trainologic

JSF

trainologic

The Need for JSF

- JSF is a framework for building web applications.
- Before we go into what JSF is and how to use it, we need to discuss the motivation behind it.
- JavaEE provides several standards for building web applications: Servlets, JSP, Taglibs, JSTL, etc..
- What's wrong with building web applications on-top of these standards?

trainologic

The Need for JSF

- Well, nothing is wrong with implementing your web application using Servlets & JSP.
- However, you will most probably find yourself dealing with a lot of plumbing code dealing with your web application's infra-structure.
- Let's highlight some of the infra-structure services that are common for web applications…

trainologic

Common Infra-Structure

- Most web applications will need the following:
  - A central place (e.g., XML file) for specifying the navigation rules for your site.
  - Validation services.
  - Mapping HTTP request parameters to POJOs.
  - Conversion utilities.
  - Error management.
  - Reusable view components.
  - Life-cycle management.

trainologic

So Hard to Choose…

- There are many web frameworks in the Java eco-system. E.g.: Spring MVC, Grails, Play!, etc…
- Frameworks can be categorized intro 3 different types:
  - Request Oriented.
  - Component Oriented (JSF is here).
  - RAD.
- Let's go over these categories…

trainologic

Component Oriented

- A component oriented framework tries to bridge over the two very different paradigms:
  - The HTTP protocol (string-based).
  - Object-oriented model of the presentation in the back-end (the Swing-like model).
- Consider the mapping of a button to a function on the back-end.

trainologic

JSF

- JSF is about enabling component-based, object-oriented development of web applications using Java.
- JSF, as we are going to see, provides a framework in which there are built-in infra-structure services that deal with the aforementioned problems.
- Note that JSF is the only standard web framework under the JavaEE umbrella.

trainologic

JSF

- JSF stands for **J**ava**S**erver **F**aces.
- Part of the JavaEE specification.
- Reference implementation is the Mojarra project.
- Many component libraries exist.
- We will focus on the excellent PrimeFaces.

trainologic

Hello World

- Let’s start by exploring a simple application:
  - A page requesting first name and last name.
  - A page displaying a welcome message addressing the full name.
- We’ll have 2 views, 2 managed beans and a navigation rule.

trainologic

index.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">


<h:form>
First Name:
<h:inputText value="#{name.firstName}"></h:inputText>
<br/>
Last Name:
<h:inputText value="#{name.lastName}"></h:inputText>
<br/>
<h:commandButton action="#{helloController.saveName}" value="Submit">
</h:commandButton>
   </h:form>
</html>
```

trainologic

hello.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core">
<h:body>
Hello <h:outputText value="#{name.fullName}"></h:outputText>


</h:body>
</html>
```

trainologic

Hello World Example

```
package com.trainologic.examples;
import javax.faces.bean.ManagedBean;

@ManagedBean(name="name")
public class User {
  private String firstName;
  private String lastName;

 public String getFirstName() { return firstName; }
 public void setFirstName(String firstName) { this.firstName = firstName; }
 public String getLastName() { return lastName;}
 public void setLastName(String lastName) { this.lastName = lastName;}

 public String getFullName() {
  return firstName + " " + lastName;
 }
}
```

trainologic

Hello World Example

```
package com.trainologic.examples;
import javax.faces.bean.ManagedBean;


@ManagedBean(name="helloController")
public class HelloController {
  public String saveName() {
    return "saved";
  }
}
```

trainologic

faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
version="2.2">
  <navigation-rule>
      <from-view-id>/index.xhtml</from-view-id>
      <navigation-case>
        <from-outcome>saved</from-outcome>
        <to-view-id>/hello.xhtml</to-view-id>
      </navigation-case>
  </navigation-rule>
</faces-config>
```

trainologic

Discussion

- Note that there is no code dealing with *HttpServletRequest* and its “response” counterpart.
- No code dealing with mapping of request parameters.
- A concise way of managing navigation.
- The view files (facelets) are simple and manageable.

- Let’s understand how it works…

trainologic

One Servlet to Rule Them All

- JSF is built on top of the Servlets API.
- A JSF implementation must provide a servlet named: *javax.faces.webapp.FacesServlet*.
- All URLs that require the JSF life-cycle processing, **must** pass through this servlet.

trainologic

FacesServlet

- This Servlet is responsible for processing all of the HTTP requests sent by the client according to a well defined life cycle.
- The magic of JSF is all in this life cycle.
- We will discuss this life cycle late…

trainologic

Configuring the FacesServlet

- The following context parameters can be used to configure a JSF application (important parameters only):
  - javax.faces.CONFIG_FILES – a comma separated list of relative-to-context configuration files to load in addition to /WEB-INF/faces-config.xml (if exists).
  - javax.faces.STATE_SAVING_METHOD – default is “server” and can be set to “client”.
  - javax.faces.PROJECT_STAGE – can be set to one out of: “Development”, “UnitTest”, “SystemTest” or “Production”. (Default is “Development”).

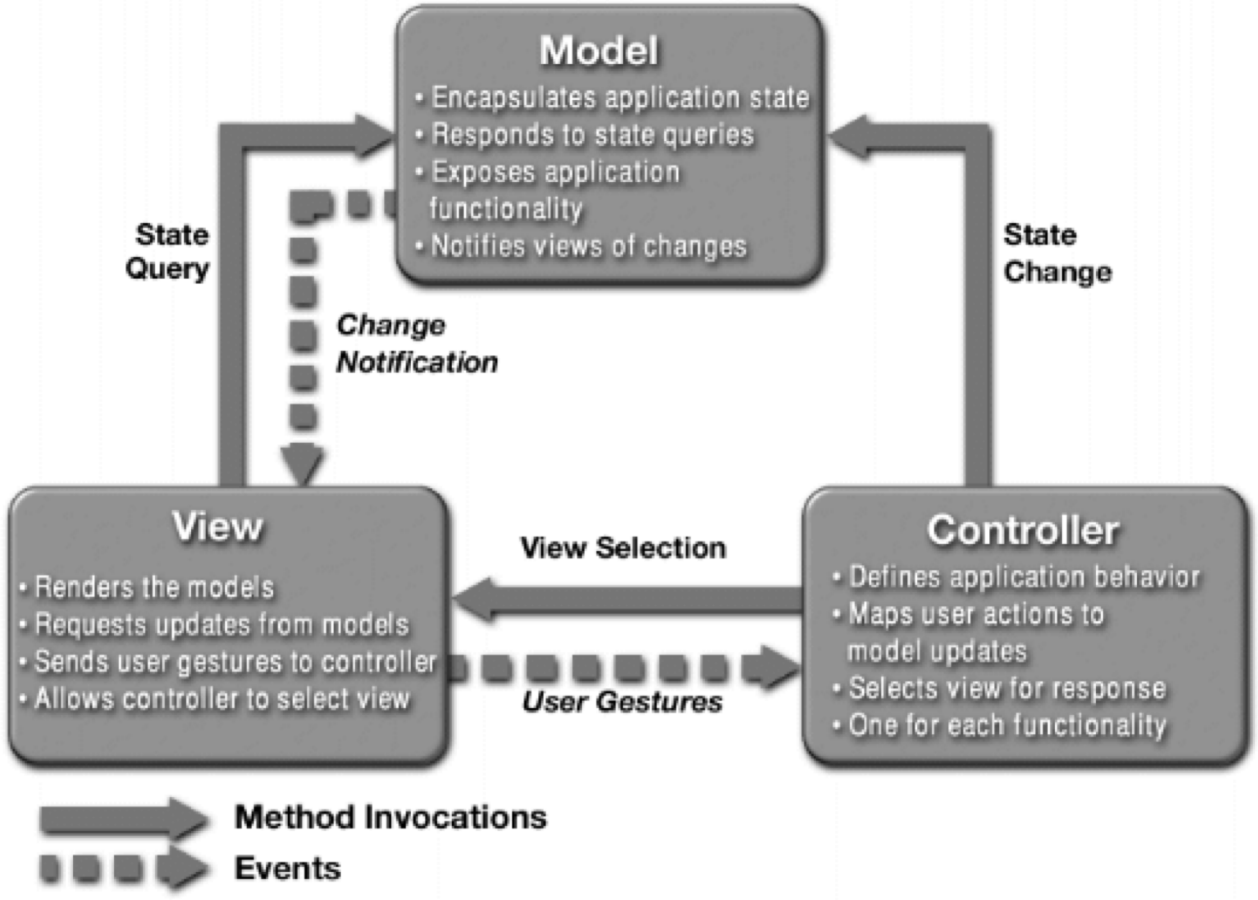
trainologic

Model-View-Controller (MVC)

- MVC is a design pattern for GUI-based application.
- This pattern provides separation between the Model (domain business logic), the View (presentation logic) and the Controller (UI event logic).
- The relationship between these decoupled components can be viewed in the following diagram…

trainologic

Model-View-Controller (MVC)



trainologic

MVC in JSF

- The Controller part in JSF:
  - The *FacesServlet*.
  - The managed beans.
- The View part in JSF:
  - JSF UI Components.
  - View handler implementations: (Facelets, JSP).
- The Model part in JSF:
  - The non-JSF business layer (e.g.: EJBs).

trainologic

The UI Components

- JSF allows you to build you web application in an object-oriented way.
- I.e., JSF provides the glue between the HTTP protocol (and the HTML representation) and the objects in our application.
- Now we are going to see how to work with those JSF objects (known as UI components) in order to develop a web UI.

trainologic

Component Hierarchy

- Every UI page in JSF is to be considered as a tree of components.
- Let's review our 'hello world' example:

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h=http://java.sun.com/jsf/html xmlns:f="http://java.sun.com/jsf/core">
<h:form>
First Name:
<h:inputText value="#{name.firstName}"></h:inputText><br/>
Last Name:
<h:inputText value="#{name.lastName}"></h:inputText><br/>
<h:commandButton action="#{helloController.saveName}" value="Submit">
</h:commandButton>
   </h:form>
</html>
```

trainologic

Component Hierarchy

- Highlighted are he JSF UI components that make the hierarchy of this page:

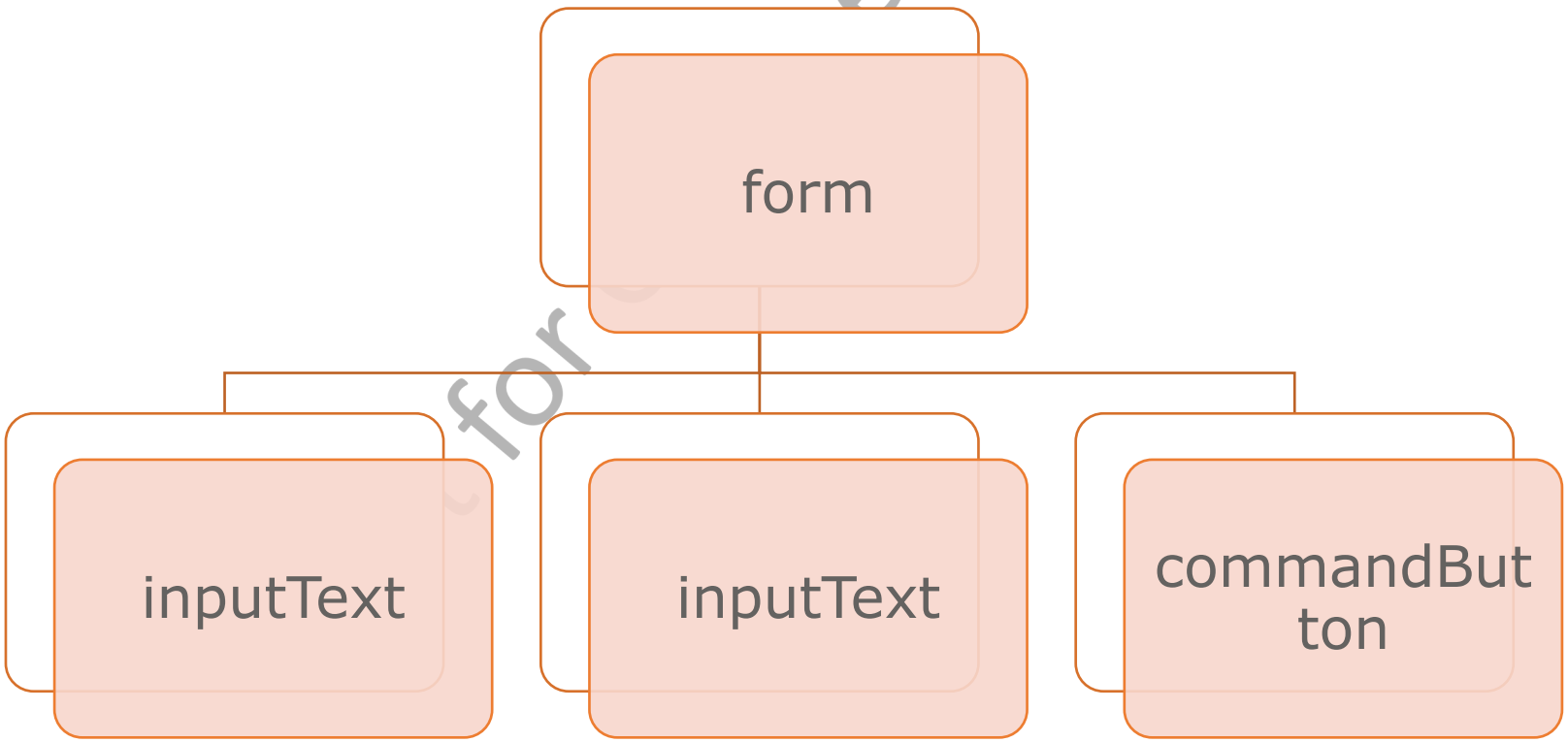
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">


<html xmlns=http://www.w3.org/1999/xhtml
xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h=http://java.sun.com/jsf/html xmlns:f="http://java.sun.com/jsf/core">
<h:form>
First Name:
<h:inputText value="#{name.firstName}"></h:inputText><br/>
Last Name:
<h:inputText value="#{name.lastName}"></h:inputText><br/>
<h:commandButton action="#{helloController.saveName}" value="Submit">
</h:commandButton>
   </h:form>
</html>
```

trainologic

The Hierarchy

- Following is a diagram of the JSF components' hierarchy of our example:



trainologic

Component Bindings

- From the example, we can see that there is some sort of special notation in the Facelets file regarding the JSF components.
- This is the binding between the UI components and the managed beans.
- The special notation is the JSF Expression Language (EL).

- Let's get to know managed beans…

trainologic

Managed Beans

- An important layer in JSF.
- Provide the ability to configure the component tree by using POJOs.
- The binding between the UI components and the managed beans is done by using EL.
- In the discussed example, there is a binding between the *inputText* components and the properties of a managed-bean called ‘name’.
- The *commandButton* is bound to a method invocation of another managed-bean.

trainologic

Defining a Managed Bean

- Managed beans are defined by using the @ManagedBean annotation.

- Accepts the following attributes:
  - name – the name of the managed bean.
  - eager – a Boolean value which defaults to *false*. If it is *true,* then only if the scope of the managed bean is “application” the bean must be instantiated on application startup.

- Scope?!?

trainologic

Scope

- A managed bean can be attached to one of the following scopes:
  - application – a single instance will be shared across the entire *ServletContext*.
  - session – an instance per-session will be created (HttpSession level).
  - request – an instance per-request will be created (HttpServletRequest level).
  - view – an instance will be bound for the *view*.
  - none – a new instance will be returned and not be bound to any scope.
  - custom – explained later.

trainologic

Scopes

- The following annotations control the scope:
  - @ApplicationScoped.
  - @CustomScoped.
  - @NoneScoped.
  - @RequestScoped.
  - @SessionScoped.
  - @ViewScoped.

trainologic

Managed Bean Life-cycle

- When a managed bean is referenced by an EL expression, the relevant scope is searched for the bean.
- If not found, the managed bean will be instantiated and bound to the scope.
- You can use the standard *@PostConstruct* annotation to specify initialization method (*void* and no-args).
- When the scope ends, the bean is being unbound and a *@PreDestroy* method is invoked (if exists).
- Not for scope = none.

trainologic

The Request Scope

- JSF is build on top of the Servlet specification.
- I.e., for every request that JSF handles, there is an *HttpServletRequest* instance behind the scene.
- When a managed bean with the scope “request” is created, it will be stored as an attribute (not parameter) under the same name.
- When the request ends, the instance will be unbound.

trainologic

The Session Scope

- When a “session” scoped managed bean is created, it will be stored in an attribute by the same name on the *HttpSession*.
- Note that for large amounts of session-scoped beans, there will be pressure on the GC.

trainologic

The View Scope

- When a “view” scoped managed bean is created, it will be stored in the *viewMap* property of the *UIViewRoot* object.
- UIViewRoot is the root of the component tree.
- It’s state storage is configuration dependent. Can be either ”server” or “client”.

trainologic

The Application Scope

- A global bean.
- Stored as an attribute under the *ServletContext* object.
- **Must be thread-safe**.

- You were warned!

trainologic

The None Scope

- Useful for creating dynamic object trees composed of managed beans.
- For example, creating a *Company* managed bean that has *ceo* and *cto* managed beans of type *Employee*.
- Wait. How can we reference from one bean to another?

trainologic

@ManagedProperty

- Allows for Dependency Injection of one managed bean (or its property) into another managed bean.
- The *value* attribute can accept either a literal value or an EL value.

trainologic

The View Technology

- When JSF was introduced, the most popular ‘view’ technology to use with it was JSP.
- That made sense as it was the only JavaEE standard one.
- However, that introduced some problems.
- JSF needs to use the ‘view’ to build the component tree.
- However, the only way to ”evaluate” a JSP is to go through the JSP/Servlet lifecycle.
- I.e., JSF can’t interact directly with JSP. It has to go through the Servlet container.

trainologic

Facelets

- That was the main reason for developing Facelets.
- The most popular view technology for JSF.
- Based on XHTML syntax.
- Doesn’t required a Servlet container to process it.
- Parsed by JSF in order to build the component tree.

trainologic

Facelets

- Facelets supports JSF tag libraries through XML namespace definitions.
- The following tag libraries are supported:
  - JSF Facelets – (ui:).
  - JSF HTML – (h:).
  - JSF Core – (f:).
  - Pass-through Elements – (jsf:).
  - Pass-through Attributes – (p:).
  - Composite Component – (cc:).
  - JSTL Core – (c:).
  - JSTL Functions – (fn:).

trainologic

Facelets Templates

- Facelets provides a templating mechanism.
- Allows to easily build a reusable consistent UI.
- A ui:composition tag can specify a base template file.
- Everything outside this tag is ignored (overridden by the template).
- The ui:composition tag will usually include ui:define elements that will provide the specific content to pass to the template.

trainologic

Facelets Templates

- In the template file itself, you can display provided content with the ui:insert tag.

- You can, instead of ui:composition, use the ui:decorator tag which doesn’t not disregard the surrounding content.

trainologic

Facelets Templates

Exercise

trainologic

The Expression Language

- The EL (Expression Language) is a simple language designed for the presentation tier of web applications.
- The JSF specification uses the Unified Expression Language defined as part of the JSP 2.1 specification.
- Wait, didn't we say that JSF doesn't depend on JSP as the view technology? (Usually its Facelets).
- In order to understand this relationship, let's discuss the history of EL…

trainologic

History of the EL

- The EL was first introduced as part of the JSTL specification (1.0) combined with JSP 1.2.
- It was usable only for attributes of JSTL tags.
- It was later adopted as part of the JSP specification (2.0) and was usable across an entire JSP page.
- When JSF was created, it branched the EL to a custom tailored solution that will support the JSF needs (for example, method expressions).

trainologic

History of the EL

- This resulted in an absurd situation, in which JSP authors who also use JSF are required to use two variants of the EL.
- It was fixed with JSP 2.1 and JSF 1.2 with the introduction of the Unified EL.
- Suitable for both JSP and JSF.

- Let's learn how to use it…

trainologic

EL Sytax

- The EL specification describes two forms for EL expressions:
- **${*expr*}** – also called immediate expression.
- **#{*expr*}** – also called deferred expression.

- In JSF we almost always use the deferred syntax (evaluated only when needed).

trainologic

Literals

- The following literals are supported in EL:
  - Boolean: *true* and *false*.
  - Integer: like in Java.
  - Floating point: 4.3E4.
  - Strings: double or single quoted.
  - Null: *null*.

trainologic

Operators

- Arithmetic:
  - +, -, *, /, div, %, mod.
- Relational:
  - ==, eq, !=, ne, <, lt, >, gt, <=, le, >=, ge.
- Logical:
  - &&, and, ||, or, !, not.
- Misc:
  - empty, ternary (?:), instanceof.

trainologic

Operator Duality

- As you can see, there are many operators for which there is more than one way to express them.
- This was defined in order to provide consistency with JavaScript and XPath.

trainologic

Implicit Objects

- The following implicit objects are provided in the context of JSF:
  - view – returns the *UIViewRoot*.
  - facesContext – returns the *FacesContext* instance.

trainologic

Method Bindings

- EL expressions can be evaluated to method bindings.
- I.e., bound to managed beans methods.
- Method bindings are only applicable where they are expected and you can’t use other types of EL in those places.
- For example, we have seen the *action* attribute of the <h:commandButton> component receiving an EL method binding.

trainologic

Exercise

Exercise

trainologic

JSF Lifecycle

- As mentioned before, the *magic* of JSF is in its lifecycle.
- Every request that is handled by the JSF framework goes through a well defined lifecycle.
- Also known as the JSF Request Processing Lifecycle.
- Crucial to the understanding of how JSF works.

trainologic

JSF Lifecycle Types

- Before we discuss the different request processing phases, it is important to note that there are 3 lifecycle types:
  - Non-Faces request that generates a Faces response.
  - Faces request that generates a Faces response.
  - Faces request that generates a non-Faces response.

trainologic

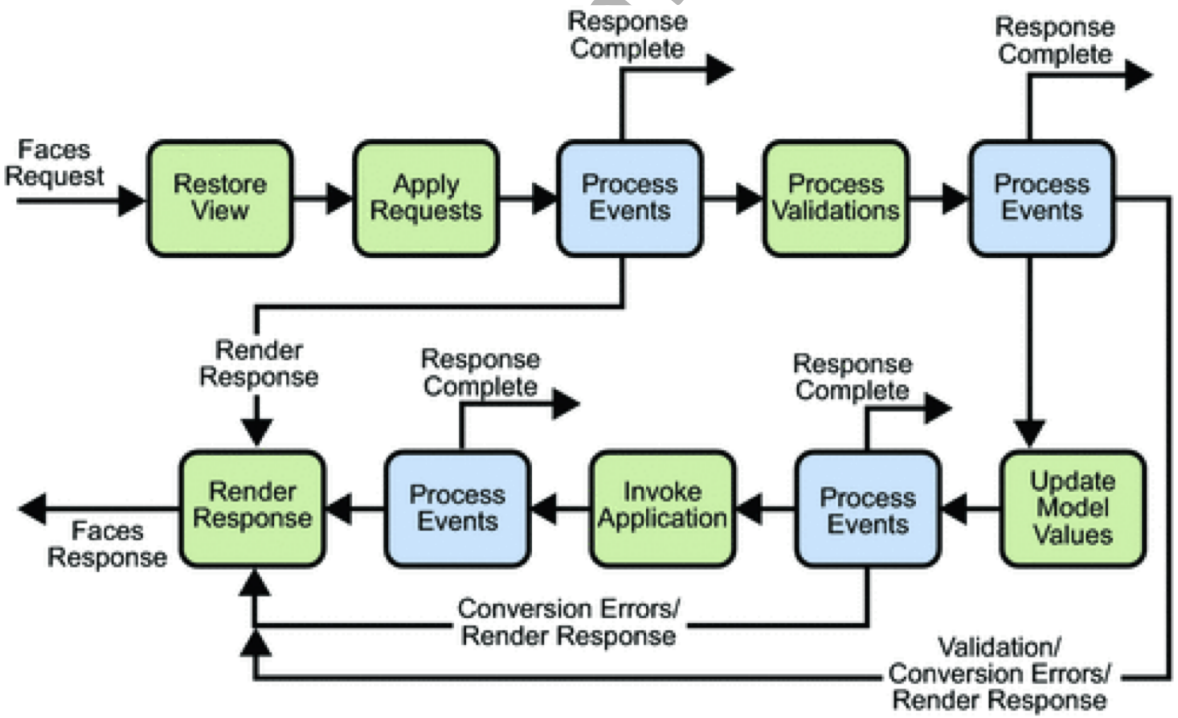
Non-Faces Request

- A request that was not generated by a JSF page.
- Have the following lifecycle:
  - First, a component tree is created according to the specified view.
  - Next the Render Response phase is invoked.

trainologic

Request Processing Phases

- A Faces request goes through several well-defined phases that constitute the request processing lifecycle:



trainologic

Restore View

- The first phase that a JSF request goes through is the Restore View phase.
- In this phase, the JSF framework will restore the component tree (the view) from which the client's request has arrived.
- I.e., the last state that we knew about.

- But, where is the view stored?

trainologic

Restore View

- The object in charge of the view-state management is: *StateManager*.
- JSF implementations must support saving the view-state on either the client-side or the server-side.
- Saving the state on the client-side will usually be done by the use of hidden fields.

trainologic

Apply Request Values

- In this phase, each JSF component will update its value from the request.
- The components can access request parameters, headers and cookies.
- The components are said to have ‘local values’ after this phase.

trainologic

Process Validations

- In this phase, validation logic is being executed.
- We will discuss how to use the built-in validation mechanisms, how to create our own validators and the relationship to the standard Bean Validation specification.
- For now, it is important to understand that the validations are performed **before** the ‘Update Model Values’ phase.
- I.e., only validated data will be used in the next phase.

trainologic

Update Model Values

- After the validations were performed, the JSF UI components update their associated managed beans.
- In this phase, each component is setting the relevant properties of the bound managed beans.
- Note that only valid values are stored in the managed beans.

trainologic

Invoke Application

- Some components, like *commandButton,* can be associated with an action.
- An action is a method to invoke when the button is pressed.
- In the “Invoke Application” phase, this action is invoked.
- The return method of this action-method represents the next ‘view’ to display.
- The navigation mechanism uses this string to decide which ‘view’ will be presented next.

trainologic

Render Response

- After the next view to display has been selected by the navigation mechanism, this phase is responsible for building the component tree of the view and rendering it.
- In case that there were errors in the previous phases, it will render the current view.

trainologic

Render Response

- At any phase of the lifecycle, it is possible to invoked *FacesContext.renderResponse()*.
- In case that this method is invoked, the rest of the lifecycle is skipped and the ’Render Response’ phase is taking place instead.
- It is also possible to invoke *FacesContext.responseComplete()*.
- In this case, the lifecycle ends and ‘Render Reponse’ phase will not happen.

trainologic

Phase Listeners

- You can implement the *PhaseListener* interface if you want to receive before/after phase events.
- Following are the methods you need to implement:
  - *getPhaseId()* – returns the id of the phase that the handler is interested in. the special value ANY_PHASE receives events for all the phases.
  - *beforePhase*() – handler logic to be invoked before the phase.
  - *afterPhase().*

trainologic

Discussion

- *PhaseListener* implementations must be thread safe.
- JSF guarantees that if the *beforePhase()* method was invoked, the respective *afterPhase()* method will be invoked regardless of exceptions that may have occurred.

trainologic

Exercise

Trace Phases

trainologic

Navigation

trainologic

The Navigation Mechanism

- Navigation in JSF is orchestrated by a single instance of a class extending *NavigationHandler*.
- JSF implementations are required to provide a default implementation that is based on application configuration files.

trainologic

Converters

- Converters are responsible for string-to-object and object-to-string mappings.
- Implementations need to implement the JSF *Convert<T>* interface.

- You can look at the *javax.faces.converter* package for built-in converter implementations.

trainologic

Using Converters

- The use a custom converter, you have to go through:
  - Implement the *Converter* interface.
  - Register the implementation.
  - Optionally, bind the converter.

trainologic

Registering Converters

- Registering converters is done through an *addConverter* method of the *Application* class.
- We have two variants:
  - Bind by converter id.
  - Bind by class.

- Use the second for implicit converters.

trainologic

Explicit Converters

- *UIOutput* and *ValueHolder* supports registering converters.
- You have support for that in the “f:” tag library.

trainologic

Tag Library

- You have the *<f:convertDateTime>* which registers a *DateTimeConverter* with the surrounding component.
- The following attributes are available for this tag:
  - dateStyle – **default**, short, medium, long, full.
  - locale.
  - pattern.
  - timeStyle – **default,** short, medium, long, full.
  - timeZone.
  - type – date, time, both.

trainologic

Tag Library

- There is also a *<f:convertNumber>* which registers a *NumberConverter* with the surrounding component.
- The following attributes are available for this tag:
  - currencyCode, currenySymbol.
  - groupingUsed – **true**, false.
  - integerOnly – true, **false**.
  - locale, maxFractionDigits, minFractionDigits, maxIntegerDigits, minIntegerDigits.
  - pattern.
  - type – **number**, currency, percent.

trainologic

Exercise

Implement a java.awt.Color Converter