จาก Entity Classes

## การทำ Filter

1. New File / Filter และตั้งชื่อ NameFilter
2. add to web.xml โดย mapping ว่าจะใช้กับServlet ตัวไหน [edit] \*\*ถ้า /\* คือทุก filter



3. เมื่อสร้างเสร็จสามารถ add ให้ไป mapping กับ Servlet ตัวไหนเพิ่มได้ใน web.xml



4. โครงสร้างไฟล์ [ใช้ method ทั้งหมดและ implements มา]

```
public class LoginFilter implements Filter {
    private FilterConfig config;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        this.config = filterConfig;
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws ServletException, IOException {
        //do something before
        chain.doFilter(request, response);
        //do something when comeback
    }

    @Override
    public void destroy() {
    }
}
```

- การดึงข้อมูล ผ่าน [`HttpServletRequest`] request

1. Session

```
HttpSession session = [(HttpServletRequest) request].getSession(false);
```

2. url

```
String url = [(HttpServletRequest)request].getRequestURI();
```

- Check Session

```
if [session ==null || session.getAttribute("name") == null]
```

- เข้าถึง ServletContext []

```
config.getServletContext().getRequestDispatcher('/Servlet').forward[request, response];
```

## ตัวอย่าง 1 : getParameter

Servlet

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String name = request.getParameter("name");
    String weighttext = request.getParameter("weight");
    double weight = Double.parseDouble(weighttext);

    request.setAttribute("name", name);
    request.setAttribute("weight", weight);

    getServletContext().getRequestDispatcher("/WeightConverterView.jsp").forward(request, response);
}
```

JSP

```
<h1>Weight Converter</h1>
<form action="WeightConverter" method="post">
    Name: <input type="text" name="name" /><br>
    Weight: <input type="number" name="weight" /> kg<br>
    <input type="submit" value="ENTER">
</form>
<div>
    <h5>DATA THAT WE GET:</h5>
    <p>
        NAME: ${name}<br>
        WEIGHT: ${weight} kg
    </p>
</div>
```

## ตัวอย่าง 2 : Session

Servlet

```
HttpSession session = request.getSession(true);
request.setAttribute("name", name);
request.setAttribute("weight", weight);
Astronomer ast = (Astronomer) session.getAttribute("ast");
if(ast==null){
    ast = new Astronomer();
    session.setAttribute("ast", ast);
}
ast.setName(name);
ast.setWeight(weight);
```

JSP

```
<div>
    <h5>DATA THAT WE GET:</h5>
    <p>
        NAME: ${ast.name}<br>
        WEIGHT: ${ast.weight} kg<br>
        <b>Convert = ${ast.result}</b>
    </p>
</div>
```

## ตัวอย่าง 3 : Authentication Filter

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
    HttpSession session = ((HttpServletRequest)request).getSession(false); //If not have session
    if(session==null||session.getAttribute("user")==null){ //if cart not have user
        config.getServletContext().getRequestDispatcher("/Login").forward(request, response);
        return;
    }else{
        chain.doFilter(request, response);
    }
}
```

## ตัวอย่าง 4 : Logout

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    System.out.println("-----Logout----");
    HttpSession session = request.getSession(false);
    if (session != null) {
        session.invalidate();
        System.out.println("-----Session Invalidate-----");
        response.sendRedirect("index.html");
    }
}
```

\*\*ใช้ sendRedirect

## ตัวอย่าง 5 :TaskServlet

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    HttpSession session = request.getSession();
    Account account = (Account) session.getAttribute("account");
    String newTask = request.getParameter("todolist");
    TaskJpaController ctrl = new TaskJpaController(utx, emf);
    if (newTask != null && newTask.trim().length() > 0) {
        Task task = new Task(newTask, account);
        try {
            System.out.println("-----TRY-----");
            ctrl.create(task);
            System.out.println("-----CREATE FINISH-----");
        } catch (Exception ex) {
            Logger.getLogger(TaskServlet.class.getName()).log(Level.SEVERE, null, ex);
        }
        //getServletContext().getRequestDispatcher("/Task.jsp").forward(request, response);
        response.sendRedirect("Task");
    } else {
        List<Task> taskList = ctrl.findTaskEntities();
        request.setAttribute("taskList", taskList);
        getServletContext().getRequestDispatcher("/Task.jsp").forward(request, response);
    }
}
```

## ตัวอย่าง 6 :LoginServlet

```

Account account = (Account) session.getAttribute("account");
if (username != null && password != null) {
    String cryptPass = cryptWithMD5(password);
    AccountJpaController ctrl = new AccountJpaController(utx, emf);
    Account acc = ctrl.findAccount(username);
    if (acc != null) {
        if (cryptPass.equals(acc.getPassword())) {
            if (account == null) {
                account = new Account();
                session.setAttribute("account", account);
            }
            account.setUsername(username);
            account.setPassword(cryptPass);
            getServletContext().getRequestDispatcher("/Task").forward(request, response);
            return;
        }
    }
    request.setAttribute("message", "Try Again");
    getServletContext().getRequestDispatcher("/Login.jsp").forward(request, response);
} else {
    getServletContext().getRequestDispatcher("/Login.jsp").forward(request, response);
}
}

```

## ตัวอย่าง 7 : AllStudent

```

public class AllStudent {
    private Map<String, Student> allData;

    public AllStudent(){
        allData = new HashMap();
    }

    public boolean add(Student std){
        Student stdInMap = allData.get(std.getId());
        if(stdInMap == null){
            allData.put(std.getId(), std);
            return true;
        }else{
            return false;
        }
    }

    public boolean remove(String id){
        Student stdInMap = allData.get(id);
        if(stdInMap!=null){
            allData.remove(id);
            return true;
        }else{
            return false;
        }
    }

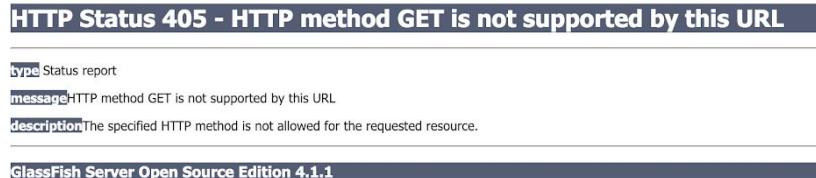
    public List<Student> getAllData(){
        return new ArrayList(allData.values());
    }
}

```

- push [key, value]
- remove[Object key]
- get[Object key]
- size[Object key]

## ERROR

- ERROR 405 : when compile and deploy servlet but not have method



- ERROR 500 : the scripting of jsp is cause of the exception
- ERROR 404 : you invoke wrong URL or the URL that did not have file or servlet