

It's Just A View

An Introduction To Model • View • Controller In XPages

Ulrich Krause, BCC GmbH

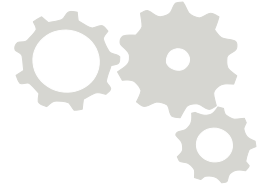
Ghent, Belgium, March 30-31, 2015

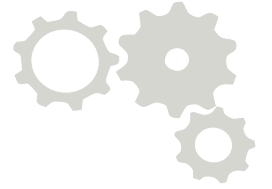
About: Ulrich Krause

- Administrator /Developer since 1993
- Senior Software Architect at BCC, Germany



- OpenNTF Contributor
- IBM Champion (2011 – 2015)
- Blog <http://www.eknori.de>
- Twitter @eknori
- Mail ulrich.krause@eknori.de





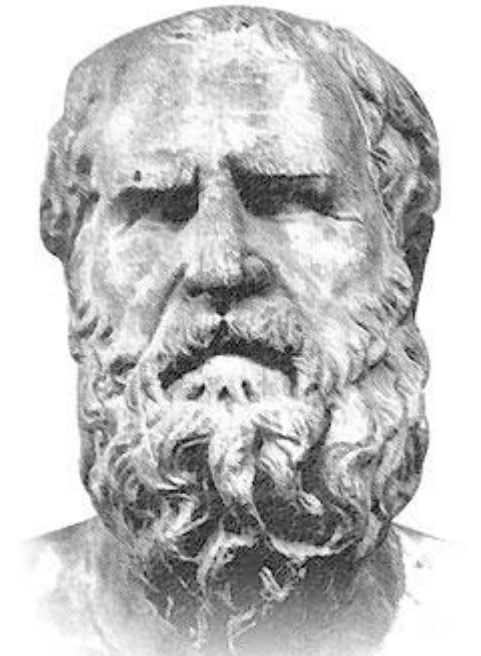
Agenda

- The constant in live and software development
- Software Quality / Maintenance
- Design Patterns
- The Basics of MVC
- Example

Headline

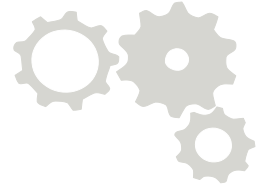
- The only **constant** in **live** is

CHANGE



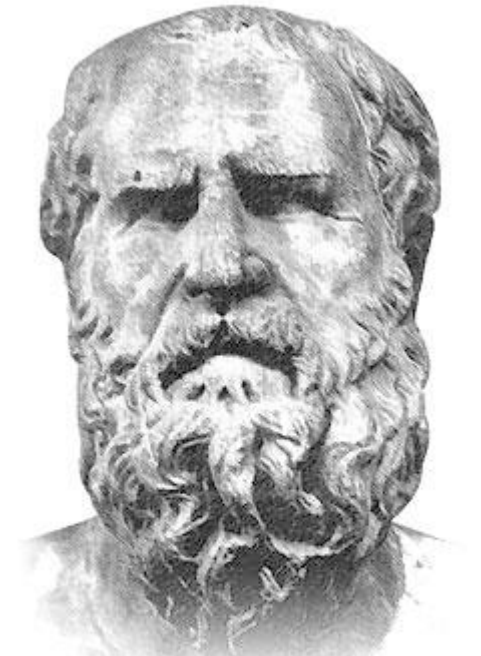
Heraclitus (520 - 460 BC)

Headline



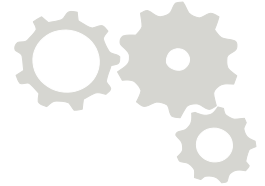
- The only **constant** in software development is

CHANGE

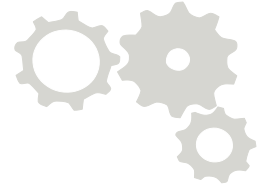


Heraclitus (520 - 460 BC)

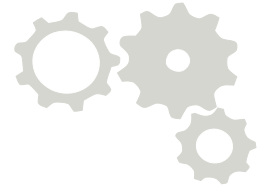
Softwarequality - Overview

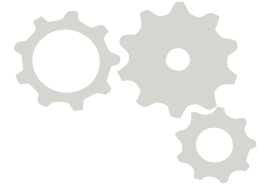


Softwarequality – User Perspective



Softwarequality – Developer Perspective

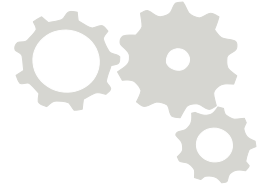




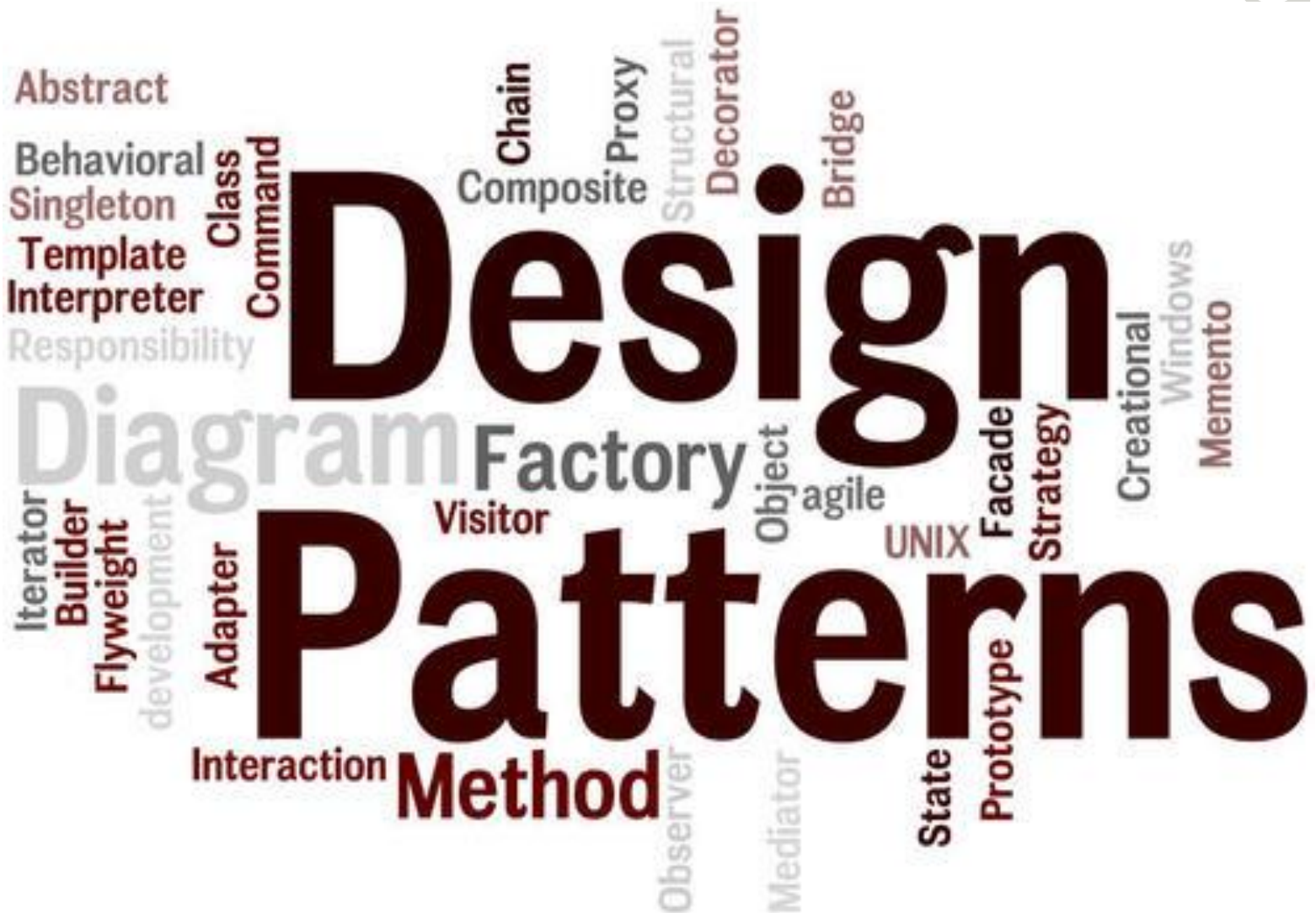
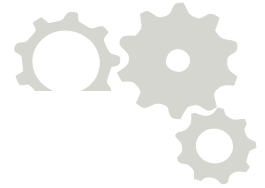
Software Quality / Maintenance

- Challenges with „historically grown“ applications
 - Code can be everywhere in the application
 - Forms, View Events, Buttons, Hotspots, Script Libraries ...
 - Use of different languages
 - LotusScript, @Formula, Javascript, Java, Simple Actions, HTML ...
 - Redundancies
- One possible solution is to separate the Frontend from the Backend Code.
 - NotesDocument
 - NotesUIDocument

Software Quality – Use The Right Tools



Design Patterns



What are Design Patterns?



- Recurring solutions to software design problems you find again and again in real-world application development.
- A general reusable solution to a commonly occurring problem in software design.
- It is a description or template for how to solve a problem that can be used in many different situations.
- Are about design and interaction of objects, as well as providing a communication platform concerning elegant, reusable solutions to commonly encountered programming challenges.
- **GoF** Patterns are considered the foundation of all design patterns.

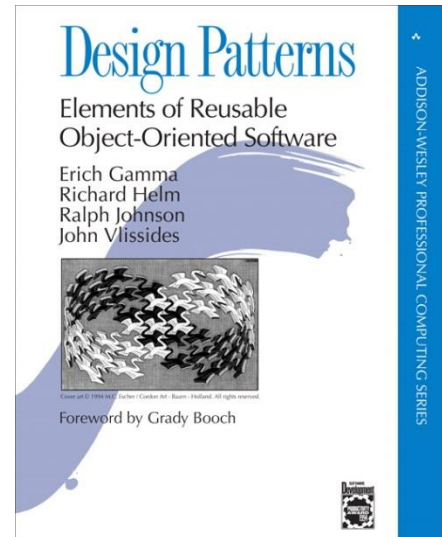
Design Patterns are NOT

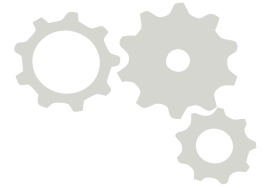


- A design pattern is not a **finished design** that can be transformed directly into source or machine code
- A design pattern is not a **code snippet** that can be copied into your code.

Gang Of Four - GoF

- Ralph Johnson, Erich Gamma, Richard Helm, John Vlissides





The 23 GoF Design Patterns

C

Abstract Factory

S

Adapter

S

Bridge

C

Builder

B

Chain of Responsibility

B

Command

S

Composite

S

Decorator

S

Facade

C

Factory Method

S

Flyweight

B

Interpreter

B

Iterator

B

Mediator

B

Memento

C

Prototype

S

Proxy

B

Observer

C

Singleton

B

State

B

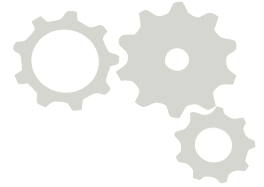
Strategy

B

Template Method

B

Visitor



Brief History Of MVC

- MVC was one of the first works to describe and implement software constructs in terms of their responsibilities.
- Trygve Reenskaug introduced MVC in the 1970s
- In the 1980s, Jim Althoff and others implemented a version of MVC .
- MVC was expressed as a general concept, in a 1988 article.

**A Cookbook for Using the Model-
View-Controller User Interface
Paradigm in Smalltalk-80**

Glenn E. Krasner
Stephen T. Pope

Father Of MVC

- **Trygve Mikkjel Heyerdahl Reenskaug** (born 1930) is a Norwegian computer scientist and professor emeritus of the University of Oslo.
- He formulated the model-view-controller (MVC) pattern for Graphic User Interface (GUI) software design in 1979 while visiting the Xerox Palo Alto Research Center (PARC).

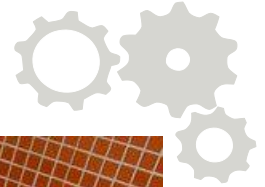


Why MVC? – An Example Project Case



- You created a superhero web application/website for a comic shop owner with a small database table.
- it is a huge success and your client is extremely satisfied.
- They ask you to change the application, they want to use a different database and, according to market demand, they definitely need both iPhone and Android apps.
- Now repeat this five times.
- The client keeps on asking for modifications and expansions.
- These can be UI related changes and even complete backend architecture .

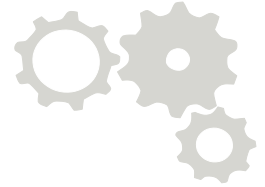
It's Official, We're In Deep Doo-Doo Now ...



IT'S OFFICIAL:
WE'RE IN
DEEP
DOO-DOO
NOW

However if you used MVC from the start

- ... you'd notice that some things would have been less painful
- And you'd been happier

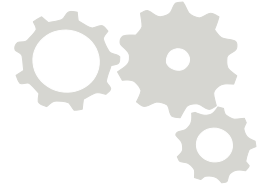


Why ?



- **90%** of the code for the web application and the mobile app will be the same, but instead of saving the user data to a Shared Object or through a web service, you'd be using a local DB for instance.
- Without **MVC**, chances are pretty high you'll be making modifications in a bunch of classes.
- The same applies to the **UI** for instance. Only the way it's presented to the user is different.

MVC Components



Model

View

Controller

Understanding MVC

The **Model** represents your data structure. Typically your model class will contain functions to retrieve, insert, and update information in the datastore

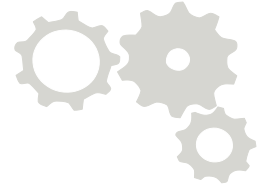


Model

View

Controller

Understanding MVC



Model



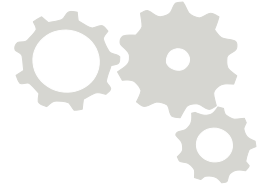
View



Controller

The **View** is the information that is being presented to the user. A View will normally be a web page, but can be any other type of "page"

Understanding MVC



Model

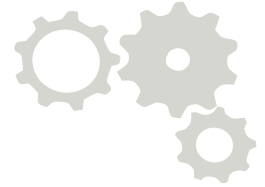


View



Controller

The **Controller** servers as an intermediary between the **Model**, the **View** and any **other resources** needed to process HTTP requests and generate a web page

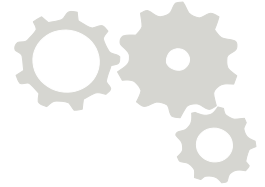


An easy way to understand MVC

- The **Model** is the data,
- The **View** is the window on the screen,
- And the **Controller** is the glue between the two



MVC Interaction



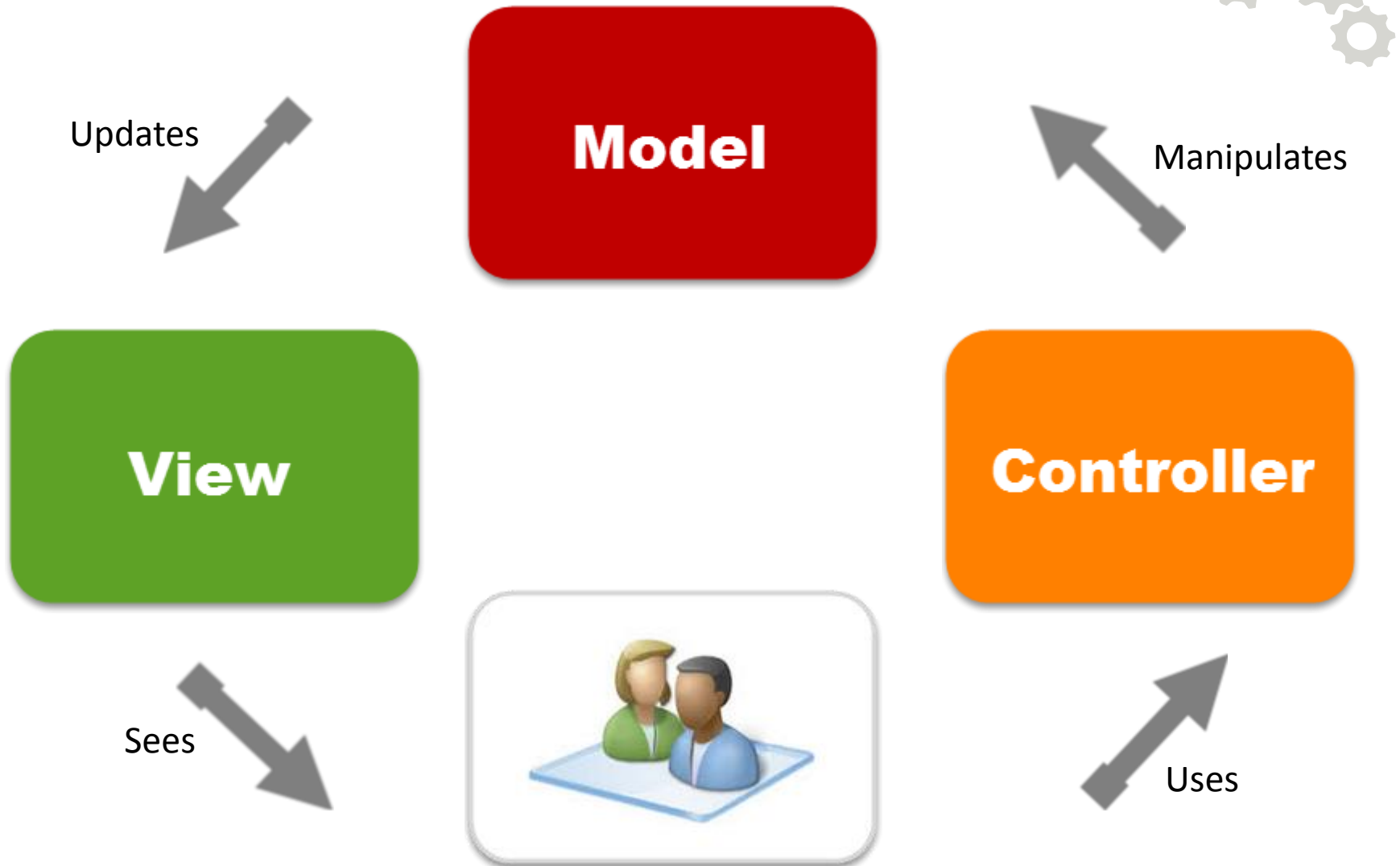
Model

View

Controller



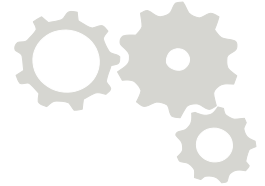
MVC Interaction





SHOW ME, DON'T TELL ME

Sample Application



Superheroes

Overview



Add Superhero

Name	Species	Description
Superman	Alien	Rocketed to Earth as an infant from the doomed planet Krypton, Kal-El was adopted by the loving Kent family and raised in America's heartland as Clark Kent. Using his immense solar-fueled powers, he became Superman to defend mankind against all manner of threats while championing truth, justice and the American way!
Hulk	Human	After being bombarded with a massive dose of gamma radiation while saving a young man's life during the testing of an experimental bomb, Dr. Robert Bruce Banner was transformed into the Incredible Hulk: a green behemoth who is the living personification of rage and pure physical strength. He has currently taken on a new persona: Doc Green.
Spiderman	Human	Peter Parker was bitten by a radioactive spider as a teenager, granting him spider-like powers. After the death of his Uncle Ben, which he could have prevented, Peter learned that "with great power, comes great responsibility." Swearing to always protect the innocent from harm, Peter Parker became the Amazing Spider-Man!
Batman	Human	Bruce Wayne, who witnessed the murder of his billionaire parents as a child, swore to avenge their deaths. He trained extensively to achieve mental and physical perfection, mastering martial arts, detective skills, and criminal psychology. Costumed as a bat to prey on criminals's fear, and utilizing a high-tech arsenal, he became the legendary Batman.

Sample Application



Heroes Of The Universe

Green Lantern



Superman



eknori



Batman




Hulk



Spiderman



 New

Heroes Of The Universe

Species

Alien

Name

Superman

Superpower

immense solar-fueled powers

Description

Rocketed to Earth as an infant from the doomed planet Krypton, Kal-El was adopted by the loving Kent family and raised in America's heartland as Clark Kent. Using his immense solar-fueled powers, he became Superman to defend mankind against all manner of threats while championing truth, justice and the American way!

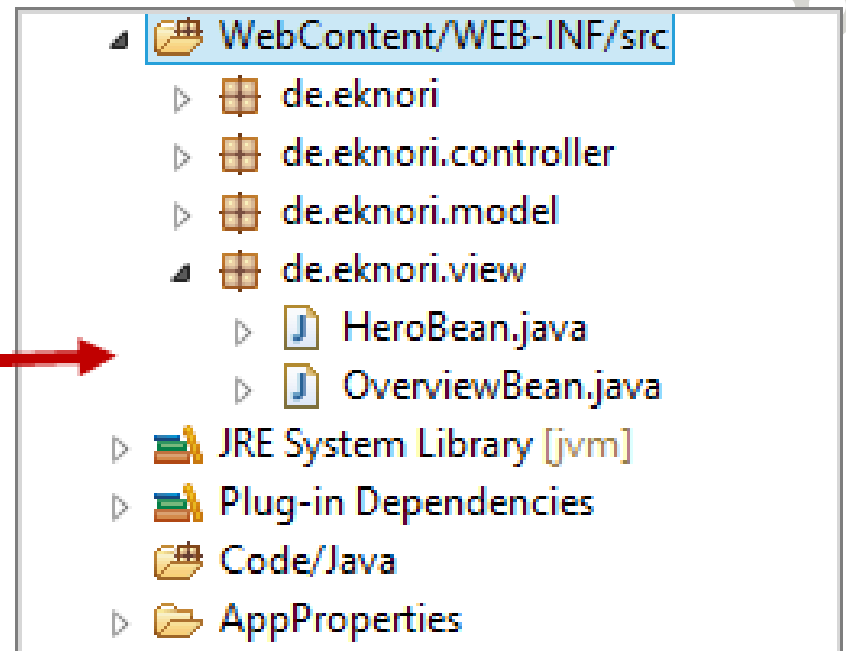
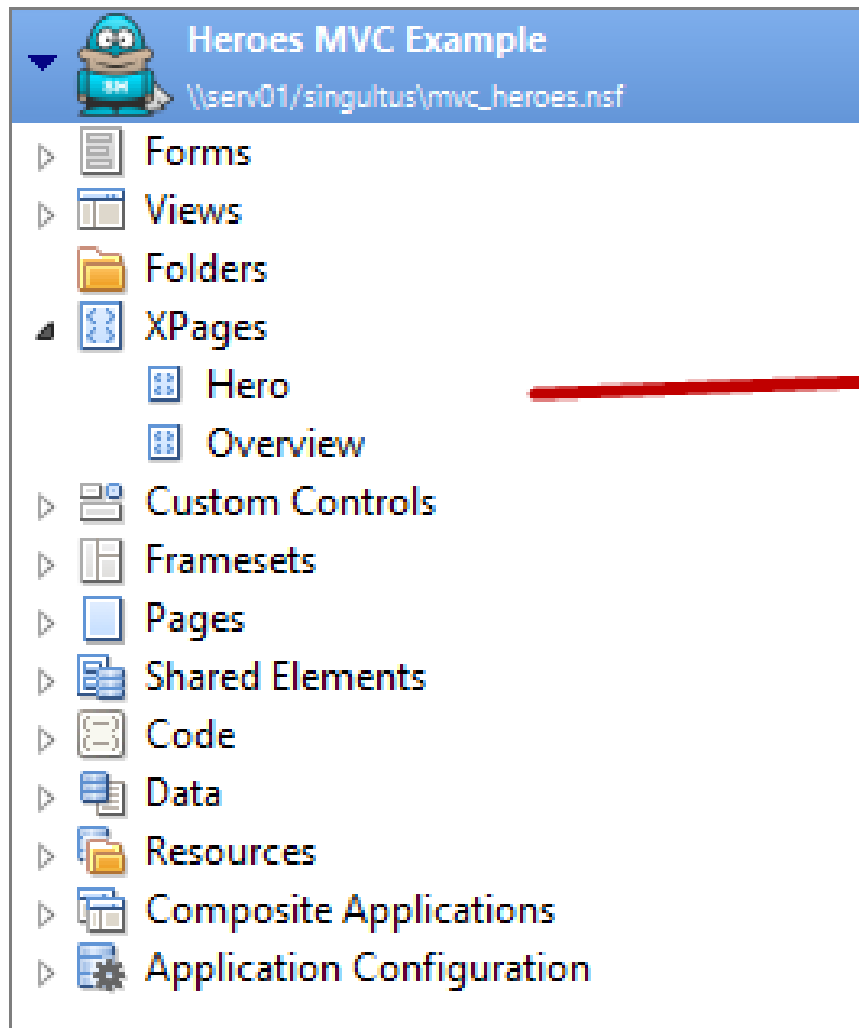
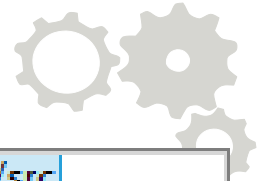
Back

Save



Demo

Sample Application Design



The Controller



```
package de.eknori.controller;

import java.util.Map;

public class AppController implements PhaseListener {
    private static final long serialVersionUID = 1L;

    public void beforePhase(PhaseEvent phase) {
        String url = UtilsJsf.getXSPContext().getHistoryUrl(0);
        String page = Utils.strLeft(
            url.substring(url.lastIndexOf("/") + 1, url
                .length()), ".");

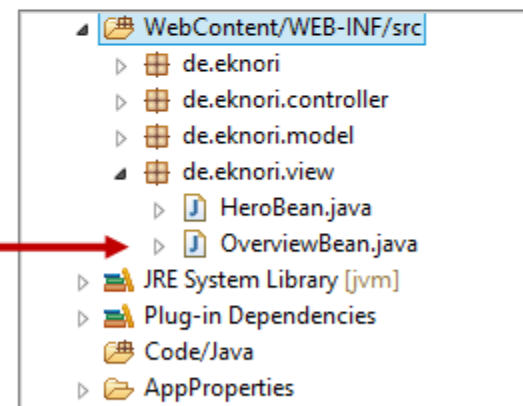
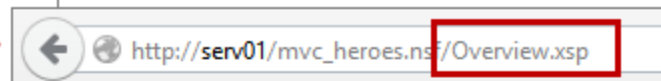
        String packageName = "de.eknori.view";
        String className = page + "Bean";
        Class<?> cl = null;
        Object pb = null;
        try {
            cl = Class.forName(packageName + "." + className);
            pb = cl.newInstance();

        } catch (Exception e) {
        }
        Map<String, Object> viewScope = UtilsJsf.getViewScope();
        viewScope.put("page", pb);

    }

    public void afterPhase(PhaseEvent phase) {
    }

    public PhaseId getPhaseId() {
        return PhaseId.RENDER_RESPONSE;
    }
}
```



View Component - Overview.xsp



```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core" xmlns:xc="http://www.ibm.com/xsp/custom">
  <xc:Layout>
    <table class="squad">
      <xp:repeat id="heroTable" value="#{page.allHeroes}" var="hero">
        <xp:tr>
          <td class="col1">
            <xp:link escape="true" text="#{javascript:page.getDisplayText(hero)}"
              value="#{javascript:page.getUrl(hero)}" />
          </td>
          <td class="col2">
            <xp:button id="btnDelete" styleClass="eknori-btn-small btn-del">
              <xp:eventHandler event="onclick" submit="true" refreshMode="complete">
                <xp:this.action><![CDATA[#{javascript:page.clickRemove(hero)}]]>
              </xp:this.action>
            </xp:eventHandler>
            <span class="fa fa-trash-o fa-fw eknori-btnIcon" />
          </xp:button>
        </td>
      </xp:tr>
    </xp:repeat>
  </table>
  <div class="btn-bar">
    <xp:button id="btnNew" value="New" styleClass="eknori-btn btn-new">
      <xp:eventHandler event="onclick" submit="true" refreshMode="complete">
        <xp:this.action><![CDATA[#{javascript:page.clickNew()}]]>
      </xp:this.action>
    </xp:eventHandler>
    <span class="fa fa-file-o fa-fw eknori-btnIcon" />
  </xp:button>
</div>
</xc:Layout>
</xp:view>
```

View Component - Hero.xsp



```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core" xmlns:xc="http://www.ibm.com/xsp/custom">
  <xc:Layout>
    <div> <xp:label value="Species" for="species" /> </div>
    <xp:inputText id="species" value="#{page.model.species}" />
    <div> <xp:label value="Name" for="inputName" /> </div>
    <xp:inputText id="inputName" value="#{page.model.name}" />
    <div> <xp:label value="Superpower" for="inputPower" /> </div>
    <xp:inputText id="inputTPower" value="#{page.model.power}" />
    <div> <xp:label value="Description" for="inputDescription" /> </div>
    <xp:inputTextarea id="inputDescription" value="#{page.model.description}" />

    <div>
      <xp:button id="btnBack" value="Back" styleClass="eknori-btn btn-back">
        <xp:eventHandler event="onclick" submit="true"
          refreshMode="complete" immediate="true" save="false">
          <xp:this.action><![CDATA[#{javascript:page.clickBack()}]]></xp:this.action>
        </xp:eventHandler> <span />
      </xp:button>

      <xp:button id="btnSave" value="Save" styleClass="eknori-btn btn-save">
        <xp:eventHandler event="onclick" submit="true"
          refreshMode="complete" save="true">
          <xp:this.action><![CDATA[#{javascript:page.clickSave()}]]></xp:this.action>
        </xp:eventHandler> <span />
      </xp:button>
    </div>
  </xc:Layout>
</xp:view>
```

Model Component: Overview

```
package de.eknori.model;

import java.io.Serializable;

public class Overview implements Serializable {
    private static final long serialVersionUID = 1L;

    public Overview() {
    }


    public List<Hero> getAllHeroes() throws Exception {
        return getDao().getAllHeroes();
    }


    public Hero getHeroById(String id) throws Exception {
        return getDao().getHeroesById(id);
    }


    public Hero createHero() throws Exception {
        Hero model = getInstanceOfHero();
        model.generateNewId();
        return model;
    }

    public Hero getInstanceOfHero() {
        return new Hero(this);
    }

    private DaoI dao = null;
    public DaoI getDao() throws Exception {
        if (dao == null) {
            dao = new DaoNsf(this);
        }
        return dao;
    }
}
```



 de.eknori.model

 **Overview**

- createHero(): de.eknori.model.Hero
- getAllHeroes(): java.util.List
- getDao(): de.eknori.model.DaoI
- getHeroById(String): de.eknori.model.Hero
- getInstanceOfHero(): de.eknori.model.Hero
- Overview(): void

Model Component: Hero

```
package de.eknori.model;

import java.io.Serializable;
import java.util.UUID;

public class Hero implements Serializable {
    private static final long serialVersionUID = 1L;

    private final Overview app;


    private String id = null;
    private String species = null;
    private String name = null;
    private String description = null;
    private String power = null;

    public Hero(Overview app) {
        this.app = app;
    }

    protected Overview getApp() {
        return app;
    }

    public String getName() {
        return name;
    }

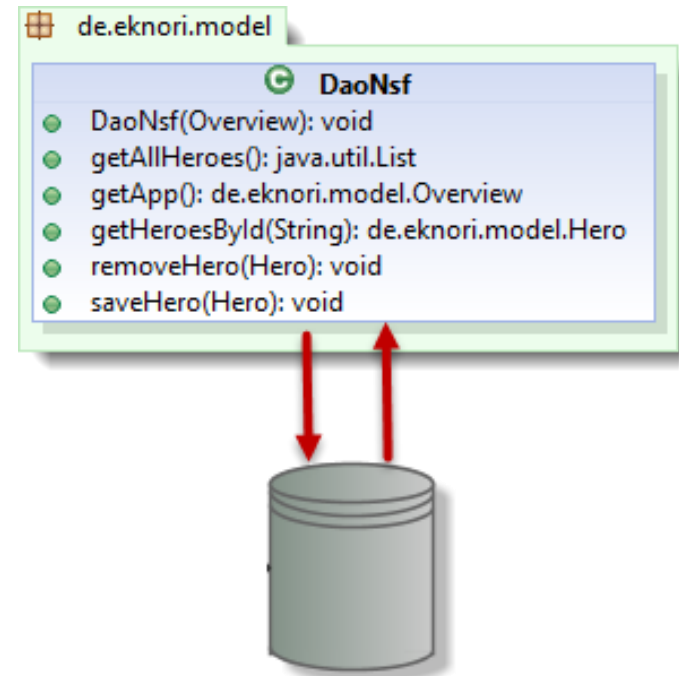
    public void setName(String name) {
        this.name = name;
    }
}
```



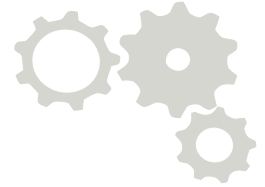
de.eknori.model	
Hero	
●	generateNewId(): void
●	getDescription(): java.lang.String
●	getId(): java.lang.String
●	getName(): java.lang.String
●	getPower(): java.lang.String
●	getSpecies(): java.lang.String
●	Hero(Overview): void
●	remove(): void
●	save(): void
●	setDescription(String): void
●	setId(String): void
●	setName(String): void
●	setPower(String): void
●	setSpecies(String): void

Data Access Object (DAO)

- A data access object (DAO) is an object that provides an abstract interface to some type of database or other persistence mechanism.
- All read / write data operations are delegated to DAO.
- No other part of the application has direct access to the underlying datastore



Changes - What Management Wants ...

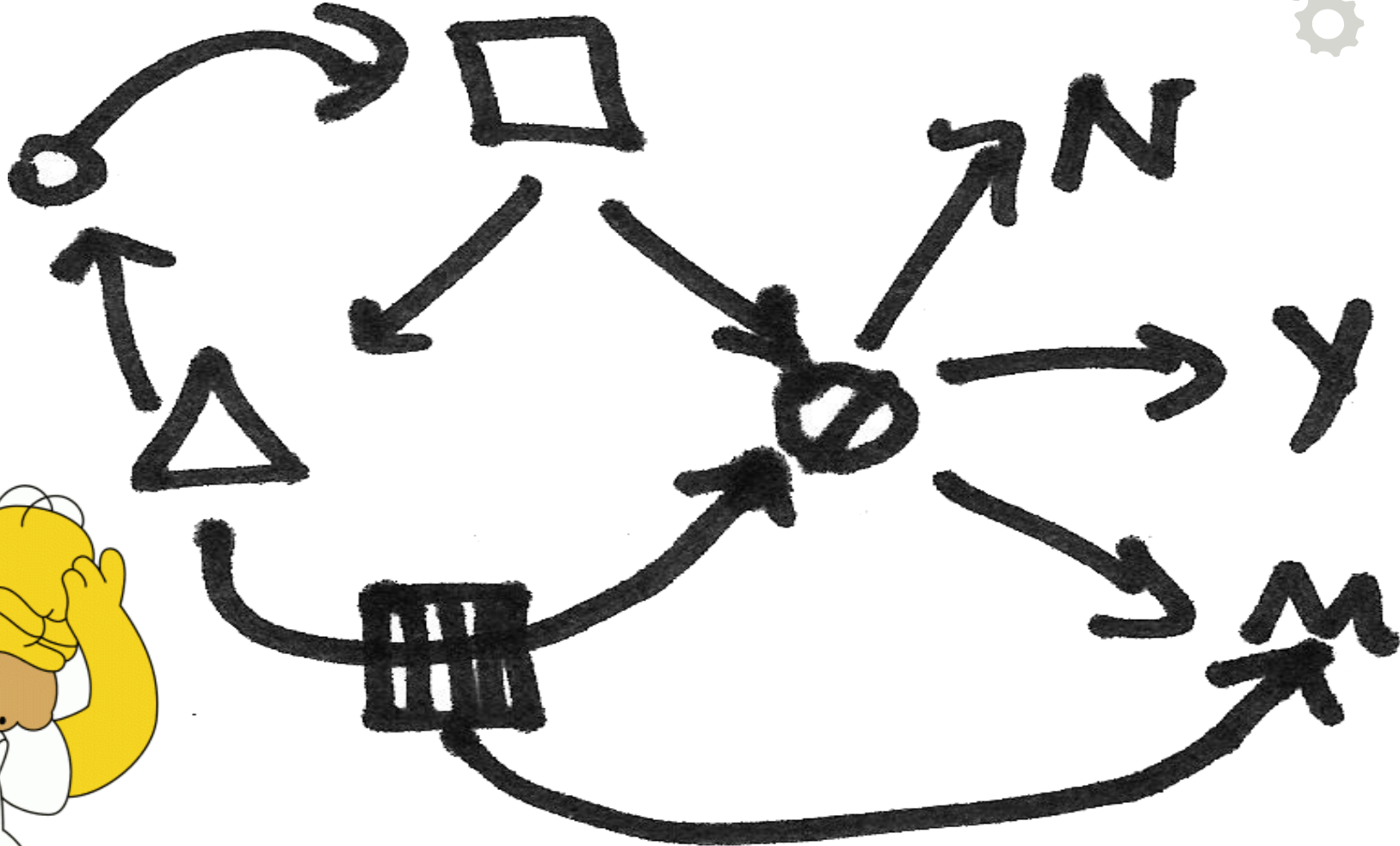
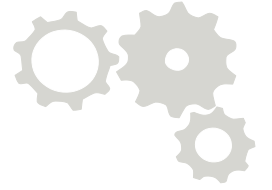


<?xml?>

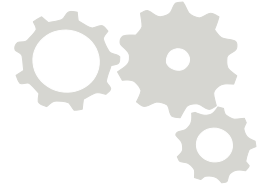
{JSON}



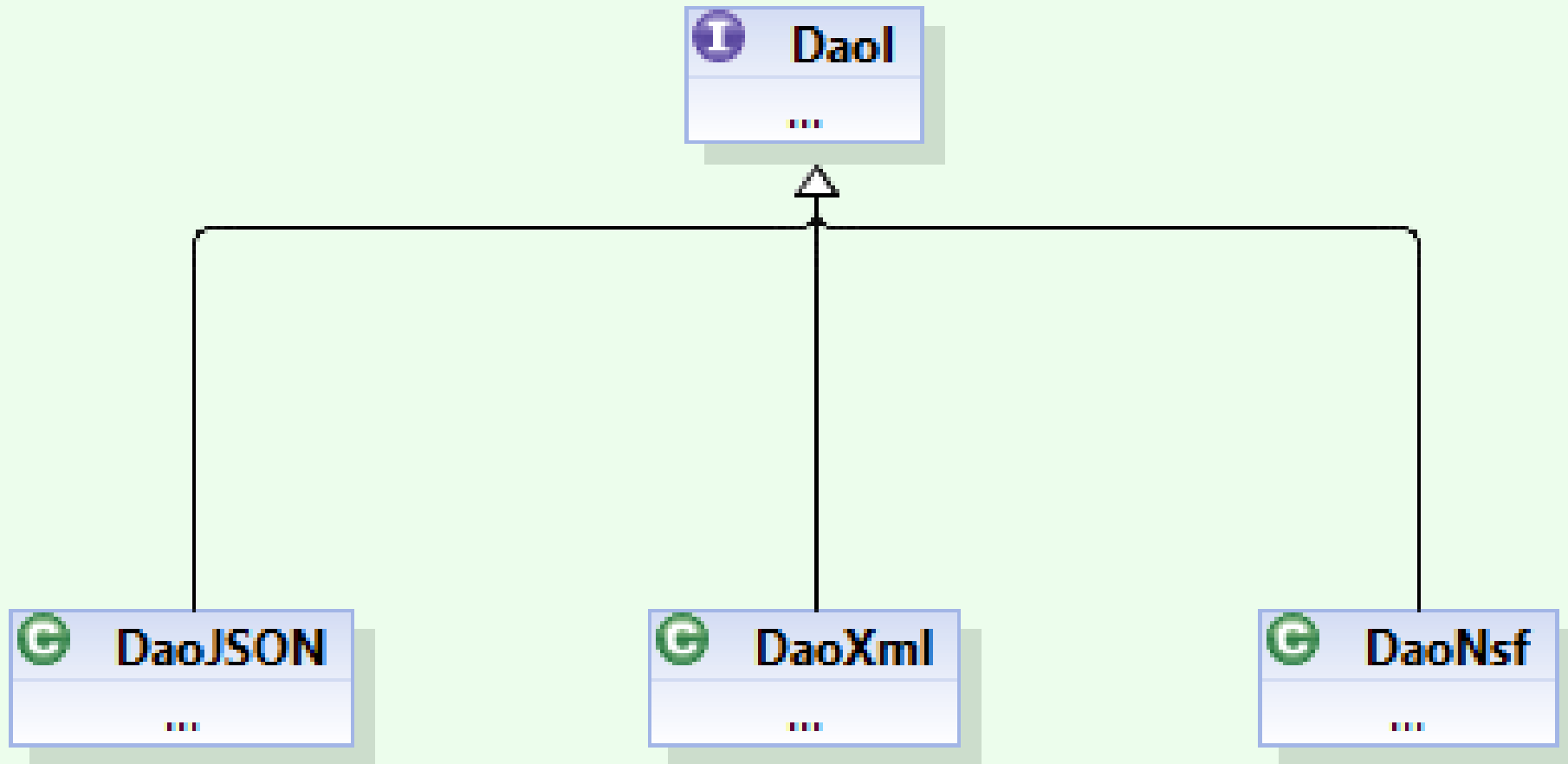
Changes - What You Think ...



Changes - What You Need ...



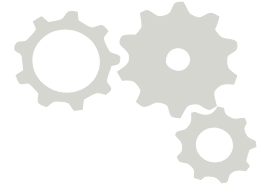
 de.eknori.model





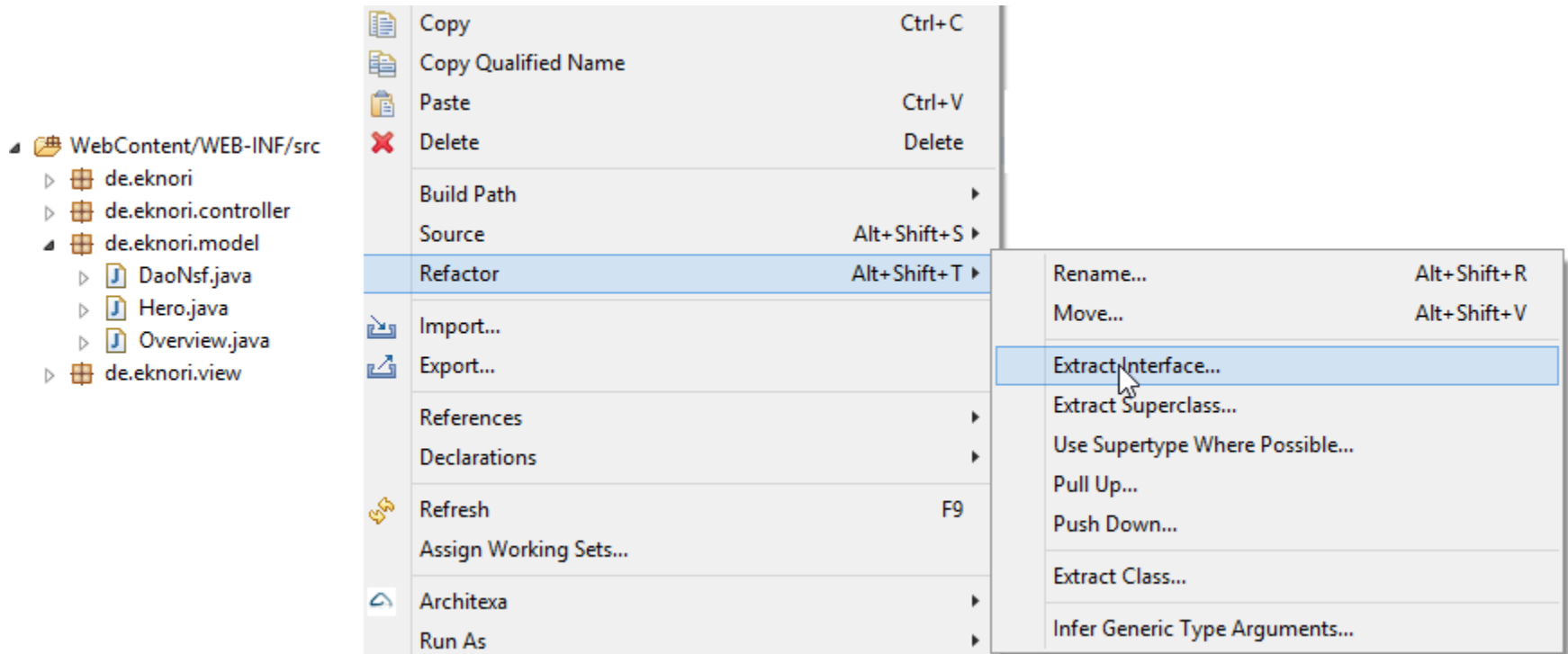
Interface - Definition

- An interface in the Java programming language is an **abstract type** that is used to specify an interface (in the generic sense of the term) that classes **must implement**.
- Interfaces are declared using the **interface keyword**, and may only contain **method signature** and **constant declarations** (variable declarations that are declared to be both static and final).



Interface - HowTo

- Refactor → Extract Interface



The screenshot shows an IDE interface with a project explorer on the left and a main menu on the right. The project explorer displays a folder structure for 'WebContent/WEB-INF/src' containing several packages and files. The main menu is open, showing the 'Refactor' option selected. A sub-menu is displayed for 'Refactor', with 'Extract Interface...' highlighted. A mouse cursor is pointing at the 'Extract Interface...' option.

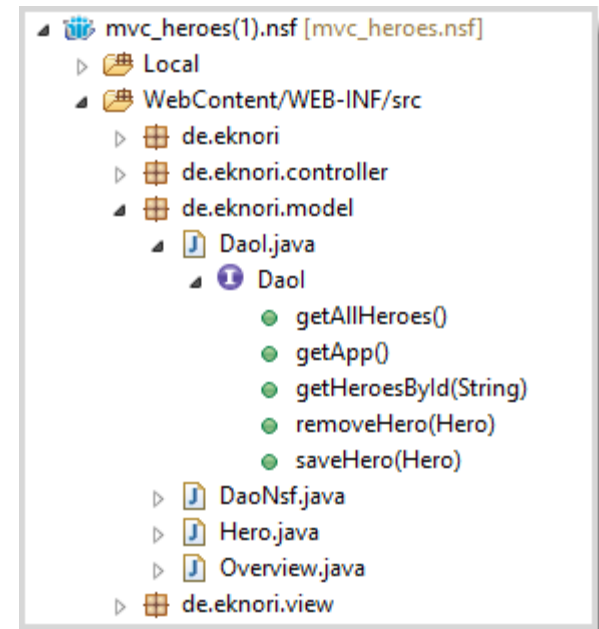
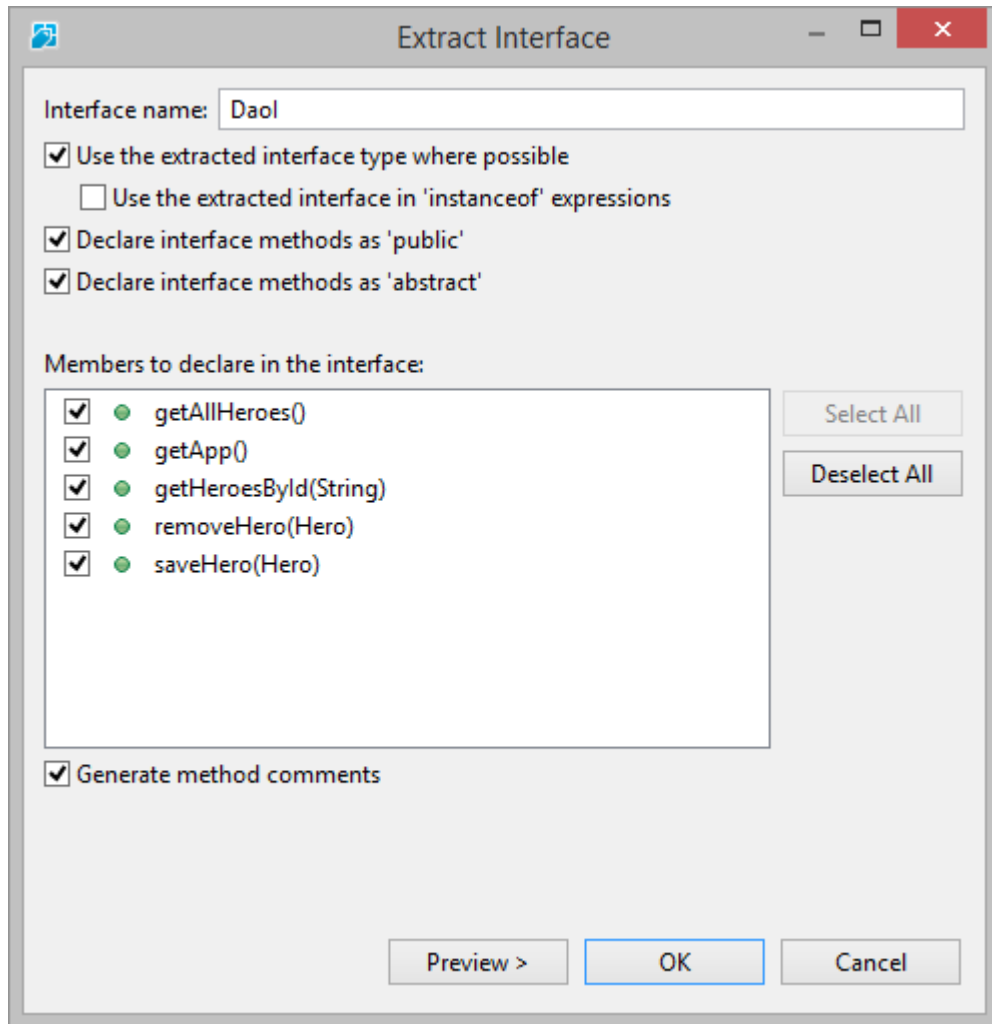
Project Explorer (Left):

- WebContent/WEB-INF/src
 - de.eknori
 - de.eknori.controller
 - de.eknori.model
 - DaoNsf.java
 - Hero.java
 - Overview.java
 - de.eknori.view

Main Menu (Right):

- Copy (Ctrl+C)
- Copy Qualified Name
- Paste (Ctrl+V)
- Delete (Delete)
- Build Path
- Source (Alt+Shift+S)
- Refactor (Alt+Shift+T)**
 - Rename... (Alt+Shift+R)
 - Move... (Alt+Shift+V)
 - Extract Interface...**
 - Extract Superclass...
 - Use Supertype Where Possible...
 - Pull Up...
 - Push Down...
 - Extract Class...
 - Infer Generic Type Arguments...
- Import...
- Export...
- References
- Declarations
- Refresh (F9)
- Assign Working Sets...
- Architexa
- Run As

Interface – How To



Interface – The Result

```
package de.eknori.model;
```

```
import java.util.List;
```

```
public interface DaoI {
```

```
    public abstract Overview getApp();
```

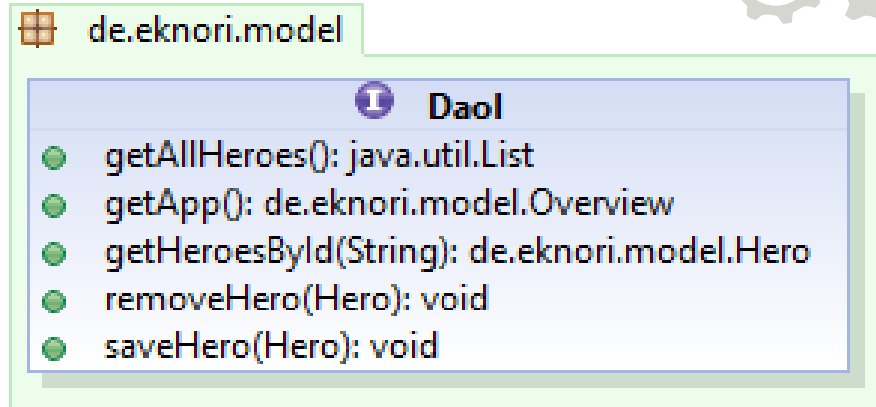
```
    public abstract List<Hero> getAllHeroes() throws Exception;
```

```
    public abstract Hero getHeroesById(String id) throws Exception;
```

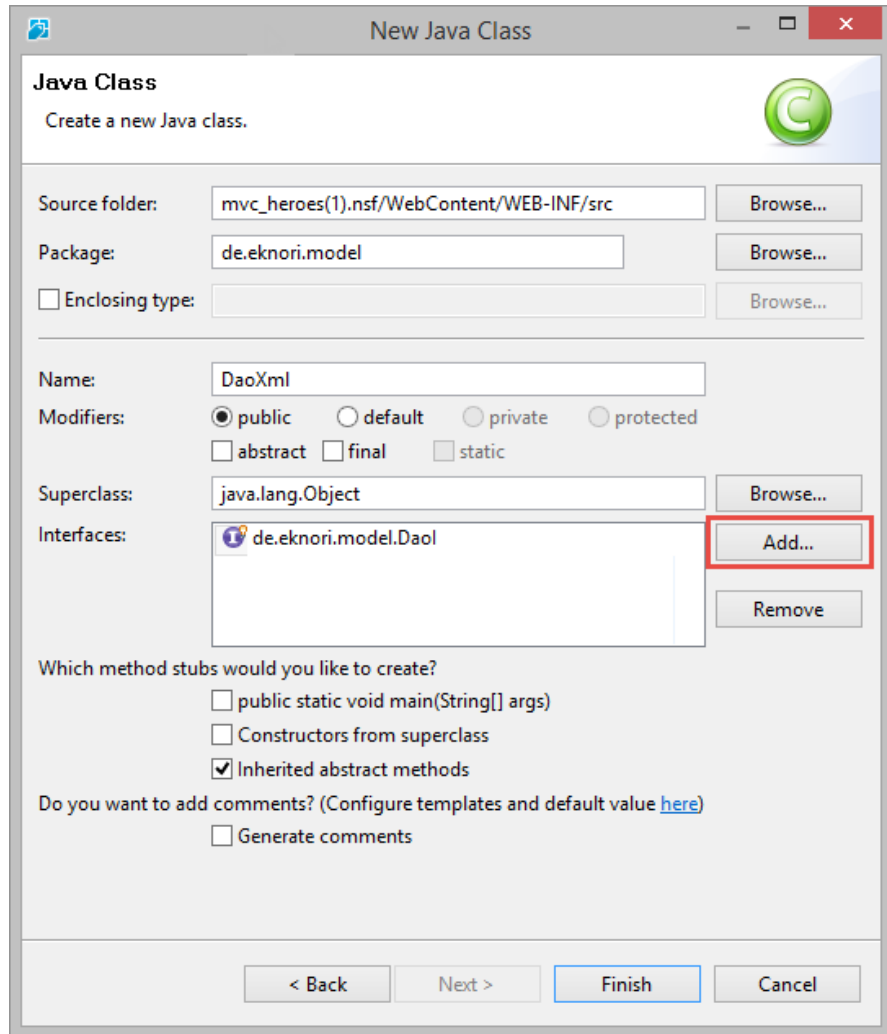
```
    public abstract void saveHero(Hero model) throws Exception;
```

```
    public abstract void removeHero(Hero model) throws Exception;
```

```
}
```



New Class: DaoXml



New Java Class

Create a new Java class.

Source folder: Browse...

Package: Browse...

☐ Enclosing type: Browse...

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: Browse...

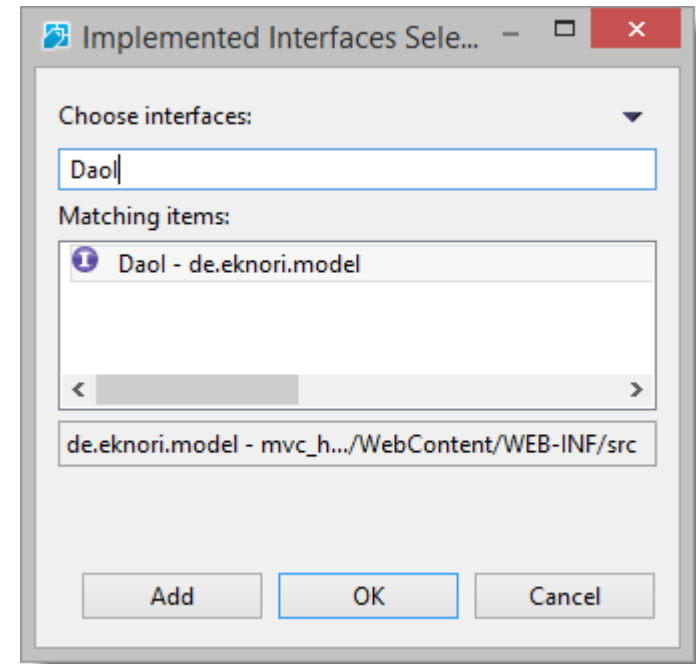
Interfaces: **Add...** Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

< Back Next > Finish Cancel



Implemented Interfaces Sele...

Choose interfaces:

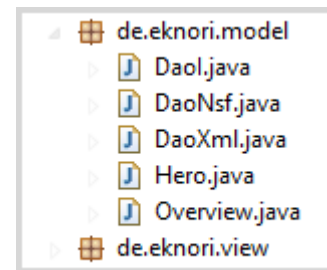
Matching items:

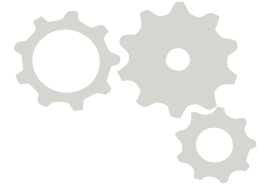
☒ DaoI - de.eknori.model

< >

de.eknori.model - mvc_h.../WebContent/WEB-INF/src

Add OK Cancel





New Class: DaoXml

- Stubs for all methods and properties
- ToDo: Write code to work with XML instead of NSF

```
package de.eknori.model;

import java.util.List;

public class DaoXml implements DaoI {

    public List<Hero> getAllHeroes() throws Exception {
        return null;
    }

    public Overview getApp() {
        return null;
    }

    public Hero getHeroesById(String id) throws Exception {
        return null;
    }

    public void removeHero(Hero model) throws Exception {
    }

    public void saveHero(Hero model) throws Exception {
    }

}
```

Implementation Usage



- Designer automagically does the necessary changes
 - in class DaoNsf

```
public class DaoNsf implements Serializable, DaoI {
```

- in class Overview

```
private DaoNsf dao = null;  
public DaoNsf getDao() throws Exception {  
    if (dao == null) {  
        dao = new DaoNsf(this);  
    }  
    return dao;  
}
```



```
private DaoI dao = null;  
public DaoI getDao() throws Exception {  
    if (dao == null) {  
        dao = new DaoNsf(this);  
    }  
    return dao;  
}
```

And Finally ...



- Depending on which datasource to use, change that one line in your code.

```
private DaoI dao = null;
public DaoI getDao() throws Exception {
    if (dao == null) {
        dao = new DaoNsf(this);
    }
    return dao;
}
```

```
private DaoI dao = null;
public DaoI getDao() throws Exception {
    if (dao == null) {
        dao = new DaoXml(this);
    }
    return dao;
}
```



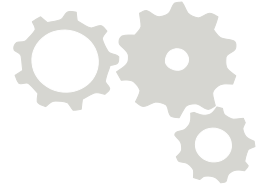
Demo



Useful Links

- [Architexa](#) helps you to understand and to document large/complex codebases.
- Design Patterns in Java [Tutorial](#)
- [Gang Of Four](#)

Q & A



Twitter @eknori

Mail ulrich.krause@eknori.de





