Lecture 2:

Olivier Liechti
AMT

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Agenda

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

13h00 - 14h00	60’	**Lecture** Managed components, inversion of control and dependency injection. Business services and EJBs.
14h00 - 15h15	75’	**Lab 03** Implementing business services with EJBs
15h15 - 15h45		Break
15h45 - 16h00	15’	**Lecture** Resource pooling
16h00 - 17h30	90’	**Lab 02** Assessing and validating the impact of HTTP sessions
17h30 - 18h00	30’	**Demonstrations & presentations**

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Back to the MVC lab…

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

3 interesting questions

- How was the servlet **instantiated** and **by who**?
- Who **invoked** the doGet method on the servlet?
- How did the servlet get a **reference** to the JSP page?

These 3 questions are related to **3 key concepts**:
managed components, *inversion of control* and *dependency injection*

How was the servlet **instantiated**?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- A servlet is a **Java object**.
- In **your code**, you never wrote anything like this:

```
HttpServlet fc = new FrontControllerServlet();
```

- The servlet instance(s) is (are) created **by the application server**.
- In other words, we say that the application server is **managing** the servlet. Or that the servlet is a **managed component**.

Who **invoked** the doGet method?

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- The HttpServletRequest interface defines a doGet method, which you have implemented in your class.
- In **your code**, you never wrote anything like this:

```
FrontController fc; HttpServletRequest request; HttpServletResponse response;
…
fc.doGet(request, response);
```

- Your code doesn’t call the Java EE code. The Java EE code calls your code, at the right time.
- This is an example of **inversion of control** (IoC).

How did the servlet **get a ref**. to the JSP?

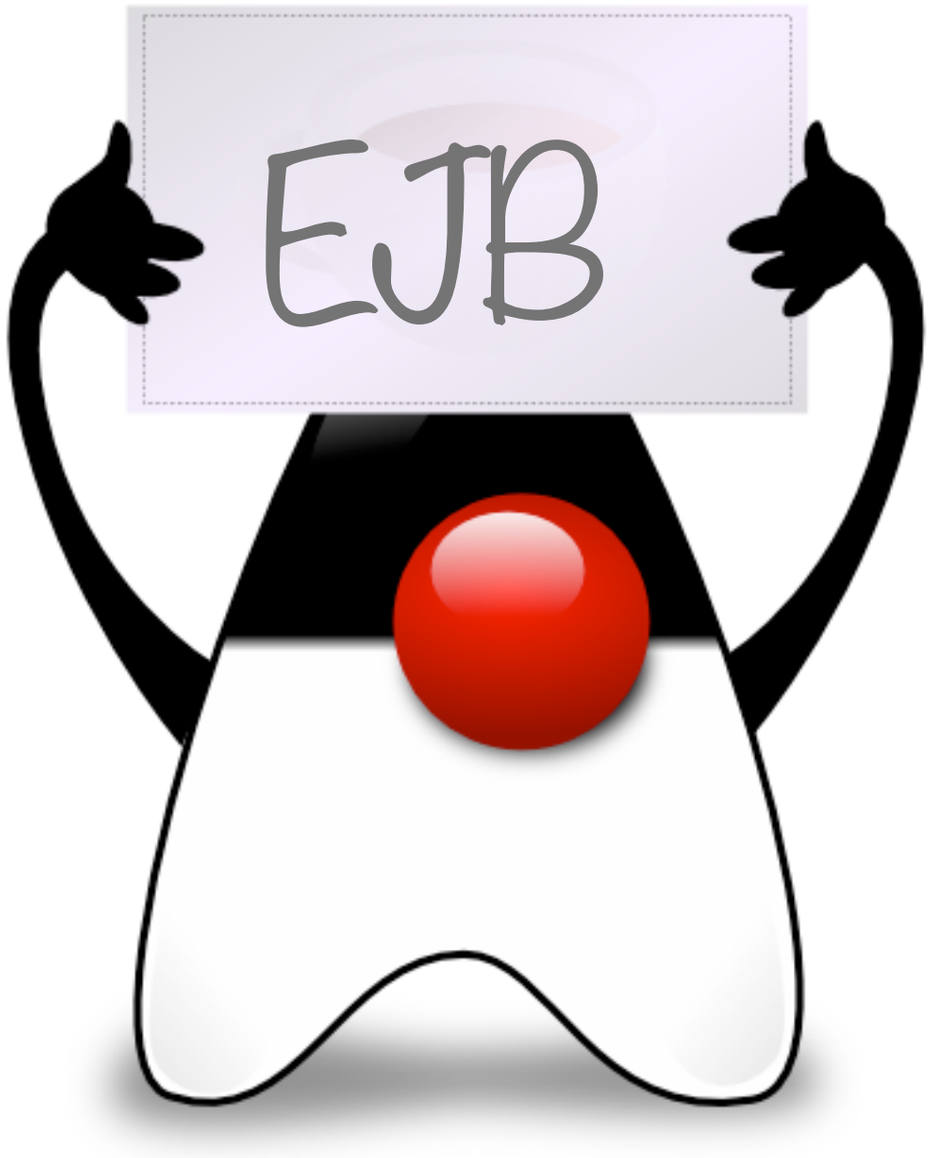
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- In **your servlet code**, you wrote something like this:

```
request.getRequestDispatcher("/WEB-INF/views/
        measures.jsp").forward(request, response);
```

- You did a **lookup operation** to get hold of the JSP.
- In other words, **you asked the application server** to give you the reference, based on a name.
- We will see that **dependency injection** is a valuable alternative to that process.

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

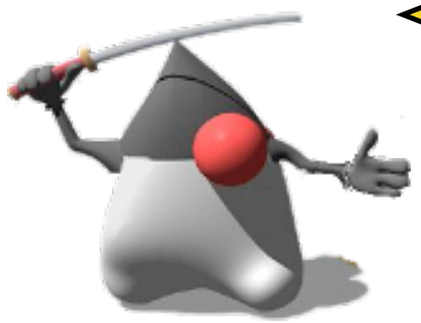


Business Services & EJB

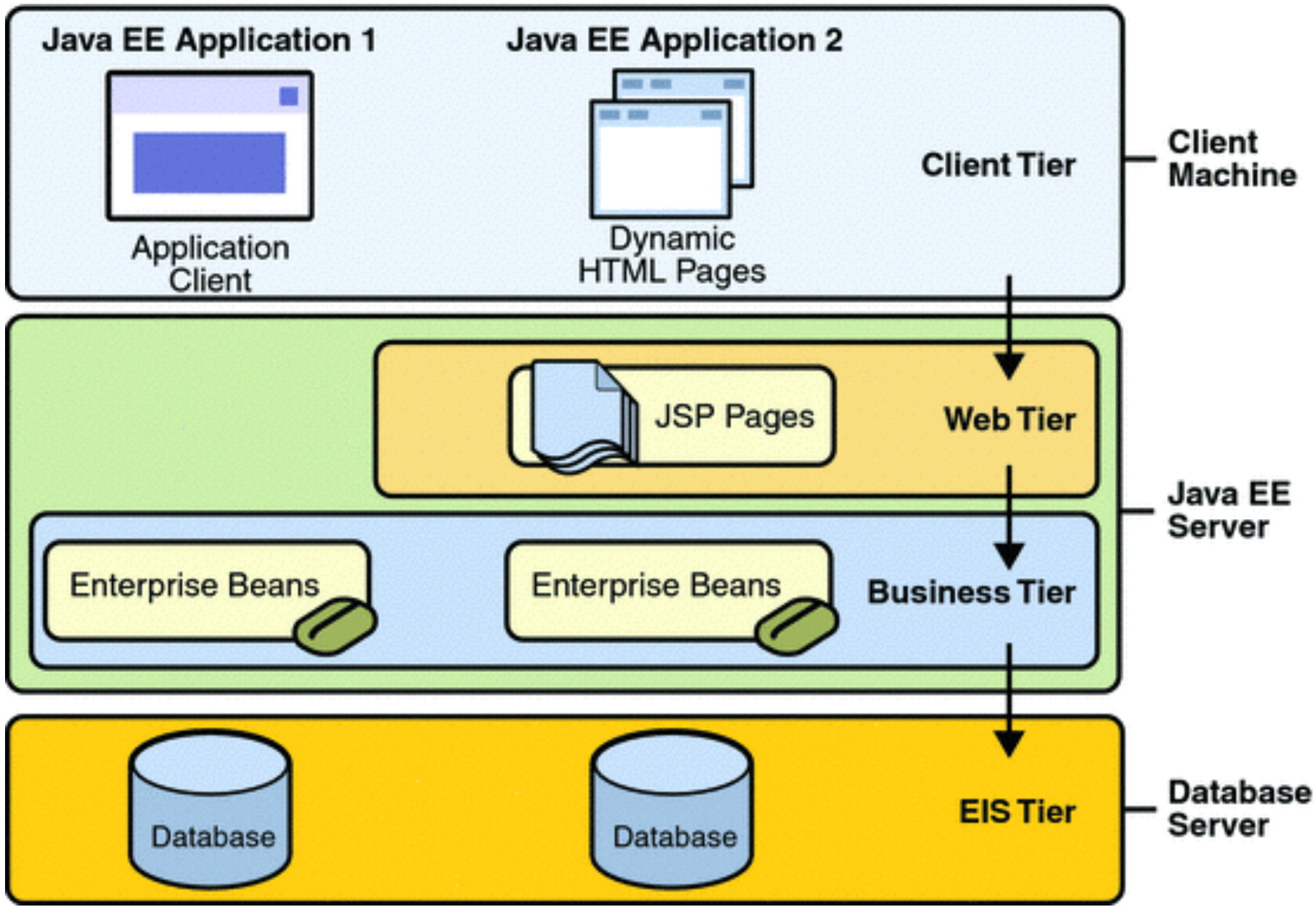
Services in a Java EE application

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- Last week, we implemented a very simple Java EE application.
- When we implemented the MVC pattern, we implemented a service as a **Plain Old Java Object** (POJO).
- **The POJO was not a managed component**. We created the instance(s) of the service (*in the web container*).
- This week, we will see an **alternative solution** for implementing Java EE services: Enterprise Java Beans (EJBs).



What is the best way to implement services, POJOs or EJBs?
There is not a single right answer to this question! There are pros and cons in both approach.



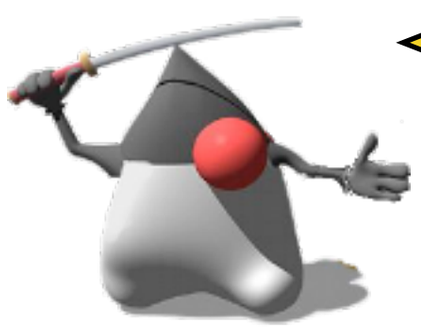
http://java.sun.com/javaee/5/docs/tutorial/doc/bnaay.html

What is an **Enterprise Java Bean** (EJB)?

- An EJB is a **managed component**, which implements **business logic** *in a UI agnostic way*.
- The EJB container manages the **lifecycle** of the EJB instances.
- The EJB container also **mediates the access** from clients (i.e. it is an “invisible” intermediary) to EJBs.
- This allows the EJB container to perform technical operations (especially related to **transactions** and **security**) when EJBs are invoked by clients.
- The EJB container manages a **pool** of EJB instances.
- Note: the EJB 3.1 API is **specified is JSR 318**.

What are the **4 types** of EJBs used today?

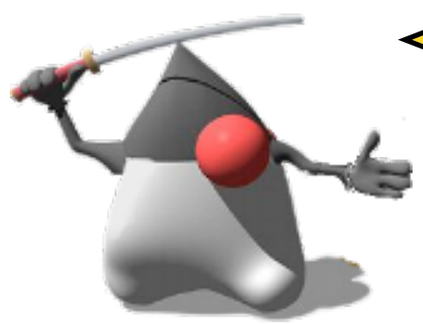
- **Stateless Session Beans** are used to implement business services, where every client request is independent.
- **Stateful Session Beans** are used for services which have a notion of conversation (e.g. shopping cart).
- **Singleton Session Beans** are used when there should be a single service instance in the app.
- **Message Driven Beans** are used together with the Java Message Service (JMS). Business logic is not invoked when a web client sends a request, but when a message arrives in a queue. We will see that later.



When you implement a stateful application in Java EE, **you have the choice to store the state in different places**. One option is to do it in the web tier (in the HTTP session). Another option is to use **Stateful Session Beans**. Many (most) developers use HTTP sessions.

In older versions of Java EE (before Java EE 5), there was another type of EJBs: **Entity Beans**.

Entity Beans were used for **accessing the database**. They were a nightmare to use and raised a number of issues. You might find them in legacy applications.



Entity Beans (as a legacy type of EJB) are **not the same thing** as **JPA Entities**, which are now widely used!

A first example

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
package ch.heigvd.amt.lab1.services;
import javax.ejb.Local;

@Local
public interface CollectorServiceLocal {

  void submitMeasure(Measure measure);

}
```

```
package ch.heigvd.amt.lab1.services;
import javax.ejb.Stateless;

@Stateless
public class CollectorService implements CollectorServiceLocal {

  @Override
  public void submitMeasure(Measure measure) {
    // do something with the measure (process, archive, etc.)
  }

}
```

These **annotations** are processed by the application server at **deployment time**.

They are an **declaration** that the service must be handled as a **managed component**!

How does a “client” find and use an EJB?

- By “**client**”, we refer to a **Java component** that wants to get a reference to the EJB and invoke its methods.
- In many cases, the client is a **servlet** or **another EJB** (i.e. a service that delegates part of the work to another service).
- The application server is providing a **naming and directory service** for managed components. Think of it as a “white pages” service that keeps track of component names and references.
- Remember that we mentioned **Dependency Injection** earlier today?

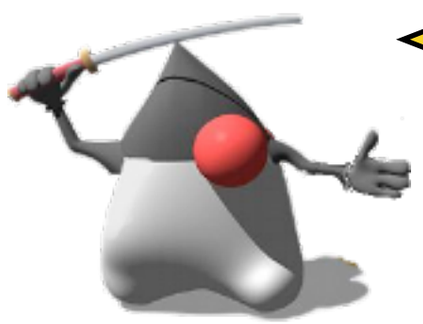
The Java Naming and Directory Interface (JNDI) provides an API to access directory services. It can be used to access an LDAP server. It can also be used to lookup components in a Java EE server.

The **first method** to find an EJB is to do an **explicit lookup**, with JNDI.

```
@WebServlet(name = "FrontController", urlPatterns = {"/FrontController"})
public class FrontController extends HttpServlet {

  private CollectorServiceLocal collectorService;

  @Override
  public void init() throws ServletException {
    super.init();
    try {
     Context ctx = new InitialContext();
      collectorService = (CollectorServiceLocal) ctx.lookup("java:module/CollectorService");
    } catch (NamingException ex) {
      Logger.getLogger(FrontController.class.getName()).log(Level.SEVERE, null, ex);
    }
  }

}
```

This gives me access to the app server's naming service

I am using the app server's naming service



Warning! These 2 JNDI operations are **costly** (performance-wise). You don't want to re-execute them for every single HTTP request!!!!
It is much better to do it once and to **cache the references** to the services.

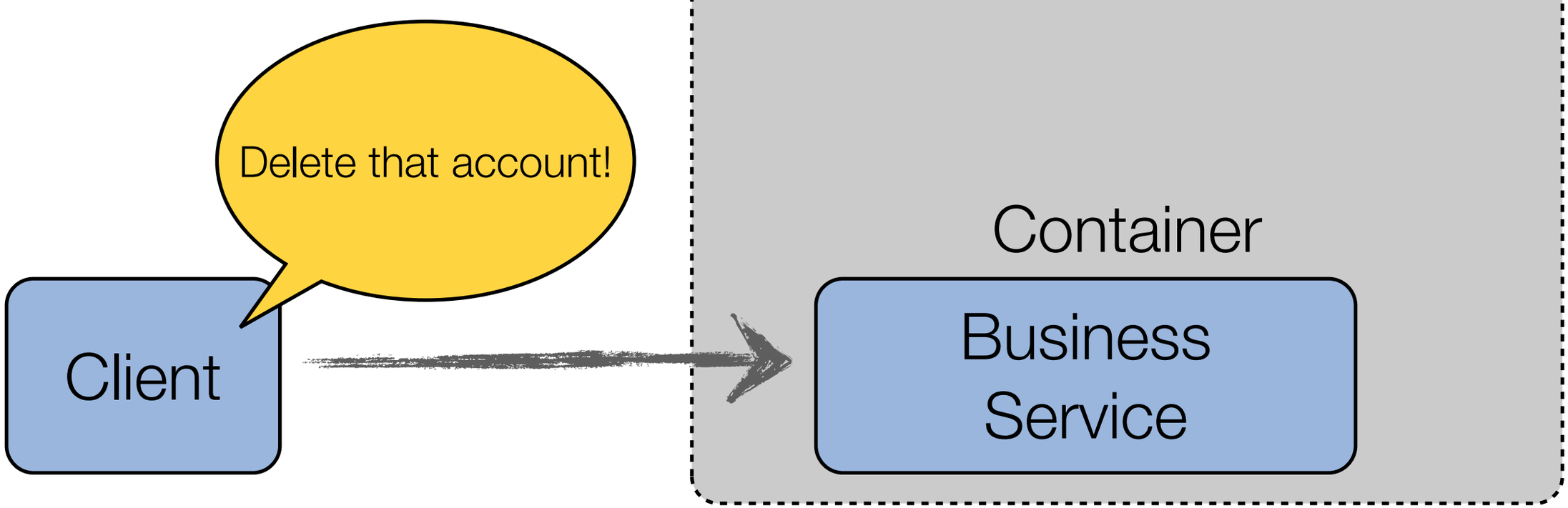
The **second method** is to ask the app server to **inject a dependency** to the service.

```
@WebServlet(name = "FrontController", urlPatterns = {"/FrontController"})
public class FrontController extends HttpServlet {

  @EJB
  private CollectorServiceLocal collectorService;

}
```

With the @EJB annotation, **I am declaring a dependency** from between my servlet and my service. The servlet *uses* the service.

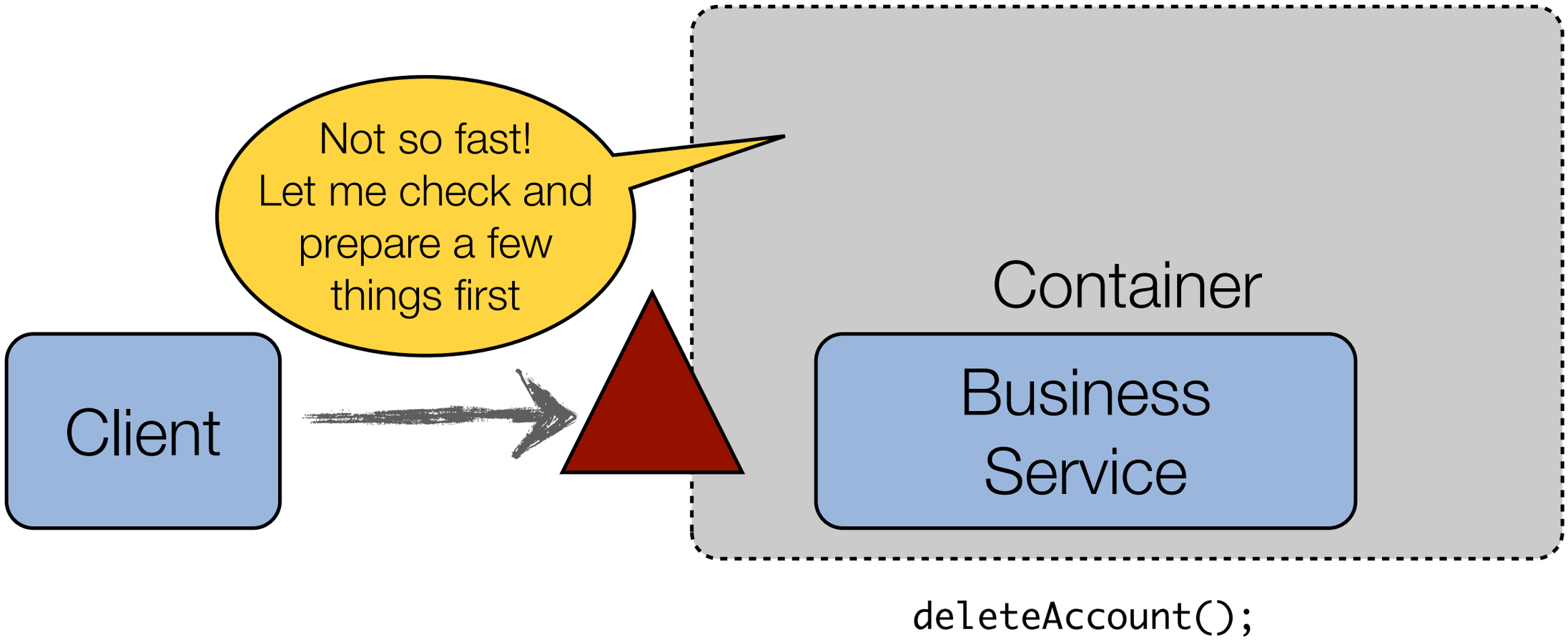
With the @EJB annotation, I am also giving instructions to the app server. The servlet and the service are **managed components**. When the app server instantiates the servlet, it **injects a value** into the **collectorService** variable.

The app server **mediates** the access between clients and EJBs. What does it mean?



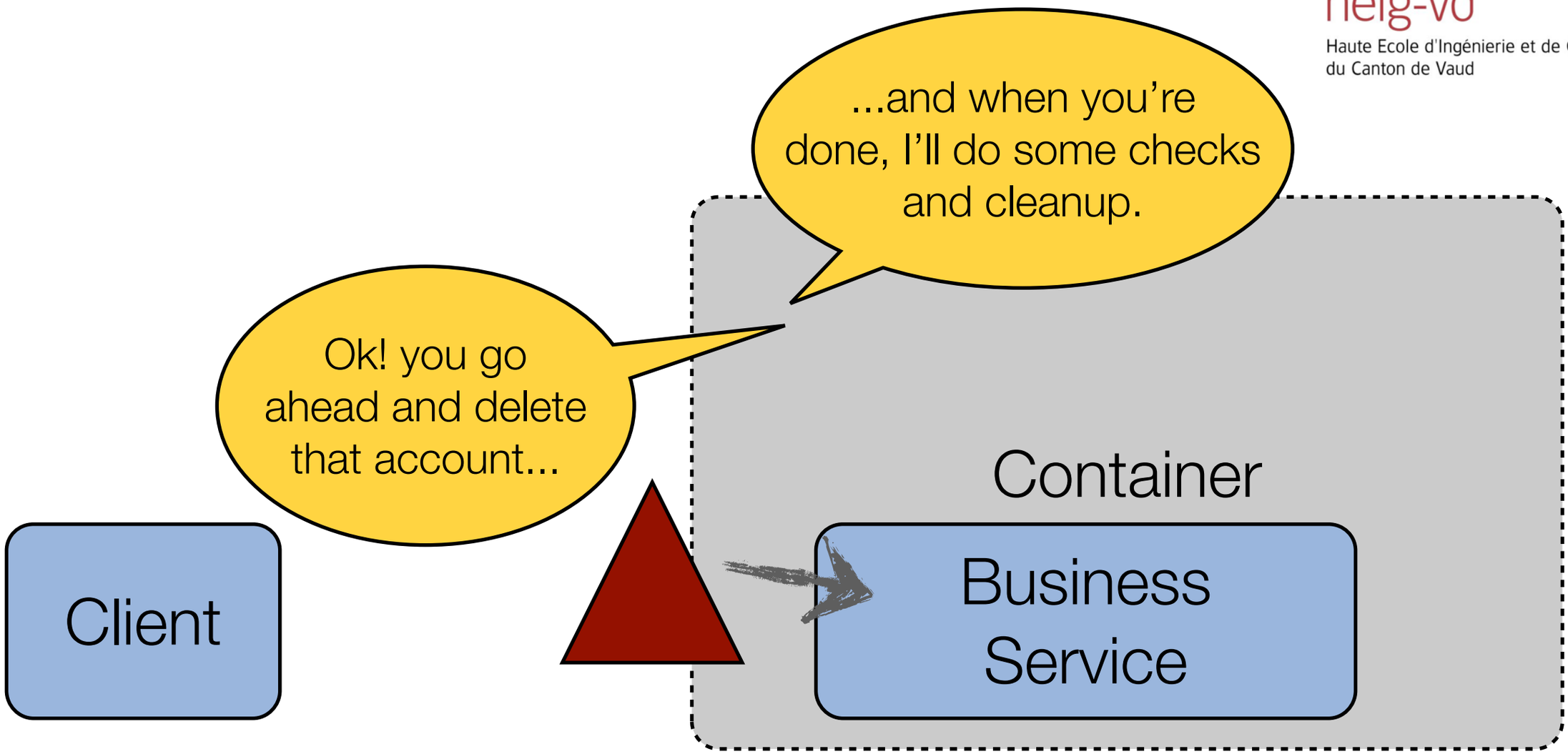

```
deleteAccount();
```

The business service, implemented as a Stateless Session Bean, is a **managed component**.
The client ***thinks*** that he has a direct reference to a Java object.
He is ***wrong***.




```
deleteAccount();
```

In reality, when the client invokes the deleteAccount() methods, the call is going **through the container**.
The container is in a position to **perform various tasks** (security checks, transaction demarcation, etc.)




```
deleteAccount();
```

When done, the container can forward the method call to the business service (your implementation).
On the way back, the response also goes back **via the container**.

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

How is that possible?
How does it work?

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

App Server
Naming Service

lookup();

I need a service for managing customer accounts... Give me a reference!

Client

Your EJB
Interface

deleteAccount();

proxy generated
by the app serv.

deleteAccount();

Your EJB
Implementation

deleteAccount();

Your service implementation implements your interface.
The container dynamically generates a class, which implements the same interface. This class performs the technical tasks and invokes your class (proxy).

Scientific Method

Formulate a **question**

"How is the container able to **intercept** method calls on EJBs?"

Propose an **hypothesis**

"I believe that it generates **dynamic proxies**"

Predict the **consequences**

"**If** I am right, then I should see these proxies in the call stack."

Experiment and **measure**

"Let's **throw an exception** in my stateless session bean and have a look at the stack trace"

Analyze

"In the stack trace, I can confirm that the servlet is not directly calling my Stateless Session Bean implementation class."

```
Caused by: java.lang.RuntimeException: just kidding
     at ch.heigvd.amt.lab1.services.CollectorService.submitMeasure(CollectorService.java:15)
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
     at java.lang.reflect.Method.invoke(Method.java:483)
     at org.glassfish.ejb.security.application.EJBSecurityManager.runMethod(EJBSecurityManager.java:1081)
     at org.glassfish.ejb.security.application.EJBSecurityManager.invoke(EJBSecurityManager.java:1153)
     at com.sun.ejb.containers.BaseContainer.invokeBeanMethod(BaseContainer.java:4786)
     at com.sun.ejb.EjbInvocation.invokeBeanMethod(EjbInvocation.java:656)
     at com.sun.ejb.containers.interceptors.AroundInvokeChainImpl.invokeNext(InterceptorManager.java:822)
     at com.sun.ejb.EjbInvocation.proceed(EjbInvocation.java:608)
     at
org.jboss.weld.ejb.AbstractEJBRequestScopeActivationInterceptor.aroundInvoke(AbstractEJBRequestScopeActivationInte
ceptor.java:46)
     at org.jboss.weld.ejb.SessionBeanInterceptor.aroundInvoke(SessionBeanInterceptor.java:52)
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
     at sun.reflect.DelegatingMeth
     at java.lang.reflect.Method.i
     at com.sun.ejb.containers.int
     at com.sun.ejb.containers.int
     at com.sun.ejb.EjbInvocation.
     at com.sun.ejb.containers.int
     at com.sun.ejb.containers.int
     at sun.reflect.NativeMethodAc
     at sun.reflect.NativeMethodAc
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
     at java.lang.reflect.Method.invoke(Method.java:483)
     at com.sun.ejb.containers.interceptors.AroundInvokeInterceptor.intercept(InterceptorManager.java:883)
     at com.sun.ejb.containers.interceptors.AroundInvokeChainImpl.invokeNext(InterceptorManager.java:822)
     at com.sun.ejb.containers.interceptors.InterceptorManager.intercept(InterceptorManager.java:369)
     at com.sun.ejb.containers.BaseContainer.__intercept(BaseContainer.java:4758)
     at com.sun.ejb.containers.BaseContainer.intercept(BaseContainer.java:4746)
     at com.sun.ejb.containers.EJBLocalObjectInvocationHandler.invoke(EJBLocalObjectInvocationHandler.java:212)
     ... 34 more
```

```java
@Stateless
public class CollectorService implements CollectorServiceLocal {

  @Override
  public void submitMeasure(Measure measure) {
    throw new RuntimeException("just kidding");
  }
```

Scientific Method

Formulate a **question**

"How is the container able to **intercept** method calls on EJBs?"

Propose an **hypothesis**

"I believe that it generates **dynamic proxies**"

Predict the **consequences**

"**If** I am right, then I should see these proxies in the call stack."

Experiment and **measure**

"Let's **inspect the call stack in the debugger**"

Analyze

"In the debugger console, I can confirm that there are container generated classes between my servlet and my service implementation."

Projects | Files | Services | Debugging

'http-listener-1(5)' at line breakpoint Collecto

- **CollectorService.submitMeasure:15**
- Hidden Source Calls
- Method.invoke:483
- Hidden Source Calls
- Method.invoke:483
- Hidden Source Calls
- Method.invoke:483
- Hidden Source Calls
  - AroundInvokeInterceptor.intercept:883
  - AroundInvokeChainImpl.invokeNext:82
  - InterceptorManager.intercept:369
  - BaseContainer.__intercept:4758
  - BaseContainer.intercept:4746
  - EJBLocalObjectInvocationHandler.invok
  - EJBLocalObjectInvocationHandlerDeleg
  - $Proxy347.submitMeasure
  - FrontController.processRequest:86
  - FrontController.doGet:103
- Hidden Source Calls
- Thread.run:745

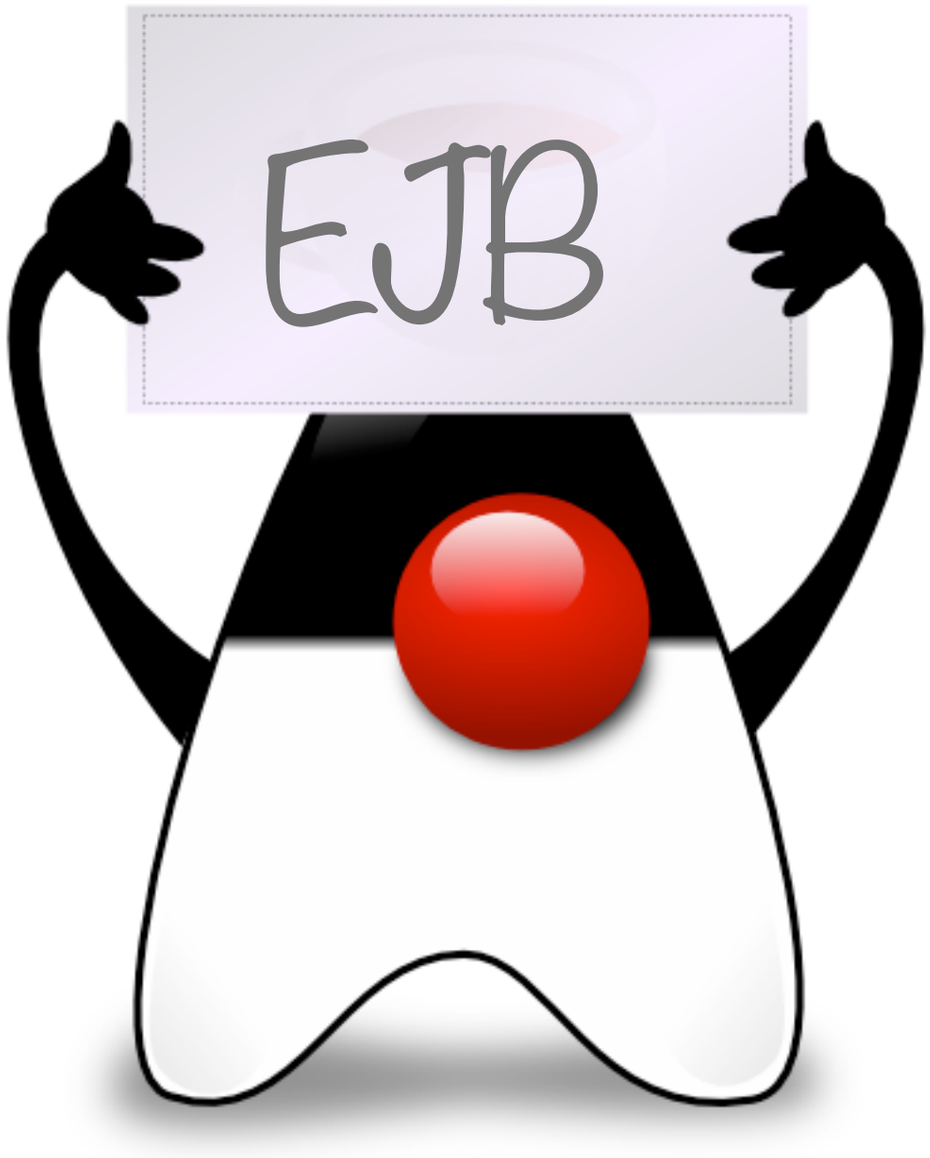
At some point, the method call is forwarded to my implementation.

The reference actually points to a proxy generated by the container. The container performs tasks that are visible in a **long call stack**!

My servlet invokes the method on its **reference** to the EJB.

*An HTTP request has arrived; GF invokes the doGet callback on my servlet (**IoC**). GF has also **injected** a **reference** to the EJB into the servlet.*

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Lab 03 - “Hello Business”

Let's expand the previous project and introduce EJBs in the business tier.

```
POST /DataLoggerServlet HTTP/1.1
Content-type: text/csv

S-01, 2.34, 1411622924142
S-99, WRONG, DATA
S-01, 11, 1411622924148
S-02, 0.82, 1411622924155
S-99, WRONG, DATA
```

```
3 measures added. 2 invalid lines.
```

<<Servlet>>

DataLoggerServlet

<<Stateless Session Bean>>

```
public void logMeasure(Measure m)
```

MeasuresManager

```
public List<Measure> getMeasures()
```

```
GET /FrontControllerServlet HTTP/1.1
Accept: text/html
```

<<Servlet>>

FrontControllerServlet

<<JSP>>

Measures.jsp

<<Singleton Session Bean>>

```
public void insertData(Object o)
```

DataManager

```
public List getData()
```

```
<table>
  <tr>
    <td></td><td></td><td></td>
  </tr>
</table>
```

Start by implementing the DataManager

- The role of the **DataManager** is to provide an **in-memory persistence** store. Later on, we will replace it with a database.
- Data should be shared by all clients, so there should be **only one instance** of DataManager. So, let's use a singleton session bean.
- The **interface** is very simple: there should be one method to **add** an object to the store and one method to **get** a list of all objects previously stored.
- **Internally**, we can store objects in a list.

Now, re-implement the MeasuresManager

- Last week, the MeasuresManager was a POJO. Let's replace it with a **Stateless Session Bean** (with a local interface).
- The MeasuresManager should use the DataManager to persist data. In other words, the MeasuresManager has a **dependency** on the DataManager and we want to inject it (with **@EJB**)
- Compared to last week, the MeasuresManager should still provide a method to get a list of measures. In addition, it should **provide a method to add a new measure**.

Adapt the FrontControllerServlet

- The FrontController servlet should only interact with the MeasuresManager (and not directly with the DataManager). We need to **inject a reference** to MeasuresManager in the servlet.
- The FrontController servlet works like last week:
  - When a request arrives, we assume that the user wants to see an HTML formatted list of measures.
  - We get the measures from the MeasuresManager service.
  - We forward the measures to the JSP, which handles the rendering.

Create a new DataLoggerServlet

- Last week, measures were generated randomly on the server side. This week, we want to be able to **create them on the client side**. We need a way to send new measures to the server.
- The role of the DataLoggerServlet is to accept HTTP requests, which carry a **list of several measures in their body** (in **CSV** format). In its **response**, the servlet informs how many measures have been **successfully** added to the store and how many lines contained **errors** (e.g. invalid numbers, etc.).
- To **parse** the request payload, you can use the getReader() method exposed by HttpServletRequest.
- The DataLogger should use the MeasuresManager to persist measures and not use the DataManager directly (inject the dependency).
- Once you are done, use the **PostMan chrome extension** to test your application.

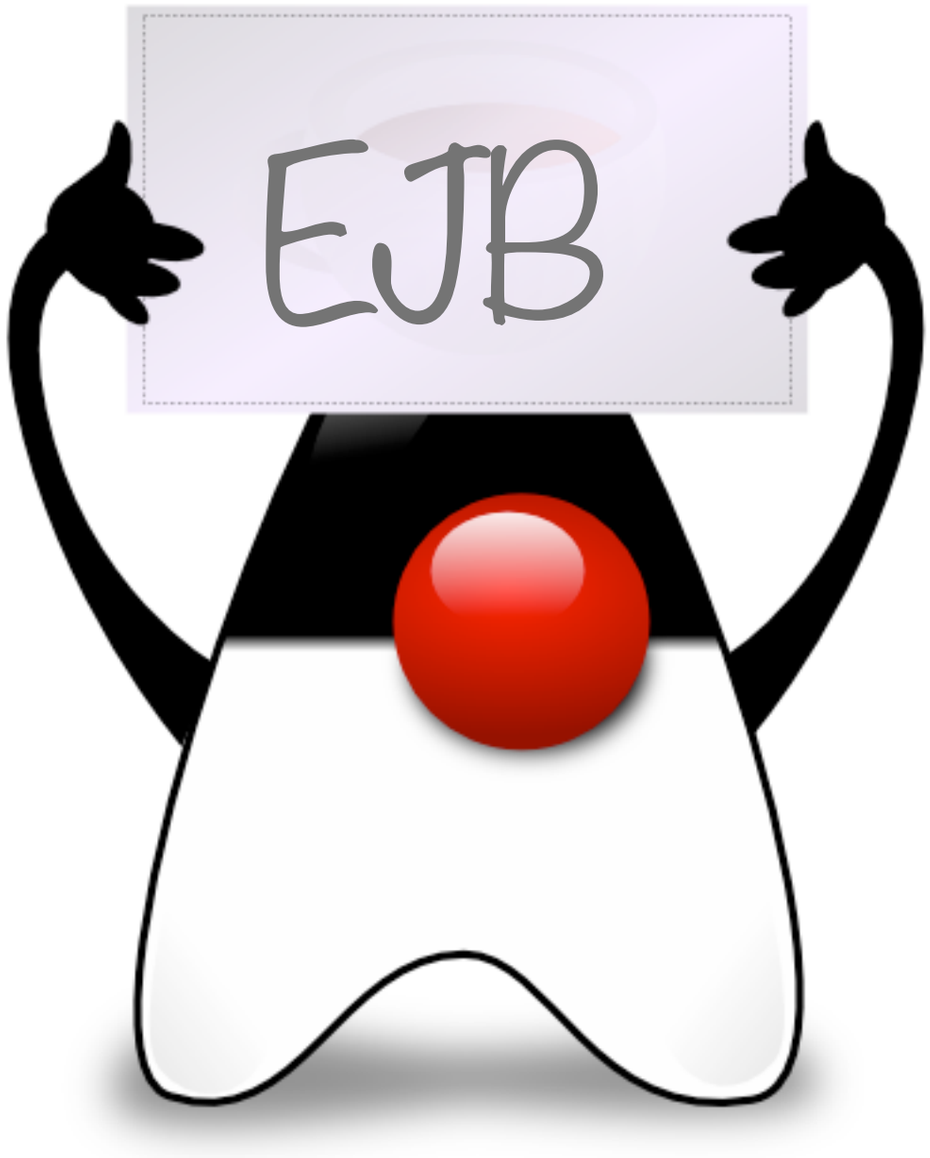
If you finished the lab before the allocated time, then work on these **extra** features

- So far, we generated a table that contained the full list of available measures. Works well for 10 measures, but when we have hundreds of them… Address this issue by **implementing pagination**.

- Note: the servlet has access to the **query string parameters** via the HttpServletRequest object. That's an easy way to pass offset and page size attributes to the servlet, in order to get a particular page of measures.

- Create a **JMeter script to populate the database** (via the DataLogger servlet). The script should generate random values. Have a look at the functions chapter in the JMeter documentation (http://jmeter.apache.org/usermanual/functions.html)

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

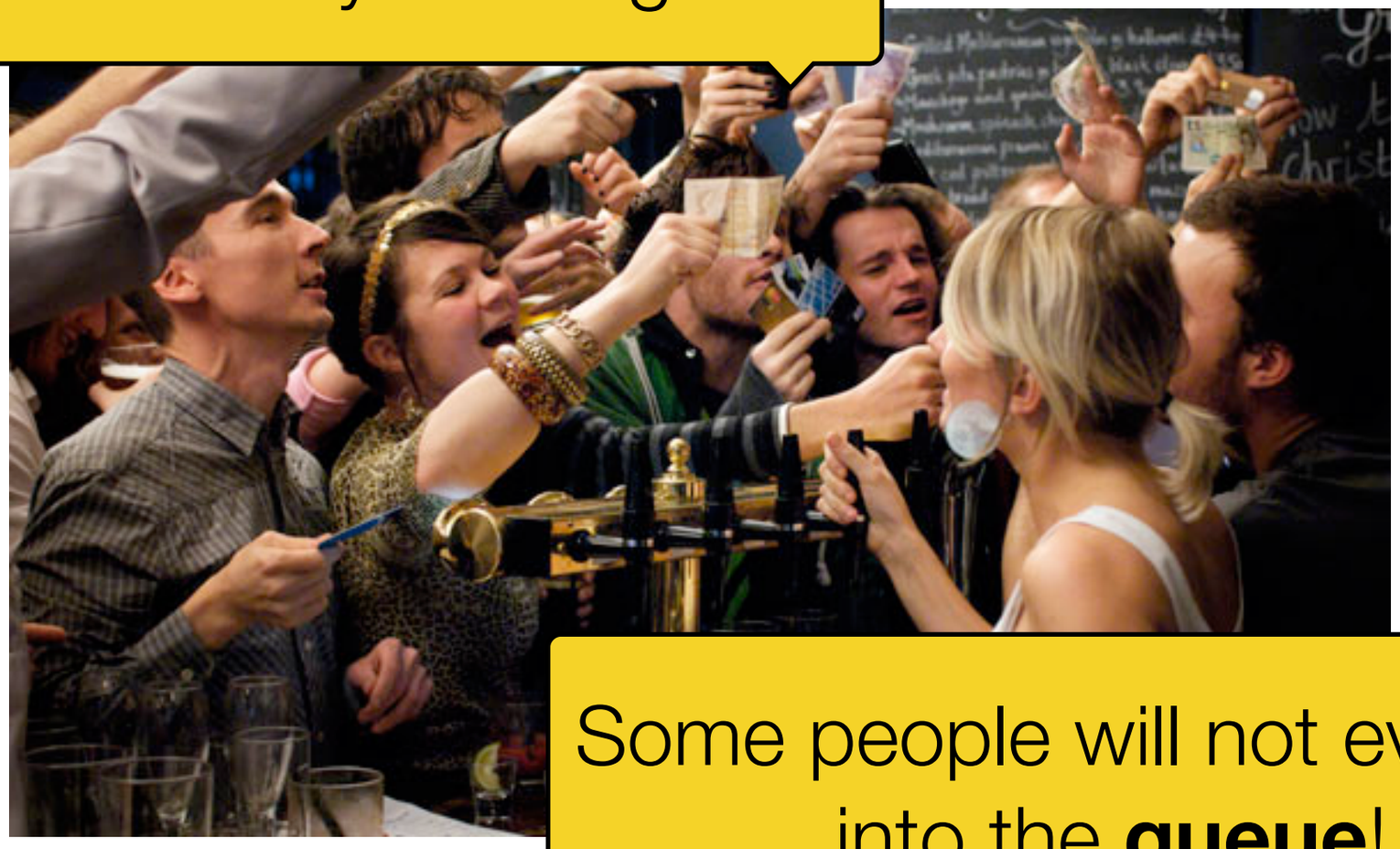


Business Services & EJB

The app server **manages a pool** of EJB instances. What does it mean?

Imagine you are a bar owner…
How do you organize your staff?

Customers will **wait too long** on Saturday evening!



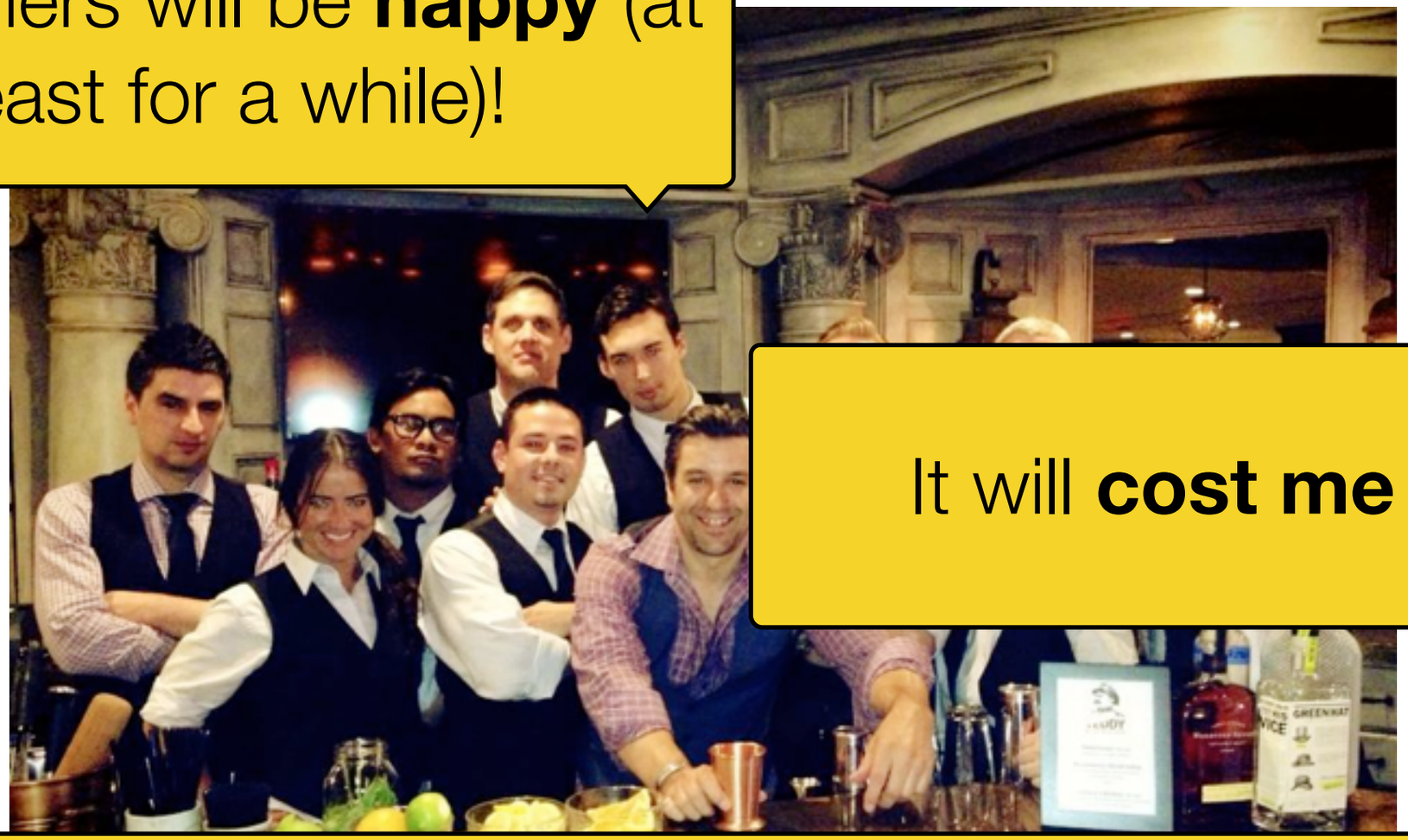
Some people will not even get into the **queue**!

"Hire only one staff member. If a customer arrives, wait until staff is done with the previous one."

If a bar tender doing too many things at the same time, he will make **mistakes**!

"Hire only one staff member. If a customer arrives, ask him to serve him at the same time as the previous one."

Customers will be **happy** (at least for a while)!



It will **cost me a lot**.

On Saturday evening, I will have 100 bar tenders. They will **compete** for the shaker and space. They will be **inefficient** and customers will have to **wait for a very long time** before getting their drink!

"As soon as a customer arrives, if no staff member is available, hire a new one."

I can **adjust the size of the pool** to **keep a balance** between the length of the queue and the contention on shared resources.
I will be able to **deal with affluence** on Saturday evening!”

“Hire a pool of staff members. Ask each staff member to serve his customers one after the other.”

Handling requests **without concurrency** does not scale, because of **queuing**.

A single **servlet** instance handles requests concurrently, but is **not thread-safe**.

Using a **new POJO instance** for every request does not scale, as it **exhausts resources**.

Using a **pool of EJBs** is a way to throttle requests in a **thread-safe** manner.

How does the EJB Container implement **resource pooling**?

- Remember that EJBs are **managed components** and are instantiated by the container.
- When the container creates an EJB instance, it **puts it in a pool**. The instance is "**ready to serve**", waiting for some job.
- When a client component (e.g. a servlet) invokes a method on a Stateless Session Bean, the container **checks if there is an available instance in the pool**:
  - If **yes**, then the instance is removed from the pool (it will not be available for other requests until done) and method is forwarded to it. When the method returns, the instance goes back into the pool.
  - If **no**, then **either** the request is **put into a queue** (until a instance is ready to serve), **or** the container **resizes the pool** by creating new EJB instances.

Monitoring Service

localhost:4848/common/index.jsf

Home | About... | Help

User: anonymous | **Domain:** domainAMT | **Server:** localhost

GlassFish™ Server Open Source Edition

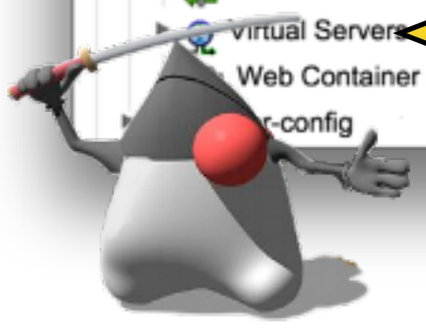
- JDBC
- JMS Resources
- JNDI
- JavaMail Sessions
- Resource Adapter Configs
- **Configurations**
- default-config
  - Admin Service
  - Availability Service
  - Connector Service
  - EJB Container
  - Group Management Service
  - HTTP Service
  - JVM Settings
  - Java Message Service
  - Logger Settings
  - Monitoring
  - Network Config
  - ORB
  - Security
  - System Properties
  - Thread Pools
  - Transaction Service
  - Virtual Servers
  - Web Container
  - -config

Deploy all MBeans needed for monitoring

Component Level Settings (16)

Level: Change Level

Select	Module	Monitoring Level
☐	Jvm	OFF
☐	Transaction Service	OFF
☐	Connector Service	OFF
☐	Jms Service	OFF
☐	Security	OFF
☐	Web Container	HIGH
☐	Jersey(RESTful Web Services)	OFF
☐	Web Services Container	OFF
☐	Java Persistence	OFF
☐	Jdbc Connection Pool	OFF
☐	Thread Pool	OFF
☐	Ejb Container	HIGH
☐	ORB (Object Request Broker)	OFF
☐	Connector Connection Pool	OFF



If we want to monitor the EJB pool, we first need **activate the monitoring**.

Application Monitoring

localhost:4848/common/index.jsf

Home | About... | Help

User: anonymous | **Domain:** domainAMT | **Server:** localhost

GlassFish™ Server Open Source Edition

Common Tasks

- Domain
- server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
  - localhost-domainAMT
- Applications
- Lifecycle Modules
- Monitoring Data
- Resources
  - Concurrent Resources
  - Connectors
  - JDBC
  - JMS Resources
  - JNDI
  - JavaMail Sessions
  - Resource Adapter Configs
- **Configurations**
  - default-config
    - Admin Service
    - Availability Service
    - Connector Service
    - EJB Container
    - Group Management
    - ...

General | Resources | Properties | Monitor | Batch | JMS Physical Destinations

Applications | Server | Resources

Application Monitoring

Refresh

Click Configure Monitoring and enable monitoring for a component or service by selecting either LOW or HIGH. See the Online Help for more information.

Instance Name: server

Application : SimpleApp **Component :** default

Monitor (23 Statistics)

Servlet Instance Statistics : SimpleApp/server/default

Name	Value	Start Time	Last Sample Time	Details	Description
RequestCount	0 count	Sep 24, 2014 5:26:03 PM	Sep 24, 2014 5:38:09 PM	--	Number of requests processed
ErrorCount	0 count	Sep 24, 2014 5:26:03 PM	--	--	Number of error responses (that is, responses with a status code greater than or equal to 400)
ProcessingTime	0 millisecond	Sep 24, 2014 5:26:03 PM	Sep 24, 2014 5:38:09 PM	--	Average response time

Even then, it seems that **no EJB monitoring data** is available in the web console…?

GlassFish Server Open Sou ×

localhost:4848/monitoring/domain/server/applications/SimpleApp/CollectorService

GlassFish Server Open Source Edition 4.1 REST Interface

- createcount
  - unit : count
  - lastsampletime : -1
  - name : CreateCount
  - count : 0
  - description : Number of times EJB create method is called or 3.x bean is looked up
  - starttime : 1411572510481
- methodreadycount
  - unit : count
  - current : 1
  - lastsampletime : 1411572807901
  - lowwatermark : 0
  - lowerbound : 0
  - upperbound : 32
  - name : MethodReadyCount
  - description : Number of stateless session beans in MethodReady state
  - highwatermark : 1
  - starttime : 1411572510481
- removecount
  - unit : count
  - lastsampletime : -1
  - name : RemoveCount
  - count : 0
  - description : Number of times EJB remove method is called
  - starttime : 1411572510481

Child Resources

bean-methods
bean-pool

/monitoring/domain/server/applications/**$APP**/**$EJB**

Using the **monitoring REST API** gives access to a huge quantity of precious information (in a **tree**)!!

Scientific Method

Formulate a **question**

"How does Glassfish manages EJB pools?"

Propose an **hypothesis**

"**I believe** that it increases the pool size if no bean is "ready to work" when a new request comes in.

Predict the **consequences**

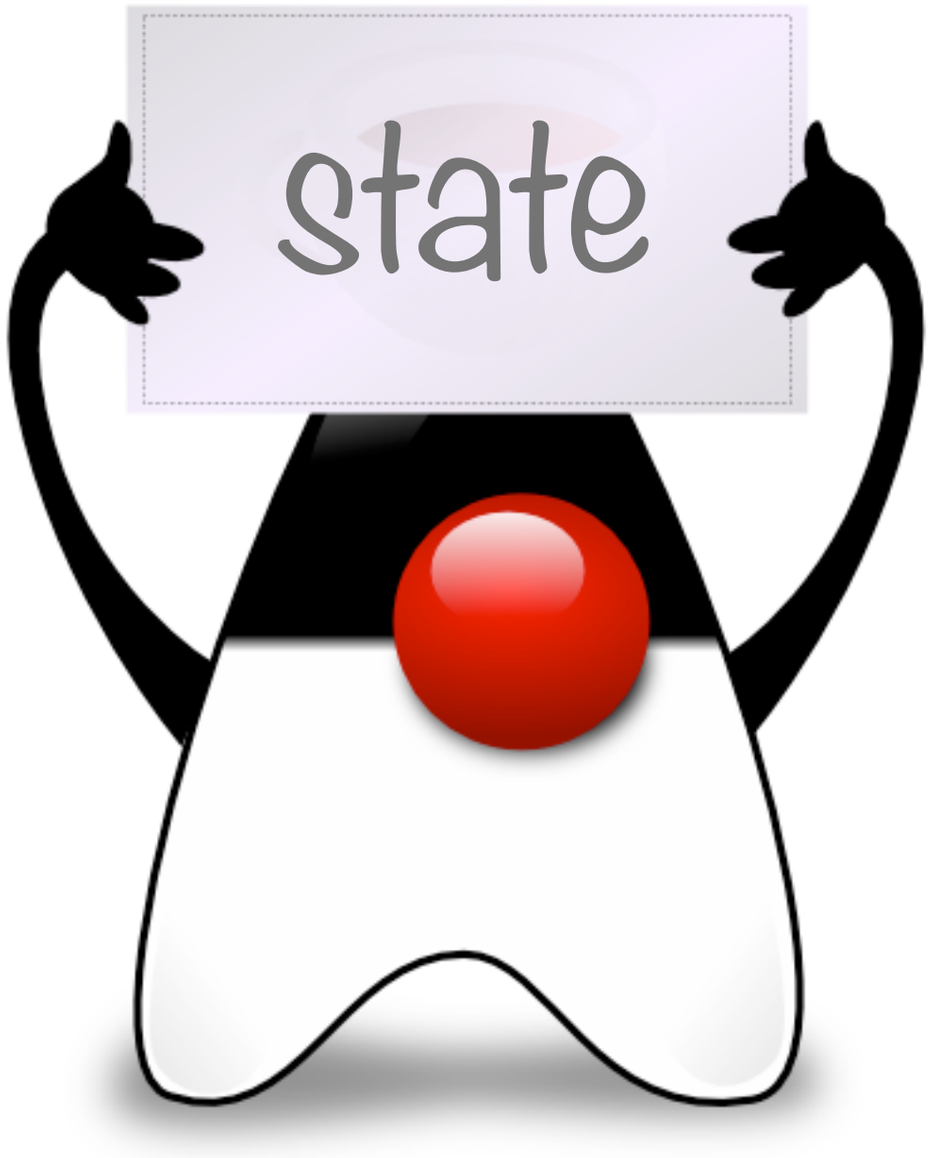
"**If I am right**, then by generating sufficient load, I should be able to trigger a "pool resize operation"

Experiment and **measure**

"Let's add a Thread.sleep(4000) statement in a EJB method to make it easier to exhaust the pool. Let's collect data via GF monitoring."

Analyze

"In the REST Monitoring API, I can validate that the pool has been resized."



heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Lab 02 - HTTP Sessions

Lab Objective

What is the impact of storing objects in **HTTP sessions** using the Servlet API, both in terms of **memory consumption** and of **performance**?

Answer this question in a systematic and structured way and provide supporting **evidence**.

Two questions, two experiments!

Question 1
What is the impact of storing objects in **HTTP sessions** using the Servlet API in terms of **memory consumption**?

Propose an **hypothesis**

Predict the **consequences**

Experiment and **measure**

Analyze

Two questions, two experiments!

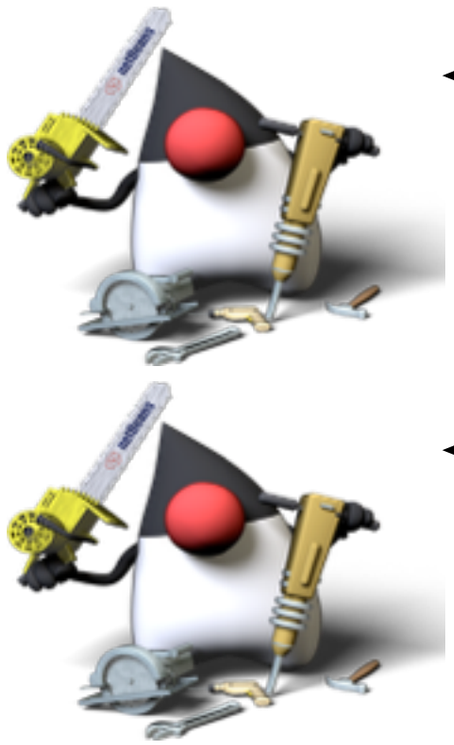
Question 2
What is the impact of storing objects in **HTTP sessions** using the Servlet API in terms of **performance**?

Propose an **hypothesis**

Predict the **consequences**

Experiment and **measure**

Analyze



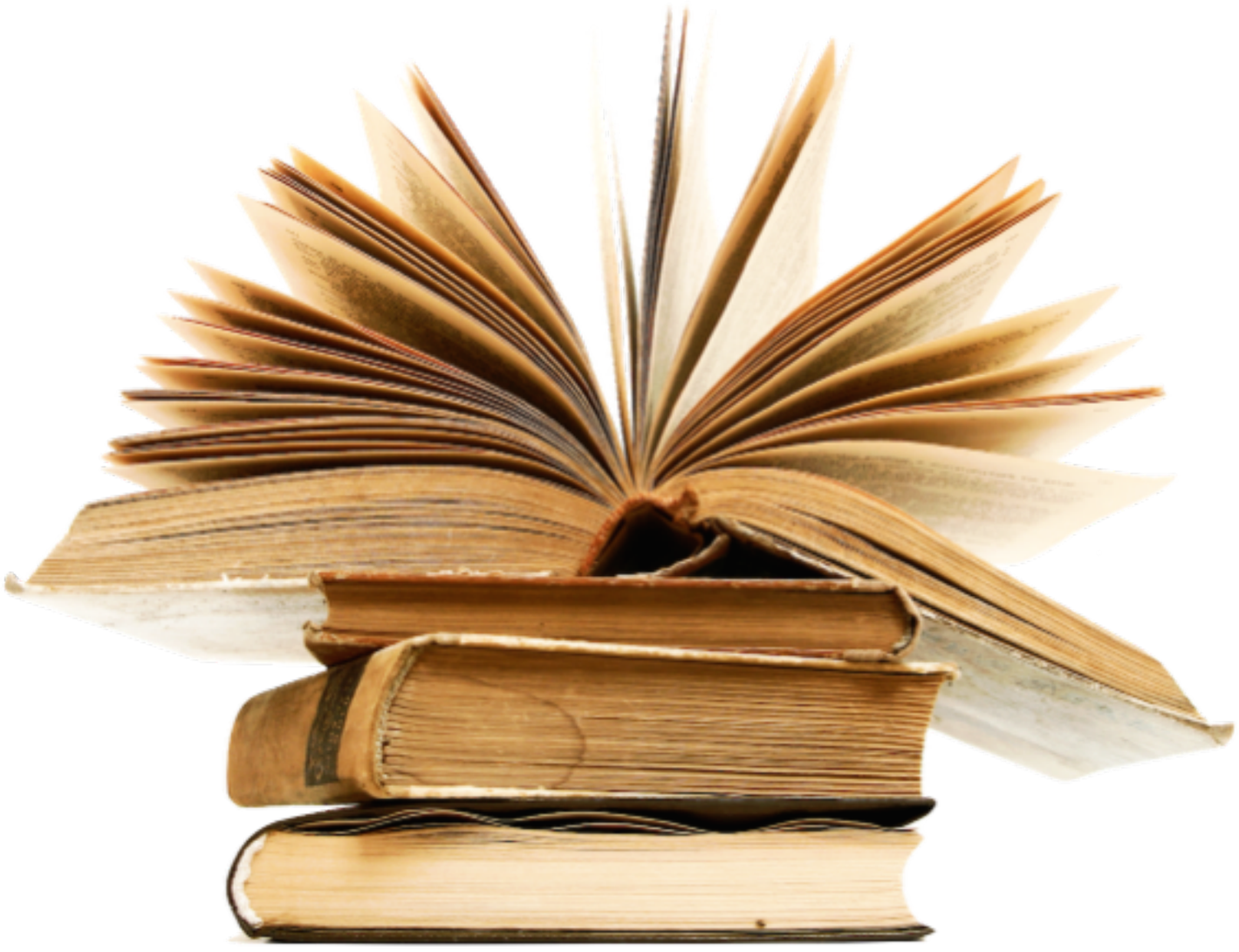
Let's start with **memory consumption**. What is your hypothesis?

If your hypothesis holds, **then** what are the **measurable consequences**?

How would you **design an experiment** to get evidence?

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



References

MUST READ for the Tests

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Selected sections from the Java EE 7 tutorial**
  - http://docs.oracle.com/javaee/7/tutorial/doc/ejb-intro.htm#GIJSZ

Why is it very important to **code in english** (i.e. to use english for variable, class and method names, to use english in your comments)?

What is the difference between **servlets** and **stateless session beans** in terms of **concurrency**?

How is **dependency injection** applied in the context of EJBs? Illustrate your answer with code examples.

In the context of servlet to EJB interactions, what is the **alternative to dependency injection**?

What is the role of the **EJB pool** managed by the application server?

How is **inversion of control** applied in the Java EE platform? Give one example and illustrate it with a code example.

How is the EJB container able to perform **security checks** when a method is invoked on a stateless session bean? Illustrate your answer with a UML **sequence diagram**.

What are **four** types of EJBs?

What code do you write in a servlet in order to use the naming service provided by Glassfish, in order to **lookup** an EJB by name?

What code do you write in a servlet in order to **inject a dependency** on an EJB? Why is this code not working in a POJO?