EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

JAT – Java Technology

David Ježek
david.jezek@vsb.cz

Tel: 597 325 874
Office: EA406

Something from history…

- 500 BC – Pythagoras founded a society of peoples that believe among other things, that beans contains souls of dead peoples, therefor eating of beans was forbidden.
- 850 AC – Arabian goatherd Khalid found, that his herd behaves in a strange way and discovered plant “Coffea arabica” and the effect of caffeine.
- End of the 17th century – cultivation of coffee beans on the island Java, name Java was connected with coffee in general
- 1995 – Sun’s programing language Oak has problem with the name, therefore was chosen name Java and language was connected with coffee.

Java Platforms

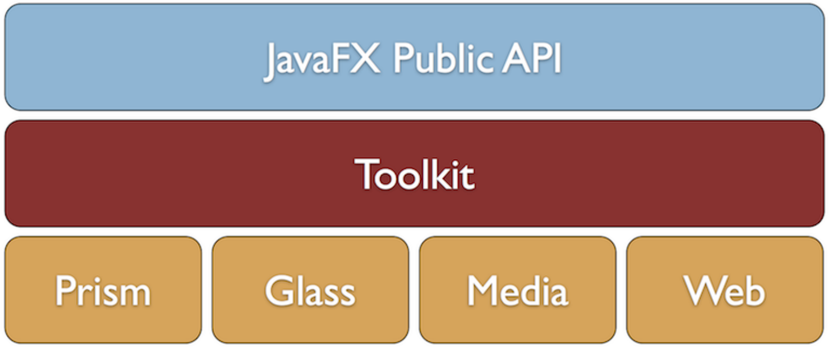
- **Standard Edition (Java SE)**
  - Java Applet, Java WebStart
- **Java FX**
  - JavaFX 2.1 - desktop, browser and mobile phones.
  - Planed: TV set-top boxes, gaming consoles, Blu-ray players and other
- **Enterprise Edition (Java EE)**
  - Servlets, JavaServerFaces, JavaServerPages
- **Micro Edition (Java ME)**
  - Mobile phone, Java TV, Java Card, other devices

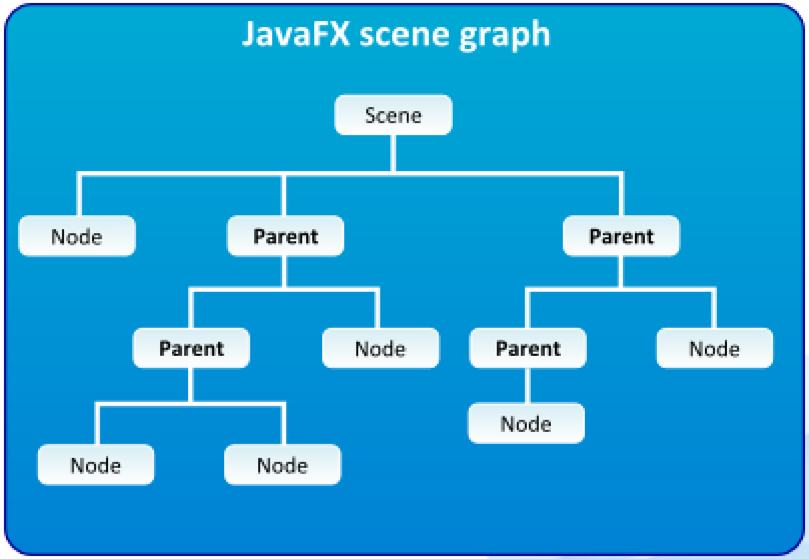
Java SE Platform

JDK							
		JRE					
Java Language	Tools and Utilities	Runtime	Java SE API				Java VM
			Base Library	Other Base Packages	Integration Libraries	User Intrface Libraries	
javac	javadoc JAR javah javap JPDA JConsole VisualVM java DB Security Internationalization RMI IDL Deployment Monitoring Troubleshooting Scripting	Java Java Web Start Applet/Plug-in	Lang and Util Collections Concurrency Utilities JAR Logging Management Preferences API Reference Objects Reflection Regular Expressions Versioning ZIP Instrumentation	Beans I18N Support I/O JMX Math Networking Override Mechanism Security Object Serialization Extension Mechanism XML	IDL JDBC JNDI RMI RMI-IIOP Scripting JNI	AWT Swing Java 2D Accessibility Drag and Drop Input Methods Image I/O Print Service Sound	HotSpot
	JVM TI						

JavaFX Platform

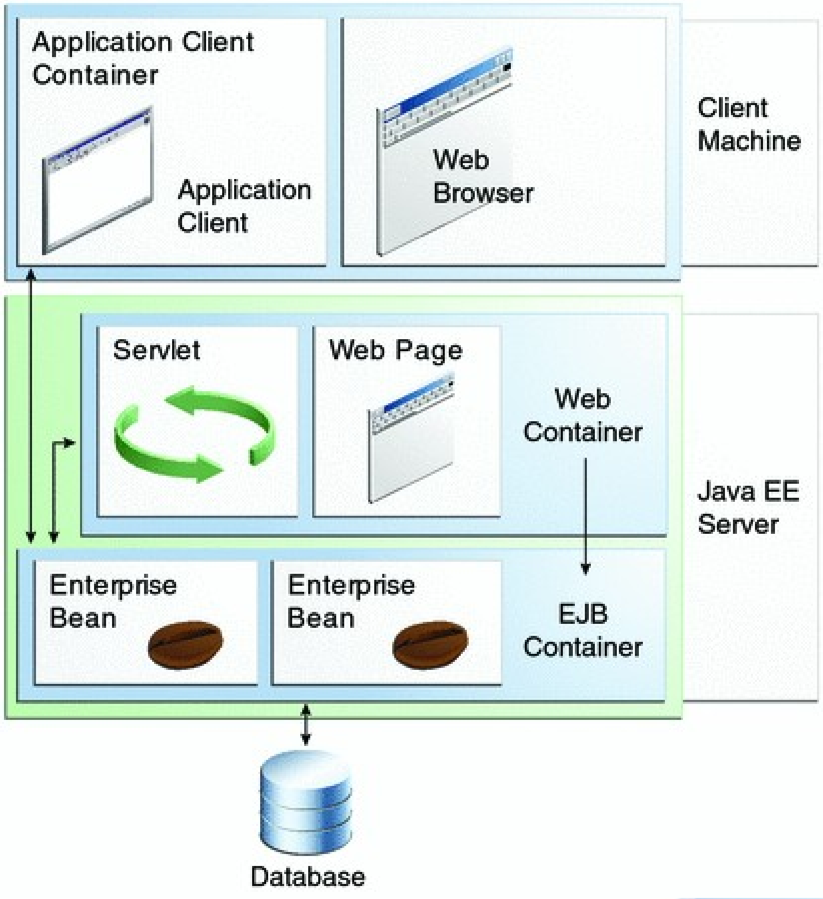
- Designed for RIA
- Variety of devices
- Media support
- Scene graph
- CSS styles





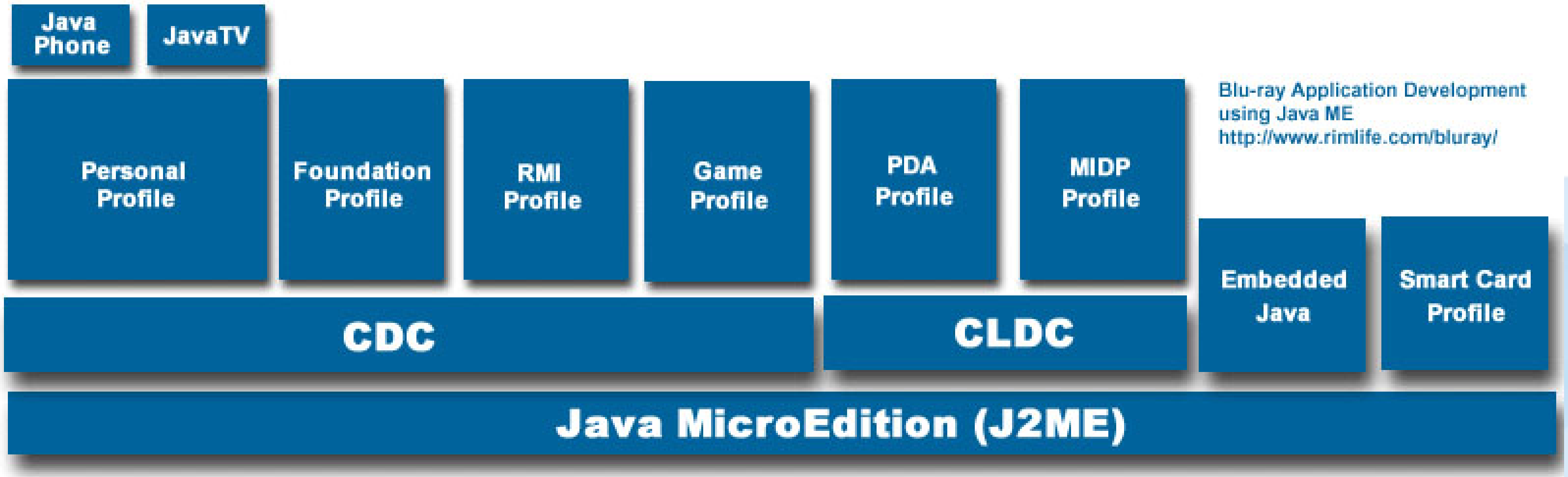
Java EE Platform

- Multi-tier architecture
- Java EE API
  - Servlets
  - Java Server Faces
  - Java Server Pages
  - Persistence API
  - …
- Runtime environment



Java ME Platform

- Designed to run java application on devices with limited resources
- Java ME API is subset of Java SE API with some extra special class libraries
- Define profiles and configuration for different types of devices



Blu-ray Application Development using Java ME
http://www.rimlife.com/bluray/

3. Component Technology – Motivation

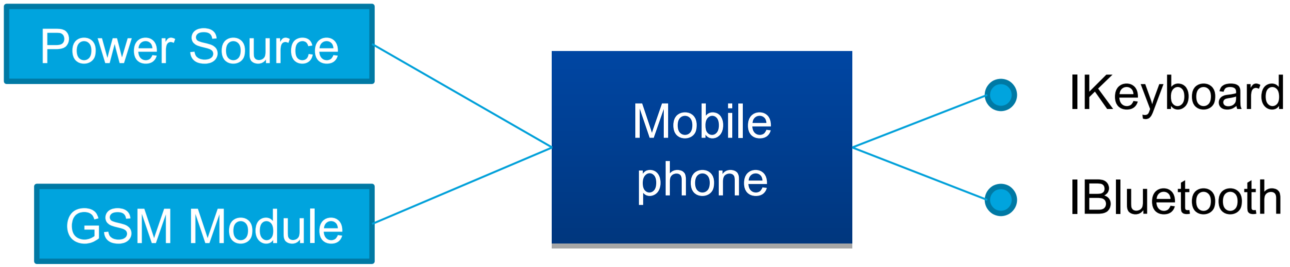
- **Development**
  - Reusability
  - Easy testing
  - The possibility of specialization of producers
- **Distribution**
  - Short time to market
  - Vendor independence
- **Service**
  - Reduced maintenance costs
  - Interchangeability – customer pressure to standardization

3.1. Application of component in IT

- **Hardware**
  - Memory, processors, motherboards
  - Peripherals – PnP, drivers
  - Communication elements
- **Software**
  - Graphical User Interface – Swing
  - Distributed application –CORBA, EJB, .NET, COM, DCOM, …
  - Databases
  - Information systems

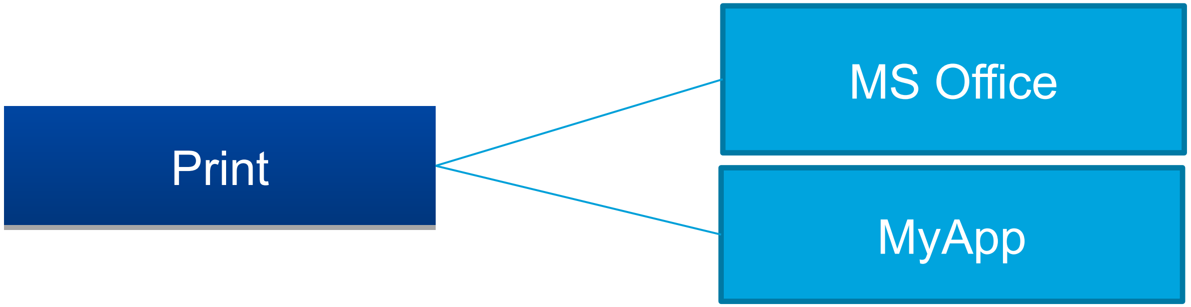
3.2. What is a component? 1/3

- Building unit with contractually defined:
  - interfaces
  - explicit contextual links



What is a component? 2/3

- It can be used independently of
  - the environment for which it was created.

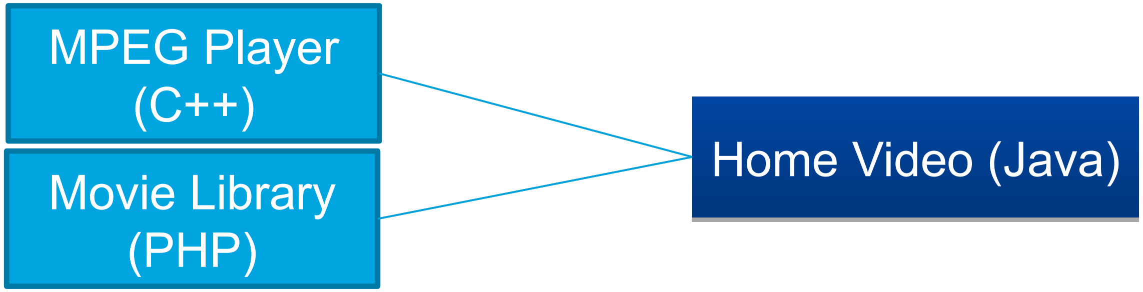


Print

MS Office

MyApp

  - the environment in which it was created.



MPEG Player (C++)

Movie Library (PHP)

Home Video (Java)

What is a component? 3/3

It is designed to integrate third party.

- **Author of component**
  - He does not know who and for what will others use the components.
  - He must meet specified interface.
- **Author of application**
  - He does not know who will supply components.
  - Communicates with component through the specified interface.
- **Integrator**
  - Link the application with the appropriate components.

Requirements for components

- Complete documentation
- Thorough testing
- Robust control of validity of inputs
- Return of sufficiently informative error messages
- Assumed that the component will be used for unexpected purposes.

Component specification

- State
  - **Properties** – read, write
- Behavior
  - **Methods** – invocation, parameters, results
- Interaction with the environment
  - **Events** – registration, notification

The life cycle of component 1/2

- **Creation of component**
  - Standards – defined by used technology CORBA, COM+, DCOM, EJB, .NET, JavaBeans
  - Binary compatibility - independent of language

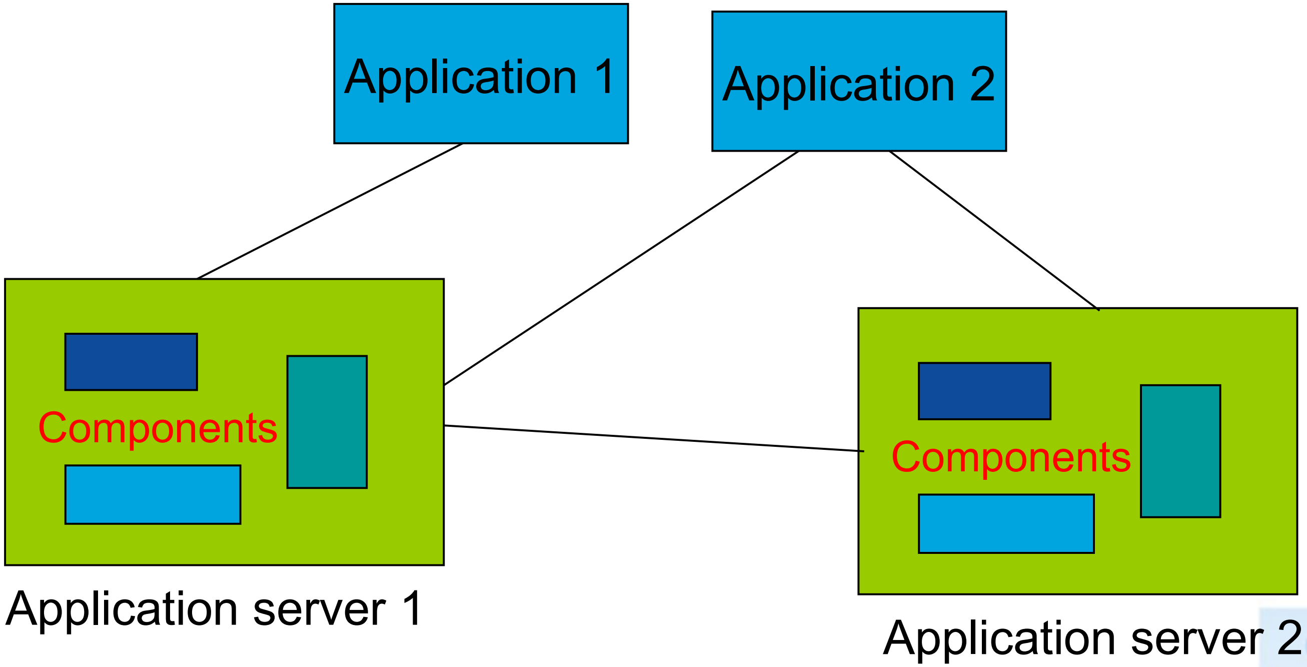
- **Publication of interface**
  - Documentation - for humans
  - Introspection - part of the components from which the client application can read the component's metadata

The life cycle of component 2/2

- **Distribution of component**
  - Publication of the component or registration of the component in one of the directory services
  - LDAP(standard from IETF), JNDI (Java), UDDI (XML web services)
  - Library distribution
- **Search for components**
  - Identification of components, realization of late binding
- **Making an application**
  - IDE support –access to the component such as to internal objects

Architecture of component-oriented systems

Architecture of system with distributed components – component can run on different servers.



Application server

- Environment for running applications and software components
  - Distributed environment
  - Resources
  - Security
  - Transaction
- Examples of Java application servers
  - JBoss, Jakarta Tomcat, BEA Weblogic, Citrix Meta Frame, IBM WebSphere, Oracle AS, Glassfish, Sun Java System AS, …

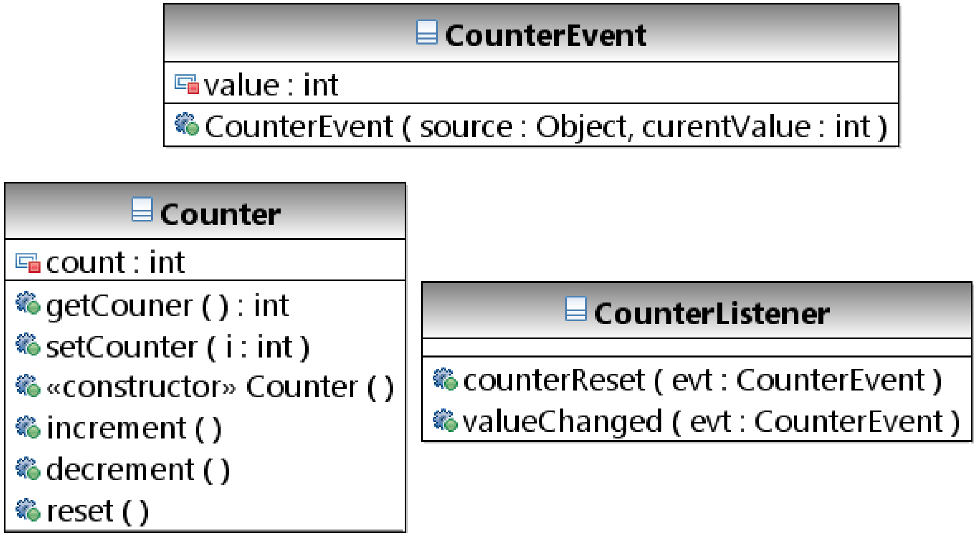
Major component technologies

- JavaBeans, EJB
  - Only Java language
  - Enterprise Java Beans – for large systems, distributed
- COM, COM+, DCOM, ActiveX
  - Binary compatible
  - The basic technology for Windows
- .NET
  - Compatibility at the language level – C++, C#, Jscript, VB.NET
- CORBA
  - Binary Compatible, Platform compatible
  - Many languages

2. JavaBeans

JavaBean component is a Java class meet a specified condition.

- Sources
  - http://java.sun.com/docs/books/tutorial/javabeans/TOC.htm
  - http://java.sun.com/javase/technologies/desktop/javabeans/docs/spec.html
  - http://www.cs.vsb.cz/behalek/vyuka/pte/prednasky/index.php



CounterEvent
- value : int
- CounterEvent (source : Object, curentValue : int)

Counter
- count : int
- getCouner ( ) : int
- setCounter (i : int)
- «constructor» Counter ( )
- increment ( )
- decrement ( )
- reset ( )

CounterListener
- counterReset (evt : CounterEvent)
- valueChanged (evt : CounterEvent)

JavaBeans - fundamentals

“A Java Bean is a reusable software component that can be manipulated visually in a builder tool.”

- Component granularity
  - Small building blocks
  - „whole application“
- Portability
- A uniform, high-quality API
- Simplicity
- Beans v. Class Libraries
- Design time vs. run-time
- Security Issues
- Internationalization
- Component persistency
  - Serialization or Externalization
- Local activation
- Multi-Threading

Java Beans – Component types

- **Visual components**
  - They have visual representation and occupying space in the window. Therefore, they must be subclass of class java.awt.Component
  - Examples: button, table, scrolling lists, HTML viewer
  - Support in visual tools
- **Invisible components**
  - Example: timer, spell checker, …
  - Support in visual tools

Java Beans – Component Structure

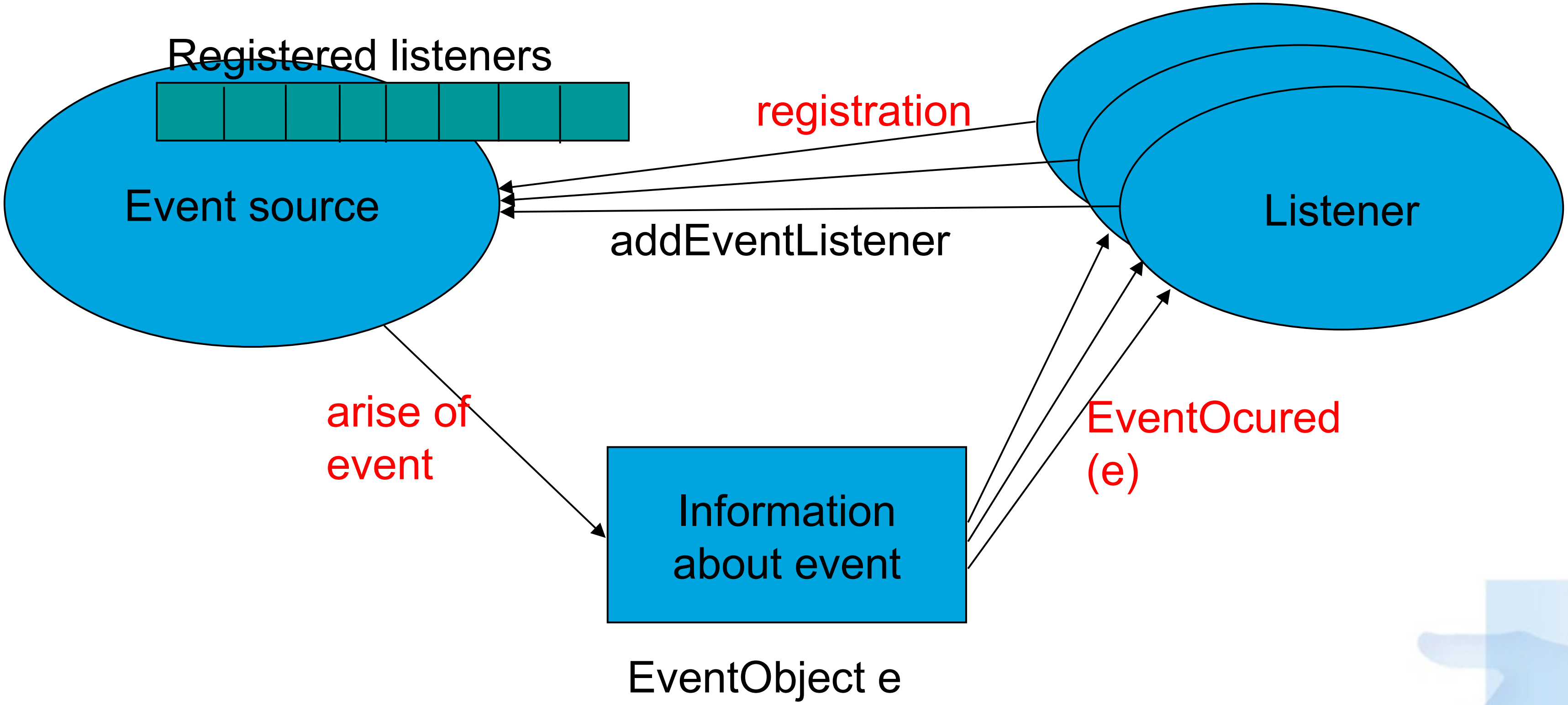
- **Properties**
  - Value of properties can be read or written through access method (get/set methods). Could not be accessed directly.
- **Methods**
  - Operation over components
- **Events**
  - Communication link between components

Java Beans – Events (1)

- **Source of events**
  - An object that generates events
  - Manages the list of registered listeners

- **Listener**
  - An object that wants to be informed about the event
  - Must be registered at the event source
  - Must implement the agreed interface

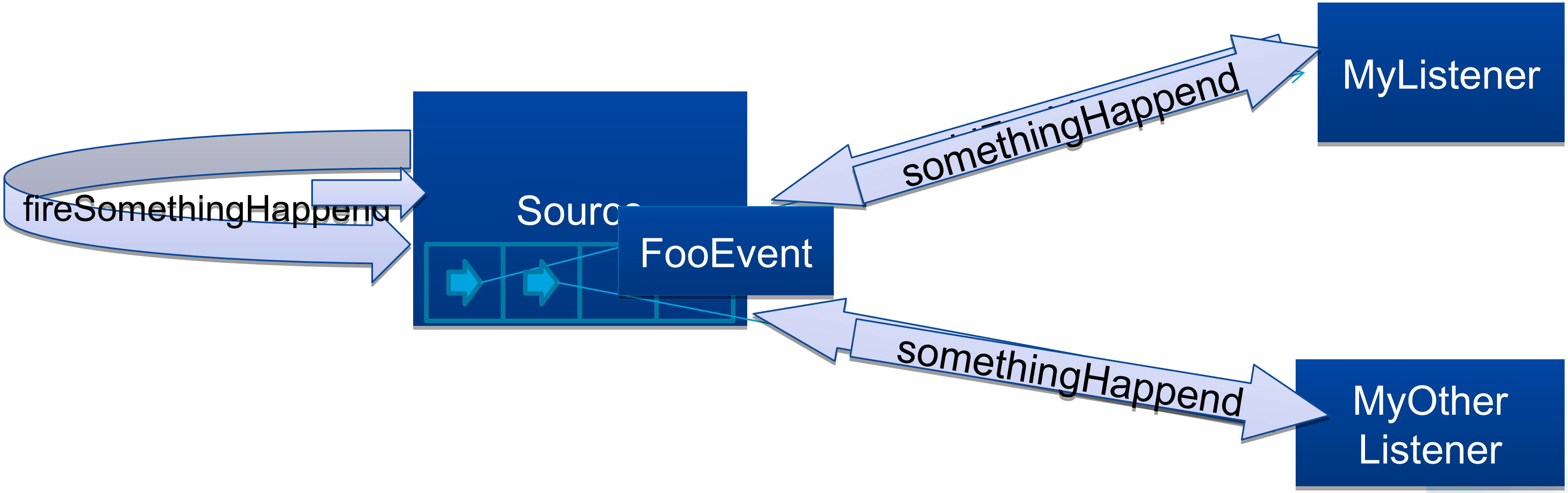
Java Beans – Events (2)



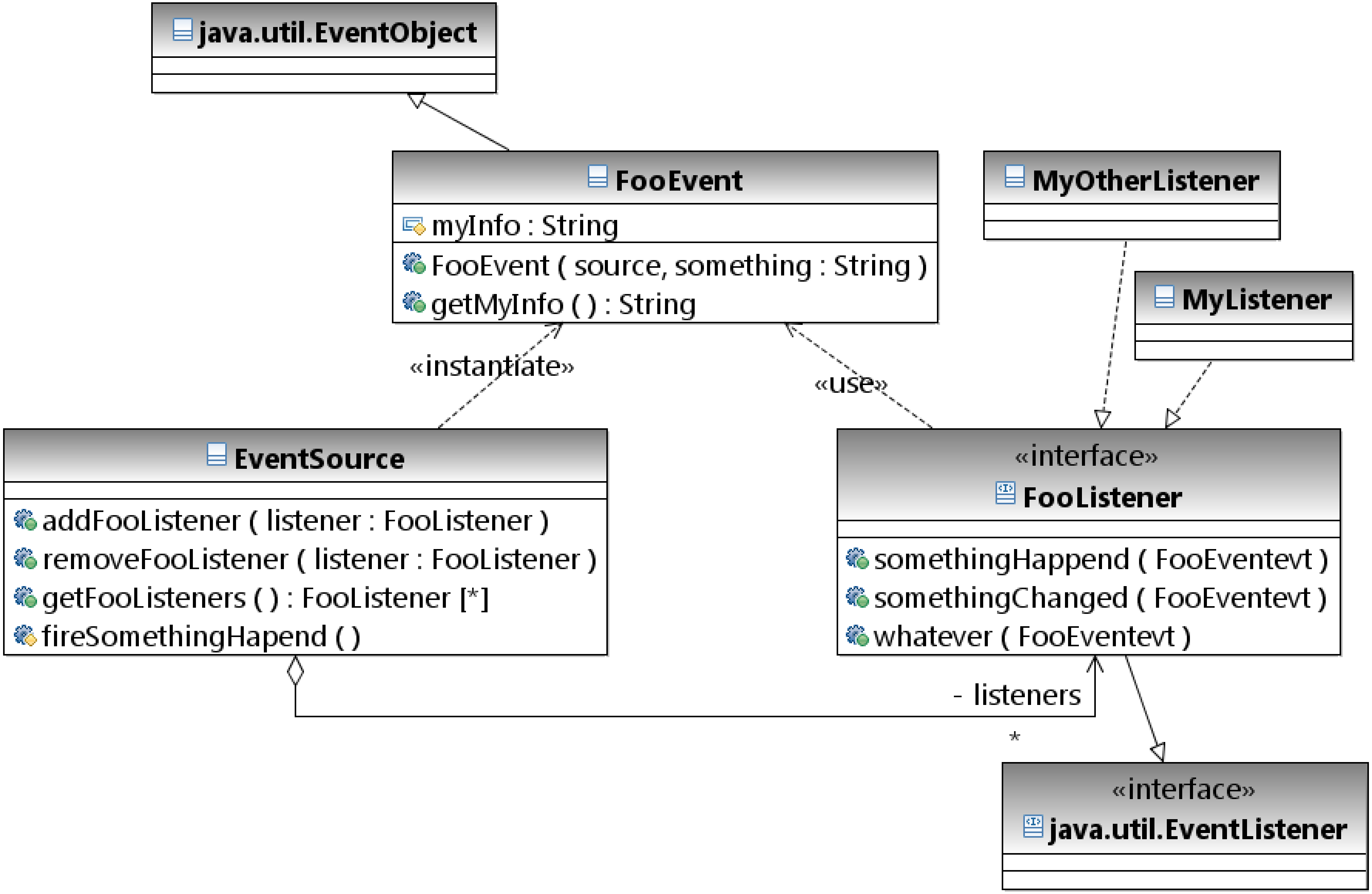
Java Beans – Events Handling

- The listener register to the event source (eg buttons, that we waiting to be pressed)
- The user press a button - an event occurred
- Event Source (button) pass through the list of registered listeners, and notify each one about occurred event:
  - Call the agreed method of listener interface
  - Pass event information to method as parameter (object of an subclass of java.util.EventObject)

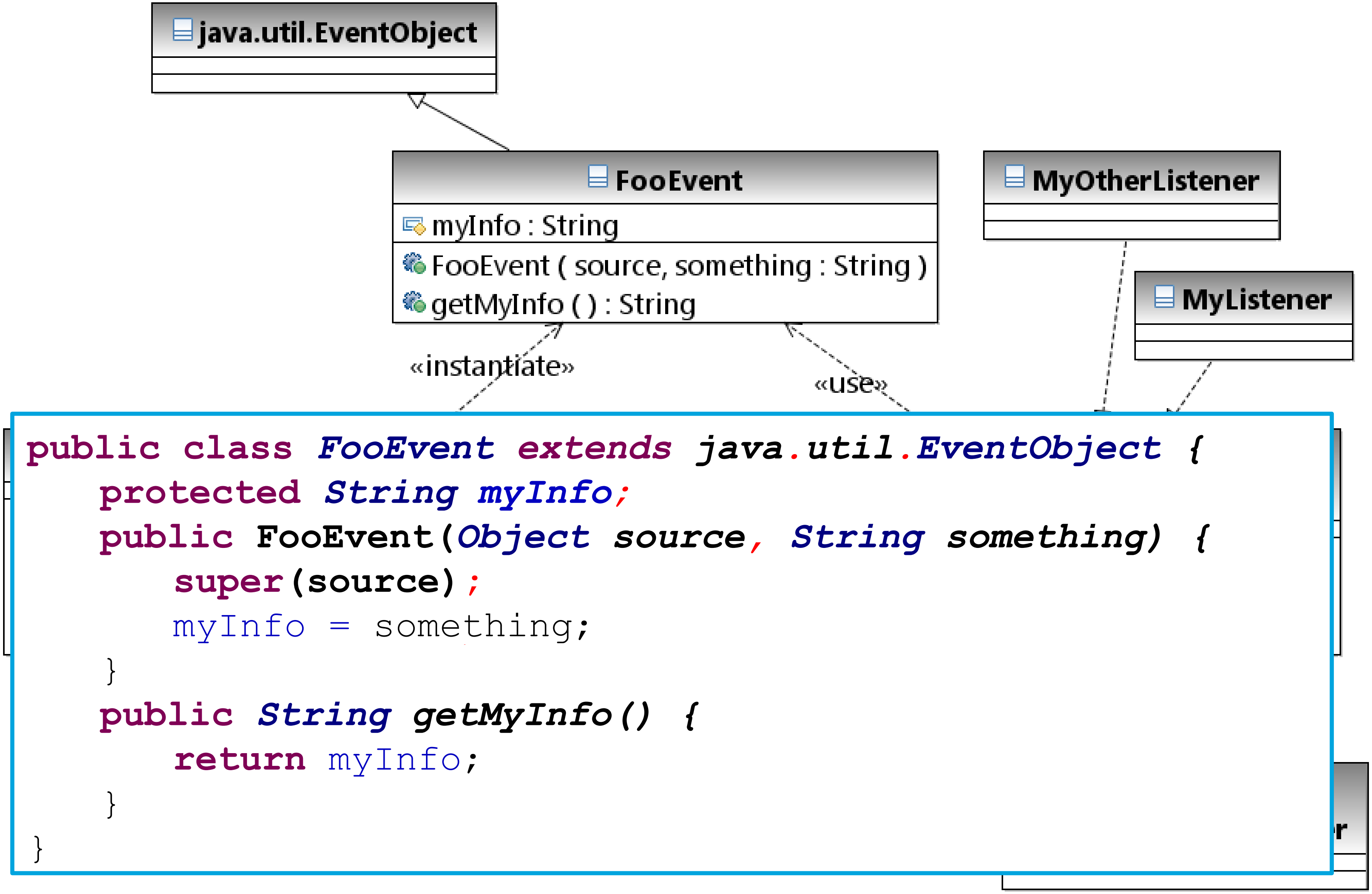
Events: Model Event – Listener Event handling



Events: Model Event – Listener

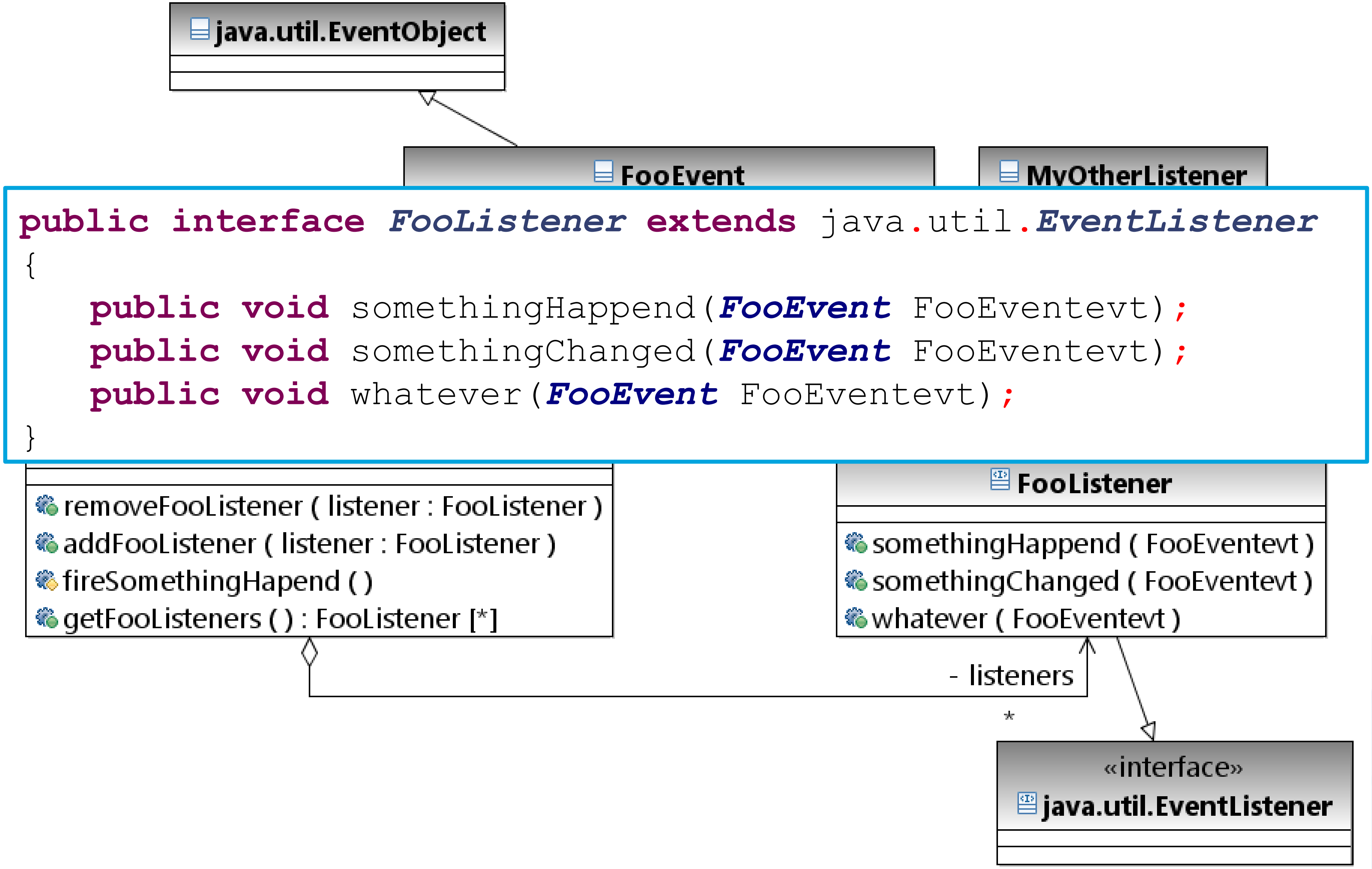


Události: Model Event – Listener



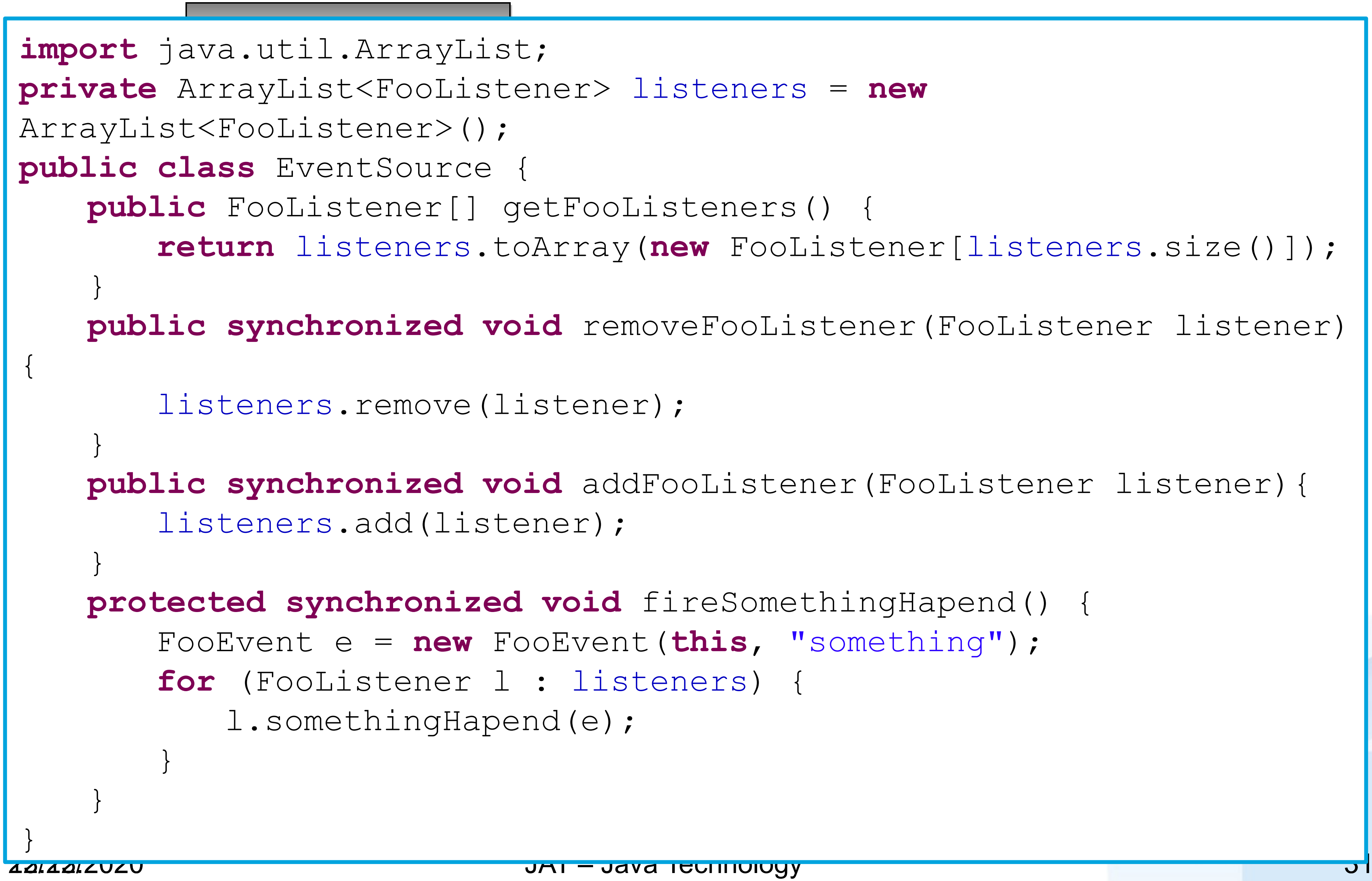
```
public class FooEvent extends java.util.EventObject {
   protected String myInfo;
   public FooEvent(Object source, String something) {
      super(source);
      myInfo = something;
   }
   public String getMyInfo() {
      return myInfo;
   }
}
```

Události: Model Event – Listener


java.util.EventObject
FooEvent
MyOtherListener
removeFooListener (listener : FooListener)
addFooListener (listener : FooListener)
fireSomethingHapend ( )
getFooListeners ( ) : FooListener [*]
FooListener
somethingHappend (FooEventevt)
somethingChanged (FooEventevt)
whatever (FooEventevt)
- listeners
*
«interface»
java.util.EventListener

```java
public interface FooListener extends java.util.EventListener
{
    public void somethingHappend(FooEvent FooEventevt);
    public void somethingChanged(FooEvent FooEventevt);
    public void whatever(FooEvent FooEventevt);
}
```

Události: Model Event – Listener



```
import java.util.ArrayList;
private ArrayList<FooListener> listeners = new
ArrayList<FooListener>();
public class EventSource {
    public FooListener[] getFooListeners() {
        return listeners.toArray(new FooListener[listeners.size()]);
    }
    public synchronized void removeFooListener(FooListener listener)
{
        listeners.remove(listener);
    }
    public synchronized void addFooListener(FooListener listener){
        listeners.add(listener);
    }
    protected synchronized void fireSomethingHapend() {
        FooEvent e = new FooEvent(this, "something");
        for (FooListener l : listeners) {
            l.somethingHapend(e);
        }
    }
}
```

JavaBeans – Events - Summary

- Class, that represent event
  - It have to be subclass of `java.util.EventObject`
  - The name must have the form `<something>Event`
  - It have to have constructor with at least one parameter (source of event)
- Interface for listeners (listener type)
  - It have to implements interface `java.util.EventListener`
  - The name must have the form `<something>Listener`
  - Each method should have one parameter of type `<something>Event`
  - The methods should return nothing (`void`)
- Class that represent source of events (JavaBean) must have methods:

```
public void add<ListenerType>(<ListenerType> listener)
  throws java.util.TooManyListenersException;
public void remove<ListenerType>(<ListenerType> listener)
public <ListenerType>[] get<ListenerType>s();
```

JavaBeans – Adapter(1)

- `EventListener` interface for a particular component can contain many methods.
- If we want to respond only to certain events:
  - We have to implement either a blank response to all other events
  - Or we use the adapter as the base class and implement only the chosen method
- The adapter implements the default response to all events

JavaBeans – Adapter(2)

```
public interface CounterListener {
    public void counterReset(CounterEvent evt);
    public void valueChanged(CounterEvent evt);
}


public class CounterAdapter implements CounterListener{
    public void counterReset(CounterEvent evt) {
    }
    public void valueChanged(CounterEvent evt) {
    }
}
```

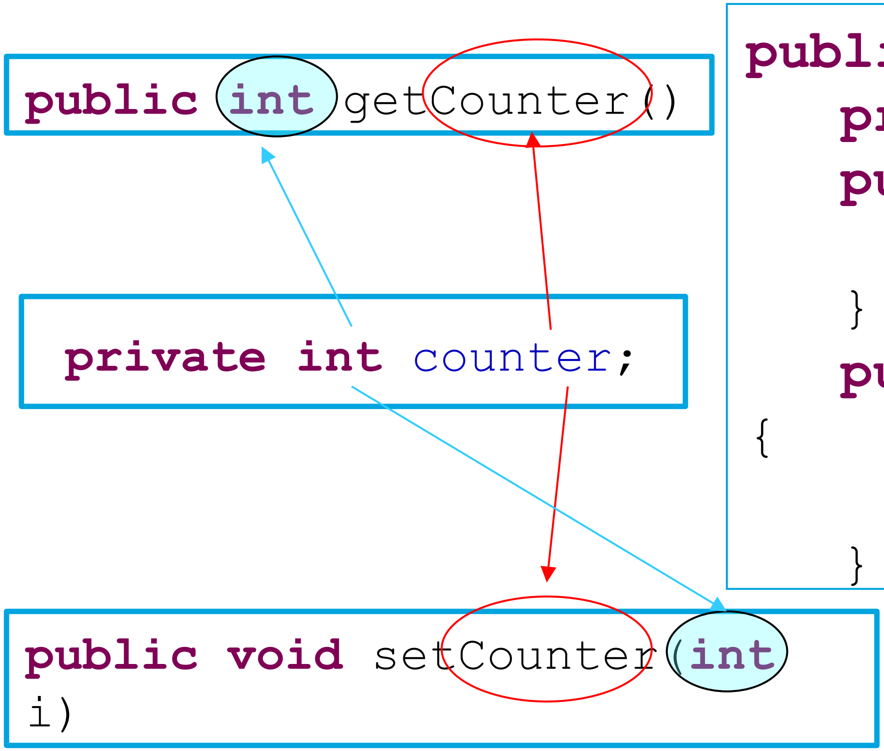
Java Beans – Adapter (3)

```
Counter counter = new Counter();
//use of anonymous inner classes
counter.AddCounterListener(new CounterAdapter() {
   @Override
   public void counterReset(CounterEvent e) {
      System.out.println("Reset");
   }
});
```

```
Counter counter = new Counter();
//use of anonymous inner classes
counter.AddCounterListener(new CounterListener() {
   public void valueChanged(CounterEvent evt) {
   }
   public void counterReset(CounterEvent e) {
      System.out.println("Reset");
   }
});
```

JavaBeans Properties

- Property with name **counter**



```
public int getCounter()
```

```
private int counter;
```

```
public void setCounter(int
i)
```

```
public class TickCounter {
    private int counter;
    public int getCounter() {
        return this.counter;
    }
    public void setCounter(int i)
{
        this.counter = i;
    }
```

JavaBeans – Properties

- **Simple properties**
  - Getter method

```
public <PropertyType> get<PropertyName>()
public boolean is<PropertyName>()
```

  - Setter method

```
public void set<PropertyName>(<PropertyType> value)
```

JavaBeans – Properties

- **Indexed properties**
  - indexed setter

```
void set<PropertyName>(int index, <PropertyType> value)
```

  - indexed getter

```
public <PropertyType> get<PropertyName> (int index)
```

  - array setter

```
public void set<PropertyName> (PropertyType values[])
```

  - array getter

```
public <PropertyType>[] get<PropertyName> ()
```

JavaBeans – Properties

```
protected ArrayList<JButton> buttons;
void setButtons(int index, JButton b){
 buttons.set(index, b);
}
public JButton getButtons(int index){
 return buttons.get(index);
}
public void setButtons(JButton values[]){
 buttons.clear();
 for(JButton b : values){
 buttons.add(b);
  }
}
public JButton[] getButtons(){
 return buttons.toArray(new JButton[buttons.size()]);
}
```

JavaBeans – Special properties

- Bound properties
  - Generate event PropertyChange from interface java.beans.PropertyChangeListener, if their value is changed.

- Constrained properties
  - Generate event VetoableChange from interface java.beans.VetoableChangeListener, if their value is changing.
  - Change can be rejected.

JavaBeans – Properties

Bound properties

```
public void addPropertyChangeListener(PropertyChangeListener x);
public void removePropertyChangeListener(PropertyChangeListener x);

public interface PropertyChangeListener extends java.util.EventListener {
    void propertyChange(PropertyChangeEvent evt);
}

public class PropertyChangeEvent extends java.util.EventObject {
    public PropertyChangeEvent(Object source, String propertyName,
     Object oldValue, Object newValue);

public String getPropertyName();
public Object getNewValue();
public Object getOldValue();
}
```

JavaBeans – Properties

Bound properties

```
public void addPropertyChangeListener(String propertyName,
             PropertyChangeListener listener);
public void removePropertyChangeListener(String propertyName,
             PropertyChangeListener listener);

public void add<PropertyName>Listener(
             PropertyChangeListener p);
public void remove<PropertyName>Listener(
             PropertyChangeListener p);
```

JavaBeans – Properties

Constrained properties

```
<PropertyType> get<PropertyName>();
void set<PropertyName>(<PropertyType> value)
         throws PropertyVetoException;

public void addVetoableChangeListener(
         VetoableChangeListener x);
public void removeVetoableChangeListener(
         VetoableChangeListener x);

public interface VetoableChangeListener extends
                  EventListener {
    public void vetoableChange(PropertyChangeEvent evt)
            throws PropertyVetoException;
}
```

JavaBeans – Properties

Constrained properties

```
void addVetoableChangeListener(String propertyName,
      VetoableChangeListener listener);
void removeVetoableChangeListener(String propertyName,
      VetoableChangeListener listener);

void add<PropertyName>Listener(
         VetoableChangeListener p);
void remove<PropertyName>Listener(
         VetoableChangeListener p);
```

JavaBeans – Properties

```
public void setPriceInCents(int newPriceInCents){

int oldPriceInCents = ourPriceInCents;
//First tell the vetoers about the change. If anyone
//objects, we let the PropertyVetoException propagate
//back to our caller.
vetos.fireVetoableChange("priceInCents", new
Integer(oldPriceInCents), new Integer(newPriceInCents));
//No one vetoed, so go ahead and make the change.
ourPriceInCents = newPriceInCents;
changes.firePropertyChange("priceInCents", new
Integer(oldPriceInCents), new Integer(newPriceInCents));


}
```

JavaBeans – Using the properties of component

- Attributes of objects in scripting languages
  - JSP, JSF
- Programmatic access through public access methods
- Access via forms (property sheets) in design tools
- Reading and writing to persistent memory

JavaBeans – Methods

- All public methods that are not getter or setter method of any property or add/remove method for registering listeners, are considered as component methods.

```
public void clear() {
   val=0;
}
public void increment() {
   val++;
}
```

Java GUI Toolkit – Introduction 1

- AWT Abstract Window Toolkit (import java.awt.*)
  - Included in JDK from first version of Java language, basic build blocks for creating complex user interface. Design of AWT use many design patters (most significant design pattern - Model-View-Controller).
  - Dynamic layout management of visual components

Java GUI Toolkit – Introduction 2

- Swing (import javax.swing.*)
  - Extends existing AWT toolkit, included in JDK from version 2, contains lot of new components, standard dialog windows, Look & Feel. Use AWT classes as parent classes and also use many design patterns.
  - Component are designed as lightweight, that means, that appearance and behavior is implemented directly in Java.
  - Dynamic layout management of visual components – 8 type of basic layout managers, but exist many others.
  - Part of JFC (Java Foundation Classes).
    - Support for data transfer (Cut/Copy/Paste, Drag & Drop).
    - Include Undo Framework (support for Undo a Redo operation).
    - Internationalization, Accessibility (disclosure of the contents visually impaired people).

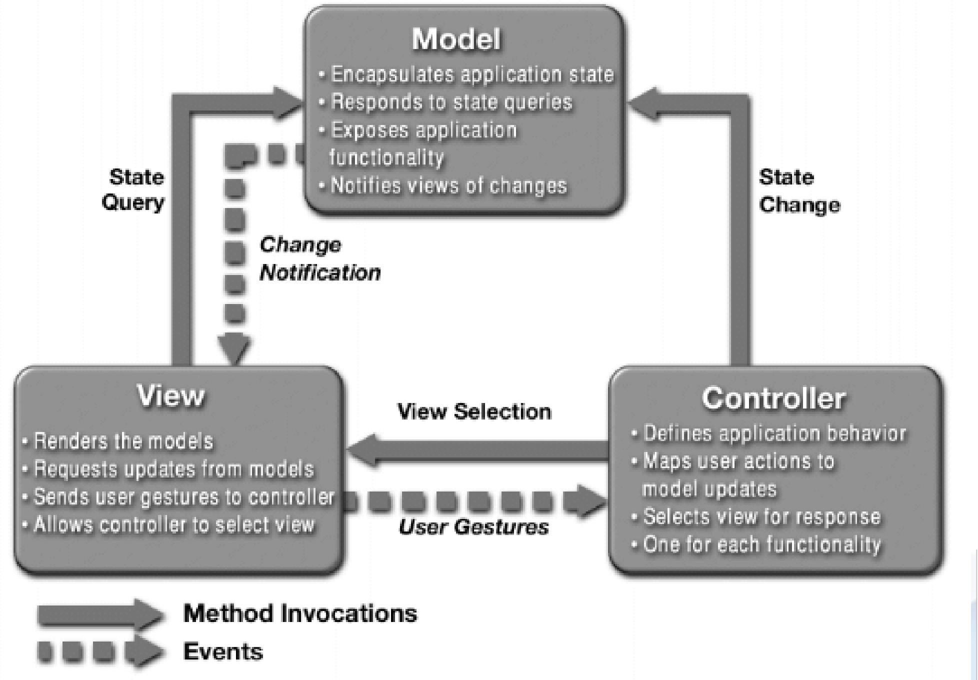
Java GUI Toolkit – Introduction 3

- JavaFX
  - The newest technology for creating user interfaces.
  - Focused on multimedia and easy creation of rich user interfaces.
  - This technology began as a single platform, but today it is applicable in the form of library.
  - Allows use of cascading style sheet (CSS) known from creation of the web pages.
- Another multiplatform alternative is SWT toolkit from IBM.

Design Pattern Model-View-Controller

- Exist many variants of MVC pattern but in Java SE is used following one:

- Properties:
  - It is possible to have
  - many views to one
  - model.
  - Reusability of model.


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

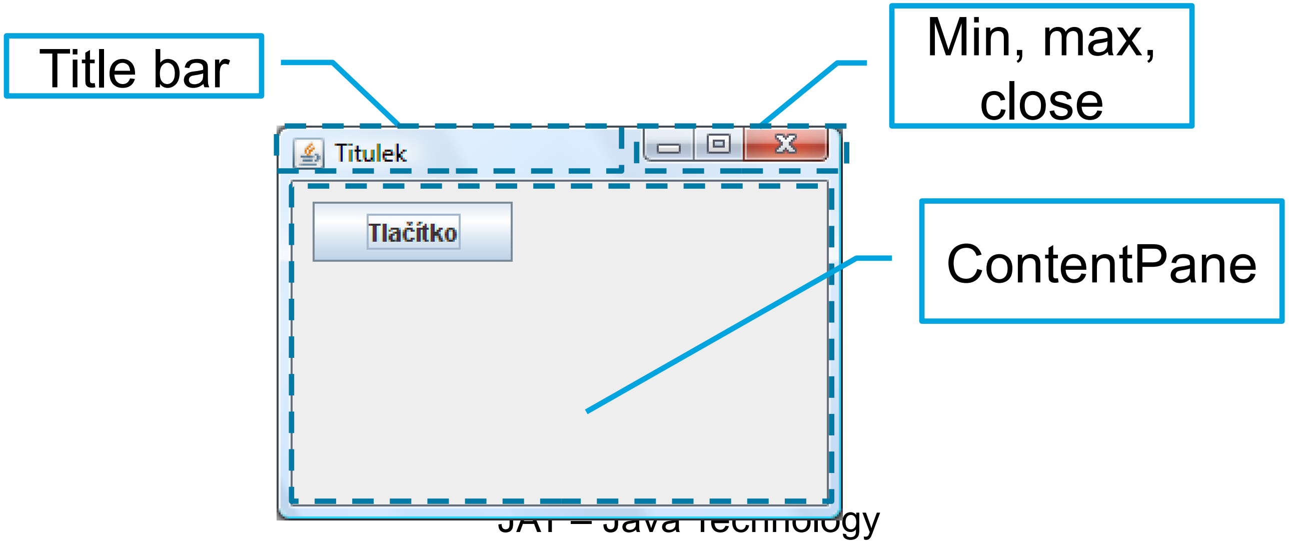
- http://java.sun.com/blueprints/patterns/MVC-detailed.html

Java GUI

- Fundamental components
  - java.awt.Component, javax.swing.JComponent
    - addMouseListener
    - addKeyListener
    - get/set[Preferred/Minimum/Maximum]Size
- Panels jawa.awt.Panel, javax.swing.JPanel
  - setLayout(LayoutManager l)
  - add(Component c, Object constraints)

The Most Frequently Used Components of Swing

- Root component for application UI: JFrame (JDialog and JApplet for application embedded in web page)
  - Application window contain standard elements, that can be hidden.
  - Communicate with operating system.
  - Contains container called ContentPane for other elements of user interface (for example JButton, JLabel, JPanel).
  - Methods for make window visible
    - pack(), setVisible(Boolean b)



Title bar

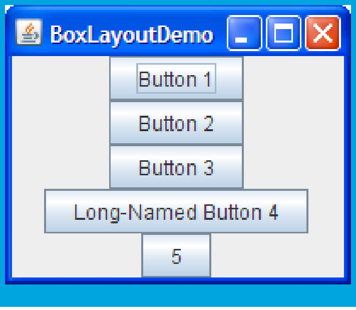
Min, max, close

ContentPane

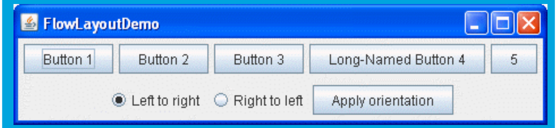
Layout Management

- Dynamic GUI
- Layout managers controls the placement of components within the container.

BoxLayout

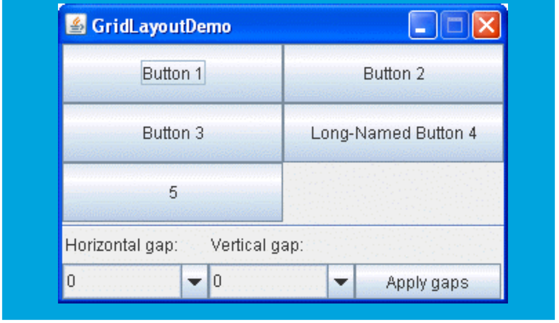


FlowLayout



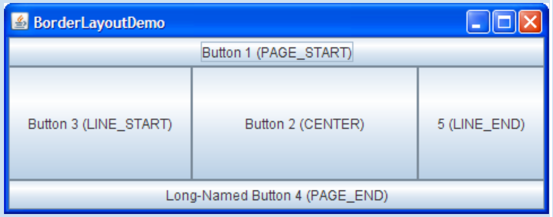
● Left to right ○ Right to left

GridLayout

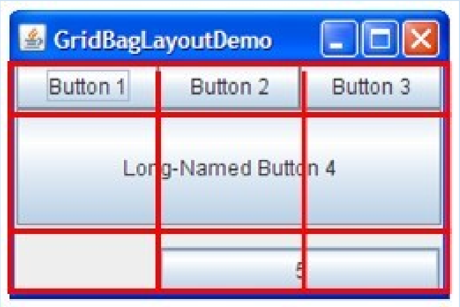


Horizontal gap: 0
Vertical gap: 0

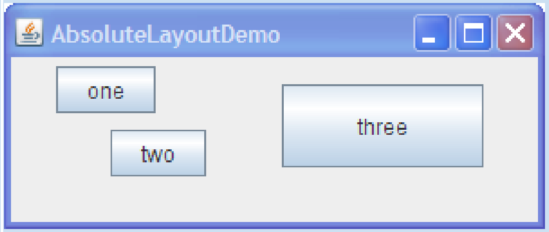
BorderLayout



GridBagLayout



“Null” Layout



Examples of Swing GUI

Examples many Swing components and their possibilities can be found in demo applications call SwingSet2 and SwingSet3. These demo applications also include source codes, that helps understand way of using swing components in custom Java code.

- <JAVA_JDK_HOME>\demo\jfc\SwingSet2\
- https://swingset3.dev.java.net/ (online version – runnable through Java WebStart technology)

Directory services

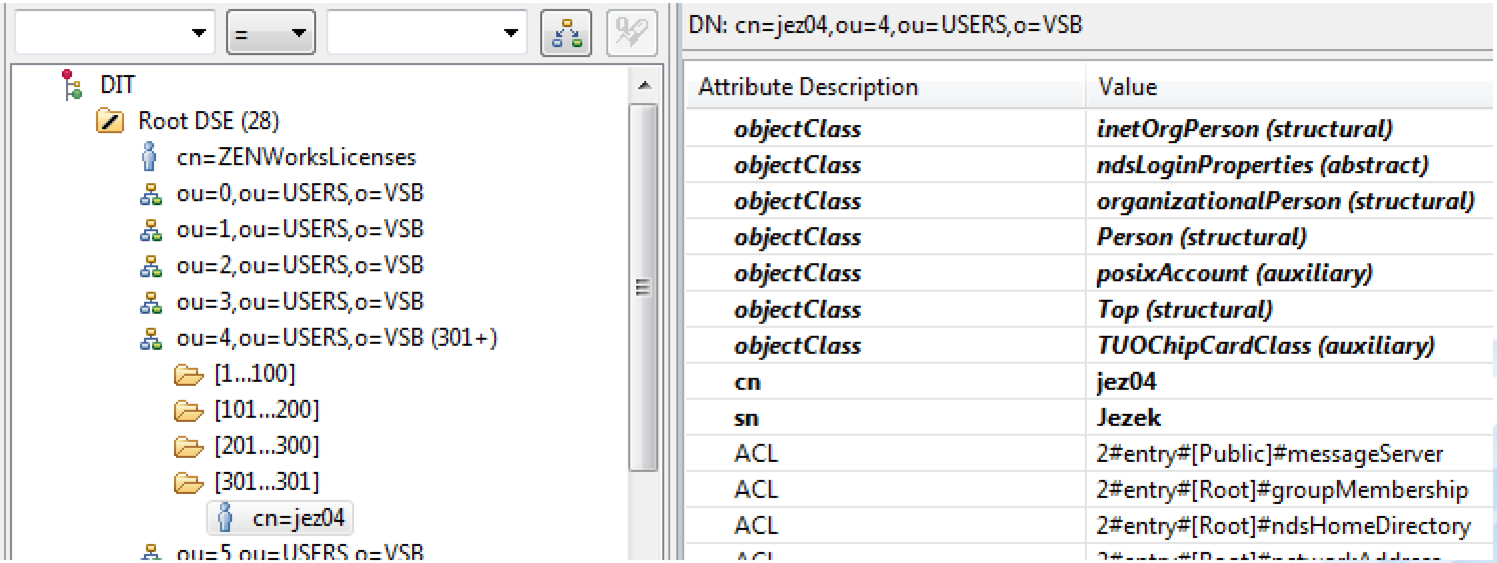
- Directory service
  - It is a software system, that store, organize and provide information in directory structure (tree structure)
- X500 series of standards for directory services
- Examples of directory services
  - DNS, LDAP, UDDI
- http://en.wikipedia.org/wiki/Directory_service

LDAP (*Lightweight Directory Access Protocol*)

- **LDAP** is defined protocol for storing and access to data in directory server. Base on this protocol are individual entries stored on server and organized to tree structure (like in real directory architecture).
- Lightweight „version“ of X500
- http://cs.wikipedia.org/wiki/LDAP

LDAP – fundamental terms

- Attribute – has type and name
- Class – set of attributes
- Schema – define concrete structure of classes
- Object(Entry) – instance of one class or more then one class



- DIT
  - Root DSE (28)
    - cn=ZENWorksLicenses
    - ou=0,ou=USERS,o=VSB
    - ou=1,ou=USERS,o=VSB
    - ou=2,ou=USERS,o=VSB
    - ou=3,ou=USERS,o=VSB
    - ou=4,ou=USERS,o=VSB (301+)
      - [1...100]
      - [101...200]
      - [201...300]
      - [301...301]
        - cn=jez04
    - ou=5,ou=USERS,o=VSB

DN: cn=jez04,ou=4,ou=USERS,o=VSB

Attribute Description	Value
objectClass	*inetOrgPerson (structural)*
objectClass	*ndsLoginProperties (abstract)*
objectClass	*organizationalPerson (structural)*
objectClass	*Person (structural)*
objectClass	*posixAccount (auxiliary)*
objectClass	*Top (structural)*
objectClass	*TUOChipCardClass (auxiliary)*
cn	**jez04**
sn	**Jezek**
ACL	2#entry#[Public]#messageServer
ACL	2#entry#[Root]#groupMembership
ACL	2#entry#[Root]#ndsHomeDirectory

LDAP – fundamental terms

- DN = Distinguished Name
  - DN: cn=jez04,ou=4,ou=USERS,o=VSB
- RDN = Relative Distinguished Name
  - RDN: cn=jez04

cn – common name

dc – domain component

mail – e-mail address and

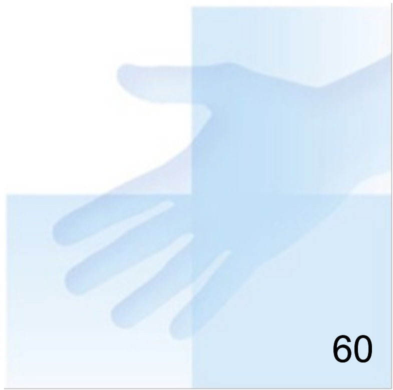
sn – surname

o – organization

ou – organization unit

Java Naming and Directory Interface (JNDI)

- JNDI is part of Java API that provide unified access to various naming and directory services for java application.
- http://java.sun.com/products/jndi/tutorial/index.html

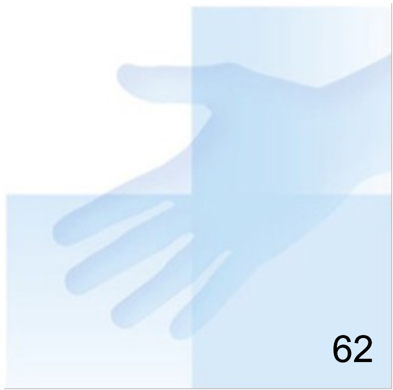


JNDI – Naming Concepts

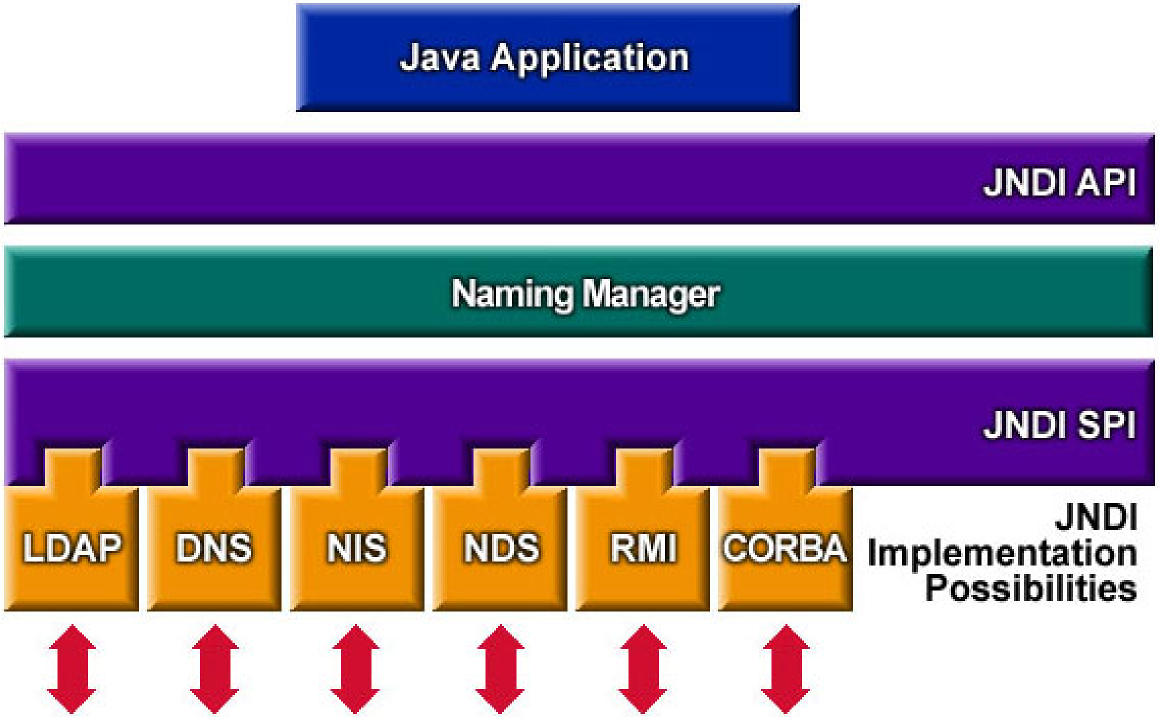
- Name – object naming
  - Name convention
    - Directory path in UNIX system: /home/fei
    - DNS: www.vsb.cz
    - LDAP: cn=jez04,ou=4,ou=USERS,o=VSB
- Bindings
  - Association of name with object
- References and Addresses
- Context
  - Context is set of bindings (name-to-object bindings)
  - Has defined name convention.
  - Subcontext
- Naming System

JNDI – Directory Concepts

- Attributes
  - Identifier
  - Value
- Directory
  - Set of objects
- Directory Service
  - Service provided basic operation such as add, delete, modify, search.



JNDI - Architecture



JNDI – Usage

Basic steps

- Download SPI library to connect to the service provider
  - Oracle provides library for the most widely used naming and directory services (LDAP, DNS, File system, Novell, …)
    - http://java.sun.com/products/jndi/serviceproviders.html
- Setting of environment variable for base context
- Obtain the context
- Using the context

JNDI – Usage: Setting

Setting of Environment Variables – Connection to LDAP

```
    Hashtable<String, String> env =
                                        new Hashtable<String,
String>();
    env.put(Context.INITIAL_CONTEXT_FACTORY,

"com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL,
                                        "ldap://
pca1035a.vsb.cz:10389/o=jat");
    env.put(Context.SECURITY_CREDENTIALS, "secret");
    env.put(Context.SECURITY_PRINCIPAL,
                                        "uid=admin, ou=system");
    env.put(Context.SECURITY_AUTHENTICATION, "simple");
```

JNDI – Usage: Setting

Setting of Environment Variables – Connection to File System

```
   Hashtable<String, String> env =
                                new Hashtable<String,
String>();
   env.put(Context.INITIAL_CONTEXT_FACTORY,

"com.sun.jndi.fscontext.RefFSContextFactory");
   env.put(Context.PROVIDER_URL,
               "file:/tmp/tutorial/");

   Context ctx = new InitialContext(env);
```

JNDI – Usage: Obtaining Context

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
   "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL,
   "ldap://pca1035a.vsb.cz:10389/o=jat");
...

try {
   Context ctx = new InitialContext(env);

   Object r  = ctx.lookup(
   "cn=homedir,cn=Jon Ruiz,ou=People/Windows");
   System.out.println(r);

   } catch (NamingException e) {
   }
```

JNDI – Usage: Using the Context

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
   "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL,
   "ldap://pca1035a.vsb.cz:10389/o=jat");

try {
   Context ctx = new InitialContext(env);

   Object r  = ctx.lookup(
   "cn=homedir,cn=Jon Ruiz,ou=People/Windows");
   System.out.println(r);

   } catch (NamingException e) {
   }
```

JNDI – Usage: Obtaining and Using the Directory Context

- Obtain of directory context

```
//Setting of environment variables
DirContext ctx = new InitialDirContext(env);
```

- Read the attributes from an entry

```
Attributes attrs = ctx.getAttributes(
        "cn = Ted Geisel, ou=People");
attrs.get("sn").get();
```

JNDI – Listing

- **`Context.list()`**
  - Only list of name
- **`Context.listBindings()`**
  - List of names and objects that are bind to them
- **`NamingEnumeration interface`**
  - ***`NamingEnumeration<NameClassPair>`***
  - ***`NamingEnumeration<Binding>`***

  Closure:
  - **`hasMore()`** return **`false`**
  - **`NamingEnumeration.close()`**
  - **`NamingException`**

JNDI – Operation with Object

- `Context.bind()`
  - Insert new specified object and bound it with specified new name
- `Context.rebind()`
  - Insert new object and bound it with existing name
- `Context.unbind()`
  - Delete name and bound object
- `Context.rename()`
  - Change the name for specified entry

JNDI – Operation with Context

- `Context.createSubcontext()`
  - Create new subcontext (node in tree structure)
- `Context.destroySubcontext()`
  - Delete existing subcontext
- `Context.rename()`
  - Change the name for specified entry even context

JNDI – Directory Operation

Obtaining attributes collection

- `Attributes answer = ctx.getAttributes("some dn");`
  - `answer.getAll(); answer.getIDs();`

Modification of collection of attributes

  - `ctx.modifyAttributes(name, DirContext.REPLACE_ATTRIBUTE, atributes);`
  - `DirContext.ADD_ATTRIBUTE`
  - `DirContext.REMOVE_ATTRIBUTE`

23/12/2020

JNDI – Directory Operation

Modification of attributes using ***ModificationItem***

- Create object of class ***ModificationItem***

```
new
  ModificationItem( DirContext.REPLACE_ATTRIBUTE,
        new BasicAttribute("mail",
        "geisel@wizards.com"));
```

- Pass collection of ***ModificationItem*** object to method **modifyAttributes**

```
ctx.modifyAttributes(name, modItems);
```

JNDI – Directory Operation

- Searching using attribute

```
// ignore attribute name case
Attributes matchAttrs = new BasicAttributes(true);
matchAttrs.put(new BasicAttribute("sn", "Geisel"));
matchAttrs.put(new BasicAttribute("mail"));
NamingEnumeration<SearchResult> answer =
ctx.search("ou=People", matchAttrs);
```

- Searching using filters

```
SearchControls ctls = new SearchControls();
String filter = "(&(sn=Geisel)(mail=*))";
NamingEnumeration<SearchResult> answer = ctx.search(
            "ou=People", filter, ctls);
```

JNDI – Directory Operation

- Searching filter syntax
  - Prefix notation of logical formulas
  - & (and), | (or), ! (not), = , ~=, >=, <=, =*, *, \
- Example

```
(| (& (sn=Geisel) (mail=*)) (sn=L*))
```

- Control of search operation `SearchControls`
  - Scope of searching
  - Required attributes
  - Time limit
  - Limitation for number of returned entry

JNDI - Names

- Can be used two types of names
  - Strings
  - Interface **`Name`**
- **`CompoundName`** implements interface **`Name`**
  - Name is composed from individual parts of tree nodes
  - Can be easily generated programmatically
  - Construction of name is independent of context system provider.
    - C:\Windows\system\
    - /home/staff/
    - cn=jez04, ou=4, ou=USERS, o=VSB
    - floreon.vsb.cz

JNDI – Names: CompoundName

- Name is composed from individual parts of tree nodes.
- Collection of parts of name contains root at position with index 0 and continue with subcontext until reach current entry.
- That means construction of name is always same even it is used file system provider or DNS provider, where name syntax has different direction (from root to leaf and from leaf to root) and different name parts separator.
- Parsing of string name to `CompoundName`

```
NameParser parser = ctx.getNameParser("");
Name cn = parser.parse(compoundStringName);
```

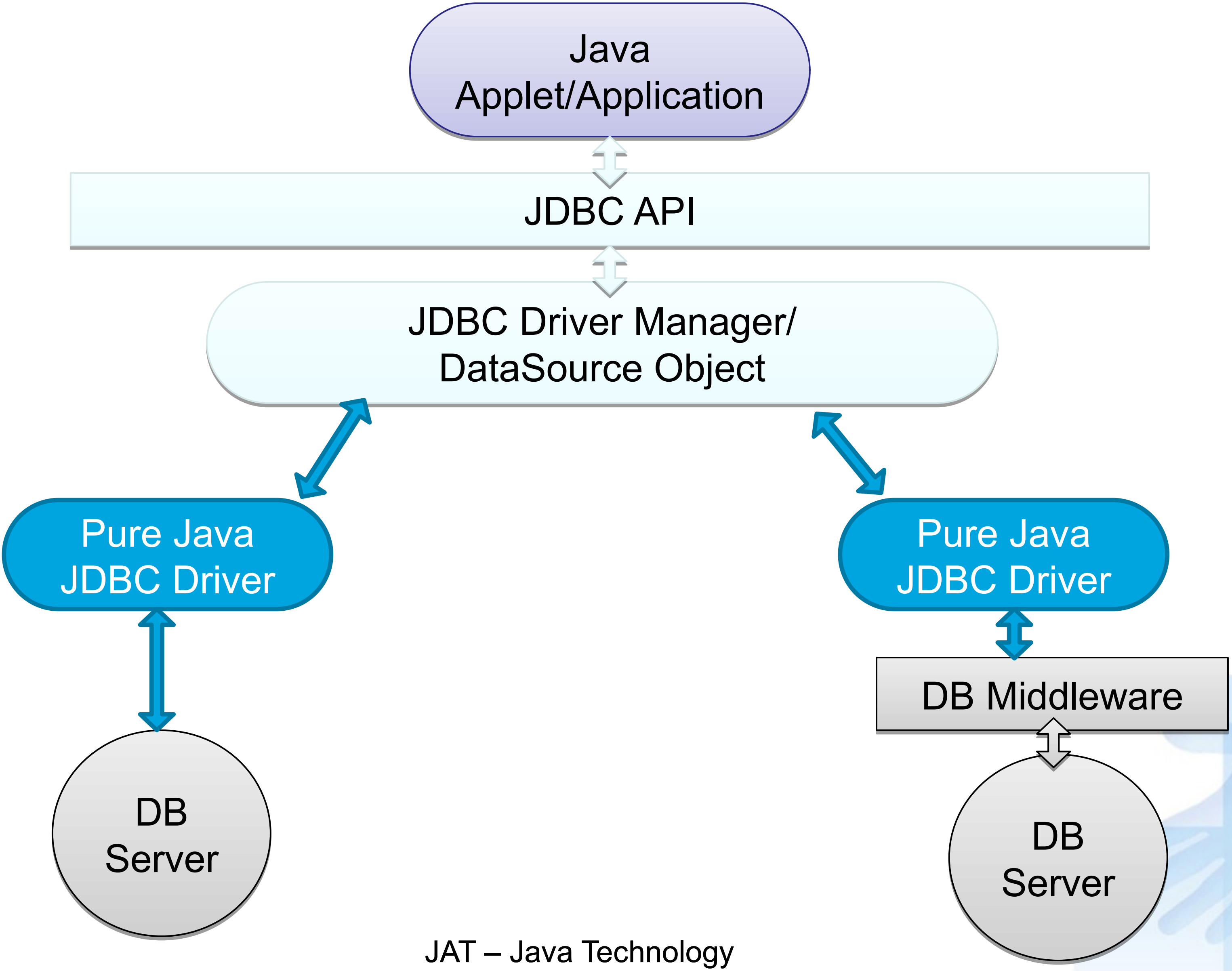
- Methods for `CompoundName` construction

```
getAll();
get(int posn);
getPrefix(int posn);
getSuffix(int posn);
add(String comp);
add(int posn, String comp);
addAll(Name comps);
addAll(Name suffix);
addAll(posn, Name suffix);
remove(posn);
```

JDBC – Java DataBase Connection

- Technology for unified access to database form Java.
- JDBC API
  - Establishing connection with database
  - Creating and sending SQL statement to database
  - Retrieving and process result of SQL query
- http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html

JDBC – Architecture



JDBC Drivers

- Types:
  - JDBC – ODBC bridge driver
  - Java and native code Driver
  - Pure Java Driver – communication with database server
  - Pure Java Driver – communicate with middleware server

- Concrete drivers for common database systems
  - MySQL Connector/JDBC
    - http://www.mysql.com/downloads/connector/j/
      - mysql-connector-java-X.X.XX-bin.jar
  - Oracle Database 11g R2 JDBC driver
    - http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html
  - JavaDB – distribution of Apache Derby database
    - Included in JDK from version 7
    - Written completely in Java
  - MSSQL, …
- **Obtained driver library must be linked to Java application in same way as other used libraries.**

JDBC – Driver Initialization

- Using DriverManager

```
try {
    Class.forName("com.mysql.jdbc.Driver")
            .newInstance();
    Class.forName("oracle.jdbc.OracleDriver")
            .newInstance();

Class.forName("org.apache.derby.jdbc.EmbeddedDriver")
            .newInstance();

} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (InstantiationException e) {
    e.printStackTrace();
} catch (IllegalAccessException e) {
    e.printStackTrace();
}
```

JDBC – Establishing Connection

- Using `DriverManager`

```
try {
   Connection conn = DriverManager.getConnection(
         "jdbc:mysql://localhost/shop" +
         "?user=shop&password=shop");
   Statement stm = conn.createStatement();
   ResultSet rs = stm.executeQuery("show tables");
   while(rs.next()){
      System.out.println(rs.getString(1));
   }
} catch (SQLException e) {
   e.printStackTrace();
}
```

- Connection string syntax
  - jdbc:oracle:thin:[user/password]@[host][:port]:SID
  - jdbc:derby:database name[;create=true]
  - jdbc:mysql://[host][,failoverhost...][:port]/[database][?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...

JDBC – Establishing Connection

- Obtaining of `DataSource`

```
Context ctx = new InitialContext(env);
DataSource ds = (DataSource) ctx.lookup(
            "java:comp/env/jdbc/myDB");
Connection con = ds.getConnection();
```

- Creation of `DataSource`

```
com.mysql.jdbc.jdbc2.optional.MysqlDataSource ds =
   new MysqlDataSource();
ds.setPort(3306);
ds.setDatabaseName("jat_example");
ds.setServerName("localhost");
ds.setUser("admin");
ds.setPassword(„heslo");
ctx.bind("java:comp/env/jdbc/myDB", ds);
```

JDBC – SQL Query

- Creating and sending SQL query to database

```
con = factory.getConnection();
Statement stmt = con.createStatement(
                    ResultSet.TYPE_SCROLL_SENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery(
            "SELECT cof_name, price FROM coffes");
```

- Other methods of class ***Statement***
  - `boolean execute(String)`
  - `ResultSet getResultSet()`
  - `int executeUpdate(String)` (for SQL INSERT, UPDATE, DELETE)
  - `close()`

JDBC – Process Results

- Process results – read data and ResultSet metadata

```
ResultSetMetaData meta = rs.getMetaData();
while(rs.next()){
for(int i=1; i<=meta.getColumnCount(); i++){
System.out.println(meta.getColumnName(i) +
": " + rs.getString(i));
}
```

- Moving cursor
  - When result set is returned cursor point before first row
  - `next()` return `false` if cursor move after last row
  - `last()`, `first()` , `previous()` , `relative(int)` , `absolute(int)`

JDBC – Process Results

- Data reading
  - get**XXX**()
    - byte, double, float, int, long, string, short, BigDecimal, Blob, Date, Time
- Meta-information
  - ***`ResultSetMetaData`*** **`getMetaData()`**

JDBC – Update of Tables

- Using SQL query (**`UPDATE myTable SET column1=‘value’ WHERE`** …)
- Using **`ResultSet`**

```
Statement stmt = con.createStatement(
ResultSet.TYPE_SCROLL_SENSITIVE,
   ResultSet.CONCUR_UPDATABLE);
ResultSet uprs = stmt.executeQuery(
   "SELECT COF_NAME, PRICE FROM COFFEES");
uprs.next();
uprs.updateString("COF_NAME", "Foldgers");
uprs.updateRow();
```

- **`cancelRowUpdates();`**
- **`updateXXX()`** - double, float, int, string, Time, …
- **`deleteRow()`**
- **`moveToInsertRow()`**

JDBC – Automatically Generated Keys

- Some database tables generate unique keys for all newly inserted records.
- If application need know these keys it must use following lines of code:

```
stmt.executeUpdate("INSERT INTO autoincSample (column1) VALUES
('Record 1')", Statement.RETURN_GENERATED_KEYS);
rs = stmt.getGeneratedKeys();
```

- Application have to pass flag *RETURN_GENERATED_KEYS* to method *executeUpdate*
- Application can obtain the keys using method *getGeneratedKeys()* and returned result set contains all generated key from executed statement.

JDBC – Prepared Statements

- Could be used if application repetitively process same statement only with different data.
- Usage of prepared statement preserve application against:
  - SQL injection attack
  - Bad data transformation to string (application side) and back to proper data type (database side)
- SQL statement is send to DBMS and compiled, after that can be processed repetitively (time saving, better security)

```
PreparedStatement prepStm = con.prepareStatement(
"UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");

prepStm.setInt(1, 75);
prepStm.setString(2, "Colombian");
prepStm.executeUpdate();
```

JDBC – Transaction

- Each single SQL statement is treated as transaction
- Don’t exist command „BeginTransaction“ it is performed automatically
- If application need more then one statement in one transaction, it have to use method `setAutoCommit()`

```
con.setAutoCommit(false);
updateSales.executeUpdate("UPDATE COFFEES SET SALES = 50 WHERE
COF_NAME LIKE ‘Colombia’");
updateTotal.executeUpdate("UPDATE COFFEES SET TOTAL = TOTAL + 50
WHERE COF_NAME LIKE ‘Colombia’");
con.commit();
con.setAutoCommit(true);
```

JDBC – Transaction

- rollback() – cancel transaction and change values in DB into state before transaction begin.
- SavePoint – allow rollback transaction to this point (SavePoint)

```
//Process some SQL statements
Savepoint svpt1 = con.setSavepoint("SAVEPOINT_1");
// Process some SQL statements
con.rollback(svpt1);
// Process some SQL statements
con.commit();
```

- `con.releaseSavepoint(svpt1);`

JDBC – Stored Procedures

- Stored procedure sreation

```
String createProcedure = “proprietary code to creating
gprocedur";
Statement stmt = con.createStatement();
stmt.executeUpdate(createProcedure);
```

- Stored procedure call

```
CallableStatement cs = con.prepareCall(
         "{call SHOW_SUPPLIERS(?, ?)}");

cs.setXXX(int, String);

ResultSet rs = cs.executeQuery();
```

JDBC – Exceptions, Warnings

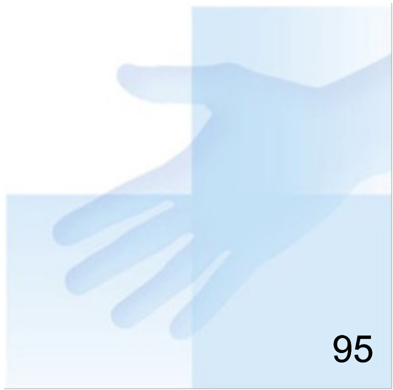
- SQL erroes and exception threw by database server are wrap to java exception with class **SQLException**
- Warnings get be get from object of class **Connection**, **Statement**, **ResultSet**

```
SQLWarning warning = stmt.getWarnings();
while (warning != null) {
    System.out.println("Message: "+
warning.getMessage());
    System.out.println("SQLState: "+
warning.getSQLState());
    System.out.print("Vendor error code: ");
    System.out.println(warning.getErrorCode());
    warning = warning.getNextWarning();
}
```

JDBC – Closing the Connection

- Don’t forget close the connection if you don’t need it anymore.

```
con.close();
```



JavaAPI for XML

- JAXP (SAX, DOM, XSLT, StAX)
  - Apache Xerces
- JAXB
  - JavaEE

XML – history

- **SGML** (*Standard Generalized Markup Language*) is a standard for defining generalized markup languages for documents. Which allows define markup language as oven subsets. SGML is a complex langugage which allows many markup syntaxes. That complexity is disadvantage for common usage.
- SGML is ISO standard called *ISO 8879:1986 Information processing—Text and office systems—Standard Generalized Markup Language (SGML)*

XML – history

- Language XML is created as profile (specialized subset) of SGML and become very popular.
- XML can be easy parsed and processed because of simplicity.

- XHTML, GML, SVG, MathML, DocBook

XML – example

```
<math xmlns=" h t t p : / /www.w3. org /1998/Math /MathML">
<mrow>
  <msup>
      <mfenced open=" [ " close=" ] ">
          <mrow>
              <mi>a< / mi>
              <mo>+< /mo>
              <mi>b< / mi>
          < /mrow>
      < /mfenced>
      <mn>260< /mn>
  < /msup>
< /mrow>
```

$$[a+b]^{260}$$

XML

View: Data model

- XML document is modeled as tree (XML tree).
- Notice: This data model was alsow presented in SGML language and in database community is known as weakly structured data.

XML – well formed document

- Element has type identified by name (known as tag). Exampel: <book>...</book>.
- Element can contains set of pairs attribute=’value’.
- In text form XML document can be identified start (start-tag) and end mark(end-tag) of element (<name>...</name>).
- Text between start and end mark is called element content.

XML – well formed document

- If element contains others tags and characters contents is called mixed content. For example:

  <a>Hi, <b>Mike</b></a>.

- Elements with no content are called empty. Short syntax:

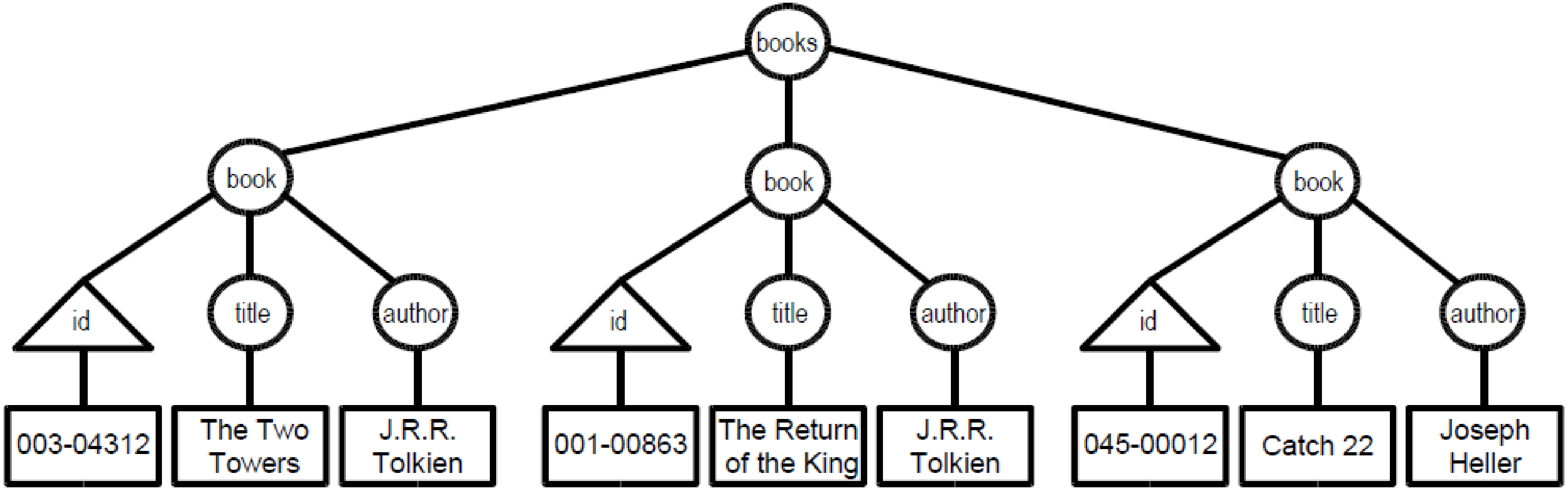
  <img src="picture.jpg"/>.

- First lien contains XML declaration, for example:

  <?xml version="1.0" ?>

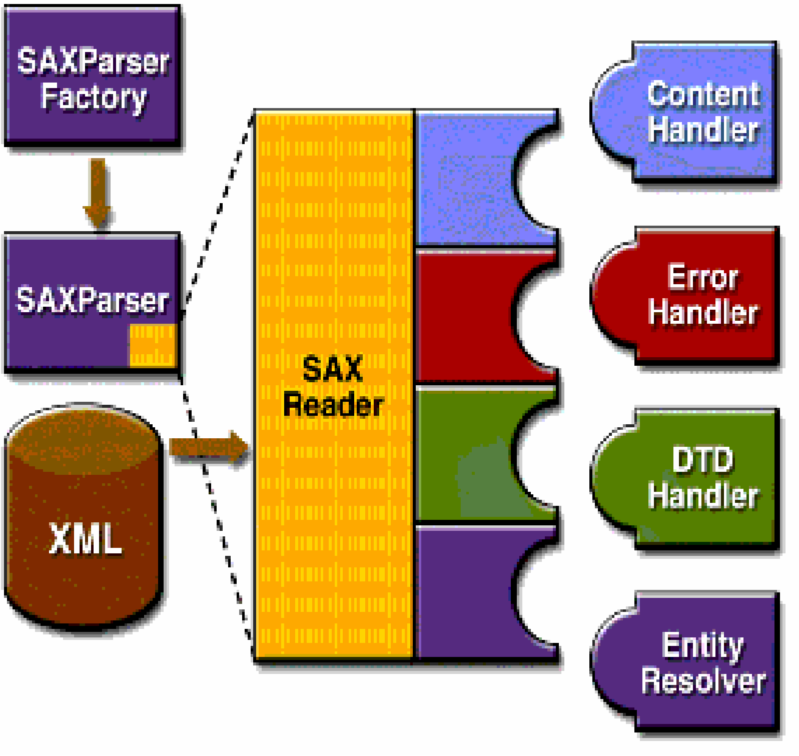
- Document is called well formed if fullfil all these rules.

XML - tree


books
book
book
book
id
title
author
id
title
author
id
title
author
003-04312
The Two Towers
J.R.R. Tolkien
001-00863
The Return of the King
J.R.R. Tolkien
045-00012
Catch 22
Joseph Heller

JavaAPI for XML – SAX

- Event model
- SAXParser inform about found start or end tags, …



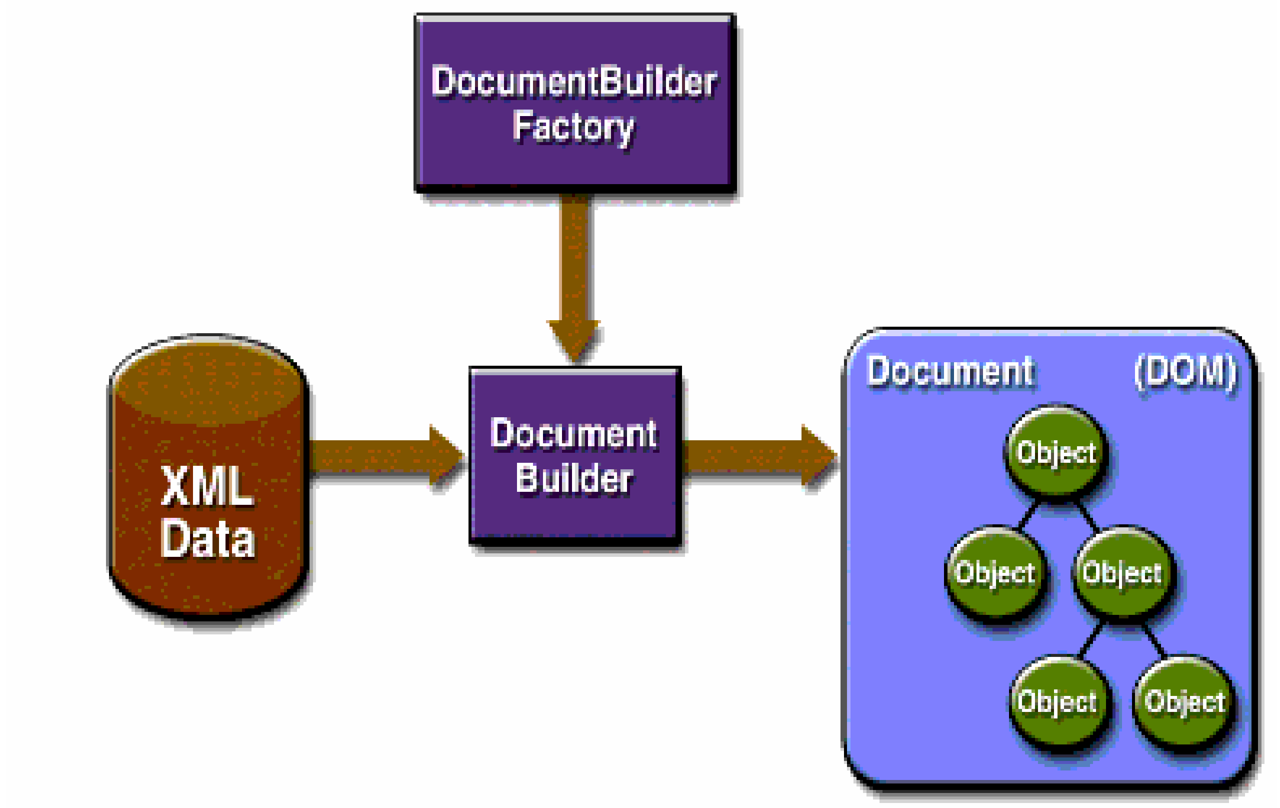
JavaAPI for XML – SAX

```
public class Parse {
   public static void main(String[] args) {
      XMLReader xr = null;
      try {
         xr = XMLReaderFactory.createXMLReader();
      } catch (SAXException e) {
         e.printStackTrace();
      }
      MyHandler h = new MyHandler();
      xr.setContentHandler(h);
      xr.setErrorHandler(h);
      try {
         InputSource source = new InputSource(
                        new FileReader("test.xml"));
         xr.parse(source);
      } catch (Exception e) {}
   }
}
```

```
        public class MyHandler extends DefaultHandler {
              protected int counter = 0;

  @Override
  public void endDocument() throws SAXException {
      System.out.println("pocet: " + counter);
  }
  @Override
  public void endElement(String uri, String localName,
String qName) throws SAXException {}
  @Override
  public void characters(char[] ch, int start, int length)
        throws SAXException {}
  @Override
  public void startDocument() throws SAXException {}
  @Override
  public void startElement(String uri, String localName,
String qName, Attributes attributes) throws SAXException {
      if(qName.equals("property")){
         counter++;
      }
  }}
```

JavaAPI for XML – DOM

- Whole document is loaded to memory and DOM tree is created.



JavaAPI for XML – DOM

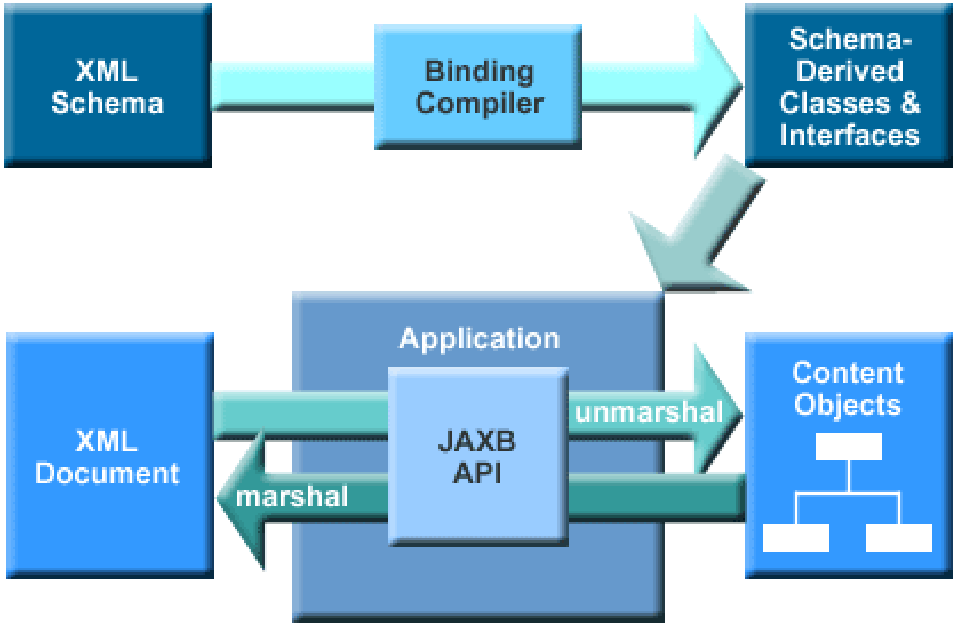
```
public class Tree {
   public static void main(String[] args) {
      DocumentBuilderFactory dbfactory =
               DocumentBuilderFactory.newInstance();
      Document doc = null;
      try {
         DocumentBuilder builder =
         dbfactory.newDocumentBuilder();
         doc = builder.parse(new File("test.xml"));
      } catch (Exception e) {}
      Element root = doc.getDocumentElement();
      NodeList nl = root.getChildNodes();
      for (int i = 0; i < nl.getLength(); i++) {
         String name = nl.item(i).getNodeName();
         System.out.println(name);
      }
   }}
```

JavaAPI for XML – DOM XPath

```
try {
  XPathFactory factory = XPathFactory.newInstance();
  XPath xPath = factory.newXPath();
  Object list = xPath.evaluate(
     "BookCatalogue/Book/*", doc,
XPathConstants.NODESET);
  NodeList nl = (NodeList)list;
  for (int i = 0; i < nl.getLength(); i++) {
    System.out.println(nl.item(i).getNodeName() + "='"
+
       nl.item(i) .getNodeValue() + "' ;" +
       nl.item(i).getTextContent());
  }
} catch (XPathExpressionException e) {
  e.printStackTrace();
}
```

JAXB – Java Architecture for XML binding

- Easy access to data in XML file and storage of data into XML
- https://jaxb.java.net/
- Included in JavaSE 7 and newer



JAXB

- Binding a schema
  - xjc.bat -p book book.xsd -d d:\Temp
- Load of XML document

```
try {
File file = new File("file.xml");
JAXBContext jaxbContext =
JAXBContext.newInstance(Setting.class);
Marshaller jaxbMarshaller =
jaxbContext.createMarshaller();
// output pretty printed
jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OU
TPUT, true);
jaxbMarshaller.marshal(setting, file);
jaxbMarshaller.marshal(setting, System.out);
} catch (JAXBException e) {}
```

JAXB

- Save of XML document

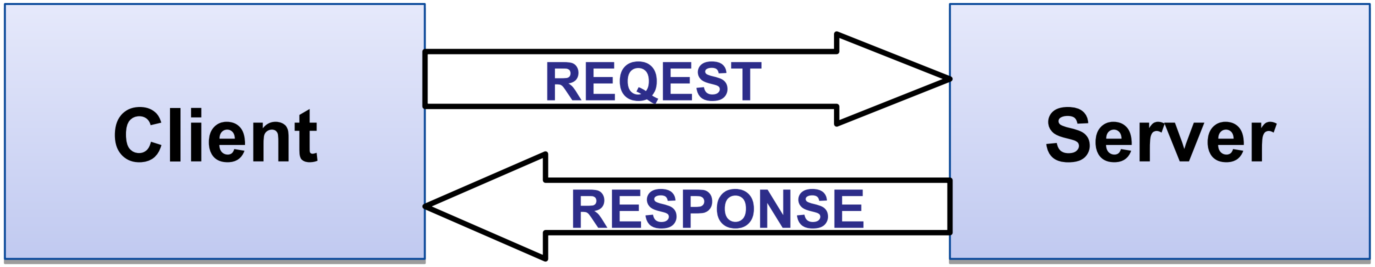
```
try {
  File file = new File("file.xml");
  JAXBContext jaxbContext =
JAXBContext.newInstance(Setting.class);
  Unmarshaller jaxbUnmarshaller =
jaxbContext.createUnmarshaller();
  Setting setting2 = (Setting)
jaxbUnmarshaller.unmarshal(file);
  System.out.println(setting2);
} catch (JAXBException e) {
  e.printStackTrace();
}
```

JAXB

```
@XmlRootElement
public class Setting {…

@XmlElement(name="file")
@XmlElementWrapper(name="files")
public void setLastUsedFiles(List<String>
lastUsedFiles) {…

@XmlAttribute
public void setPort(int port) {…
```

Hypertext Transfer Protokol (HTTP)

- Text protocol for transfer data between web server and client (often web browser). Protocol HTTP use port 80 in most cases.
- Client – Server: client sent request to the server and server sent requested data as a response.



- HTTP is a protocol from Application Layer (ISO-OSI model) and work over TCP protocol implicitly on port 80.
- Methods: **GET, PUT, POST**, HEAD, DELETE, OPTIONS, TRACE, CONNECT
- Actual version 1.1, can use persistent connection (HTTP keep-alive) – one TCP connection is used for sending more then one requests and responses.

HTTP - Request

```
method URL_document version_HTTP
head
empty_line
request_body
```

- **Example:**

```
GET /papers/content.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE
   5.0; Windows NT)
Host: www.server.cz


```

Method

Head

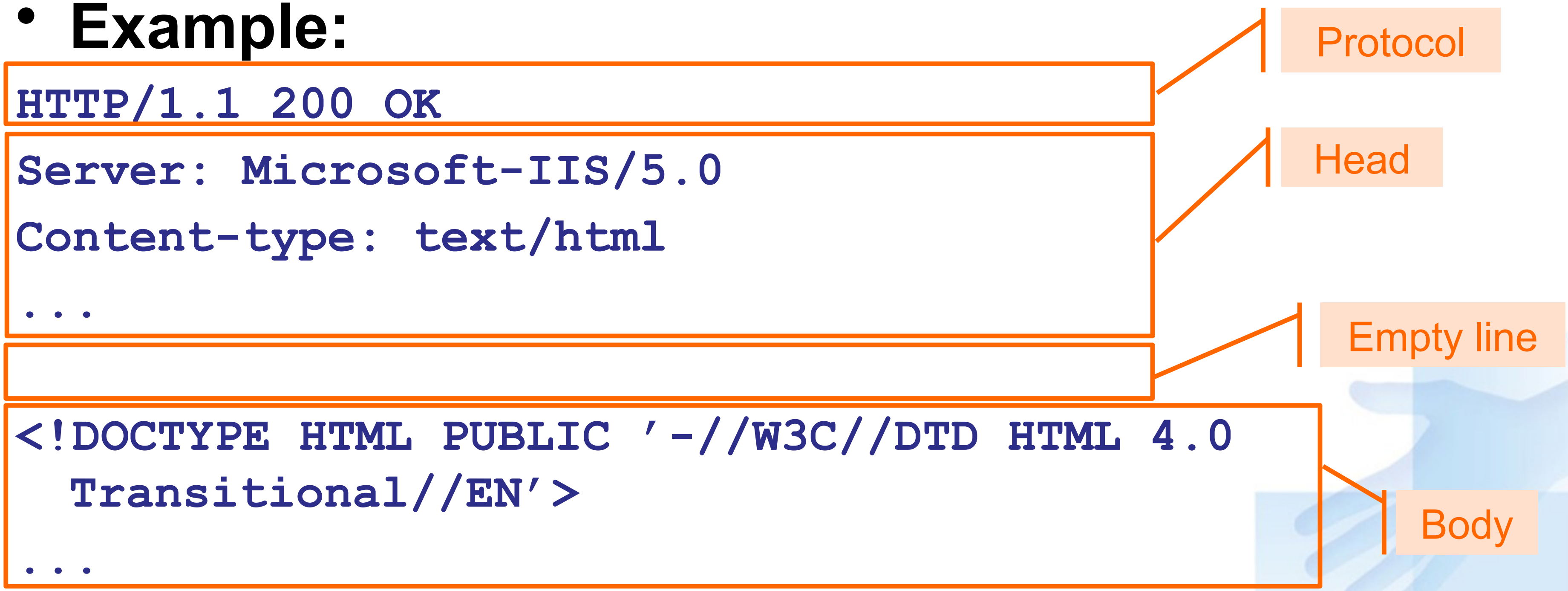
Empty line

No body

HTTP Response

```
Protocol status_code status_message
Head
Empty_line
Response_body
```

- **Example:**



```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-type: text/html
...

<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0
  Transitional//EN'>
...
```

HTTP and security

- Protocol contains only plain text data ⇒
  - Readability, simplicity
  - It is dangerous send data (password) through method GET:

```
GET /do.login?login=jez04&password=myPasswd ...
```

  - With method POST are data send in request body so they cannot be seen directly in browser but it is no problem capture data on way to server.
- In case of sending sensitive information application should use encrypted transfer like HTTPS.

Restriction of HTTP – stateless 1/2

- Protocol is stateless: server don’t have permanent connection to client so they cannot be uniquely identified – complication for web application.
- How to identify client in secure way, that already pass through authentication?
- **Bad Solutions:**
  - Transferring identification data in URL and in hidden fields of HTTP forms.

    ⇒ Transferring all identification data in all request is dangerous.
  - Cookies – Mechanism for storing data sent by server in browser. That data are automatically send to server in each request.

    ⇒ Storing and transferring all identification data in all request is dangerous even with cookies.

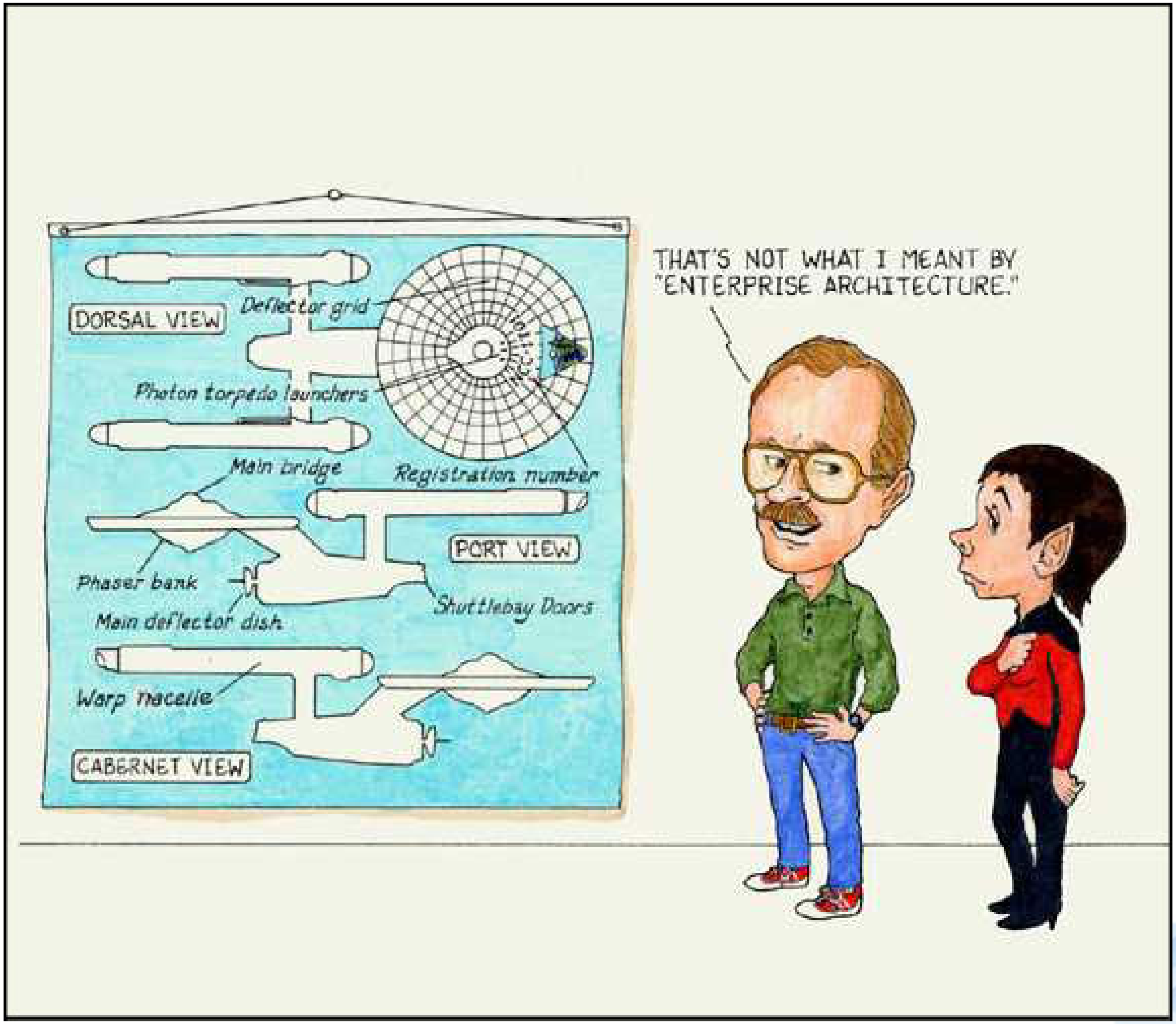
Restriction of HTTP – stateless 2/2

- Described disadvantages led to the introduction of sessions:
  - An identifier (called session id) is assigned to each new client and on server is stored pair of information the session id and client identification.
  - Session id is transferred to server with each request using cookies, parameter in URL or hidden form filed.
  - Advantage: Only this session id is transferred, complete identification is stored on server.
  - The support of session is important for development of web applications.

HTTP – Reference

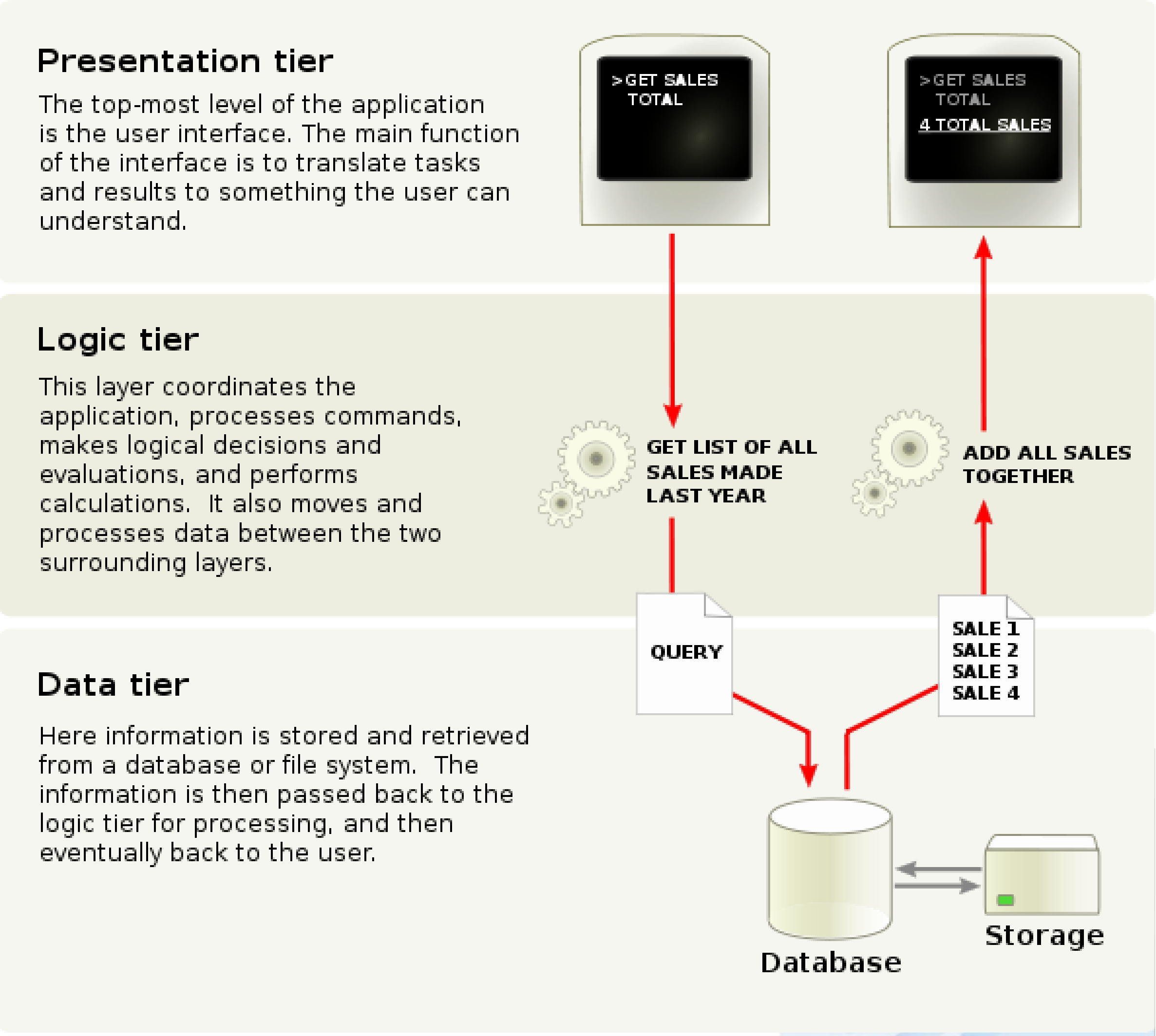
- World Wide Web Consortium: http://www.w3.org/, http://www.w3.org/Protocols/

Example of „Good“ Architecture



IS Architecture

- Multi-tier architecture
  - Subsystems are logically structured to tiers.
    - Typical tiers: presentation, application logic, data persistency.
  - Each tier represents abstraction with own responsibility. Advantages:
    - Tier can be easily understood and used.
    - Development inside tier is easier..
  - Tiers are isolated form changes in implementation of other tiers.



Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

Application Servers

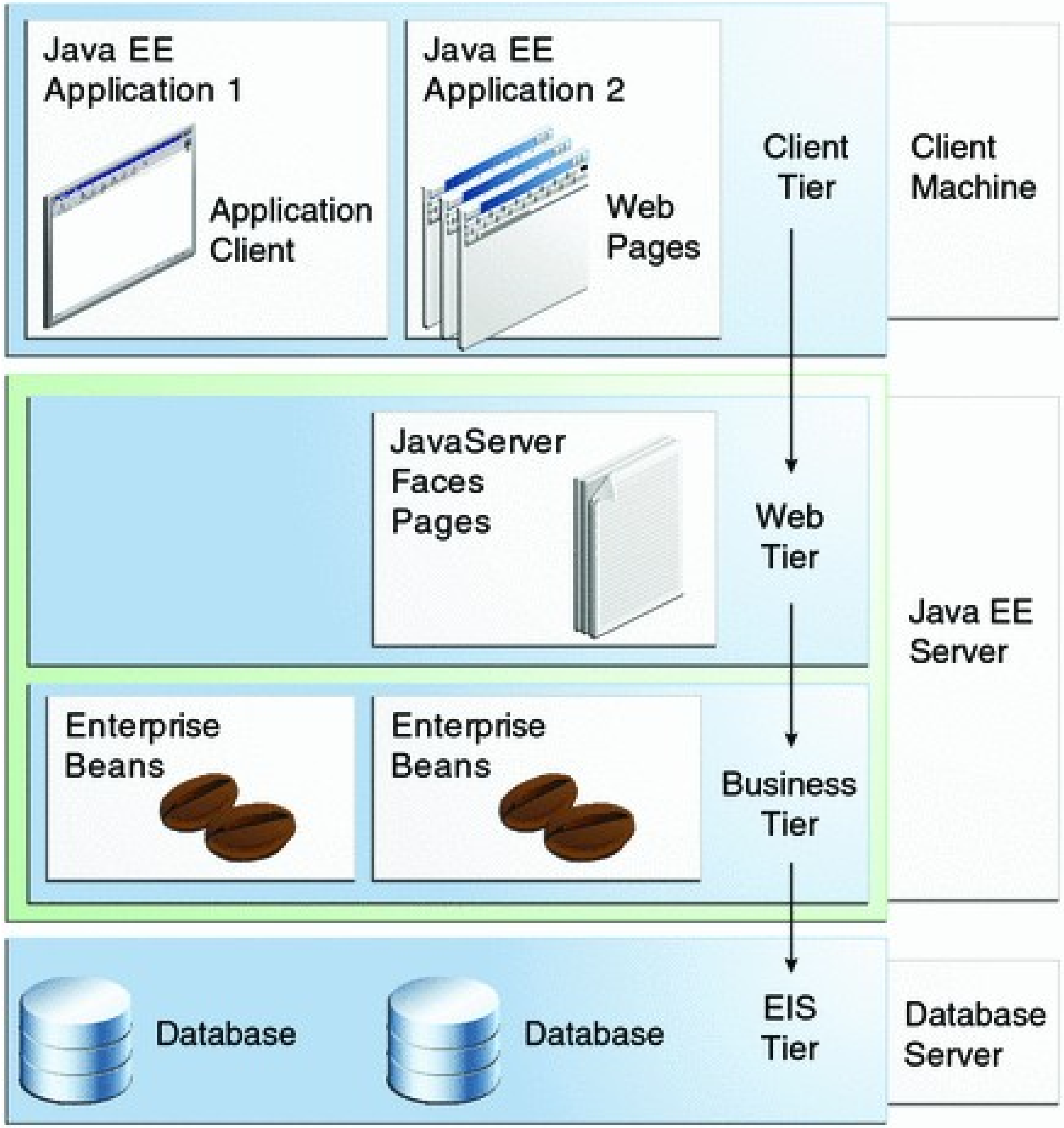
- Application server is software framework that provides an environment in which application can run.
- Application servers support:
  - Clustering
  - Fail-over – Automatic switching to a backup server, if primary server collapses.
  - Load balancing
- Application server for multi-tier architecture provide API to expose:
  - Presentation tier – Web tier used by web browsers.
  - Application logic tier – Business logic tier used by client applications.
- Application server is often integrated with web server.

Java EE Application Servers

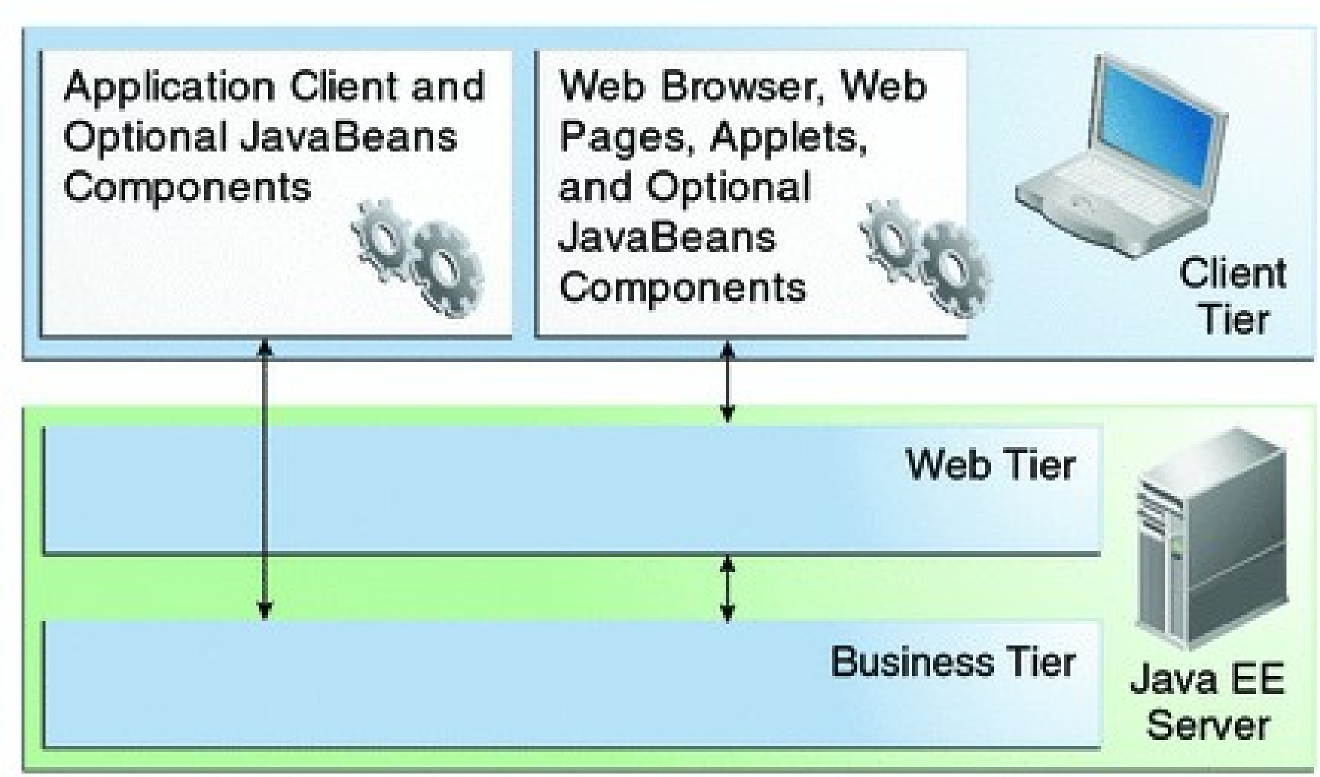
- Java EE Application server implements application container specified by Java EE platform. That container provides interface between Java EE applications and low-level platform-specific services.

- Application server GlassFish
  - Application server implementation from company Sun (now days Oracle). GalshFish is reference implementation of Java EE specifications and is free for download (Open Source).
- Apache Tomcat
  - Apache Tomcat application server do not implements fully Java EE specification. This server implements only Java Servlet and Java Server Pages (JSP) technologies, but is often included like core module in application servers that fully implements Java EE specification.
- JBoss AS
  - JBoss Application Server fully implements Java EE specification and use Tomcat as core module. This server is one of most used open source Java EE application server.

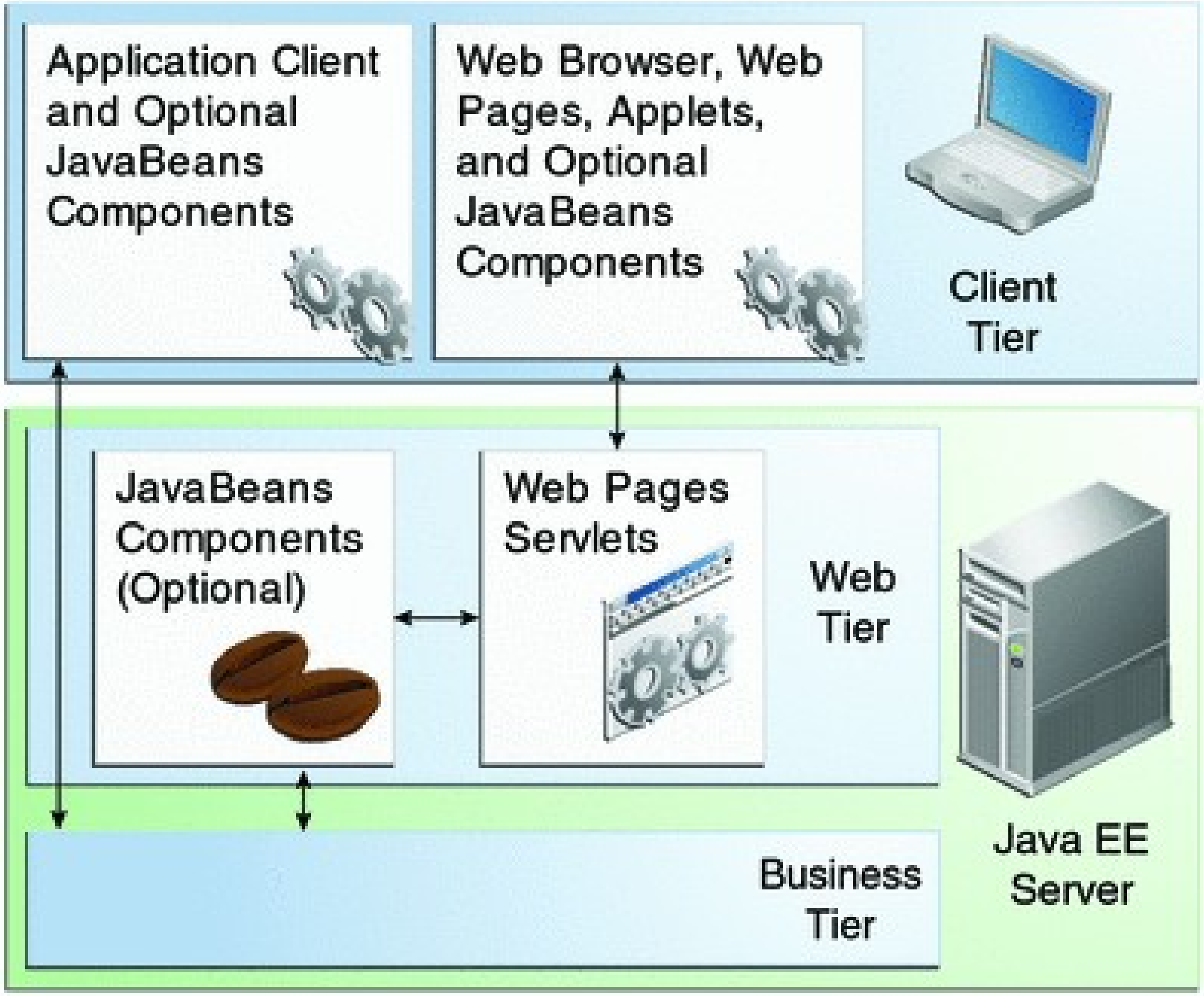
Java EE – Application Model



Java EE – Communication with Server


Application Client and Optional JavaBeans Components
Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
Web Tier
Business Tier
Java EE Server

Java EE – Web Components



Application Client and Optional JavaBeans Components

Web Browser, Web Pages, Applets, and Optional JavaBeans Components

Client Tier

JavaBeans Components (Optional)

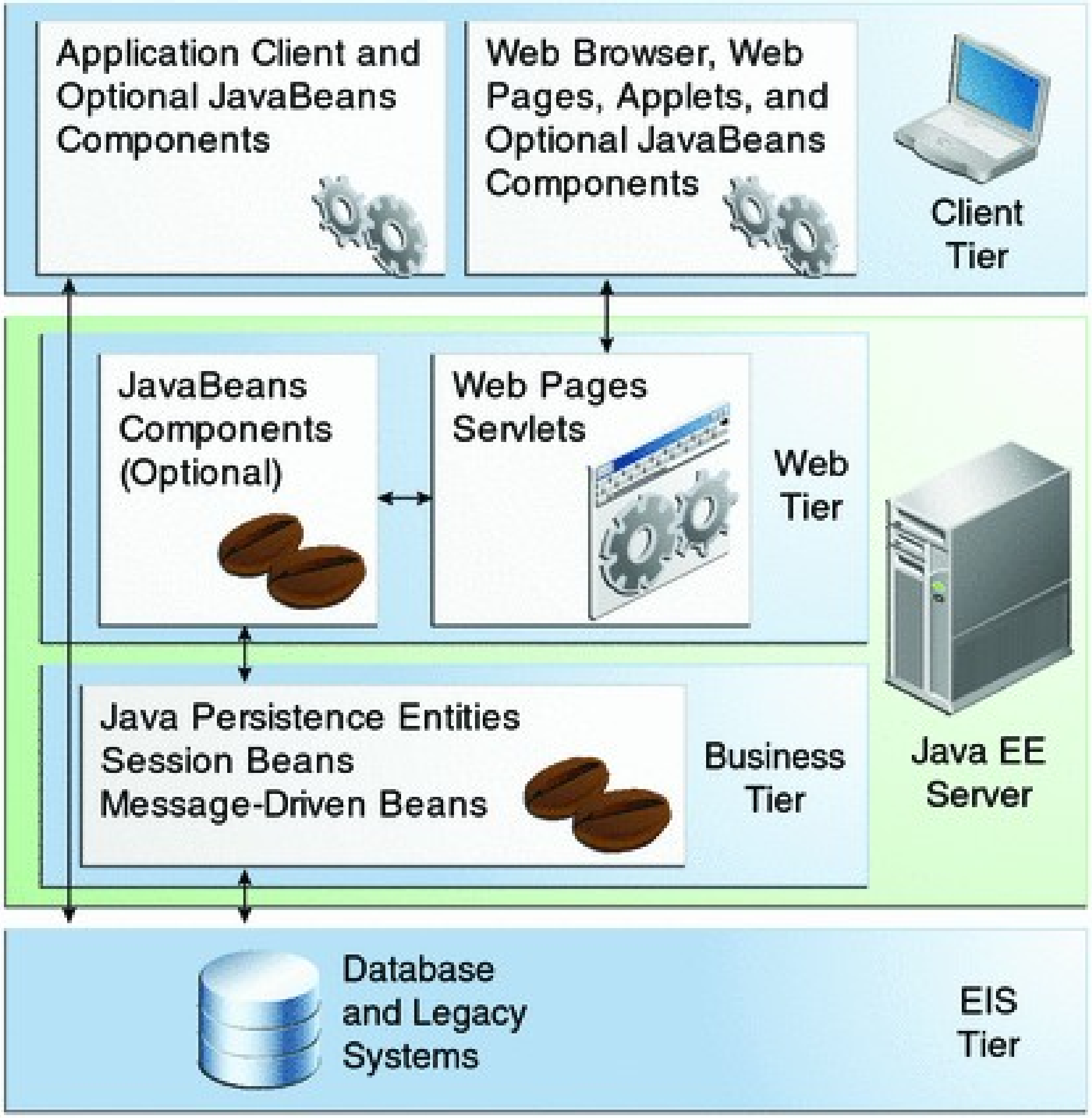
Web Pages Servlets

Web Tier

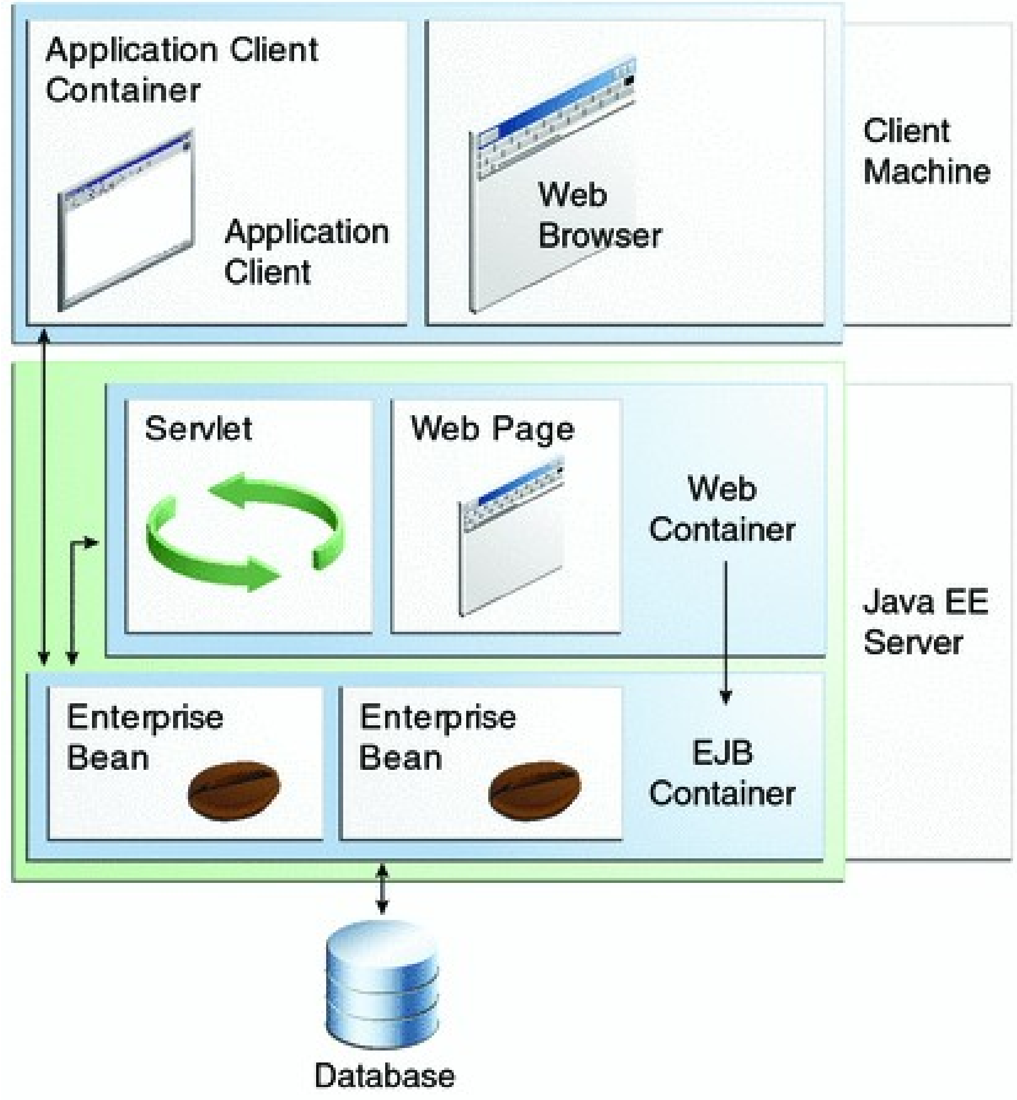
Business Tier

Java EE Server

Java EE – Business Components

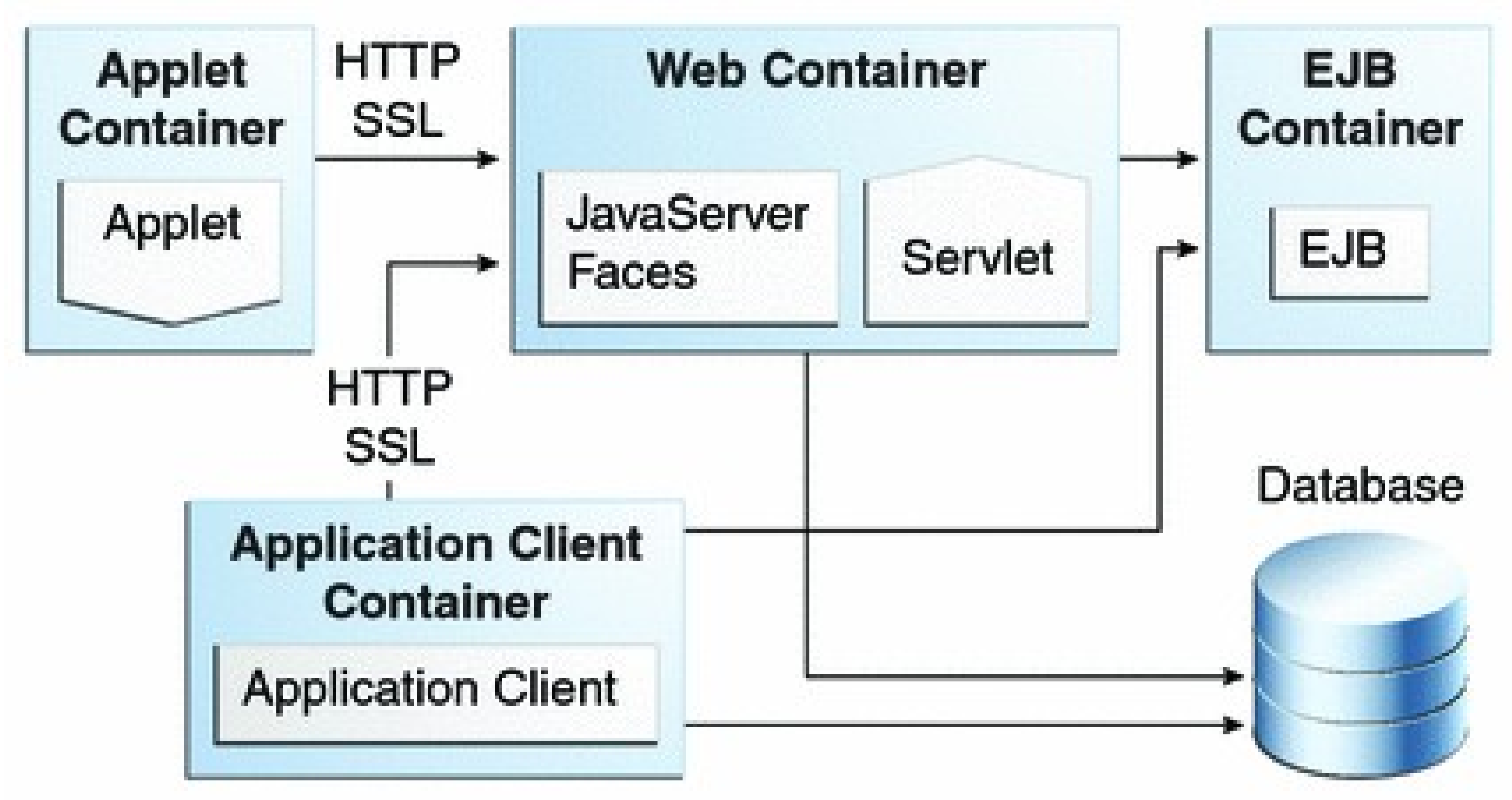

Application Client and Optional JavaBeans Components
Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
JavaBeans Components (Optional)
Web Pages Servlets
Web Tier
Java Persistence Entities Session Beans Message-Driven Beans
Business Tier
Java EE Server
Database and Legacy Systems
EIS Tier

Java EE – Containers



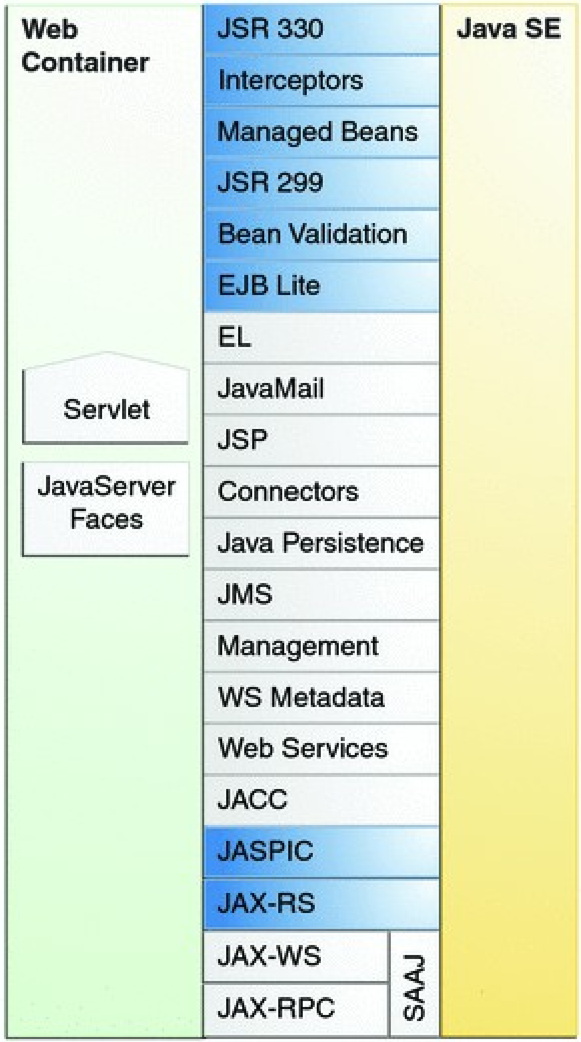
12/12/2020

Java EE – Containers



Java EE API in Containers

Web Container



Web Container

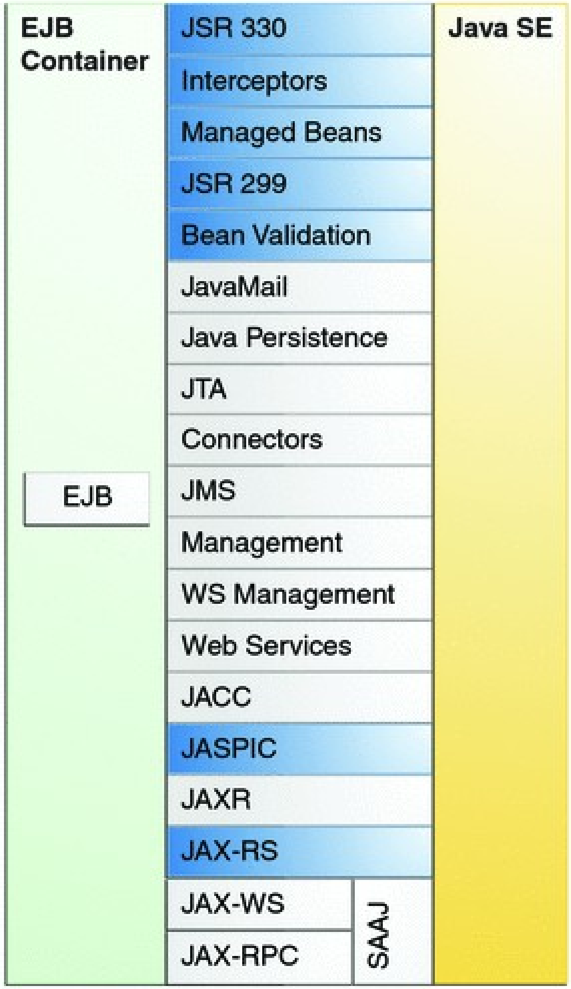
Servlet

JavaServer Faces

JSR 330
Interceptors
Managed Beans
JSR 299
Bean Validation
EJB Lite
EL
JavaMail
JSP
Connectors
Java Persistence
JMS
Management
WS Metadata
Web Services
JACC
JASPIC
JAX-RS
JAX-WS
JAX-RPC
SAAJ

Java SE

EJB Container



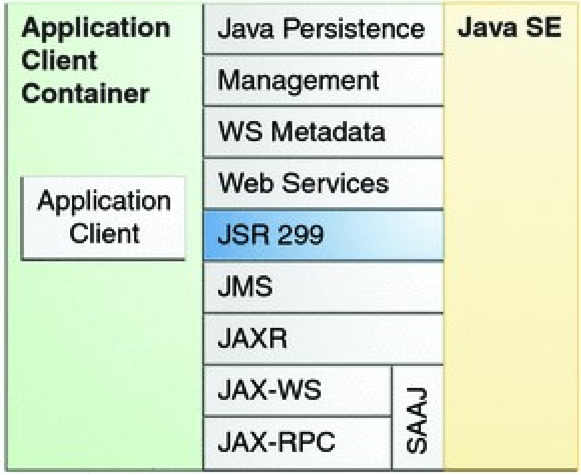
EJB Container

EJB

JSR 330
Interceptors
Managed Beans
JSR 299
Bean Validation
JavaMail
Java Persistence
JTA
Connectors
JMS
Management
WS Management
Web Services
JACC
JASPIC
JAXR
JAX-RS
JAX-WS
JAX-RPC
SAAJ

Java SE

Client Container



Application Client Container

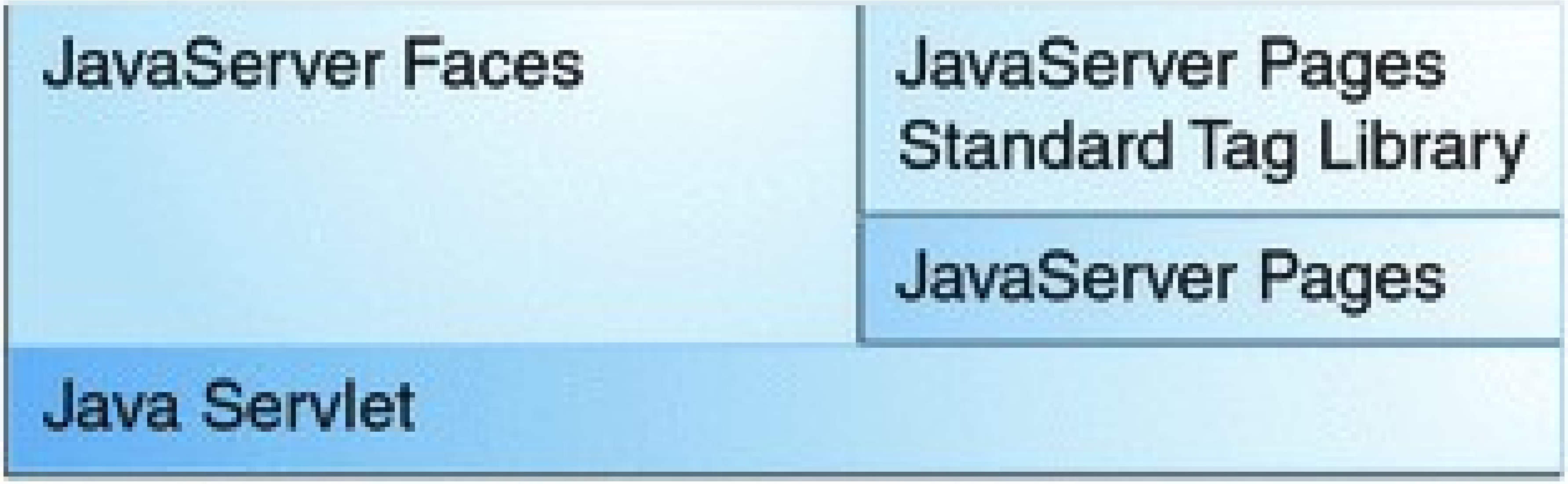
Application Client

Java Persistence
Management
WS Metadata
Web Services
JSR 299
JMS
JAXR
JAX-WS
JAX-RPC
SAAJ

Java SE

🟦 New in Java EE 6

Java EE – Web Application Technologies



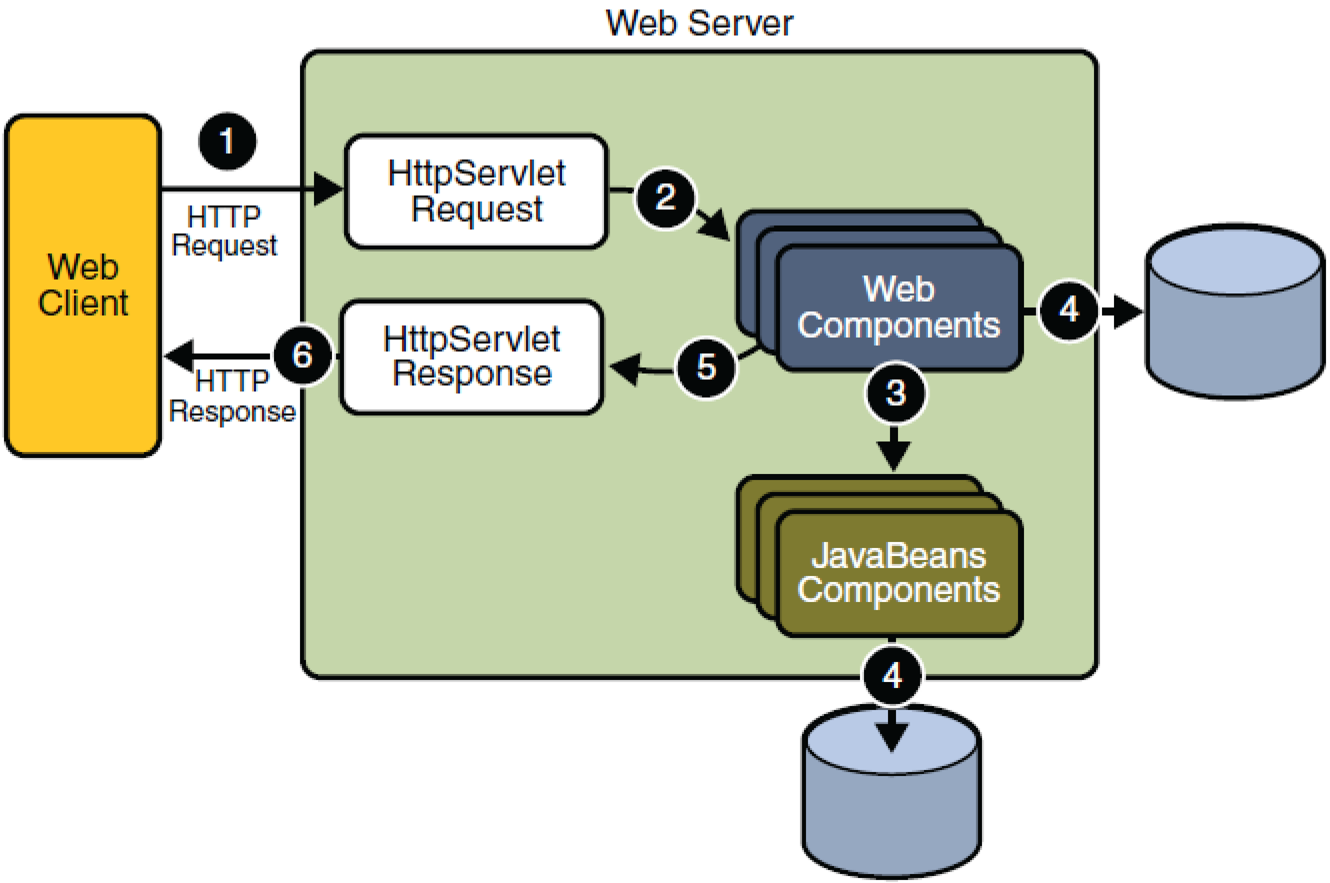
JavaServer Faces

JavaServer Pages Standard Tag Library

JavaServer Pages

Java Servlet

Java EE – Web Application



Java Servlet

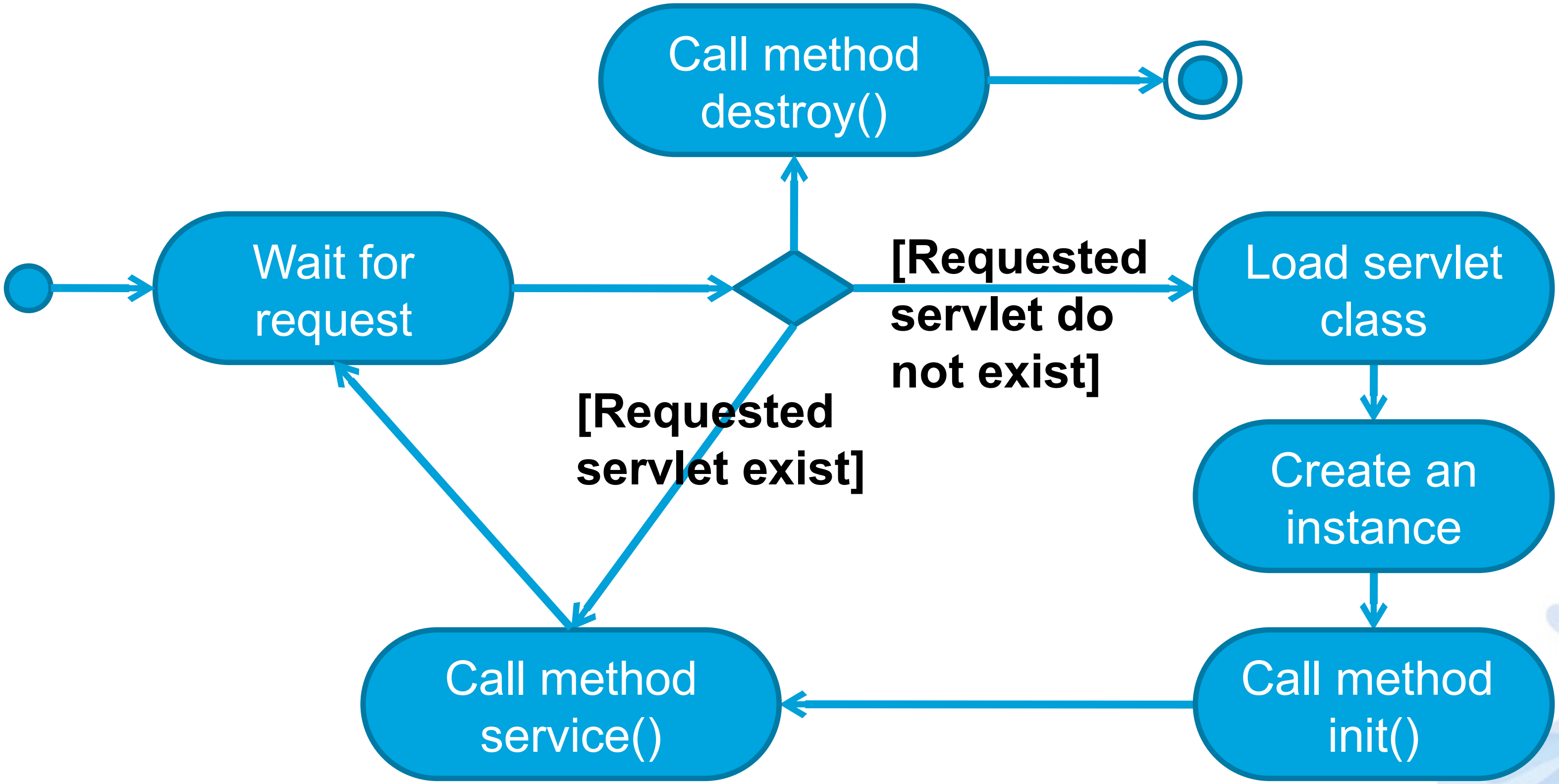
A Servlet

is a Java programming language class used to extend the capabilities of servers that can be accessed by a host application via a request-response programming model.

- Servlets can respond to any type of request, not only HTTP requests but HTTP requests are the most common requests.

Servlet – Lifecycle

- The Web container is responsible for managing the lifecycle of servlets and mapping a URL to a particular servlet.



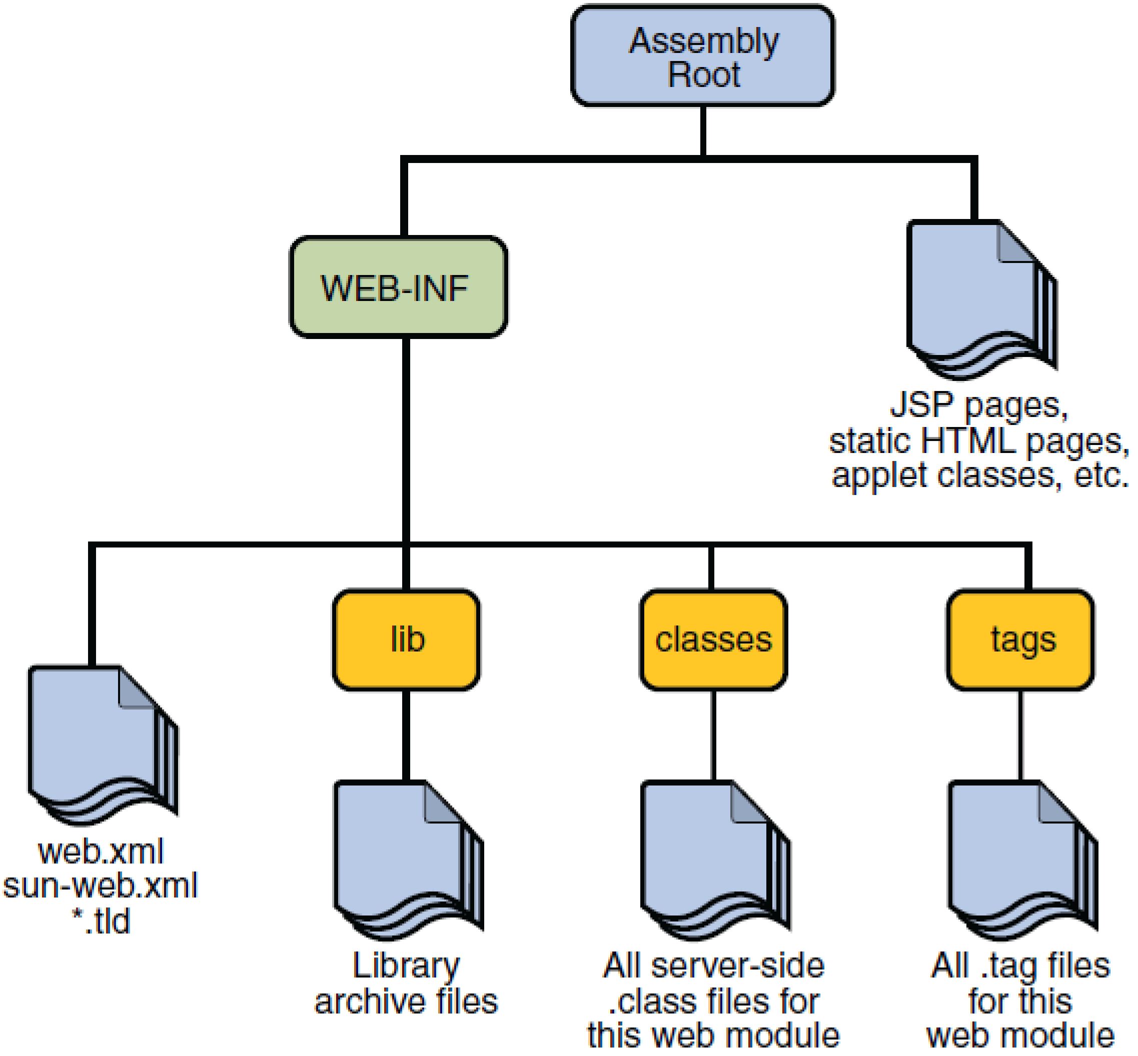
Servlet vs. HTTPServlet

- Interface **Servlet** is a general and is not bind with protocol HTTP. Code for handling requests should be contained in method **service()**.
- Class **HTTPServlet** implements interface **Servlet**. Method **service()** is already implemented and parse attributes from requests and call one of the method **doPost()**, **doGet()**, … according to http request method (GET, POST, …).

HTTPServlet - example

```
@WebServlet(description = "desc", urlPatterns = {"/MyServlet"})
public class MyFirstServlet extends HttpServlet {
    public MyFirstServlet() {
        super();
    }
    public String getServletInfo() {
   return “My first servlet";
   }
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
   doPost(request, response);
   }
   @Override
   protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
   PrintWriter pw = response.getWriter();
   pw.println("<html><body>Hello world!</body></html>");
   pw.close();
   }
}
```

Java EE – Web Module Structure


Assembly Root
WEB-INF
JSP pages, static HTML pages, applet classes, etc.
web.xml
sun-web.xml
*.tld
lib
classes
tags
Library archive files
All server-side .class files for this web module
All .tag files for this web module

Servlet Deployment

- WEB-INF\web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app …"– namespace specification --">
  <display-name>JSPExample</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
 <servlet>
    <display-name>MyFirstServlet</display-name>
    <description>pokus pokus</description>
    <servlet-name>MyFirstServlet</servlet-name>
    <servlet-class>MyFirstServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyFirstServlet</servlet-name>
    <url-pattern>/MyFirstServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Servlet – Request Information

- **String request.getParameter(“parameter name”)**
- HTTP request
  - http://[*host*]*:*[*port*][*request-path*]*?*[*query-string*]
  - Request-path: /MyServletApp/MyFristServlet/something
    - Context path: /MyServletApp
    - Servlet path: /MyFirstServlet
    - Path info: /something

Servlet – Scope Objects

- javax.servlet.ServletContext
  - getServletConfig().getServletContext(),
  - Contains: attributes, context path
- javax.servlet.http.HttpSession
  - HTTPServletRequest.getSession()
  - Contains: attributes
- javax.servlet.ServletRequest
  - Parameter of methods service() a doGet (), doPost (), …
  - Contains: attributes
- javax.servlet.jsp.PageContext

Servlet – Response

- response.setContentType("text/html");
- setCharacterEncoding()
- response.getWriter()
- response.getOutputStream()
- HTTP
  - addCookie()

Servlet – Initialization and Destruction

- **Servlet.init()**
  - Web container create object of servlet class (constructor is performed automatically), inject required resources to specified properties and call method **init()**. Method **init()** can initialize other resource like images, database connections and others.
- **Servlet.destroy()**
  - If web container decide based on internal mechanisms that servlet will be destroyed and removed from memory, then call method **destroy()** before remove servlet from memory.

Servlet – Lifecycle Monitoring

- Web container generates events if initialize or destroy servlet context, session and request. Listener of such events have to implements following interfaces:

javax.servlet.ServletContextListener

javax.servlet.ServletContextAttributeListener

javax.servlet.http.HttpSessionListener

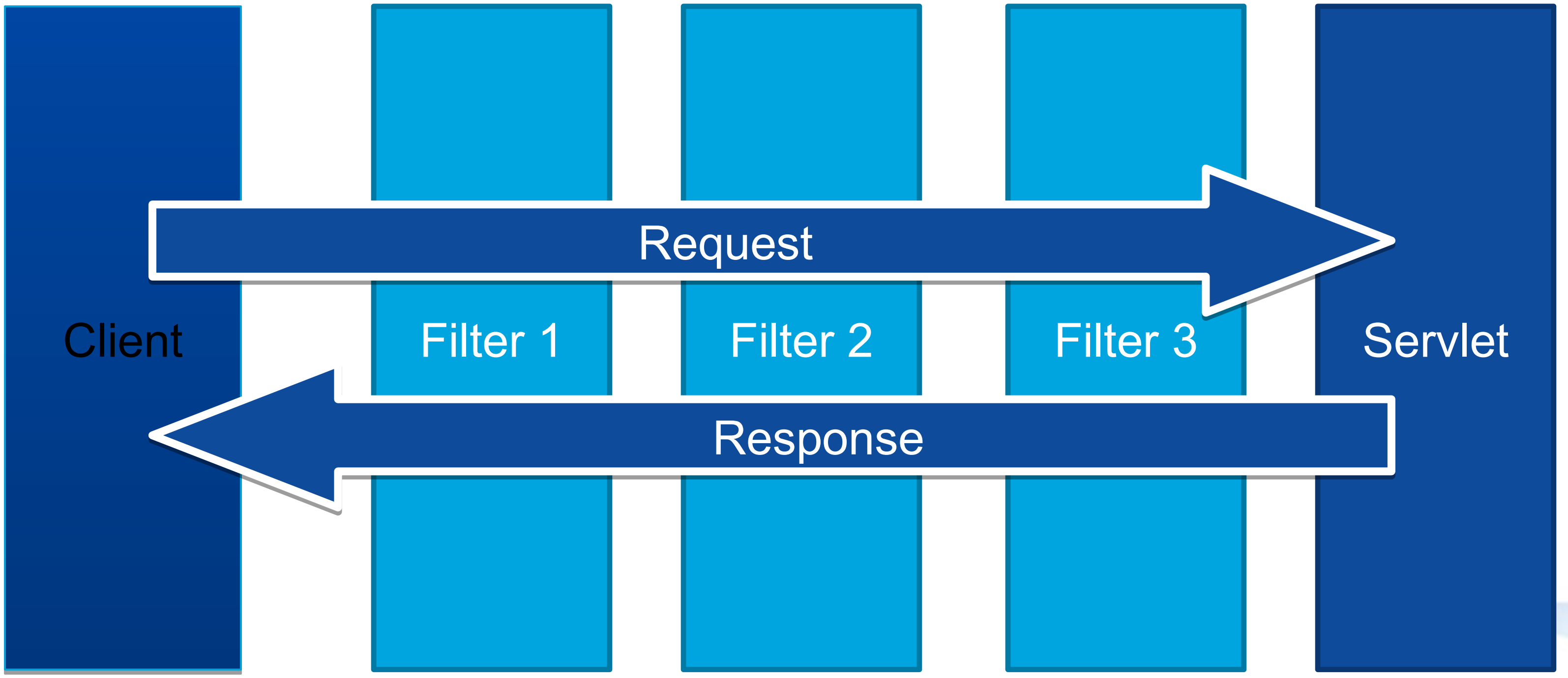
javax.servlet.http.HttpSessionActivationListener

javax.servlet.http.HttpSessionAttributeListener

javax.servlet.ServletRequestListener

javax.servlet.ServletRequestAttributeListener

Servlet - Filters



Request

Client

Filter 1

Filter 2

Filter 3

Servlet

Response

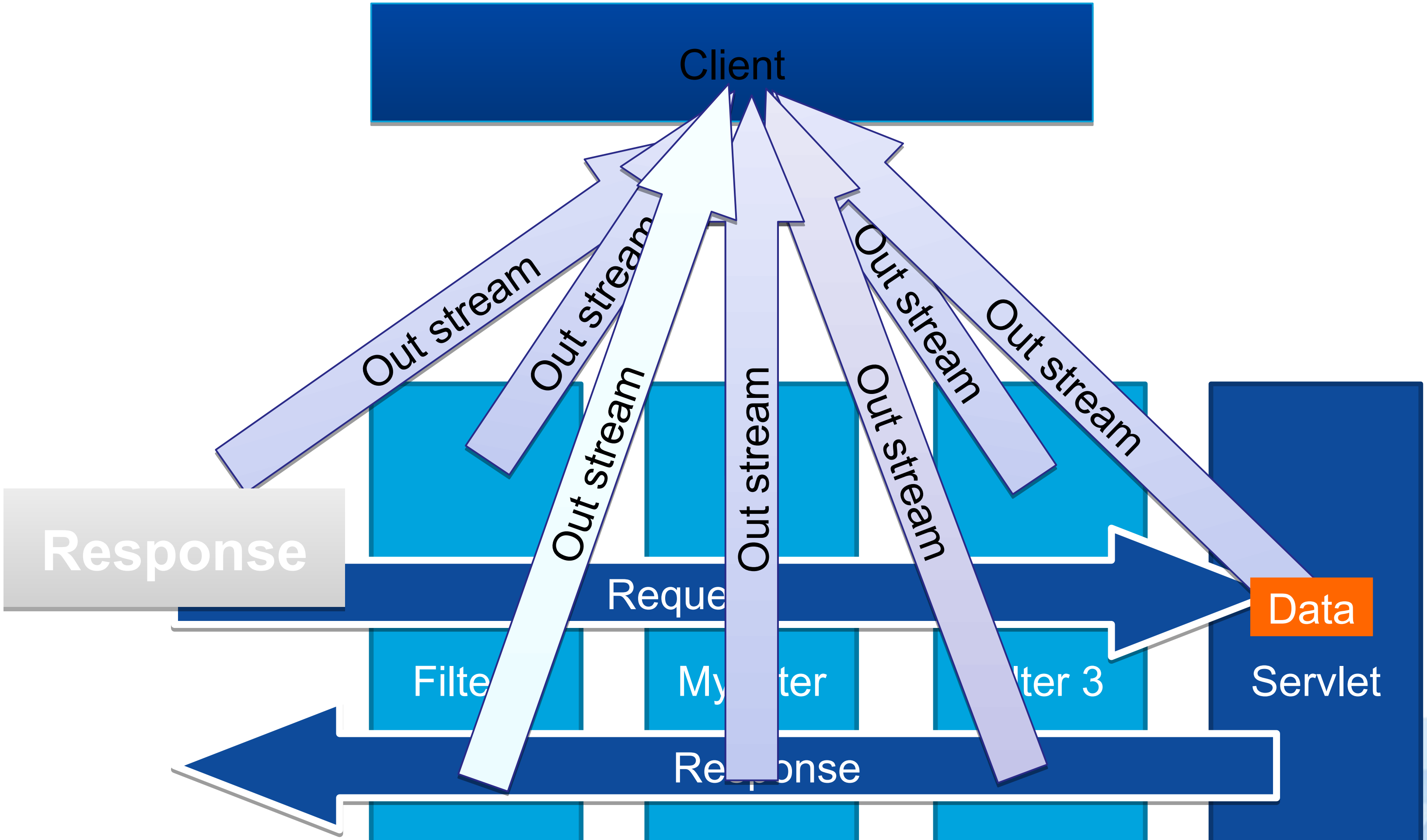
Servlet - Filters

- Filter can change request before and response after servlet processing.
- Each filter have to implements interface **javax.servlet.Filter**
  - Filter method **doFilter()** is most important, because this method performs filtering.
- Interface **javax.servlet.FilterChain** is a parameter of method **Filter.doFilter()** and each filter should call method **FilterChain.doFilter()** to pass control to next filter in chain.

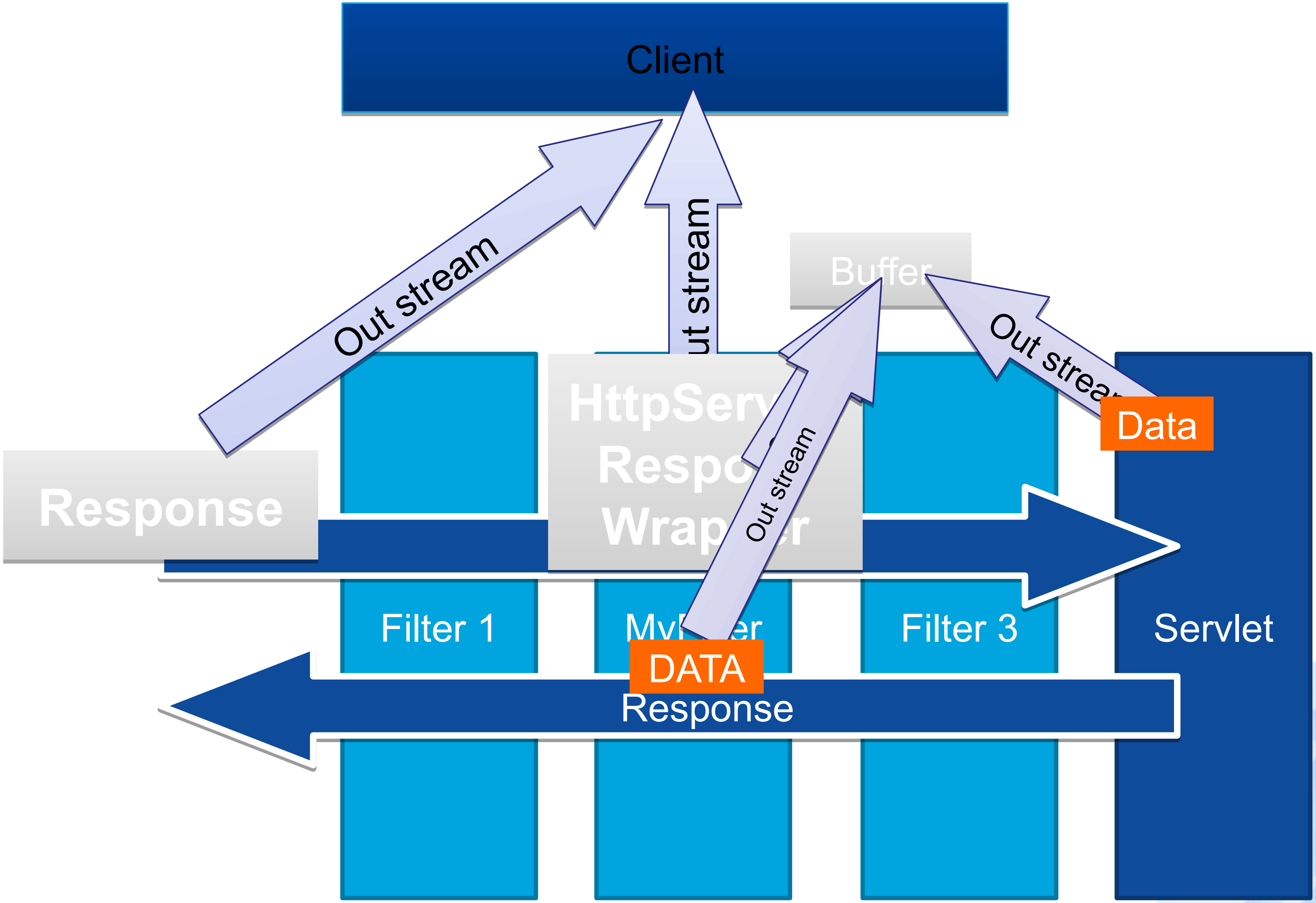
Servlet – Filter Example

```
@WebFilter(filterName="/MyFilter",
urlPatterns={"",""})
public class MyFilter implements Filter {
   public MyFilter() {
   }
   public void doFilter(ServletRequest request,
         ServletResponse response, FilterChain chain)
            throws IOException, ServletException {

      MyWrappedHttpResponse wrapper = new

MyWrappedHttpResponse( (HttpServletResponse)
            response);

      chain.doFilter(request, wrapper);

      response.getWriter().write(wrapper.toString()
         .toUpperCase());
   }
```

Servlet - Filters



Servlet - Filters

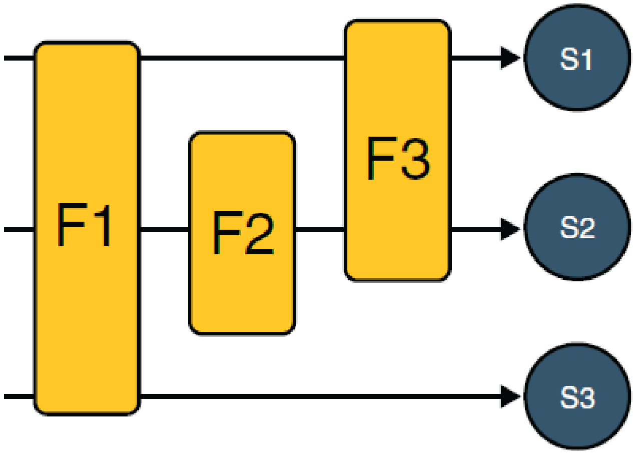


Servlet - Filter

```
public class MyWrappedHttpResponse extends
            HttpServletResponseWrapper {
    private CharArrayWriter buffer;
    public MyWrappedHttpResponse(
         HttpServletResponse response) {
        super(response);
        buffer = new CharArrayWriter();
    }
    public String toString() {
        return buffer.toString();
    }
    public PrintWriter getWriter() {
        return new PrintWriter(buffer);
    }
}
```

Servlet – Filter Mapping

```
<filter>
   <display-name>MyFilter</display-name>
   <filter-name>MyFilter</filter-name>
   <filter-class>MyFilter</filter-class>
 </filter>
 <filter-mapping>
   <filter-name>MyFilter</filter-name>
   <url-pattern>/*</url-pattern>
   <dispatcher>REQUEST</dispatcher>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>INCLUDE</dispatcher>
   <dispatcher>ERROR</dispatcher>
</filter-mapping>
```



Servlet - Include

- Implementation of include

```
RequestDispatcher dispatcher =
   getServletContext().getRequestDispatcher("/banner");
if (dispatcher != null){
   dispatcher.include(request, response);
}
```

- Included servlet can write data to response.
- Included servlet cannot change HTTP header setting (encoding, response type, etc.) and cannot create HTTP Cookies.

Servlet – Transfer Control

- Implementation of forward

```
RequestDispatcher dispatcher =
   request.getRequestDispatcher("/another_ser");
if (dispatcher != null)
   dispatcher.forward(request, response);
```

- The request URL is changed to new one.
- Origin request URL is stored to the request attributes with names:
  - javax.servlet.forward.*request-uri*
  - javax.servlet.forward.*context-path*
  - javax.servlet.forward.*servlet-path*
  - javax.servlet.forward.*path-info*
  - javax.servlet.forward.*query-string*
- Non one can write data to response before forward or exception arise during forwarding.

Servlet - Session

- Session is created automatically when method **getSession()** is called.
- Session can store object between client requests. It is realized by methods **getAttribute()** and **setAttribute()**.

```
HttpSession s = request.getSession();
Object o = s.getAttribute("counter");
if(o == null){
   o = new Counter();
   s.setAttribute("counter", o);
}
Counter c = (Counter)o;
```

- Session is terminated when method **Session.invalidate()** is called or if it is not used during timeout period.

Servlet – Session

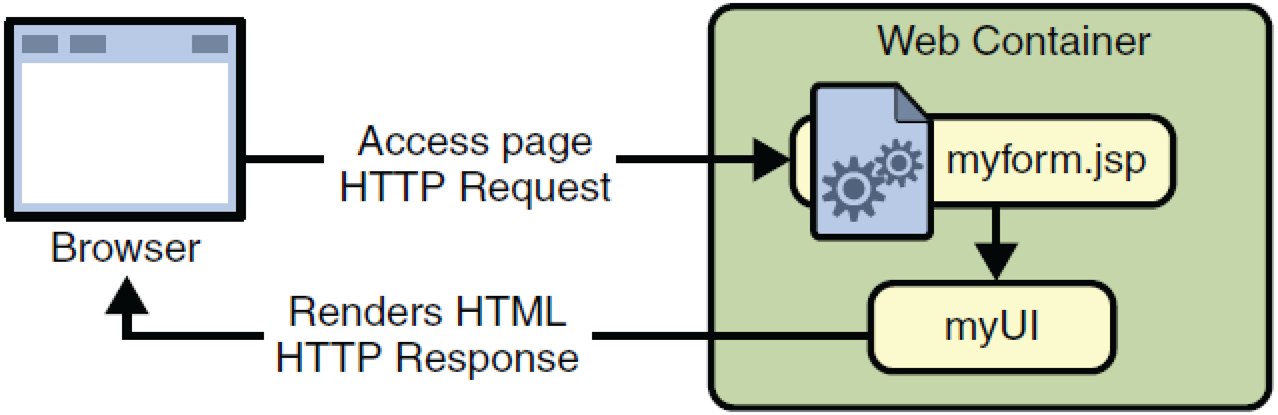
- Session ID is stored in Cookies.
- If cookies are switch off, session ID have to be stored as request parameter.

```
out.println("<p> &nbsp; <p><strong><a href=\"" +
   response.encodeURL(request.getContextPath() +
   "/bookcatalog") +
   "\">ContinueShopping </a>");
```

- Special method HttpServletResponse.encodeURL() exist to add session ID as parameter to encoded URL. The method determine if cookies are switch off and if yes add parameter with session ID to encoded URL

JSF – Java Server Faces

- JSF – component technology for creating server-side user interface of web applications.
- Main configuration file: faces-config.xml

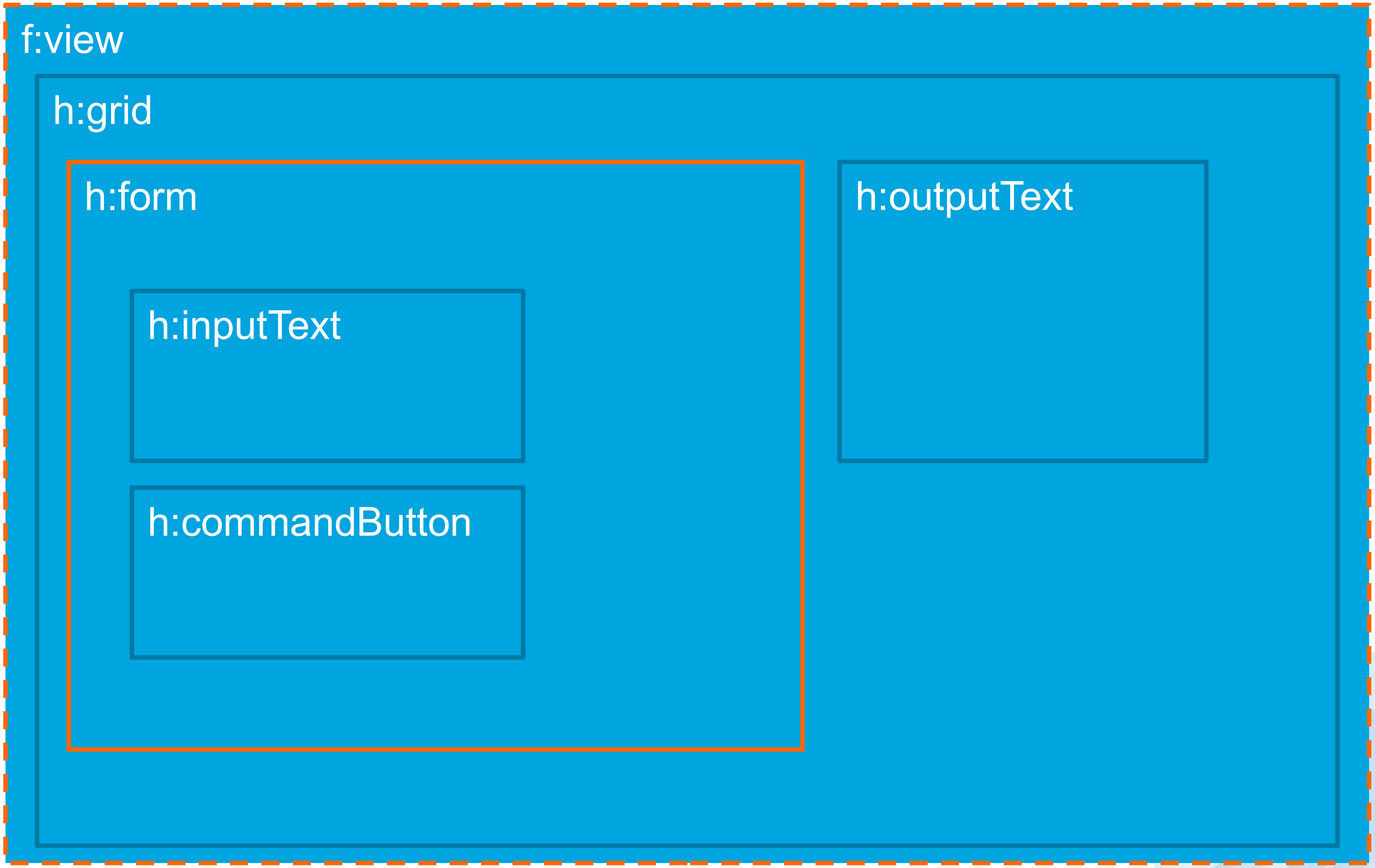


JSF – FacesServlet mapping

Programmer has to add following lines to web.xml to use JSF

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/guess/*</url-pattern>
  </servlet-mapping>
```

JSF – page structure



f:view

h:grid

h:form

h:inputText

h:commandButton

h:outputText

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:f="http://java.sun.com/jsf/core">
        <h:head>
            <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
            <title>Facelet Title</title>
        </h:head>
        <h:body>
            <h:panelGrid border="1" columns="2">
                <h:form>
                    <h:inputText
value="#{myBean.name}"></h:inputText>
                    <h:commandButton type="submit" value="Send"
                            action="ok"></h:commandButton>
                </h:form>
                <h:outputText value="#{myBean.age}"></h:outputText>
            </h:panelGrid>
        </h:body>
</html>
```

JSF - Namespaces

```
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:f="http://java.sun.com/jsf/core
   xmlns:h="http://java.sun.com/jsf/html"

xmlns:ui="http://java.sun.com/jsf/facelets
"

   >
```

JSF – GUI - View and Form

- JSF components are organized in components tree. The structure of component tree corresponds with structure of tags.
- All JSF components have to be inside component view.
  - From version JavaEE 6 component view is not required. Whole page automatically represent one view component.
- All JSF components that handle inputs form user have to by inside component form

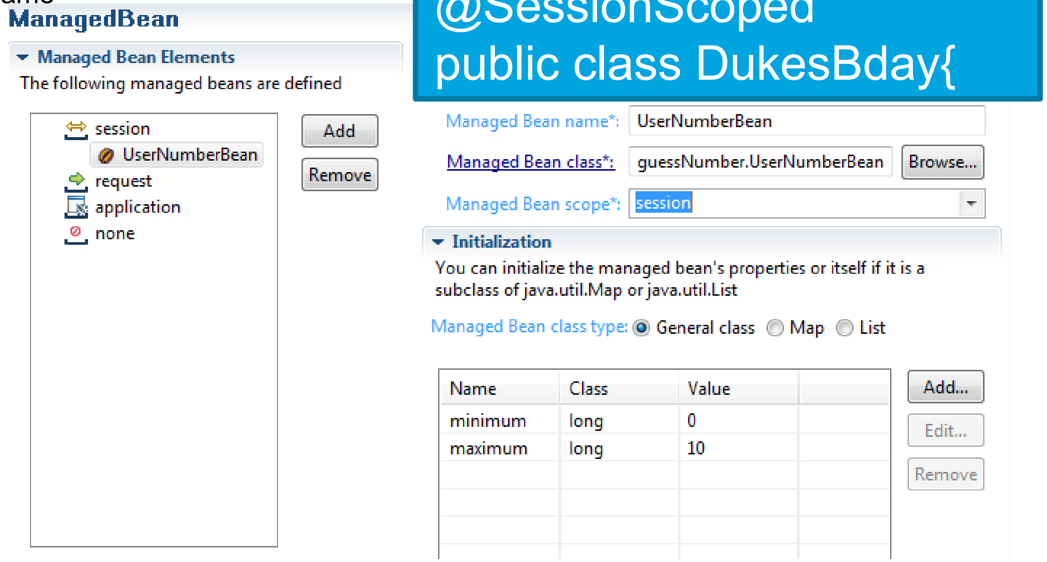
```
<f:view>
   <h:form id="helloForm1">
   </h:form>
</f:view>
```

JSF – Managed Bean

- JavaBeans which creation and state are managed by JavaEE container. Can be defined in configuration file (faces-config.xml) or by annotations.

```
<managed-bean>
<managed-bean-name>UserNumberBean</managed-bean-name>
<managed-bean-class>
guessNumber.UserNumberBean
</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
<managed-property>
<property-name>minimum</property-name>
<property-class>long</property-class>
<value>0</value>
</managed-property>
<managed-property>
<property-name>maximum</property-name>
<property-class>long</property-class>
<value>10</value>
</managed-property>
</managed-bean>
```

```
@ManagedBean
@SessionScoped
public class DukesBday{
```



ManagedBean

Managed Bean Elements

The following managed beans are defined

- session
  - UserNumberBean
- request
- application
- none

Managed Bean name*: UserNumberBean

Managed Bean class*: guessNumber.UserNumberBean

Managed Bean scope*: session

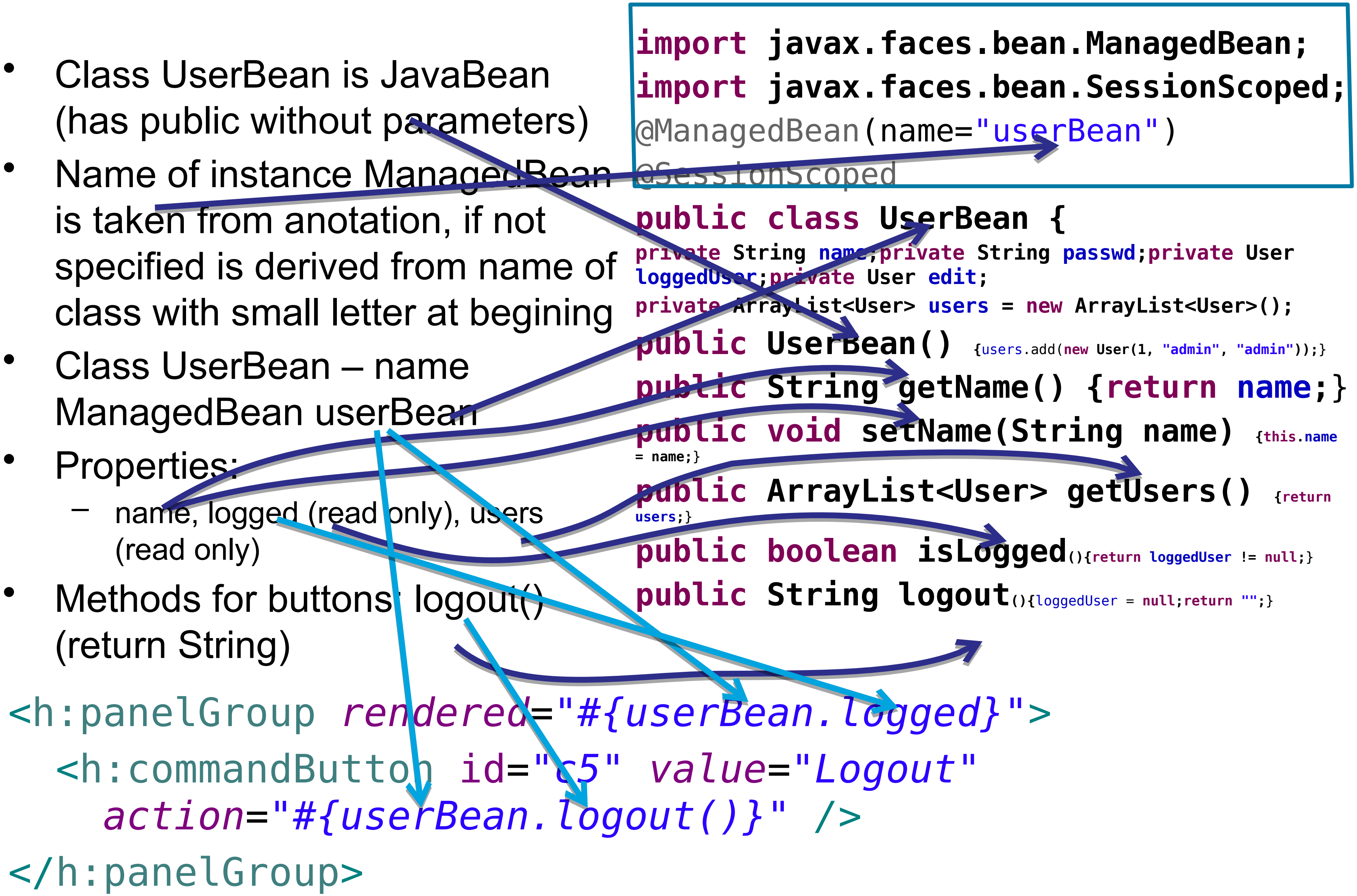
Initialization

You can initialize the managed bean's properties or itself if it is a subclass of java.util.Map or java.util.List

Managed Bean class type: ● General class ○ Map ○ List

Name	Class	Value
minimum	long	0
maximum	long	10

JSF – Managed Bean



- Class UserBean is JavaBean (has public without parameters)
- Name of instance ManagedBean is taken from anotation, if not specified is derived from name of class with small letter at begining
- Class UserBean – name ManagedBean userBean
- Properties:
  - name, logged (read only), users (read only)
- Methods for buttons: logout() (return String)

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
@ManagedBean(name="userBean")
@SessionScoped
```

```
public class UserBean {
private String name;private String passwd;private User
loggedUser;private User edit;
private ArrayList<User> users = new ArrayList<User>();
public UserBean() {users.add(new User(1, "admin", "admin"));}
public String getName() {return name;}
public void setName(String name) {this.name
= name;}
public ArrayList<User> getUsers() {return
users;}
public boolean isLogged(){return loggedUser != null;}
public String logout(){loggedUser = null;return "";}
```

```
<h:panelGroup rendered="#{userBean.logged}">
  <h:commandButton id="c5" value="Logout"
    action="#{userBean.logout()}" />
</h:panelGroup>
```

JSF – Managed Bean (CDI)

- Class UserBean is JavaBean (has public without parameters)
- Name of instance ManagedBean is taken from anotation, if not specified is derived from name of class with small letter at begining
- Class UserBean – name ManagedBean userBean
- Properties:
- name, logged (read only), users (read only)
- Methods for buttons: logout() (return String)

```
import
javax.enterprise.context.SessionScoped;
import javax.inject.Named;
@Named("userBean")
@SessionScoped
public class UserBean {
private String name;private String passwd;private User
loggedUser;private User edit;
private ArrayList<User> users = new ArrayList<User>();
public UserBean() {users.add(new User(1, "admin", "admin"));}
public String getName() {return
name;}
public void setName(String name) {this.name
= name;}
public ArrayList<User> getUsers() {return
users;}
public boolean isLogged(){return loggedUser != null;}
public String logout(){loggedUser = null;return "";}
```

```
<h:panelGroup rendered="#{userBean.logged}">
  <h:commandButton id="c5" value="Logout"
    action="#{userBean.logout()}" />
</h:panelGroup>
```

JSF – Managed Bean

- Application (@ApplicationScoped)
- Session (@SessionScoped)
- **View (@ViewScoped)**
- Request (@RequestScoped)
- None (@NoneScoped)
- Custom (@CustomScoped)

EL – Unified Expression Language

- Immediate expressions
  - `${expression}`
  - Example: `${sessionScope.cart.total}`
- Deferred expressions
  - `#{expression}`
  - Example:

```
<h:inputText id="name"
value="#{customer.name}"/>
```

- Expressions can be used to generate dynamic content in text document (often HTML – JSP technology)
- Can be used as parameters of tags (JSF technology)

EL – Value Expression

- Expressions can access to:
  - JavaBeans, collections, enum types, implicit objects
- Expression can access JavaBean properties in two ways
  - ${customer.name}
  - ${customer["name"]}
- Both access methods can be combined
  - ${customer.address["street"]}

EL – Value Expression

- Expression ${customer} process search of properties with name “customer” in context of page, request, session and application.

- If program define enum type Animals and property myAnimal of type Animals

```
public enum Animals{dog, cat, fish,
bird}
```

- Expression can use construction like this:

```
${ myAnimal == "dog"}
```

EL – Value Expression

- Expression can access to collections
  - Expression access to any element

```
${customer.orders[1]}
```

  - Expression access to first element

```
${customer.orders.orderNumber}
```

- Expressions can access to maps

```
${customer.favourite["computers"]}
```

- Expressions allow use constants

```
${"text"}
${customer.age + 20}
${true}
${57.5}
```

EL – Method Expression

- Expression can call methods
  - No paramaters

```
   <h:inputText id="name"
value="#{customer.name}"
validator="#{customer.validateName()}"/>
```

  - With parameters

```
<h:inputText
value="#{userNumberBean.userNumber('5')}">
```

EL – Operators

- **Arithmetic:** +, - (binary), *, / and div, % and mod, - (unary)
- **Logical:** and, &&, or, ||, not, !
- **Relational:** ==, eq, !
  =, ne, <, lt, >, gt, <=, ge, >=, le
- **Empty:** The `empty` operator is a prefix operation that can be used to determine whether a value is null or empty.
- **Conditional:** A ? B : C

JSF – GUI Components

- `<h:outputText lang="en_US" value="#{UserNumberBean.minimum}"/>`

- `<h:graphicImage id="waveImg" url="/wave.med.gif" />`

- `<h:inputText id="userNo" label="User Number" value="#{UserNumberBean.userNumber}"></h:inputText>`

JSF - GUI Components

```
<h:panelGroup style="border-bottom-
style: double; border-top-style:
double; border-left-style: double;
border-right-style: double">

<h:outputLink
value="somePage.xhtml">

</h:panelGroup>
```

JSF - GUI Components

```
<h:panelGrid border="1" columns="2">
  <h:outputText value="item1"></h:outputText>
  <h:outputText value="item2"></h:outputText>
  <h:outputText value="item3"></h:outputText>
  <h:outputText value="item4"></h:outputText>
</h:panelGrid>
```

JSF - GUI Components

```
<h:commandButton value="Send"
outcome="success"></h:commandButton>

<h:commandLink>
   <h:outputText value="CommandLink">
   </h:outputText>
</h:commandLink>
```

JSF – GUI - Table

```
<h:dataTable var="p"
value="#{personAgenda.allPositions}">
<h:column>
<h:outputText value="#{p.description}" />
</h:column>
<h:column>
<h:commandButton value="Edit"
action="#{personAgenda.editPosition(p)}"
/>
</h:column>
</h:dataTable>
```

JSF – GUI - Table

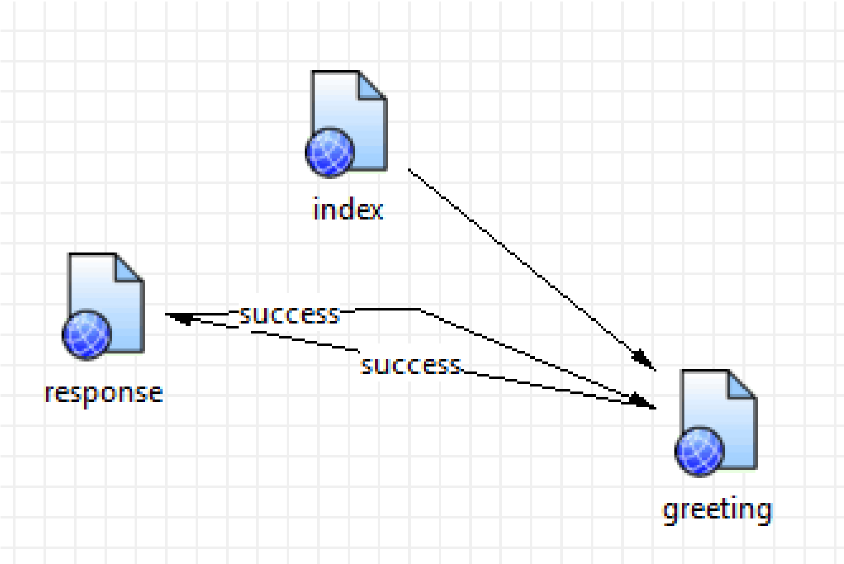
```
<h:dataTable var="p"
value="#{personAgenda.allPositions}"
binding="#{personAgenda.positionTable}">
<h:column>
  <f:facet name="header">
    <h:outputText value="Name"/>
  </f:facet>
  <h:outputText value="#{p.description}" />
</h:column>
<h:column>
  <h:commandButton value="Edit"
    action="#{personAgenda.editPosition}" />
</h:column>
</h:dataTable>
```

JSF - buttons

```
<h:button value="Send"
outcome="success"></h:commandButto
n>


<h:commandLink
action="#{myBean.doSomething()}">
  <h:outputText
value="CommandLink">
  </h:outputText>
</h:commandLink>
```

JSF – Navigation Model

- Navigation through rules and action

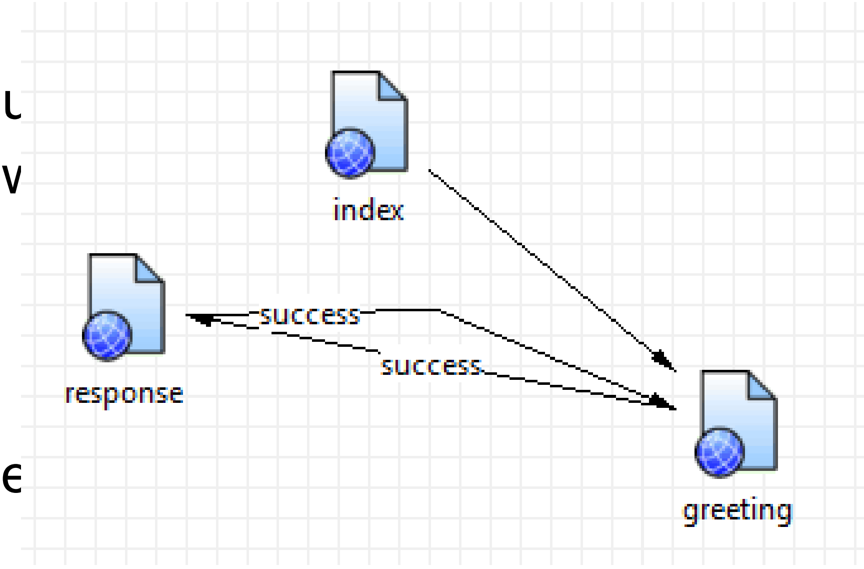


- Method of BackingBean
  - <h:commandButton id="submit" action="#{userNumberBean.deliveryOrder()}" value="Submit" />

Method must return String and usually has no parameters

JSF - navigation

```
<navigation-rule>
    <from-view-id>/greeting.jsp</from-view-id>
    <navigation-case>
        <from-outcome>success</from-ou
        <to-view-id>/response.jsp</to-view
    </navigation-case>
</navigation-rule>
<navigation-rule>
    <from-view-id>/response.jsp</from-vie
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/greeting.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```



JSF – navigation rules

```
<navigation-rule>
  <from-view-id>*</from-view-id>
  <navigation-case>
   <from-outcome>editPosition</from-outcome>
   <to-view-id>/editPosition.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
   <from-outcome>logout</from-outcome>
   <to-view-id>/Logout</to-view-id>
   <redirect/>
  </navigation-case>
<navigation-rule>
  <from-view-id>/login.xhtml</from-view-id>
  <navigation-case>
   <to-view-id>/index.xhtml</to-view-id>
   <redirect/>
  </navigation-case>
```

Code to get ManagedBean

- Using DI:

```
@ManagedProperty("#{personMB}")
private PersonMB personMB;
```

- Resolving EL expresion:

```
StarShipDB controller = (StarShipDB)
arg0.getApplication().getELResolver().
      getValue(arg0.getELContext(), null, "starShipDB");
```

- Using JNDI (Session Bean):

```
try {
   starShipSB = InitialContext.doLookup(
   "java:/global/jat2017cv11v1/StarShipSB");
   System.out.println("starshipSB in converter class: " +
starShipSB.getClass().getCanonicalName());
} catch (NamingException e) {e.printStackTrace();}
```

JSF – Facelets - Template

```
<h:body>
<div id="top" class="top">
  <ui:insert name="top">Top Section</ui:insert>
</div>
<div>
  <div id="left">
   <ui:insert name="left">Left Section</ui:insert>
  </div>
  <div id="content" class="left_content">
    <ui:insert name="content">Main
Content</ui:insert>
  </div>
</div>
</h:body>
```

JSF – Facelets – Usage of Template

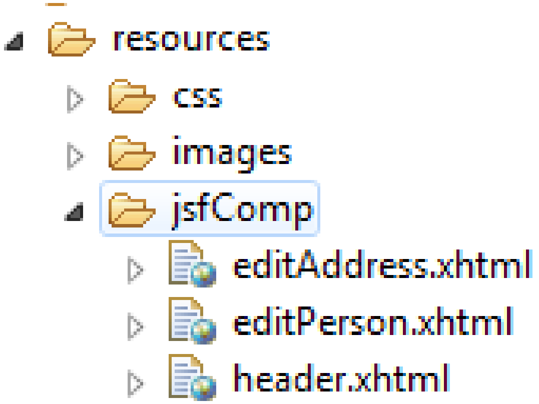
```
<html xmlns=…
<h:body>
<ui:composition template="./template.xhtml">
<ui:define name="top">
    Welcome to Template Client Page
</ui:define>
<ui:define name="left">
    <h:outputLabel value="You are in the Left Section"/>
</ui:define>
<ui:define name="content">
    <h:graphicImage value="#{resource['images:wave.med.gif']}"/>
    <h:outputText value="You are in the Main Content Section"/>
</ui:define>
</ui:composition>
</h:body>
</html>
```

JSF – composite components

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:cc="http://java.sun.com/jsf/composite"
      xmlns:h="http://java.sun.com/jsf/html">
  <cc:interface>
    <cc:attribute name="addressObject"
required="true"/>
  </cc:interface>
  <cc:implementation>
    <h:inputText
value="#{cc.attrs.addressObject.street}"/>
    <h:inputText
value="#{cc.attrs.addressObject.city}"/>
  </cc:implementation>
</html>
```

JSF – composite components

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE composition …>
<ui:composition
xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:my="http://java.sun.com/jsf/composite/jsfComp"
   template="/template_secured.xhtml">
 <ui:define name="content">
  <h:form>
<my:editPerson addressObject="#{personAgendaMB.
   editedCustomer.address} />
  </h:form>
 </ui:define>
</ui:composition>
```



JSF - konverze

```
<h:inputText id="valueEdit"
   value="#{counterHolder.counter.value}"
   label="Counter value">
       <f:converter converterId="myConverterID" />
</h:inputText>
```

```
@FacesConverter("myConverterID")
public class CounterHolder implements
   javax.faces.convert.Converter
```

JSF – Conversion Model

- Component connection to server-site JavaBeans needs data type conversion
  - Model view: data are represented in java data types (int, long, java.util.Date, …)
  - Presentation view: data represented in human readable form in HTML (text)
- Example:
  - Java.util.Date vs. „28.9.2009“
- Implementation of users converters – interface `javax.faces.convert.Converter`

JSF – Conversion Model

```
<h:inputText id="valueEdit"
   value="#{counterHolder.counter.value}"
   label="Counter value"
   converter="#{converterFactory.employeeConverter}">
      <f:converter binding="#{counterHolder}"/>
</h:inputText>
```

Expresion have to return object which implements interface **Converter**

```
<managed-bean>
<managed-bean-name>counterHolder</managed-bean-name>
<managed-bean-class>bean.CounterHolder</managed-bean-
class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

```
@ManagedBean
public class CounterHolder implements
   javax.faces.convert.Converter
```

JSF - Conversion Model

```
<h:inputText id="valueEdit"
   value="#{counterHolder.currentCompany}"
   label="Counter value">

</h:inputText>
```

```
@FacesConverter(forClass=Company.class)
public class CounterHolder implements
   javax.faces.convert.Converter
```

JSF – Validation Model

- Validation is performed before data are set into properties connected with component through EL expression.
- Validation is performed after data is converted into java data type coresponding with bean property
- Set of standard validtors:
  - validateDoubleRange, validateLength, validateLongRange, validateRegex
- Method of BackingBean

  ```
  public void validate(FacesContext context, UIComponent
  component, Object value) throws ValidatorException
  ```
- Implementation of interface

  `javax.faces.validator.Validator`

JSF - validations

-
  ```
  <h:inputText id="userNo" label="User Number" value="#{userNumberBean.userNumber}" validatorMessage="message"><f:validateLongRange minimum="#{userNumberBean.minimum}" maximum="#{userNumberBean.maximum}" /></h:inputText>
  ```

-
  ```
  <h:message showSummary="true" showDetail="false" style="color: red; font-family: ’New Century Schoolbook’, serif; font-style: oblique; text-decoration: overline" id="errors1" for="userNo"/>
  ```

JSF - validations

-
  ```
  <h:inputText id="userNo" label="User
  Number"
  value="#{userNumberBean.userNumber}"
  validatorMessage="message">
  <f:validator validatorID="myValidatorID" />
  </h:inputText>
  ```

```
@FacesValidator(value="myValidatorID")
public class CounterVelidator implements
  javax.faces.validator.Validator
```

JSF - validations

-
  ```
  <h:inputText id="userNo" label="User
  Number"
  value="#{UserNumberBean.userNumber}"
  validatorMessage="zpráva"
  validator="#{counterVelidator.validate}">

  </h:inputText>
  ```

Reference to **method**

```
@ManagedBean
public class CounterVelidator implements
  javax.faces.validator.Validator

public void validate(FacesContext context, UIComponent
component, Object value) throws ValidatorException
```

JSF – in code evaluation of EL expresion

```
public Object getAsObject(FacesContext facesContext,
UIComponent component, String value) {


PersonMB controller = (PersonMB)
facesContext.getApplication().getELResolver().
   getValue(facesContext.getELContext(), null,
"personMB");
```

JSF – custom validation messages of standard validators

```xml
<application>
  <message-bundle>jat.validation-message</message-bundle>
</application>
```

```
javax.faces.converter.DateTimeConverter.DATE={2}: ''{0}'' could not be understood as a date.
javax.faces.converter.DateTimeConverter.DATE_detail=Invalid date format.

javax.faces.validator.LengthValidator.MINIMUM=Minimum length of ''{0}'' is required.
```

JSF – face messages

```
FacesContext ctx = FacesContext.getCurrentInstance();

FacesMessage msg = new FacesMessage
(FacesMessage.SEVERITY_INFO, errorMessage,
detailMessage);

ctx.addMessage(null, msg);
```

JSF – GUI - CombBox

```
<h:selectOneMenu value="#{personMB.editedEmployee}"
converter="#{converterFactory.employeeConverter}">
<f:selectItems
value="#{personMB.allEmployeesASSelectItem}"/>
</h:selectOneMenu>
```

```
public List<SelectItem> getAllEmployeesASSelectItem(){
  Collection<Employee> allEmp = getAllEmployees();
  ArrayList<SelectItem> selItems = new
ArrayList<SelectItem>(allEmp.size());
  for(Employee e : allEmp){
   selItems.add(new SelectItem(e, e.getName() + " " +
e.getSurename()));
  }
  return selItems;
}
```

JSF – GUI - CombBox

```
 <h:selectOneMenu
value="#{personAgendaMB.editedEmployee}"
converter="#{converterFactory.employeeConverter}">
  <f:selectItem noSelectionOption="true"
itemValue="#{null}" itemLabel="None" />
  <f:selectItems
value="#{personAgendaMB.allEmployees}" var="p"
itemLabel="#{p.name}" itemValue="#{p}"/>
</h:selectOneMenu>

public Collection<Person> getAllEmployees(){
  Collection<Employee> allEmp = getAllEmployees();
  return allEmp;
}
```

JSF - localization

```
<application>
<resource-bundle>
  <base-name>jat.messages</base-name>
  <var>msg</var>
</resource-bundle>
<locale-config>
  <default-locale>en</default-locale>
  <supported-locale>cs</supported-locale>
</locale-config>
</application>
```

File:
jat/messages.properties

```
userNoConvert=The value you entered is not a number.
```

```
<h:inputText id="userNo" label="User Number"
  value="#{…}"
  validatorMessage="#{msg.userNoConvert}">
```

JSF - localization

```
<f:view locale="#{languageMB.locale}">
public String setENLocale(){
FacesContext.getCurrentInstance().
getViewRoot().setLocale(Locale.ENGLISH);
return "";
}
public String setCZLocale(){
FacesContext.getCurrentInstance().
getViewRoot().setLocale(new Locale("cs"));
return "";
}
```

JSF – localization based on language of web browser

```
private String locale;
public String getLocale() {
  if(locale == null) {
  String languages =FacesContext.getCurrentInstance().
    getExternalContext().getRequestHeaderMap().
    get("Accept-Language");
  if(languages != null) {
   return languages.split(",")[0];
    }
  }
 return locale;
}
```

JSF – UI component model

- Coplet set of UI component
- Extensibility
- Base class UIComponentBase
  - UIColumn, UICommand, UIData, UIForm, UIGraphic, UIInput, UIMessage, UIMessages, UIOutput, UIPanel, UIParameter, UISelectBoolean, UISelectItem, UISelectItems, UISelectMany, UISelectOne, UIViewRoot
- Behavioral interfaces
  - ActionSource, ActionSource2, EditableValueHolder, NamingContainer, StateHolder, ValueHolder

JSF – Model of component rendering

- Separation of component beghavior from rendering
- One component can be represented by several diferend TAGs
- UISelectOne
  - Radio buttons
  - Combo box
  - List box

JSF – Event – Listener model

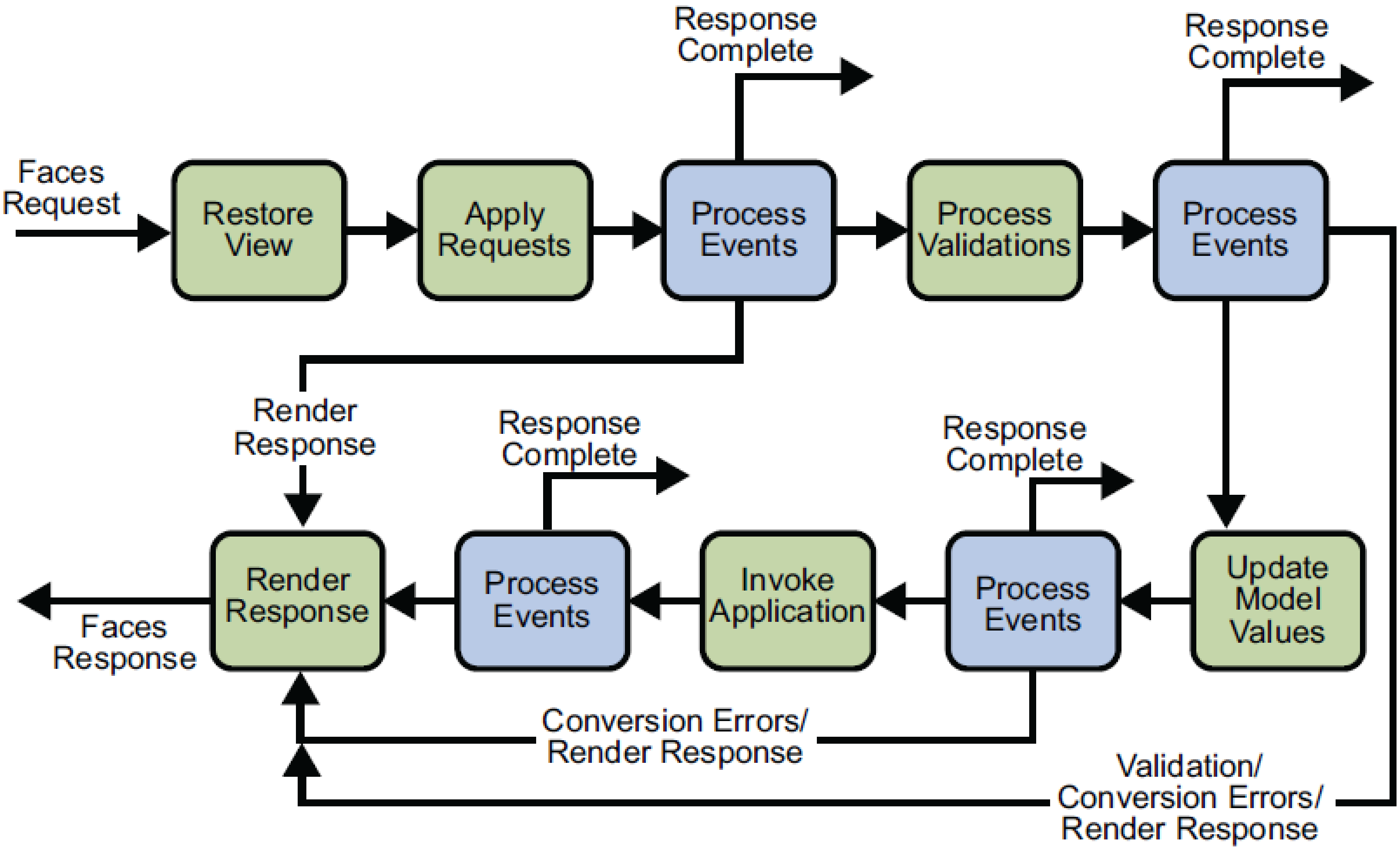
- Like Evenl-Lisener model from JavaBeans
- 3 type of events: value-change events, action events, data-model events
  - Implementation of method in BackingBean and usage of EL expression for method reference in atribute of component TAG.
  - Implementation of listener

Class which implement interface **listener**

```
<h:inputText id="name" size="50“
         value="#{cashier.name}" required="true">
  <f:valueChangeListener type="listeners.NameChanged"/>
  <f:valueChangeListener binding="#{mBean.property}"/>
</h:inputText>
```

Expresion have to return object which implements interface of **listener**

JSF – Life Cycle



P6

- EJB - Enterprise JavaBeans

Enterprise Java Beans (EJB)

- Specification of architecture for development and deployment of distributed transactional object component on server side
- Conventions + set of interfaces(EJB API)
- Target = ensure compatibility between products from diferent suppliers
  - components
  - Container
- EJB 3.0

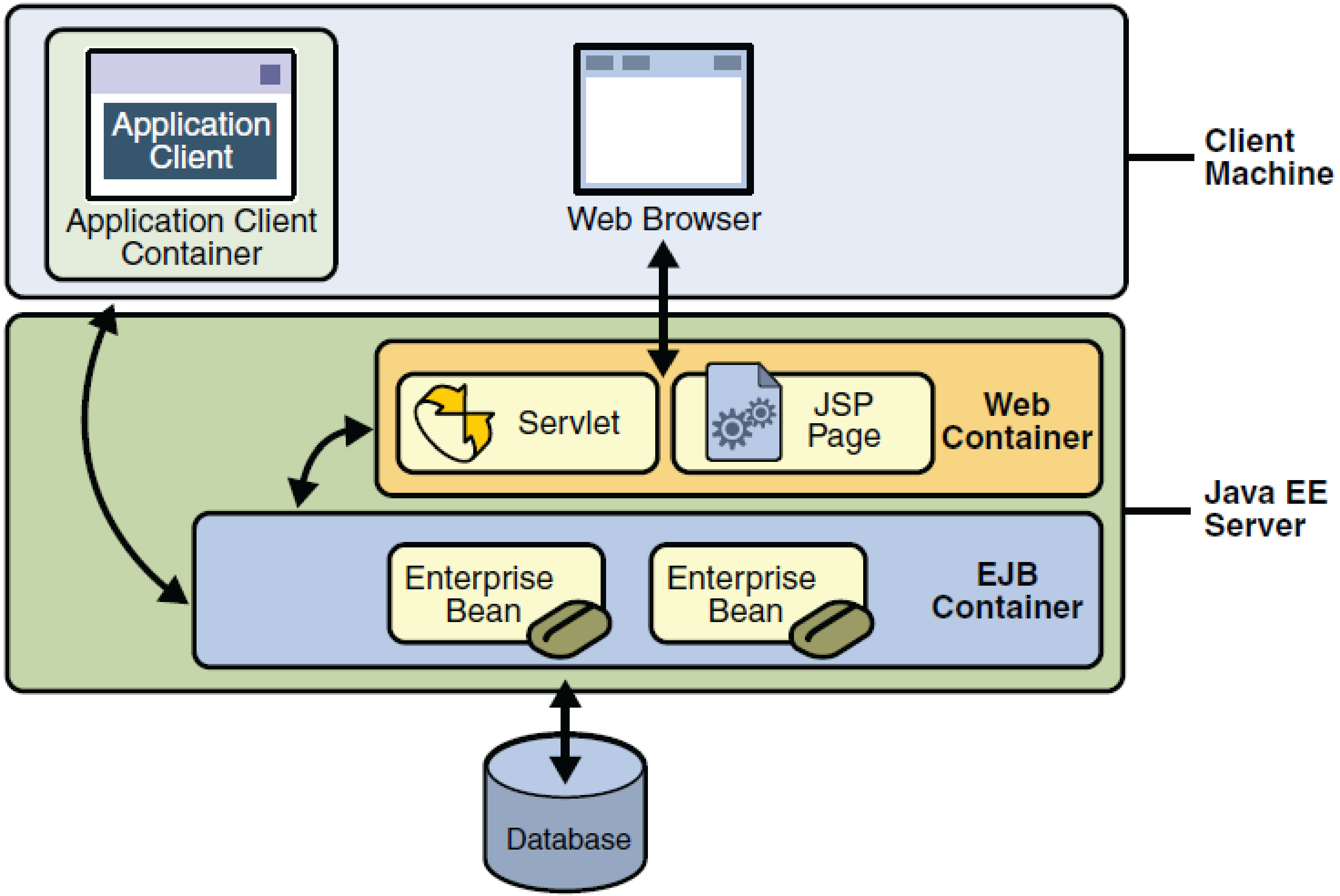
Enterprise JavaBeans

- EnterpriseBean are components implementing technology Enterprise JavaBeans (EJB)
- EnterpriseBean runs inside EJB container or web container
- EnterpriseBean is server side component encapsulating business logic
- EnterpriseBean can be invoked remotely

EJB container

- Environment for component
  - Remote access
  - security
  - transaction
  - Parallel access
  - Access to resources and their sharing
- Isolation of component from application
  - Independent of container producer
  - Development of application is easier

EJB – usage


Application Client
Application Client Container
Web Browser
Client Machine
Servlet
JSP Page
Web Container
Java EE Server
Enterprise Bean
Enterprise Bean
EJB Container
Database

Types of EJB components

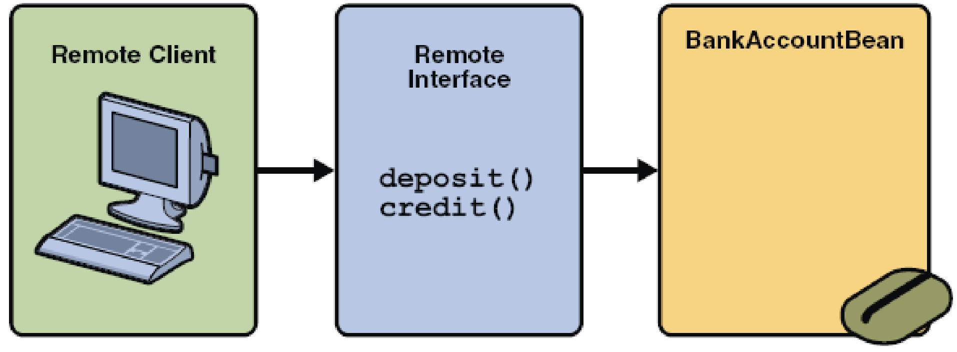
- Session Bean:
  - Stateless session bean
  - Statefull session bean
  - Singelton
- Message-Driven Beans
  - Stateless service which can be call asynchronously

SessionBean - usage

- To concrete instance access only one client at time
- Sate is not persistent, onlz short time (hours)
- Web services
- Statefull
  - Interaction between SB and client, hold information between calls SB
- Stateless
  - No information are stored for concrete client
  - General tasks

SessionBean - Interface

- Clienat access using interface (business interface)
- One bean can have more then one business interfaces

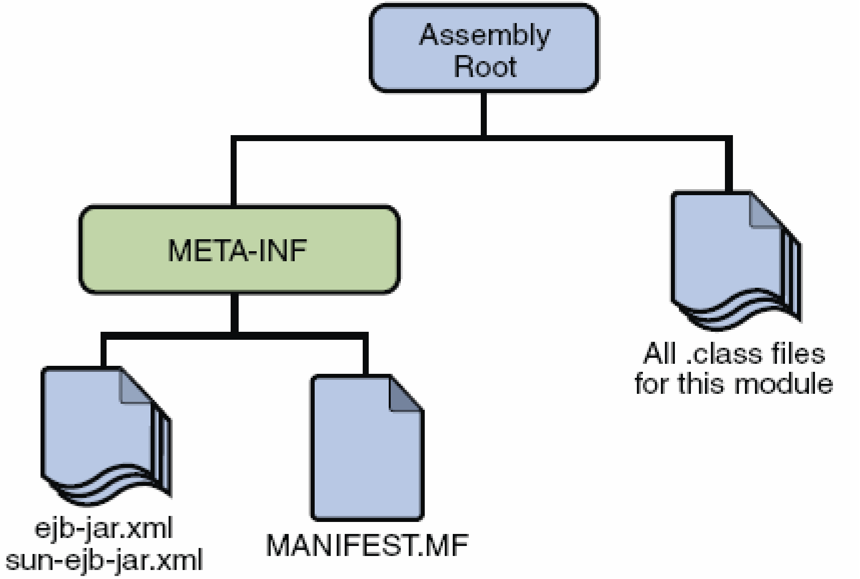


```
@Remote
public interface Account {
}
@Stateless
public class AccountBean implements Account {
    public AccountBean() {
    }
}
```

SessionBean - implementation

- Interface: `interface NameOfInterface`
- Class of component

  `class NameOfInterfaceBean implements NameOfInterface`
- Support classes



SessionBean – Remote vs. Local

Remote client

- Can run on remote JVM (other JVM)
- Client can be:
  - Web component
  - Application
  - Another EJB
- Big isolation of method's parameters
  - Client and bean work with diferent copies of objects
  - Beter security
- Data granularity

Local client

- Must run in same JVM
- Client can be:
  - Web component
  - Another EJB
- Weak isolation
  - Client and bean work with same object
  - Change done in bean has efect in client
  - Lower security

Statefull SessionBean – life cycle

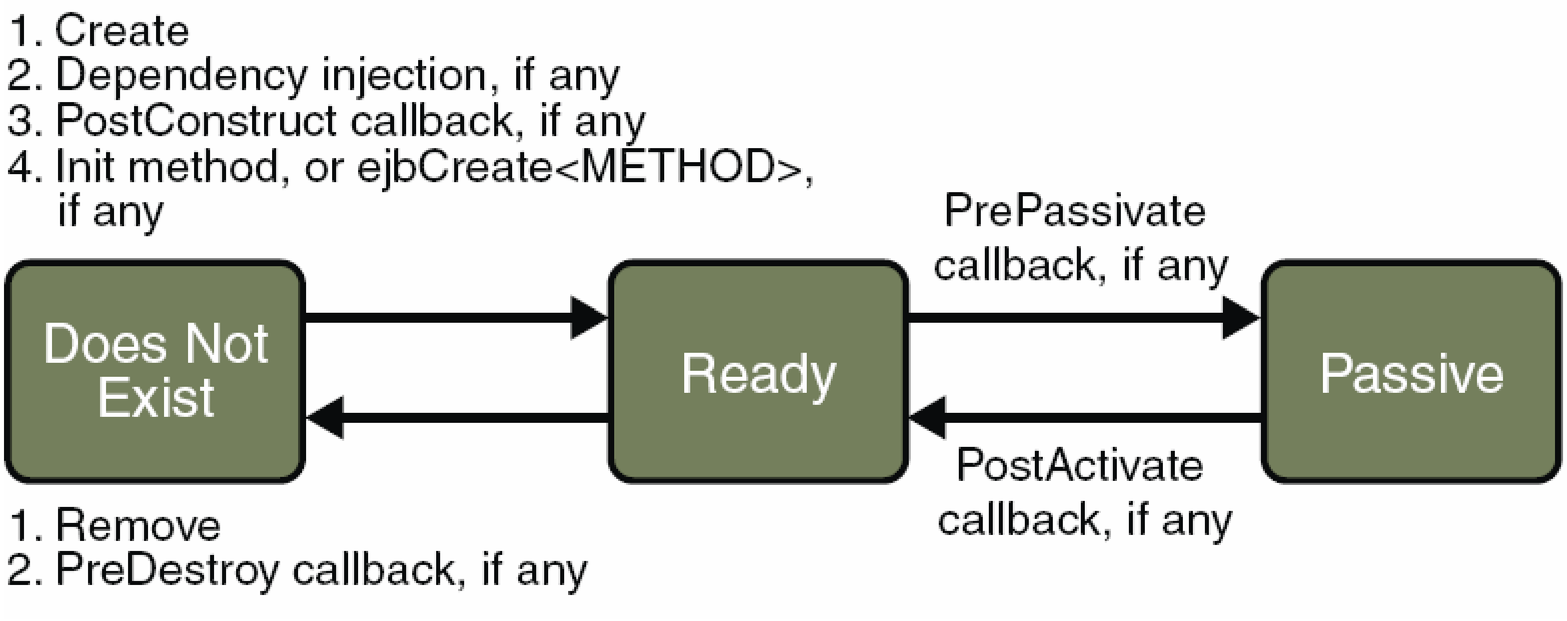
@PostConstruct

@PreDestroy

@Remove – client call method with this annotation to mark conversation as finished

@PrePassivate

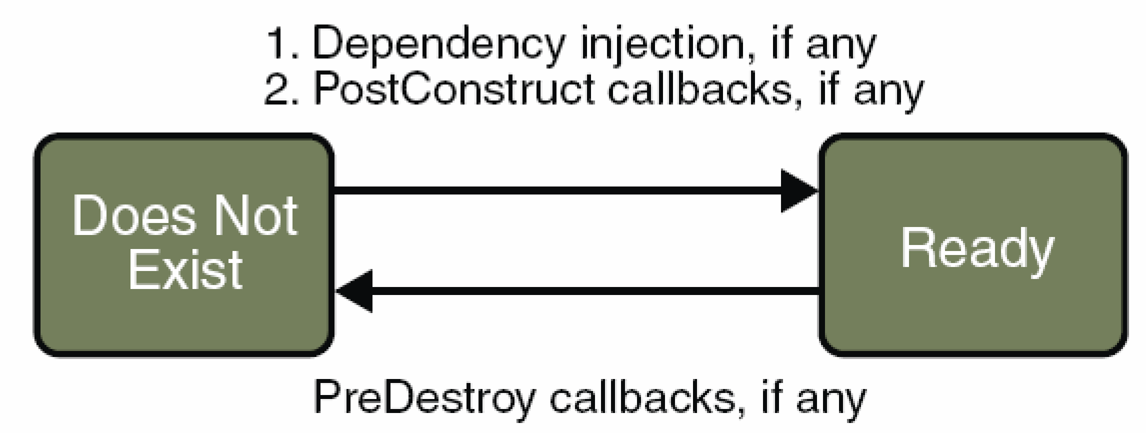
@PostActivate


1. Create
2. Dependency injection, if any
3. PostConstruct callback, if any
4. Init method, or ejbCreate<METHOD>, if any
Does Not Exist
Ready
PrePassivate callback, if any
Passive
PostActivate callback, if any
1. Remove
2. PreDestroy callback, if any

Stateless SessionBean – life cycle

@PostConstruct

@PreDestroy


1. Dependency injection, if any
2. PostConstruct callbacks, if any
Does Not Exist
Ready
PreDestroy callbacks, if any

SessionBean – server implementation

```
@Remote
public interface Account {
    public void deposite(String accNum, float amounght);
    public void remove(String accNum, float amounght);
}
```

<projectName>EAR/<BeanName>Bean/remote

```
@Stateless(mappedName = "comp/env/ejb/Account")
public class AccountBean implements Account {
    public AccountBean() {
    }
    @Override
    public void deposite(String accNum, float amounght)
{
    }
    @Override
    public void remove(String accNum, float amounght) {
    }
}
```

SessionBean – interfaces

```
@Remote

@Local

@LocalBean
```

SessionBean – server implementation

```
@Remote
public interface Account {
    public void deposite(String accNum, float amounght);
    public void remove(String accNum, float amounght);
}
```

<projectName>EAR/<BeanName>Bean/remote

```
@Stateless(mappedName = "comp/env/ejb/Account")
public class AccountBean implements Account {
    public AccountBean() {
    }
    @Override
    public void deposite(String accNum, float amounght)
{
    }
    @Override
    public void remove(String accNum, float amounght) {
    }
}
```

SessionBean – client implementation

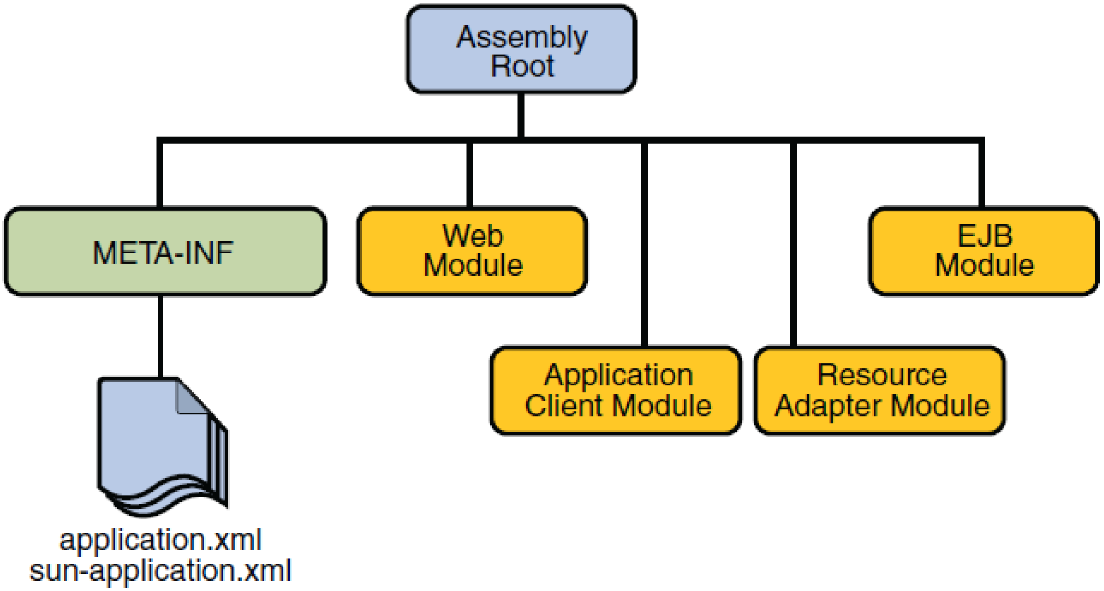
- Client have to have access to Business interface (for example as *.jar)
- Client use libraries from JavaEE
- Types of clients
  - Application Client Container
  - Normal application
  - Web client (war)

SessionBean – client implementation of normal application

```
public class Main {

   protected static Account a;
   public static void main(String[] args) {
      Properties props = new Properties();

props.setProperty(Context.INITIAL_CONTEXT_FACTORY,

"org.jnp.interfaces.NamingContextFactory");
      props.setProperty(Context.URL_PKG_PREFIXES,
                           "org.jboss.naming.client");
      props.setProperty(Context.PROVIDER_URL,
                              "jnp://localhost:1099");
      InitialContext ctx = new InitialContext(props);
      a = (Account)ctx.lookup("comp/env/ejb/Account");
System.out.println("Amount:" +
               a.deposite("1234/0300", 10.25f));
   }
```

SessionBean – client implementation ACC

```
public class RunClient {
    @EJB
    static protected Account ac;
    public static void main(String[] args) {
        System.out.println("Amount: " +
                  ac.deposite("1234/0300", 10.25f));
    }
}
```



Assembly Root

META-INF

Web Module

Application Client Module

Resource Adapter Module

EJB Module

application.xml
sun-application.xml

SessionBean – client implementation web application - servlet

```
public class EnterpriseServlet extends HttpServlet {
   @EJB
   protected Account ac;
   public EnterpriseServlet() {
        super();
   }
   protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
      response.setContentType("text/html");
      PrintWriter pw = response.getWriter();
      pw.write("<html><body>" + ac.deposite("", 17.5f)
+
               "</body></html>");
   }
}
```

Session Bean - Injection

- Resource injection can be done only for objects where for their instantiation is in responsibility of JavaEE container
  - Servlets
  - ServletContextListener
  - Managed Backing Beans in JSF

Session Bean - JSF

- Can be easily initialized using dependency injection in Managed Bean
- Managed Bean call method of EJB
- Old version of JSF has no direct support for session invalidation

```
counter.release();
HttpSession s = (HttpSession)(FacesContext.
      getCurrentInstance().getExternalContext().
      getSession(false));
s.invalidate();
```

Stateful SessionBean

- Bean hold state (values of instance variables) between individual calls
- Bean has only one client (that guarantee). The lifetime of bean is same as lifetime of variable with annotation @EJB

Stateful Session Bean - JSF

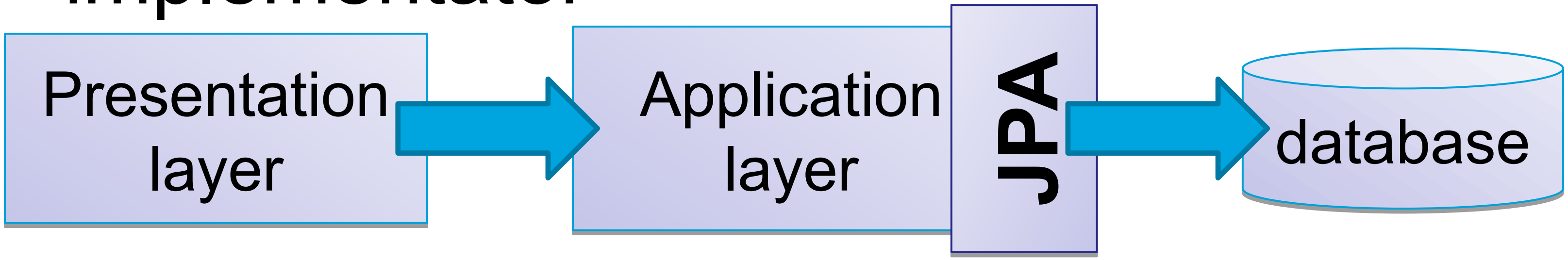
```
public String logout()
{
  FacesContext.getCurrentInstance().getExternalContext().
    invalidateSession();
  return "/home.xhtml?faces-redirect=true";
}

public void logout() throws IOException {
  ExternalContext ec =
    FacesContext.getCurrentInstance().getExternalContext();
  ec.invalidateSession(); ec.redirect(ec.getRequestContextPath() +
    "/home.xhtml");
}
```

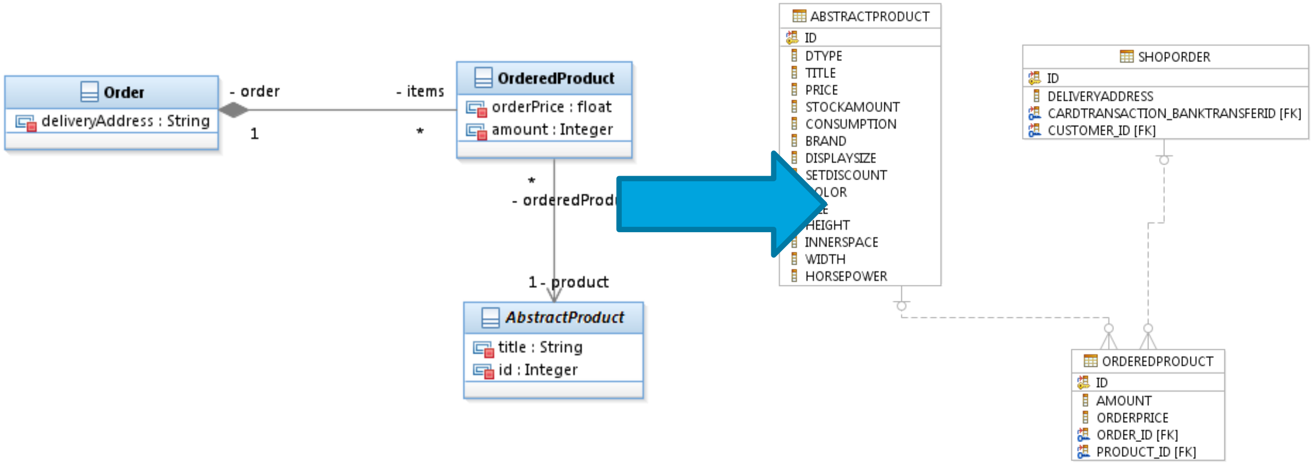
P7

- Java Persistence API
  - Language QL
- Hibernate
  - HQL
  - http://docs.jboss.org/hibernate/stable/core/reference/en/html/tutorial.html
  - http://www.manning.com/bauer2/chapter2.pdf

JPA – Java Persistent API

- API for persistance using object-relation mapping
- Only interface, it is necessary add implementator





JPA - Entity

- **Entity** – it is light-weight object from persistence domain. Typically are connected with database table.
  - Eache object is related to one record in database table.
- **Persistent state of entity** is reprezented by instance variables and class properties.
  - Mapping between database and properties is defined by annotations.

JPA – Entity class

- Class have to has annotation `javax.persistence.Entity`
- Class have to has public or protected constructor with no parameter (can have another constructors)
- Class and methods and instance variables cannot be declared as `final`

JPA – Entity class

- If is entity used in remote EJB interface have to implemented interface `Serializable`
- Entity class can be descendant of entity class or non-entity class. Non-entity classes can by descendat of entity class.
- Persistance instance variables have to be declared as private, protected or package-private. They should be accessed through set and get methods.

JPA – Entity class - example

```
@Entity
@Table(name="ShopOrder")
public class Order {
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;
  @OneToOne
  private Transaction cardTransaction;
  @ManyToOne()
  private Customer customer;
  @OneToMany(mappedBy="order")
  private Set<OrderedProduct> items;
  private String deliveryAddress;
  …
}
```

JPA – persistence properties, instance variables

- Instance variables – persistence provider access directly to them
- Properties – Persistence access properties using get, set method
  - Can be used: Collection, Set, List, Map even generic versions

- **`hashcode() equals()`**
- Types: Java primitive data types
  - java.lang.String, other serializable types (classes represented primitive data tzpes, java.math.BigInteger, java.math.BigDecimal, java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.TimeStamp, user serialzable types, byte[], Byte[], char[], Character[], enum types, other entities, collections of entities

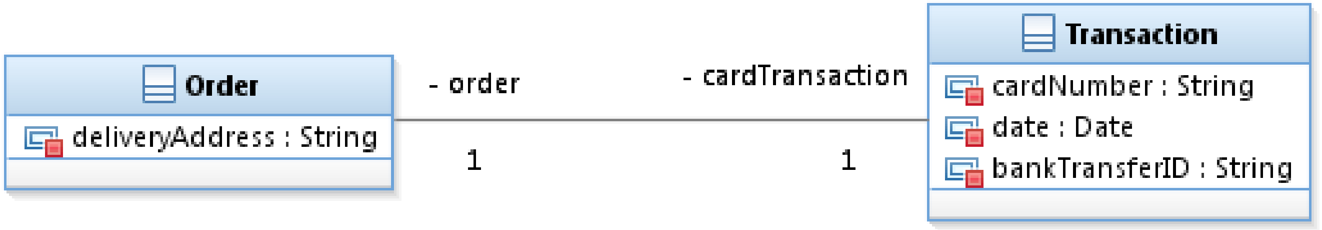
JPA – primary keys

- Each entity have to have owen primary key.
- javax.persistence.Id
- Composite Primary Key
  - Have to exisit class which define composite key
  - javax.persistence.EmbeddedId
  - javax.persistence.IdClass
  - Have to be composed from types:
    - Java Primitive data types (and coresponding embedded classes)
    - java.lang.String
    - java.util.Date (DATE), java.sql.Date

JPA – relation 1-1

```
@Entity
public class Order {
@OneToOne
private Transaction
cardTransaction;
…
```

```
@Entity
public class Transaction {
@OneToOne
private Order order;
…
```

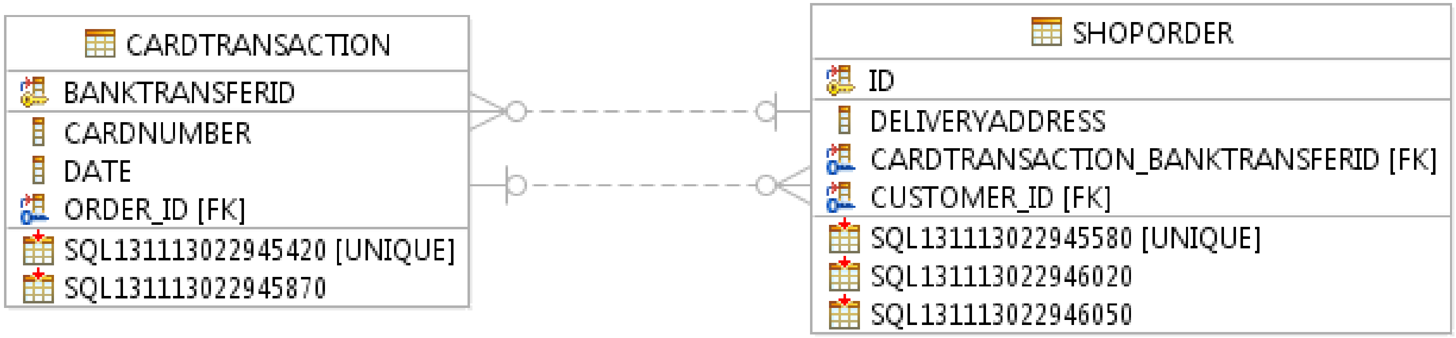


Order
deliveryAddress : String

- order
1

- cardTransaction
1

Transaction
cardNumber : String
date : Date
bankTransferID : String



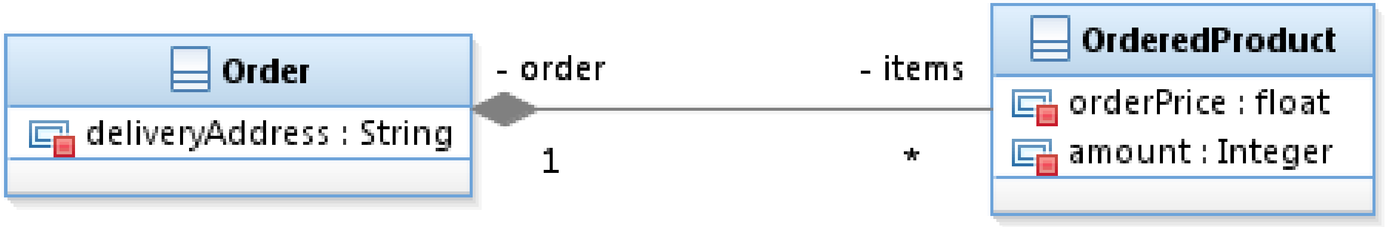
CARDTRANSACTION
BANKTRANSFERID
CARDNUMBER
DATE
ORDER_ID [FK]
SQL131113022945420 [UNIQUE]
SQL131113022945870

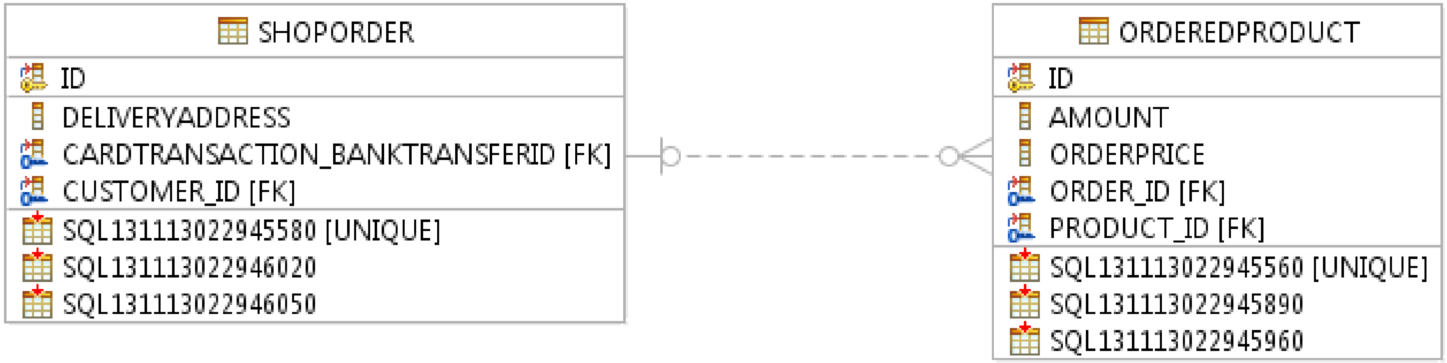
SHOPORDER
ID
DELIVERYADDRESS
CARDTRANSACTION_BANKTRANSFERID [FK]
CUSTOMER_ID [FK]
SQL131113022945580 [UNIQUE]
SQL131113022946020
SQL131113022946050

JPA – relation 1-N

```
@Entity
public class Order {
@OneToMany(mappedBy="order")
private Set<OrderedProduct>
items;
…
```

```
@Entity
public class
OrderedProduct {
@ManyToOne
private Order order;
…
```





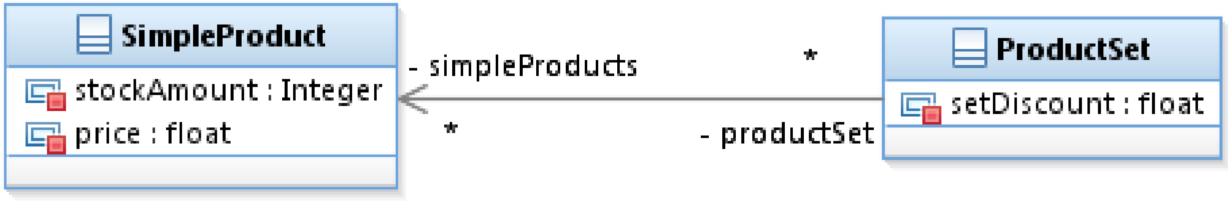
SHOPORDER
ID
DELIVERYADDRESS
CARDTRANSACTION_BANKTRANSFERID [FK]
CUSTOMER_ID [FK]
SQL131113022945580 [UNIQUE]
SQL131113022946020
SQL131113022946050

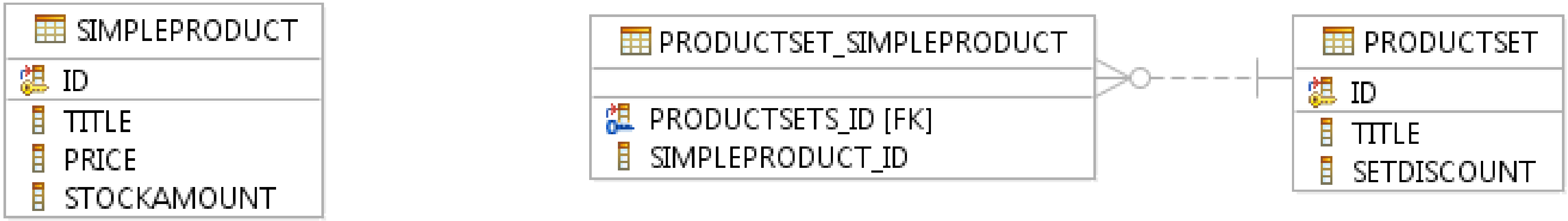
ORDEREDPRODUCT
ID
AMOUNT
ORDERPRICE
ORDER_ID [FK]
PRODUCT_ID [FK]
SQL131113022945560 [UNIQUE]
SQL131113022945890
SQL131113022945960

JPA – relation M-N

```
@Entity
public class SimpleProduct
extends AbstractProduct {
@ManyToMany(mappedBy="simpl
eProduct")
private List<ProductSet>
productSets;
}
```

```
@Entity
public class ProductSet
extends @ManyToMany
private List<SimpleProduct>
simpleProduct;
private float setDiscount;
}
```





JPA – inheritance

- **Entity can be extended from non entity calss**
- **Entita – can be extend from abstract class**

```
@Entity
public abstract class Employee {
@Id
protected Integer employeeId;
}
@Entity
public class FullTimeEmployee
    extends Employee {
protected Integer salary;
}
```

```
@Entity
public class PartTimeEmployee
    extends Employee {
protected Float hourlyWage;
}
```

JPA – inheritance mapping strategy

- One table peer class hierarchy
- One table for non-abstract entity
- Join strategy

```
public enum InheritanceType {
   SINGLE_TABLE,
   JOINED,
   TABLE_PER_CLASS
};
@Inheritance(strategy=JOINED)
```

JPA – inheritance mapping strategy

One table peer class hierarchy

@Inheritance(strategy=SINGLE_TABLE)

@DiscriminatorColumn(

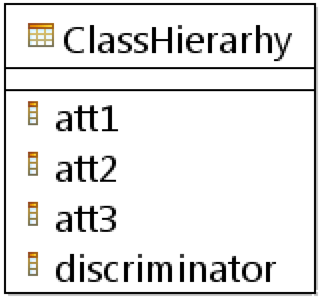
String name

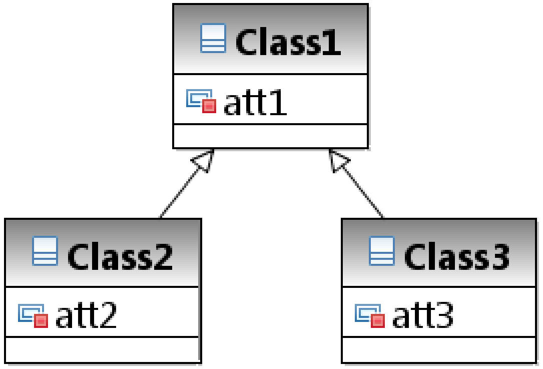
DiscriminatorType discriminatorType

String columnDefinition

String length)

```
public enum DiscriminatorType {
    STRING,
    CHAR,
    INTEGER
};
```



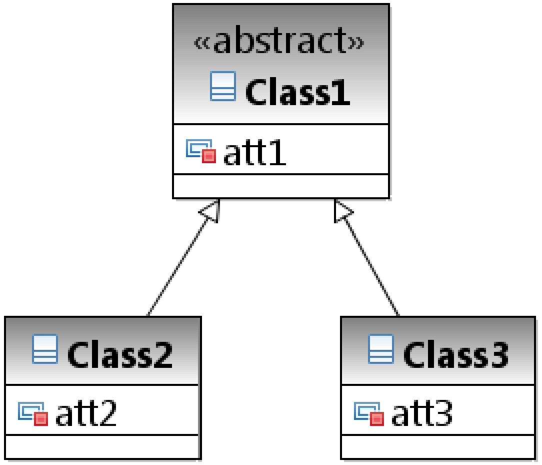


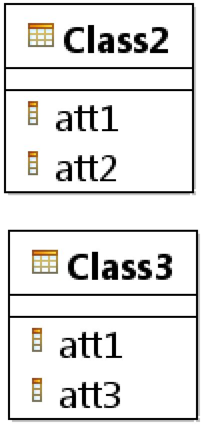
@DiscriminatorValue

JPA – inheritance mapping strategy

One table for non-abstract entity

@Inheritance(strategy=TABLE_PER_CLASS)

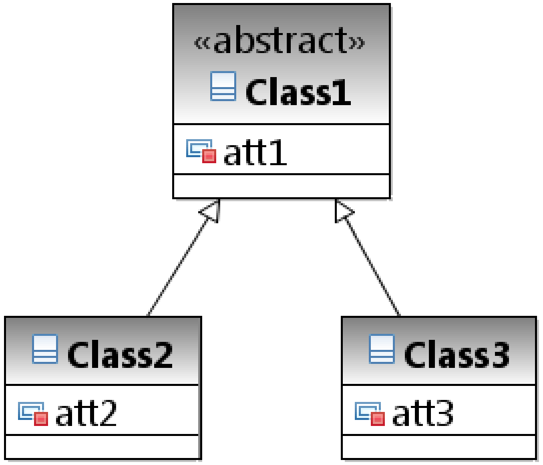


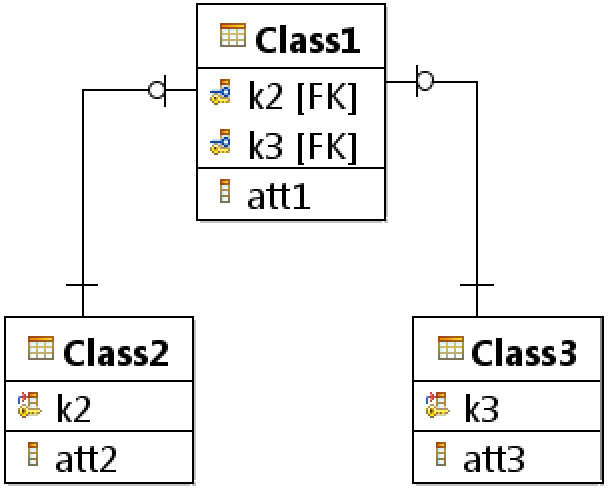


JPA – inheritance mapping strategy

Join strategy

@Inheritance(strategy=JOINED)





JPA – MappedSuperclass

```
@MappedSuperclass
public class Person {
@Column(length=50)
private String name;
@Column(length=50)
private String surename;
@Column(length=50)
private String email;
@Column(length=50)
private String password;
}
```

```
@Entity
public class Customer extends
Person {
@Id
@GeneratedValue(strategy=Gener
ationType.IDENTITY)
private int id;
@OneToMany(mappedBy="customer"
)
private Set<Order> orders;
```

```
@Entity
public class Employee extends Person {
@Id
@Column(length=50)
private String login;
private float sallary;
@Column(length=50)
private String depsrtment;
}
```

JPA – entity manager

- **Persistence context**: set of entities which exisit in concrete data storage
- EntityManager
  - Create, delete entities, find entities, execute queries

- Container managed entity manager

```
@PersistenceContext
EntityManager em;
```

JPA – find entities

```
@PersistenceContext
EntityManager em;

public void enterOrder(int custID, Order newOrder) {
    Customer cust = em.find(Customer.class, custID);
    cust.getOrders().add(newOrder);
    newOrder.setCustomer(cust);
}
```

JPA – entity life cycle

- New
- Managed
- Detached
- Removed

```
@PersistenceContext
EntityManager em;
...
public LineItem createLineItem(Order order,
    Product product, int quantity) {
    LineItem li = new LineItem(order, product,
    quantity);
    order.getLineItems().add(li);
    em.persist(li);
    return li;
}

em.remove(order);
em.flush();
```

JPA - queries

```
public List findWithName(String name) {
   return em.createQuery(
   "SELECT c FROM Customer c WHERE
   c.name LIKE :custName")
   .setParameter("custName", name)
   .setMaxResults(10)
   .getResultList();
}
.setFirstResult(100)
```

JPA – named queries

Class annotation - entity

```
@NamedQuery(
    name="findAllCustomersWithName",
    query="SELECT c FROM Customer c WHERE c.name
    LIKE :custName"
)

@PersistenceContext
public EntityManager em;
...
customers = em.createNamedQuery("findAllCustomersWithName")
    .setParameter("custName", "Smith")
    .getResultList();
```

JPA – parameters in queries

- Named

```
return em.createQuery(
    "SELECT c FROM Customer c WHERE c.name LIKE :custName")
    .setParameter("custName", name)
    .getResultList();
```

- Numbered

```
return em.createQuery(
    “SELECT c FROM Customer c WHERE c.name LIKE ?1”)
    .setParameter(1, name)
    .getResultList();
```

JPA – Persistence Units

- Package contains all entiy classes mapped into one database storage (DB).
- Have to contains file persistence.xml
- Can be part of EAR, WAR, EJB JAR

JPA – persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
<persistence-unit name="Slaids">
<provider>org.hibernate.ejb.HibernatePersistence</provider>
<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
<jta-data-source>java:/jdbc/slaids</jta-data-source>
<properties>
<property name="javax.persistence.schema-generation.database.action" value="create"/>
<property name="hibernate.hbm2ddl.auto" value="create"/>
<property name="hibernate.dialect"
value="org.hibernate.dialect.DerbyTenSevenDialect"/>
```

- <property name="*eclipselink.ddl-generation*" value="*create-tables*" />
- <property name="*eclipselink.ddl-generation.output-mode*" value="*database*" />
- <property name="*eclipselink.target-database*" value="*Derby*"/>

```
</properties>
</persistence-unit>
</persistence>
```

JPA – Query Language

Select Statement

- SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY

Update, Delte Statement

- UPDATE Player p SET p.status = ’inactive’ WHERE p.lastPlayed < :inactiveThresholdDate
- DELETE FROM Player p WHERE p.status = ’inactive’ AND p.teams IS EMPTY

JPA – queries - examples

- SELECT p FROM Player AS p
- SELECT DISTINCT p FROM Player AS p WHERE p.position = ?1
- SELECT DISTINCT p FROM Player AS p, IN(p.teams) AS t
- SELECT DISTINCT p FROM Player AS p JOIN p.teams AS t
- SELECT DISTINCT p FROM Player AS p WHERE p.team IS NOT EMPTY
- SELECT t FROM Team AS t JOIN t.league AS l WHERE l.sport = ’soccer’ OR l.sport =’football’
- SELECT DISTINCT p FROM Player AS p, IN (p.teams) AS t WHERE t.city = :city

JPA – queries - examples

- SELECT DISTINCT p FROM Player AS p, IN (p.teams) AS t WHERE t.league.sport = :sport

JPA – query - LIKE

- SELECT p FROM Player p WHERE p.name LIKE ’Mich%’
- _ - pattern matches exactly one character
- % - pattern can match zero or more characters
- ESCAPE – can define another escape character
  - LIKE ’_%’ ESCAPE ’\’
- NOT LIKE

JPA – gueries – NULL, IS EMPTY

- SELECT t FROM Team t WHERE t.league IS NULL
- SELECT t FROM Team t WHERE t.league IS NOT NULL
- Nelze použít WHERE t.league = NULL

- SELECT p FROM Player p WHERE p.teams IS EMPTY
- SELECT p FROM Player p WHERE p.teams IS NOT EMPTY

JPA – queries between, in

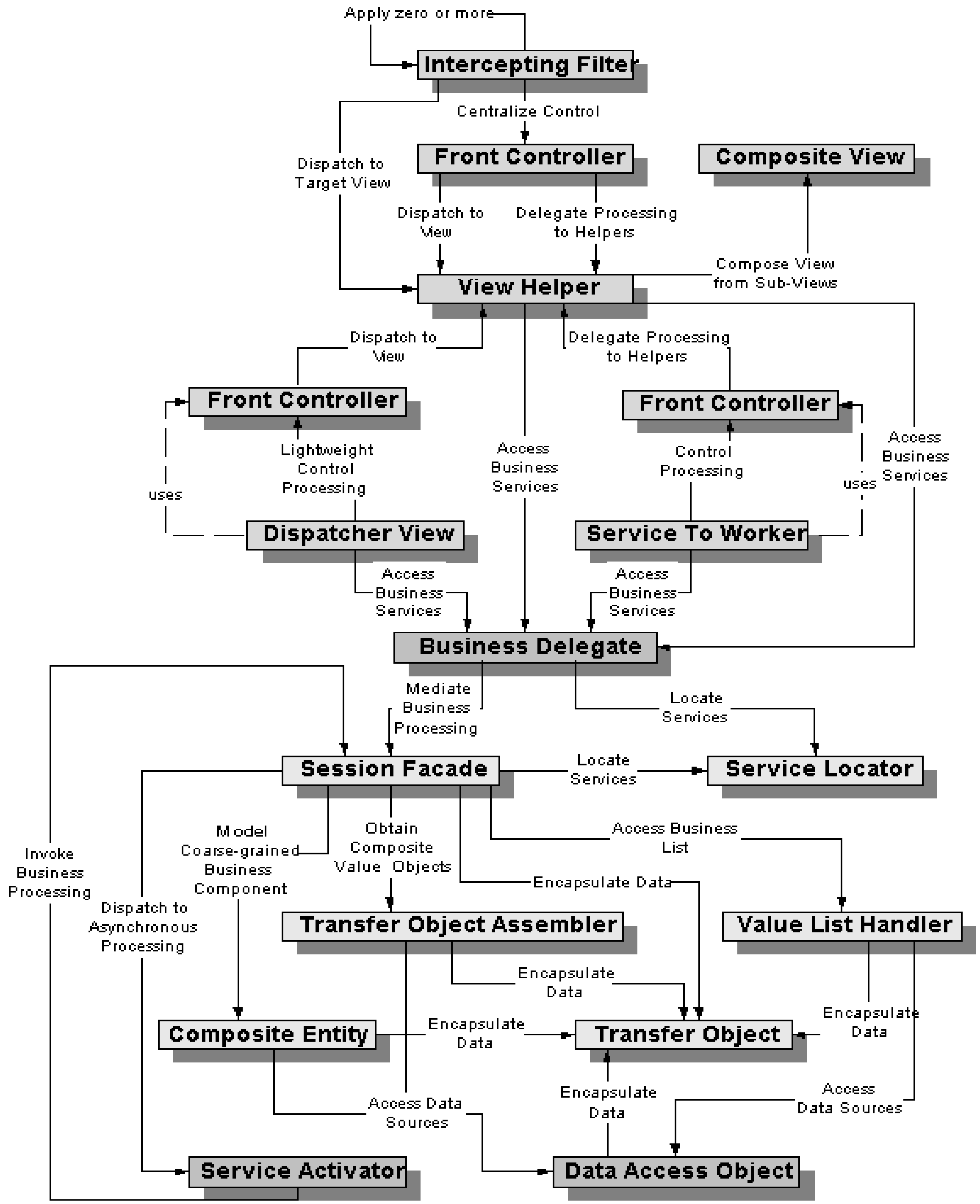
- SELECT DISTINCT p FROM Player p WHERE p.salary BETWEEN :lowerSalary AND :higherSalary
- p.salary >= :lowerSalary AND p.salary <= :higherSalary

- o.country IN (’UK’, ’US’, ’France’)

P8

- Design patterns JavaEE
  - DAO
- http://java.sun.com/blueprints/corej2eepatterns/Patterns/

Design patterns JavaEE



DAO – Data Access Object

Problem

- Different types of data storage need different access methods.
- Change data another storage lead to big code rework.

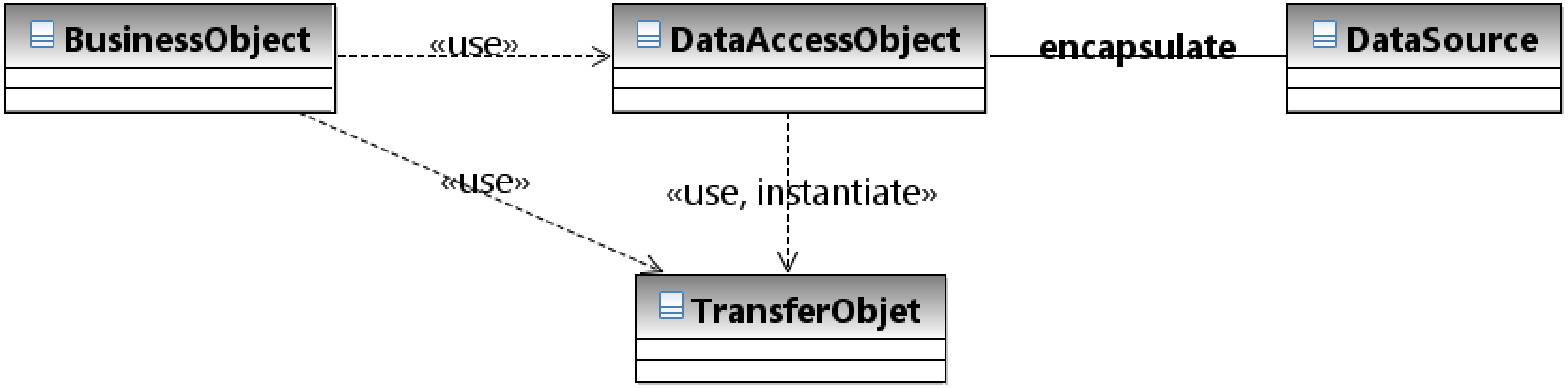
Forces

- Components need store data
- Access to each storage is different
- Components usualy use proprietar API to access storage
- Reduced portability of components
- Components should be transparents to storage implementation and should allow simple migration

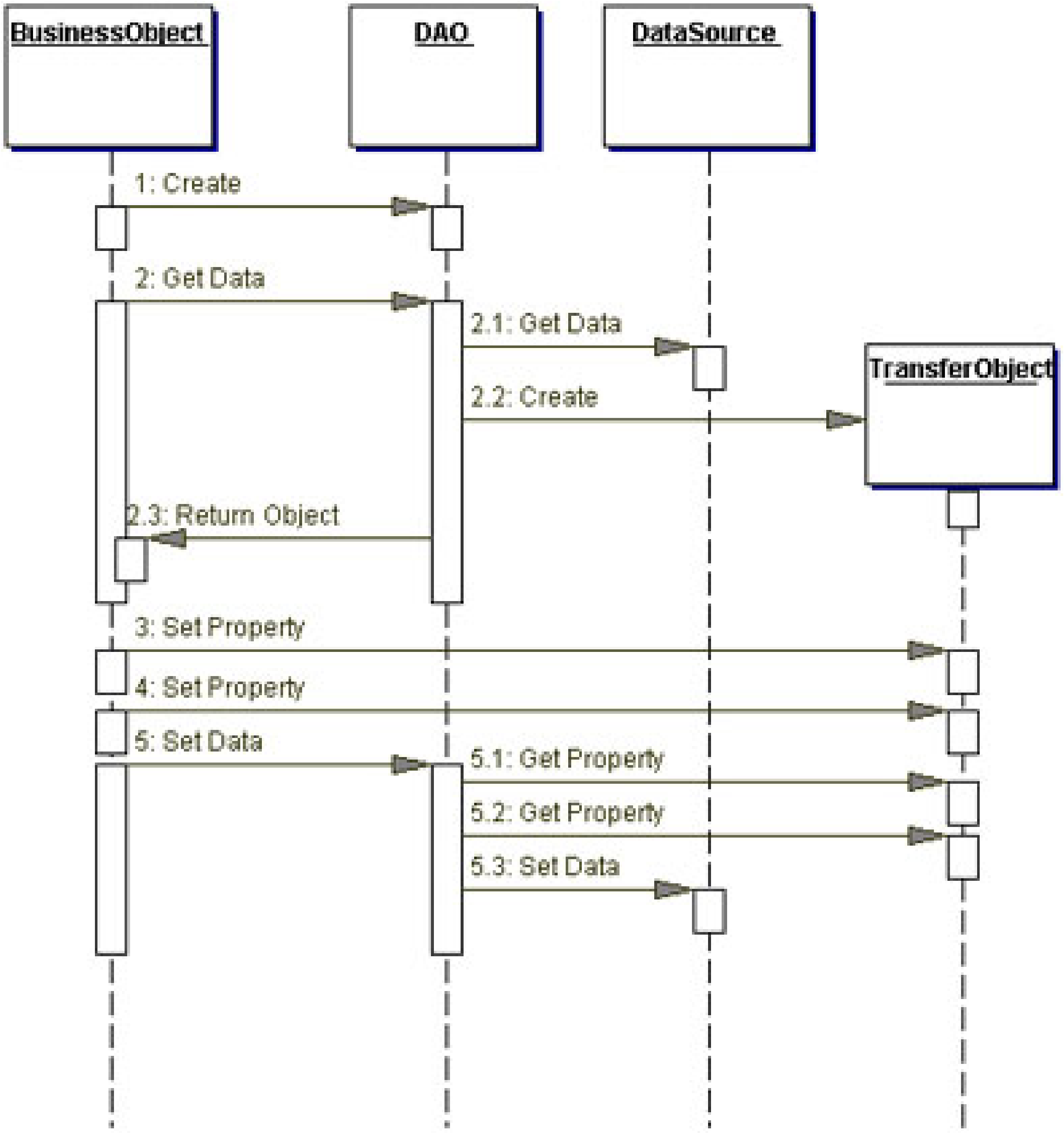
DAO – Data Access Object

Solution

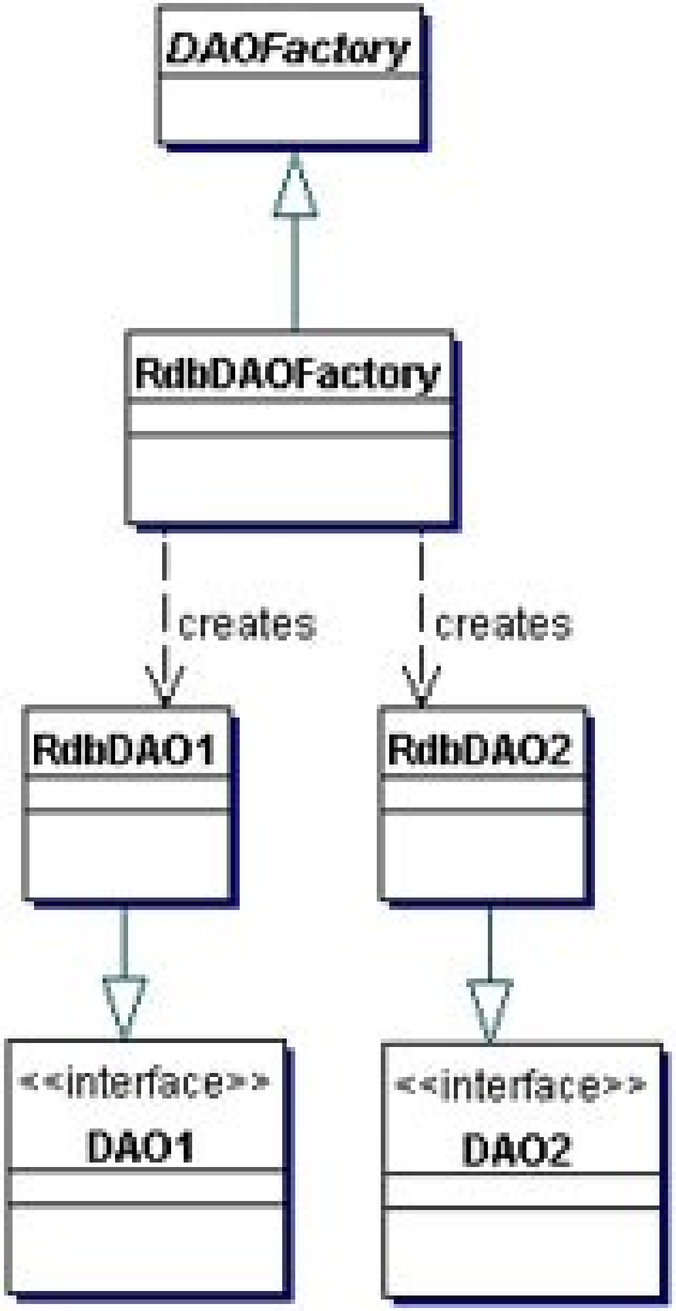
- Usage of DAO object for encapsulation all accesses to storage. DAO take over responsibility for connection to storage and store or retrive data.
- DAO provides simple and storage-independent interface.



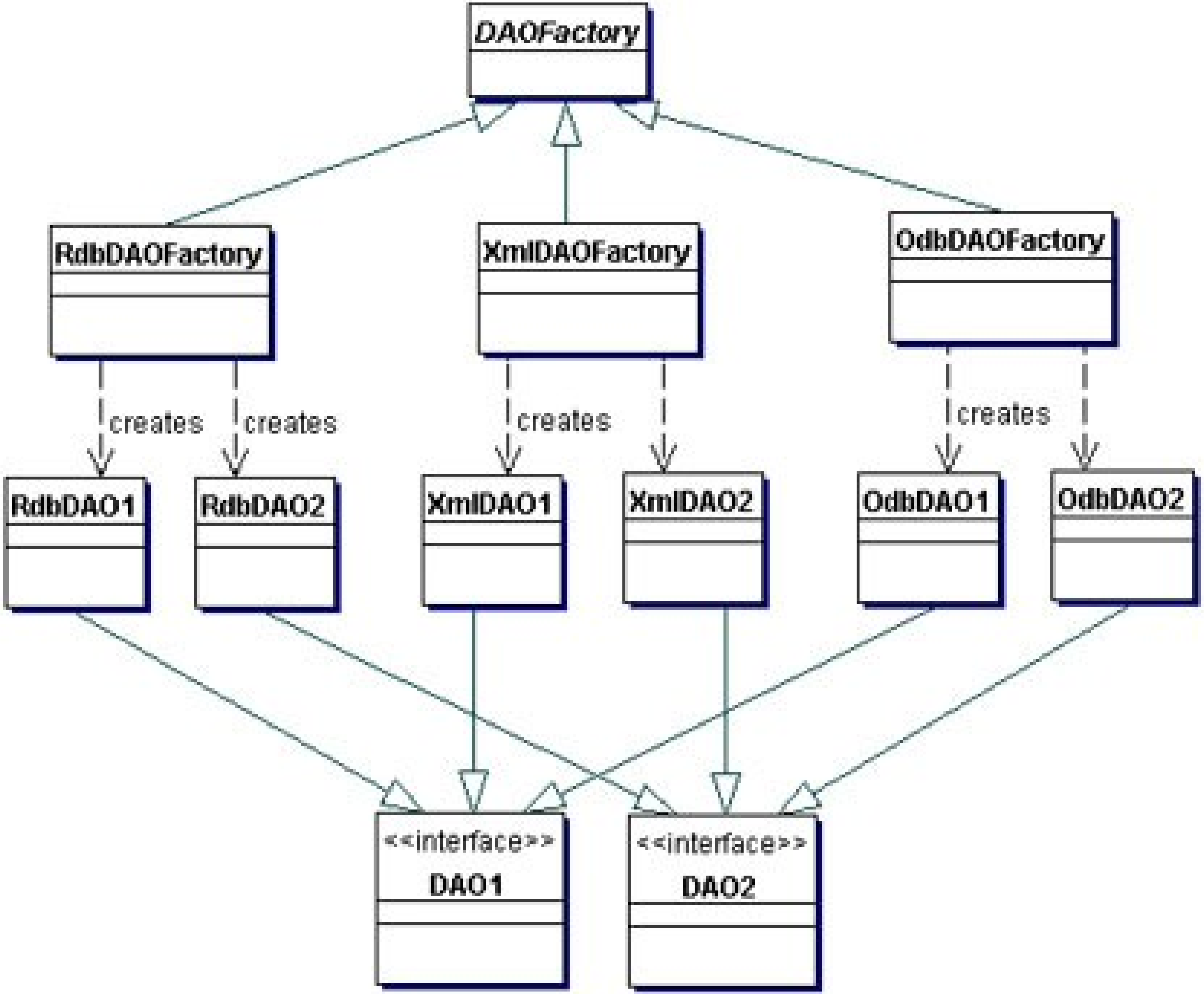
DAO – Data Access Object



Factory for DAO



Factory for DAO



P10

- Web services

What are web services

- Interface to application which is accesible throught computer network, based on standard internet technologies.
- Generally: if is application accessibel through network by protocols like HTTP, XML, SMTP or Jabber, it is web service.
- Layer between server side application program and program on client side.

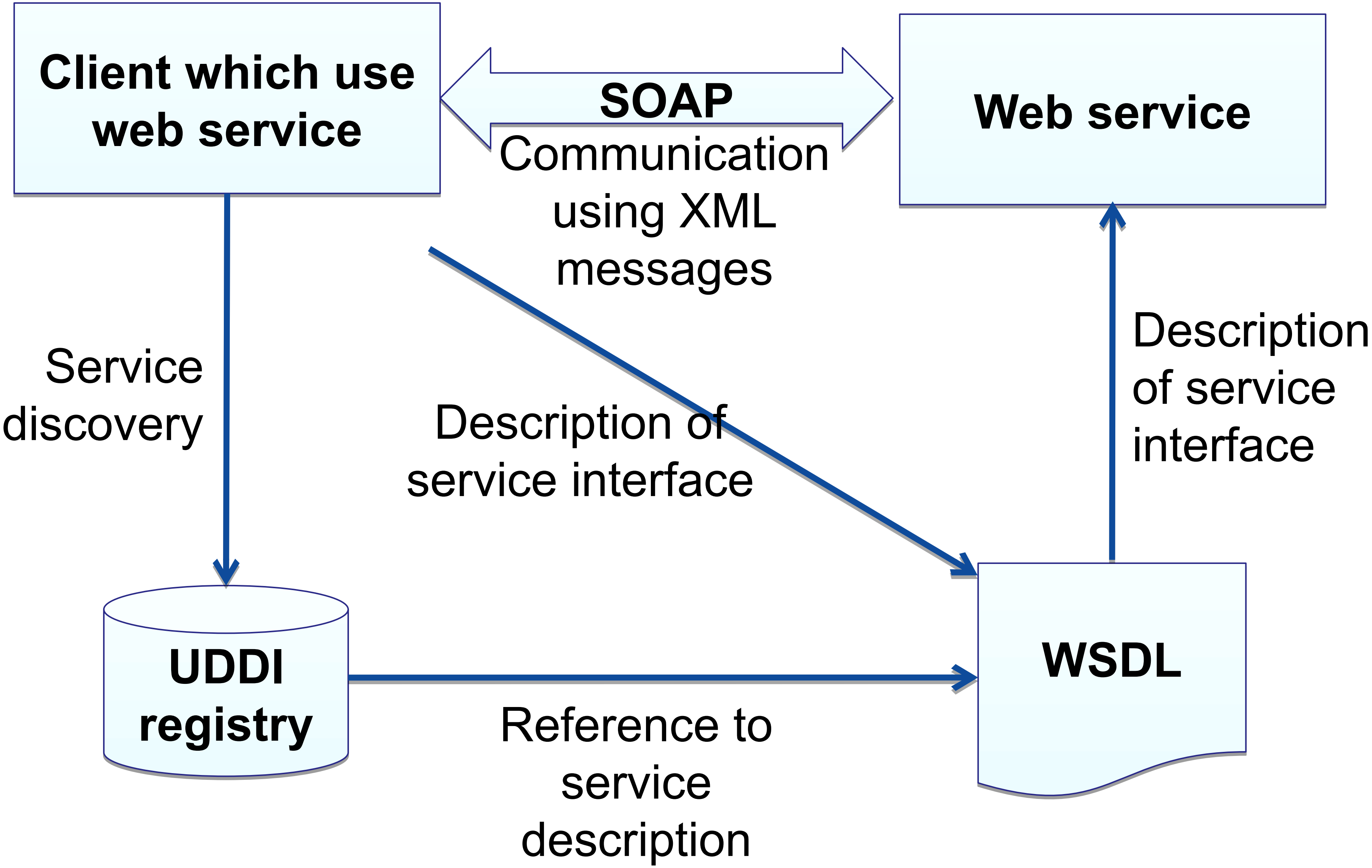
What are web services

- Functionality of service is not dependent on programming language of client or server (Java, C++, PHP, C#, ...).
- Example: HTML pages:
  - server=WWW server, client=browser
- Nowadays, we do not understand web services like this in general, a web service is a set of specific specifications from W3C.
- Available services: exchange rates, stock exchange, search services (Google), maps, weather.
- Components of distributed application?

Architecture of web services

- Set of protocols, http://www.w3.org/2002/ws/:
  - Message transfer – SOAP,
    - http://www.w3.org/2000/xp/Group/.
  - Description of service – WSDL,
    - http://www.w3.org/2002/ws/desc/.
  - Service discovery – UDDI.

Architecture of web services



Client which use web service

SOAP

Communication using XML messages

Web service

Service discovery

Description of service interface

Description of service interface

UDDI registry

WSDL

Reference to service description

Web Services Description Language (WSDL)

- Description of web service based on XML.

- IBM, Microsoft, nowadays W3C.

- WSDL file with definition of service interface, XML document, contains definition of:
  - Methods,
  - Parameters (data types).

Example of WSDL

```
<wsdl:definitions
    targetNamespace=" http://tempur i.org / ">
<wsdl : types>
    <s:schema elementFormDefault="qualified "
         targetNamespace=" http://tempuri.org / ">
. . .
    <s:element name="Query">
         <s:complexType><s:sequence>
              <s:element minOccurs="1" maxOccurs="1"
                   name="dbId" type="s:int" / >
              <s:element minOccurs="0" maxOccurs="1"
                   name="query" type="s:string"/>
         </s:sequence></s:complexType>
    </s:element>
. . .
```

Simple Object Access Protocol (SOAP)

- Standard protocol for messages (envelope + set of rules for data representation in XML).
- Message SOAP can be used in different protocols for example HTTP or RPC (Remote Procedure Call).
- It is copmposed from three parts:
  - envelope – define what message contains and how should be processed.
  - Set of coding rules – for exampel serialization of primitives data types for RPC or message passing through HTTP.
  - Konvention for calling remote procedures.

Simple Object Access Protocol (SOAP)

- SOAP based on XML.
- SOAP is quite simple
- It does not deal with transactions and security.
- Message contains element Envelope, which contains :
  - header – meta-information,
  - body – information.

Example SOAP 1.2, request 1/2

```
POST /AmphorAWS/AmphorAWS.asmx HTTP/1.1
Host : localhost
Content−Type: application/soap+xml;charset=utf−8
Content−Length: length

<?xml version="1.0" encoding="utf−8" ?>
<soap12:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xmlns:xsd="http://www.w3. org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/
  soap−envelope">
```

Example SOAP 1.2, request 2/2

```
<soap12:Body>
    <Query xmlns="http://tempuri.org/">
        <dbId>1</dbId>
        <query>
            doc(’books.xml’)/books/book[author/last=’Fernadez’]
        </query>
    </Query>
</soap12:Body>
</soap12:Envelope>
```

Example SOAP 1.2, response 1/2

```
HTTP/1.1 200OK
Content−Type: application/soap+xml ; charset=utf−8
Content−Length: length

<?xml version="1.0" encoding="utf−8" ?>
<soap12:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap−envelope">
```

Example SOAP 1.2, response 2/2

```
<soap12:Body>
    <QueryResponse xmlns="http://tempuri.org/">
        <QueryResult>string</QueryResult>
    </QueryResponse>
</soap12:Body>
</soap12:Envelope>
```

Universal Description, Discovery and Integration (UDDI)

- Registration and discovery of web services.
- Offers a public database (registry). Two biggest databases was managed by ~~IBM a Microsoft.~~
- UDDI regiostry contains four types of entities:
  - business entity.
  - business service.
  - binding template, description by WSDL.
  - service type.

Java web services

- Standard JavaEE web appiucation
- Definition of class:

```
@WebService(name="TestWS")
public class MyWebService {
   @WebMethod
   public String sayHallo(int nTimes){
      String ret = "";
      for(int i=0; i<nTimes; i++){
         ret += "Ahoj ";
      }
      return ret;
   }
}
```

Java web services – old way

- WEB-INF/web.xml (only JBoss server)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http… >
  <display-name>EnterpriseWeb</display-name>
  <servlet>
    <description></description>
    <display-name>Hello</display-name>
    <servlet-name>Hello</servlet-name>
    <servlet-class>webService.MyWebService</servlet-
class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/Hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Java WS - client

Client code generation:

- Eclipse UI
- Old way

<jboss-install-dir>/bin

  - wsconsume.bat -v -k -p ws -o "\EnterpriseWebClient\src"
  - wsimport.bat –keep WSDL_URI

http://localhost:8080/EnterpriseWeb/Hello?wsdl

~~http://localhost:8080/EnterpriseWeb/Hello?Tester~~

Java WS - client

- Usage of generated code in JavaSE application:

```
public class WebServiceClient {
   public static void main(String[] args){
      MyWebService ws = new

MyWebServiceLocator().getMyWebServicePort();
      String response = ws.sayHallo(5);
      System.out.println("Web service response:" +

                     response);
   }
}
```

P11

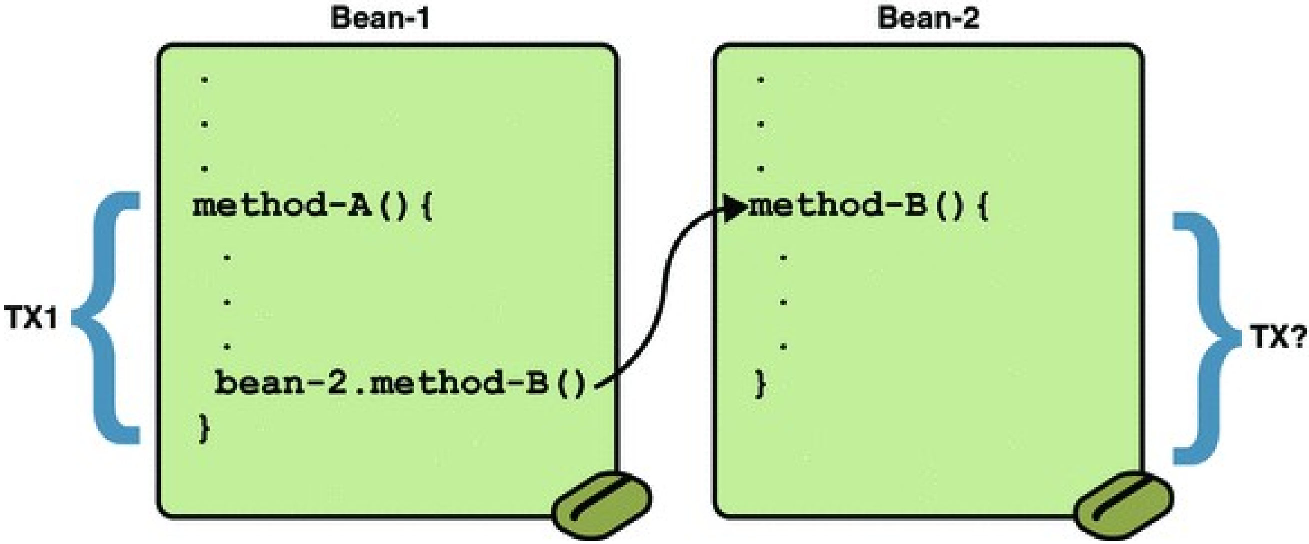
- JTA – Java Transaction
- JMS – Java Message Services
- Message-Driven Beans

- Reference: Java EE Tutorial
  - http://java.sun.com/javaee/5/docs/tutorial/doc/

JTA – Java Transaction

Conatiner managed transaction

- Inside one EJB method is not allowed more then one transactions or nested transactions.
- Transaction interface
  - Required
  - RequiresNew
  - Mandatory
  - NotSupported
  - Supports
  - Never



JTA

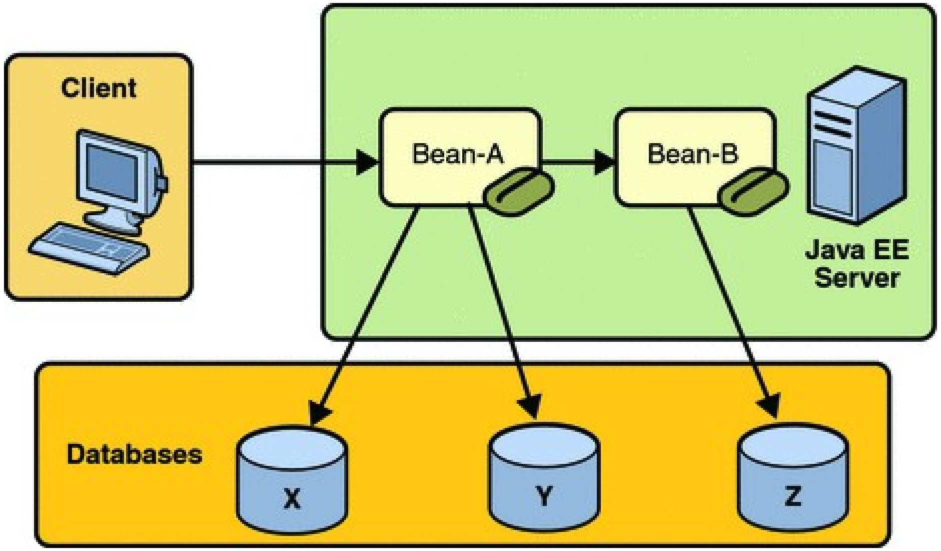
Transaction Attribute	Client’s Transaction	Business Method’s Transaction
`Required`	None	T2
	T1	T1
`RequiresNew`	None	T2
	T1	T2
`Mandatory`	None	error
	T1	T1
`NotSupported`	None	None
	T1	None
`Supports`	None	None
	T1	T1
`Never`	None	None
	T1	Error

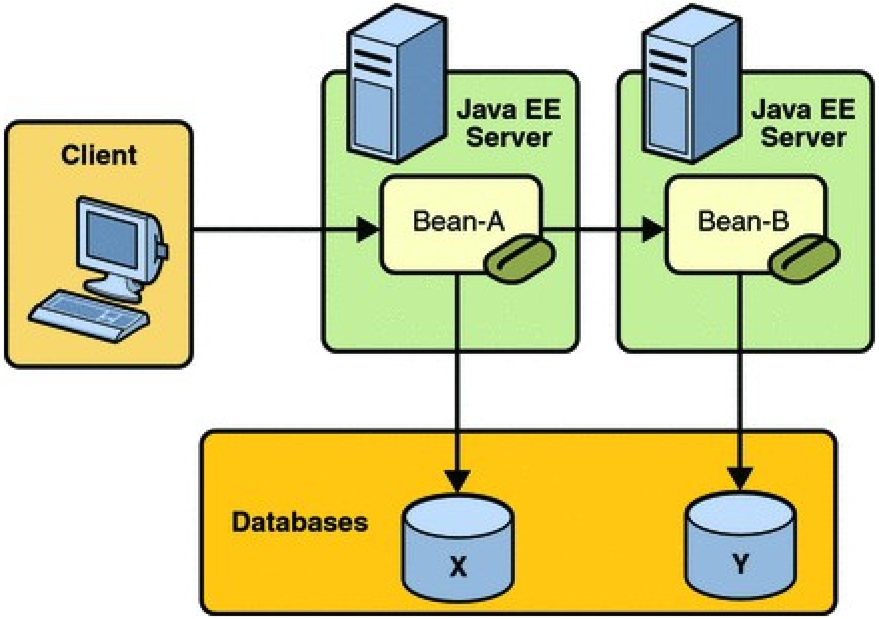
JTA

```
@TransactionAttribute(NOT_SUPPORTED)
@Stateful
public class TransactionBean{
    @TransactionAttribute(REQUIRES_NEW)
    public void firstMethod() {…}
    @TransactionAttribute(REQUIRED)
    public void secondMethod() {…}
    public void thirdMethod() {…}
    public void fourthMethod() {…}
```

```
@Resource
private SessionContext sctx;

sctx.setRollbackOnly();
```

JTA


Client
Bean-A
Bean-B
Java EE Server
Databases
X
Y
Z


Java EE Server
Java EE Server
Client
Bean-A
Bean-B
Databases
X
Y

JTA

Transaction managed by application

```
<non-jta-data-source>jdbc/pokus3</non-jta-data-source>
```

- Allowe more ten one transaction in method
- More lines of code

```
@Resource
SessionContext context;


UserTransaction utx = context.getUserTransaction();

utx.begin();
// Do work
utx.commit();
```