

Anno Accademico 2007-2008

Laboratorio di Tecnologie Web

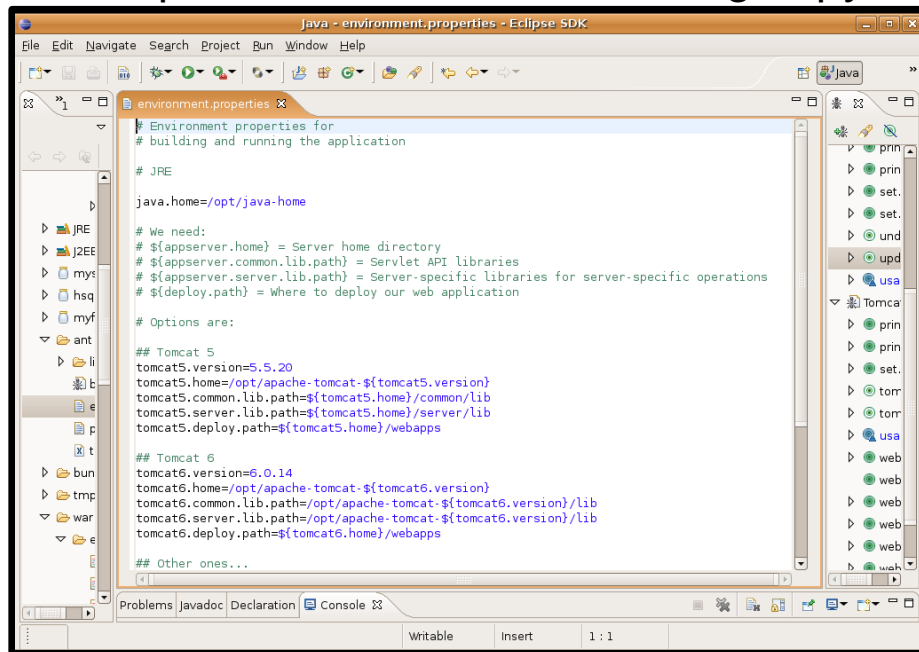
Esempio di applicazione web
basata su JSF e DAO

<http://www-lia.deis.unibo.it/Courses/TecnologieWeb0708/>

Another project from the course web site

> You can handle it as usual:

- download the *CdCollectionManager.zip* archive file
- import it in Eclipse
- set up 'J2EE 1.4 API' user library (by referring Tomcat libraries), code compliance (must be 5.0) and installed JRE (found a JDK5 on your machine)
- modify *ant/environment.properties* accordingly to your machine settings (or replace this file with a working copy from a previous project)



```
Environment properties for
# building and running the application

# JRE
java.home=/opt/java-home

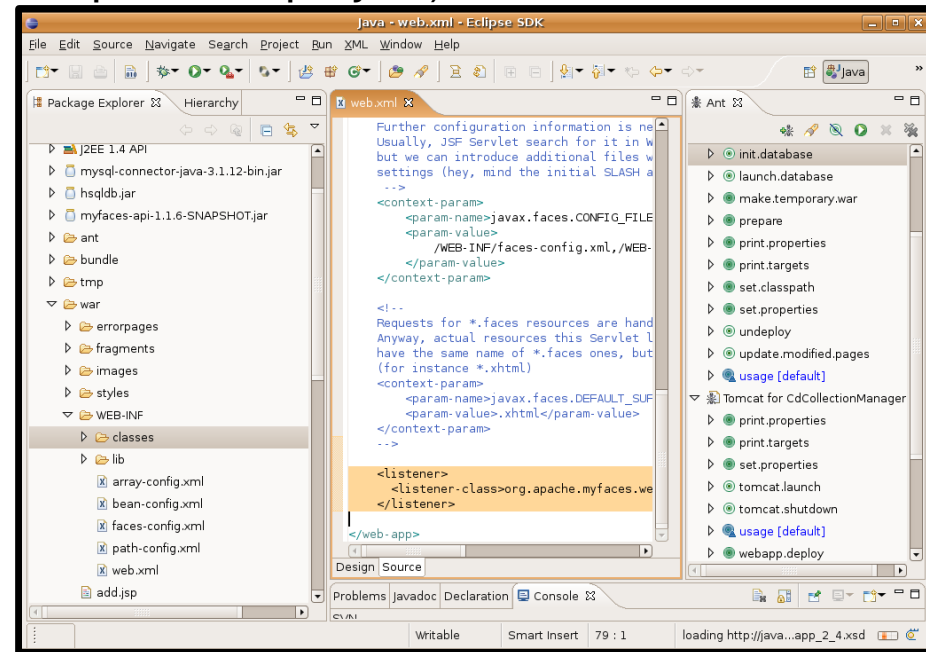
# We need:
# ${appserver.home} = Server home directory
# ${appserver.common.lib.path} = Servlet API libraries
# ${appserver.server.lib.path} = Server-specific libraries for server-specific operations
# ${deploy.path} = Where to deploy our web application

# Options are:

## Tomcat 5
tomcat5.version=5.5.20
tomcat5.home=/opt/apache-tomcat-${tomcat5.version}
tomcat5.common.lib.path=${tomcat5.home}/common/lib
tomcat5.server.lib.path=${tomcat5.home}/server/lib
tomcat5.deploy.path=${tomcat5.home}/webapps

## Tomcat 6
tomcat6.version=6.0.14
tomcat6.home=/opt/apache-tomcat-${tomcat6.version}
tomcat6.common.lib.path=/opt/apache-tomcat-${tomcat6.version}/lib
tomcat6.server.lib.path=/opt/apache-tomcat-${tomcat6.version}/lib
tomcat6.deploy.path=${tomcat6.home}/webapps

## Other ones...
```



```
Further configuration information is ne
Usually, JSF Servlet search for it in w
but we can introduce additional files w
settings (hey, mind the initial SLASH a
...

<context-param>
  <param-name>javax.faces.CONFIG_FILE
  <param-value>
    /WEB-INF/faces-config.xml,/WEB-
  </param-value>
</context-param>

<!--
Requests for *.faces resources are hand
Anyway, actual resources this Servlet l
have the same name of *.faces ones, but
(for instance *.xhtml)
<context-param>
  <param-name>javax.faces.DEFAULT_SUF
  <param-value>.xhtml</param-value>
</context-param>

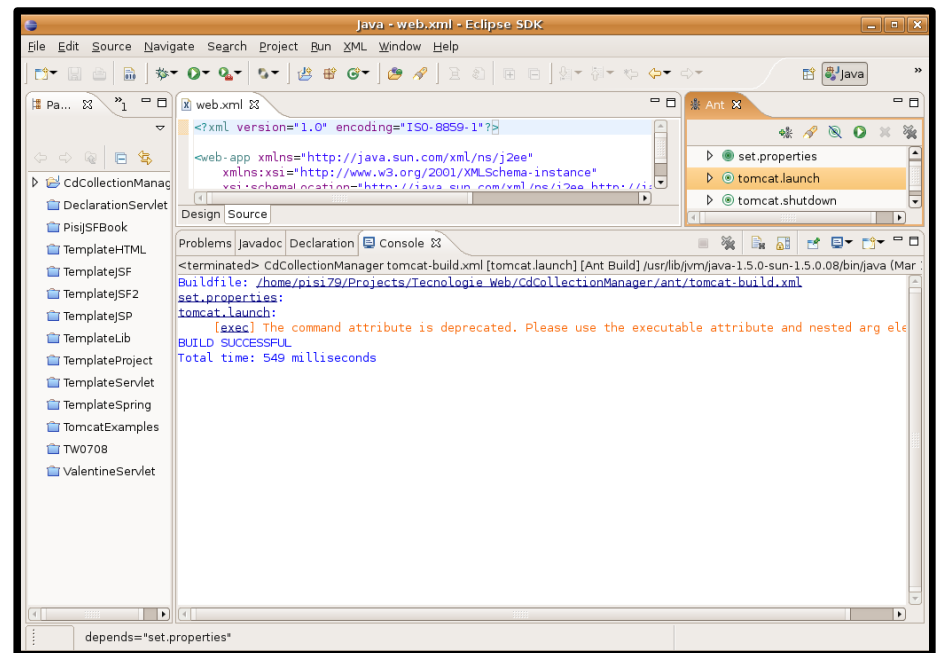
<listener>
  <listener-class>org.apache.myfaces.we
</listener>

</web-app>
```

Another project from the course web site

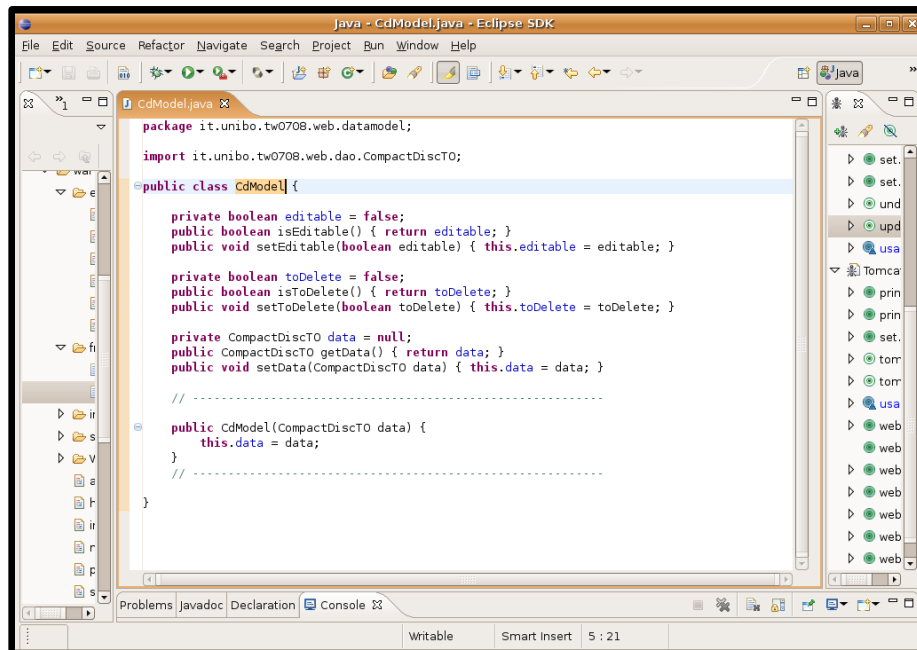
> You can handle it as usual:

- setup Tomcat users (**either** in *ant/environemnt.properties* or in *\$TOMCAT_HOME/conf/tomcat-users.xml* – you can find a sample file in the project *tmp* directory)
- launch Tomcat (**either** by issuing prompt commands or via the *ant/tomcat-build.xml*'s *tomcat.launch* target)



Another project from the course web site

- > Besides, this time (the same way you did in the *TemplateJSF* project, only):
- launch the HSQLDB database server (via the *ant/build.xml*'s *launch.database* target)
 - initialize database tables (via the *ant/build.xml*'s *init.database* target)



The screenshot shows the Eclipse IDE with the `CdModel.java` file open. The code defines a `CdModel` class that manages a `CompactDiscTO` object. It includes methods for setting and getting the `data` field, and methods for managing the `editable` and `toDelete` flags.

```
package it.unibo.tw0708.web.datamodel;

import it.unibo.tw0708.web.dao.CompactDiscTO;

public class CdModel {

    private boolean editable = false;
    public boolean isEditable() { return editable; }
    public void setEditable(boolean editable) { this.editable = editable; }

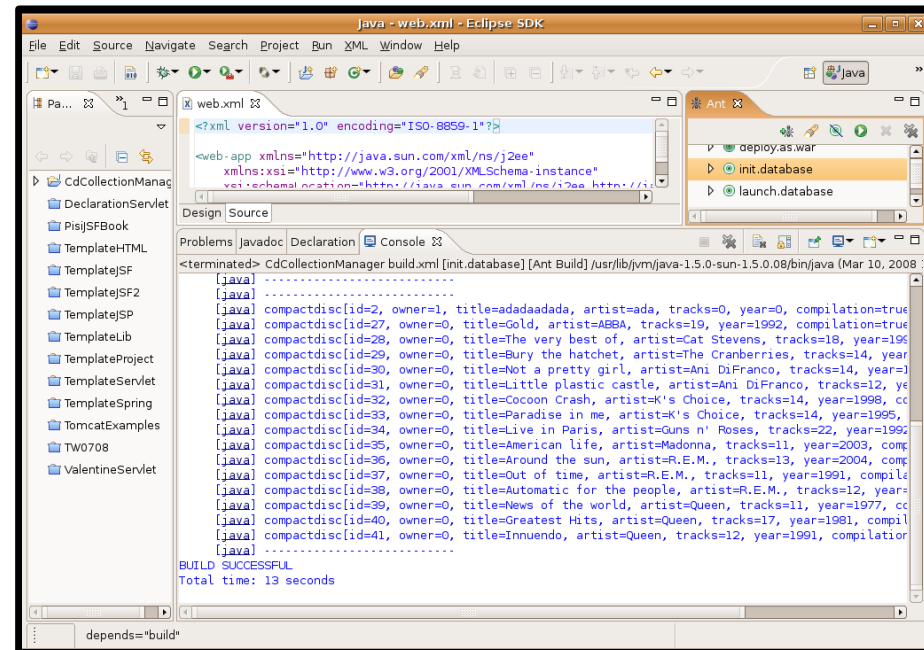
    private boolean toDelete = false;
    public boolean isToDelete() { return toDelete; }
    public void setToDelete(boolean toDelete) { this.toDelete = toDelete; }

    private CompactDiscTO data = null;
    public CompactDiscTO getData() { return data; }
    public void setData(CompactDiscTO data) { this.data = data; }

    // .....

    public CdModel(CompactDiscTO data) {
        this.data = data;
    }

    // .....
}
```



The screenshot shows the Eclipse IDE with the `web.xml` file open. The XML file is configured for a Java web application. The Ant console output shows the successful execution of the `init.database` target, which initializes the database tables.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
</web-app>
```

```
<terminated> CdCollectionManager build.xml [init.database] [Ant: Build] /usr/lib/jvm/java-1.5.0-sun-1.5.0.08/bin/java (Mar 10, 2008)
[java] .....
[java] compactdisc[id=2, owner=1, title=adadaadada, artist=ada, tracks=0, year=0, compilation=true
[java] compactdisc[id=27, owner=0, title=Gold, artist=ABBA, tracks=19, year=1992, compilation=true
[java] compactdisc[id=28, owner=0, title=The very best of, artist=Cat Stevens, tracks=18, year=199
[java] compactdisc[id=29, owner=0, title=Bury the hatchet, artist=The Cranberries, tracks=14, year
[java] compactdisc[id=30, owner=0, title=Not a pretty girl, artist=Ani DiFranco, tracks=14, year=
[java] compactdisc[id=31, owner=0, title=Little plastic castle, artist=Ani DiFranco, tracks=12, ye
[java] compactdisc[id=32, owner=0, title=Cocoon Crash, artist=K's Choice, tracks=14, year=1998, cc
[java] compactdisc[id=33, owner=0, title=Paradise in me, artist=K's Choice, tracks=14, year=1995, cc
[java] compactdisc[id=34, owner=0, title=Live in Paris, artist=Guns n' Roses, tracks=22, year=1992
[java] compactdisc[id=35, owner=0, title=American life, artist=Madonna, tracks=11, year=2003, comp
[java] compactdisc[id=36, owner=0, title=Around the sun, artist=R.E.M., tracks=13, year=2004, comp
[java] compactdisc[id=37, owner=0, title=Out of time, artist=R.E.M., tracks=11, year=1991, compil
[java] compactdisc[id=38, owner=0, title=Automatic for the people, artist=R.E.M., tracks=12, year
[java] compactdisc[id=39, owner=0, title=News of the world, artist=Queen, tracks=11, year=1977, cc
[java] compactdisc[id=40, owner=0, title=Greatest Hits, artist=Queen, tracks=17, year=1981, compil
[java] compactdisc[id=41, owner=0, title=Innuendo, artist=Queen, tracks=12, year=1991, compilation
[java] .....
BUILD SUCCESSFUL
Total time: 13 seconds
```

Are you ready?

> Now you can deploy the web application:

- by just copying the war archive (**xor** its corresponding exploded dir) to the *\$TOMCAT_HOME/webapps* path (via the *ant/build.xml*'s *deploy.as.war* or *deploy.as.war* target)
- by remotely managing Tomcat (via the *ant/tomcat-build.xml*'s *webapp.redeploy* target)
 - see the chain of dependencies among target that this entails!
 - NOTICE: double-clicking this target in the *Ant View* is sufficient to **redeploy your web application if you modify it**: it does everything that's needed to completely perform this task (i.e., cleaning, recompiling, re-packaging, undeploying, deploying and removal of temporary files)

...provided you configure Tomcat's users correctly, of course.

Let's see what's there... (1/3)

- > A few JSF beans
 - coded in the `it.unibo.tw0708.web.beans` package
 - mapped in the `war/WEB-INF/bean-config.xml` file
- > A few JSP pages that uses tags from the JSF framework
 - *by just declaring their corresponding tag libraries*
 - `war/*.jsp` and `war/errorpages/*.jsp` files
- > A few static resources (such as pictures to associate to particular input controls)
 - mapped in the `war/WEB-INF/array-config.xml` file
- > A few navigation rules (especially for the user selection process, as we'll see)
 - mapped in the `war/WEB-INF/path-config.xml` file
- > Reusable HTML/JSP/JSF fragments
 - for creating footers and navigation tabs (same as in *TemplateJSP* project)
 - `war/fragmens/*` files
- > ...

Let's see what's there... (2/3)

- > ...
- > Some graphics
 - *war/images/** files
- > Bundle messages for internationalization
 - *bundle/*** files
 - ANT targets package these files just as they were classes, in the */WEB-INF/classes* path
- > DAO pattern classes
 - transfer objects and abstract/interface definitions are coded in the *it.unibo.tw0708.web.dao* package
 - HSQLDB implementation is in *it.unibo.tw0708.web.dao.hsqldb* one
- > *javax.faces.model.DataModel* -related classes
 - to work fine with *<h:dataTable>* tags
- > Utilities (already seen in previous projects and perhaps partially unused here)
 - (optionally) do it yourself: *log4j* is a better tool to perform logging → modify *it.unibo.tw0708.web.utils.Logger* class to support it

Let's see what's there... (3/3)

> ...

> Client classes

- coded in the `it.unibo.tw0708.client` package
- provide `main()`'s methods to initialize db tables and verify data
- these classes too leverage the DAO objects to access the database!

About DAO: noticeables

- > Only *Hypersonic* implementation this time
 - you can provide yourself a *MySql / Postgres / DB2 / whatever* one if you need it.. ..or simply want to try
 - you need the database server running
 - you need the JDBC connector libraries in the *WEB-INF/lib* path
 - you also need to add this library to the project's Eclipse build-path
 - or you can safely remove the *mysql-connector-java-3.1.12-bin.jar* library
 - from the project's Eclipse build-path
 - and from the *WEB-INF/lib* path on the file system



- ...I left it there intentionally, as **I created the CdCollectionManager by cloning the *TemplateJSF* one!!!**
 - copied and pasted that project in my Eclipse workspace
 - changed the project name
 - changed the webapp name property in *ant/project.properties*
 - changed the project name in *ant/build.xml* and *ant/tomcat-build.xml*

About DAO: things in place

> Two DAO objects: `it.unibo.tw0708.web.dao.UserDAO` and `it.unibo.tw0708.web.dao.CompactDiscDAO`

- and their corresponding transfer ones:

`it.unibo.tw0708.web.dao.UserTO`,

`it.unibo.tw0708.web.dao.CompactDiscTO`

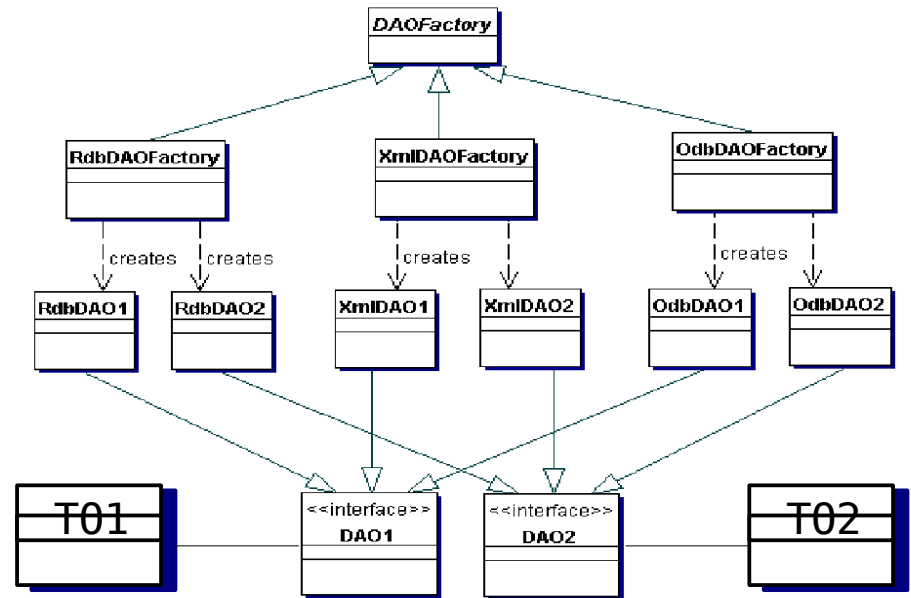
- DAO objects provide database access by wrapping JDBC calls and exchanging data in the form of transfer objects with their invokers

> Abstract factory `it.unibo.tw0708.`

`web.dao.DAOFactory` lets you instantiate a concrete one

- `it.unibo.tw0708.web.dao.hs`
`qldb.HsqldbDAOFactory`

concrete factory (the only one available) provides *HSQldb* versions of the DAO objects



About the JSF beans

> `it.unibo.tw0708.web.beans.User` provides user selection / addition facilities (data storing, navigation actions, ...)

- it is mapped as `user` session-scoped bean in `WEB-INF/bean-config.xml` file

> `it.unibo.tw0708.web.beans.LocaleChanger` changes the locale in use

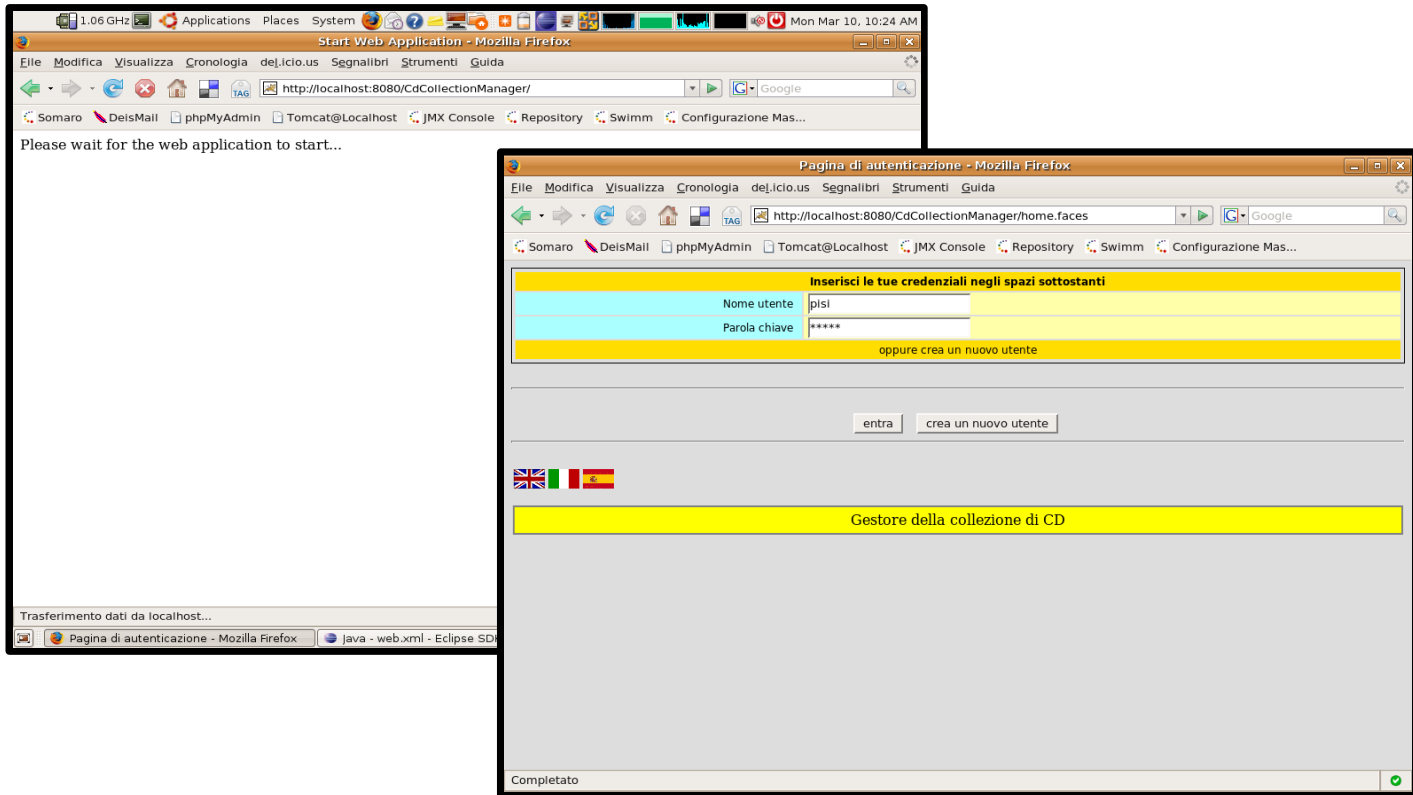
- it is mapped as `localeChanger` session-scoped bean in `WEB-INF/bean-config.xml` file
- all stuff we've already seen in the `TemplateJSF` project
 - hey, notice! this time not all locales provide all message bundles and sometimes not all expected messages are preset in a certain bundle → see how the framework falls back to messages from the default locale (`en`), as mapped in the `WEB-INF/faces-config.xml` file

> `it.unibo.tw0708.web.beans.CdManager` backs the most part of the web application

- it is mapped as `manager` session-scoped bean in `WEB-INF/bean-config.xml`
- we are gonna analyze its behaviour page by page in the following

Introductory pages (1/3)

> *index.html* shows the usual 'please wait' message and redirects to the actual JSF home page



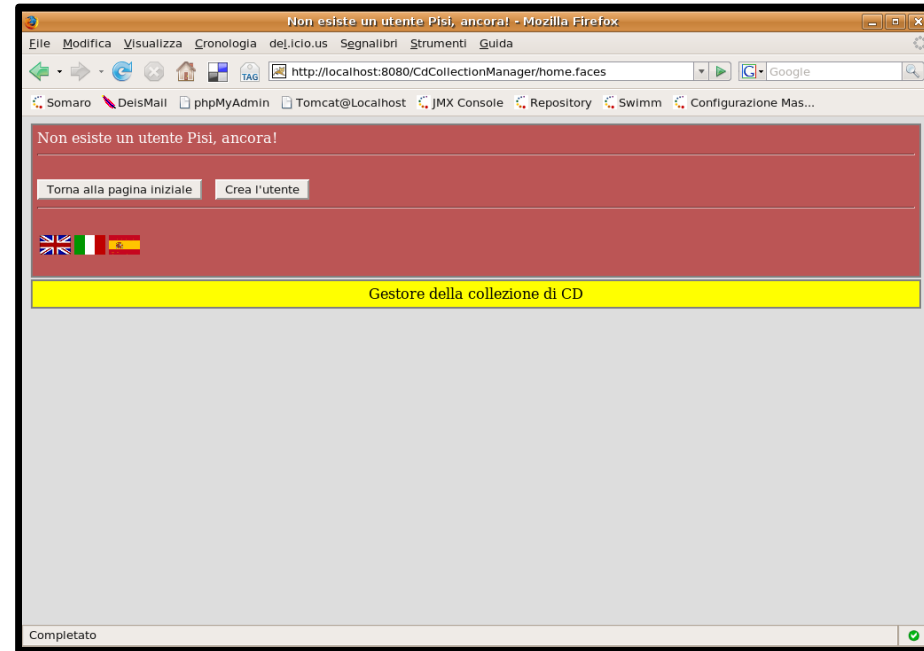
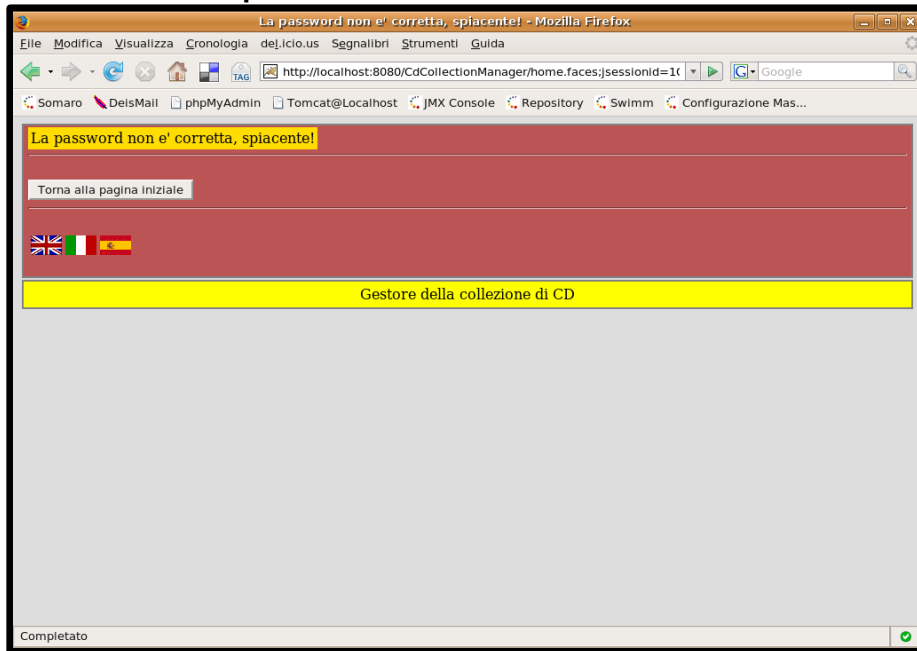
> *home.jsp* provide input controls for specifying user credentials and to change the locale currently in use

- locale changing works exactly the same as in *TemplateJSF* project

Introductory pages (1/3)

> *home.jsp* stores user input in member variables of the *user* bean and invokes an action method from it

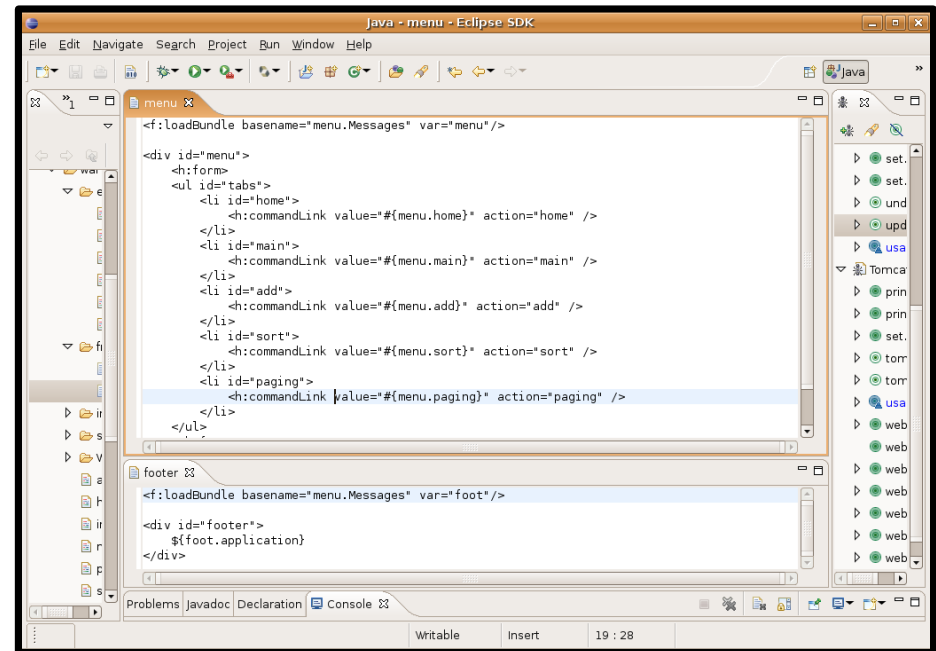
- navigation rules in *WEB-INF/path-config.xml* lead to *errorpages/<err_type>_error.jsp* pages in case something wrong happens
- this time too, **this is not a real authentication process as we have not defined any security constraint** in the *WEB-INF/web.xml* web application descriptor



Introductory pages (3/3)

> *fragments/menu* and *fragments/footer* fragments provide code excerpts for rendering a common footer and a tab navigation pane in (almost) all pages of the web application

- we have already embedded JSP fragments in JSP pages in the past → it also works with JSF parts, though it can be somewhat more difficult to keep consistency: look at the message bundle imports and tag library declaration...



The screenshot shows the Eclipse IDE with two JSP fragments open. The top fragment, named 'menu', contains the following code:

```
<f:loadBundle basename="menu.Messages" var="menu" />

<div id="menu">
  <h:form>
    <ul id="tabs">
      <li id="home">
        <h:commandLink value="#{menu.home}" action="home" />
      </li>
      <li id="main">
        <h:commandLink value="#{menu.main}" action="main" />
      </li>
      <li id="add">
        <h:commandLink value="#{menu.add}" action="add" />
      </li>
      <li id="sort">
        <h:commandLink value="#{menu.sort}" action="sort" />
      </li>
      <li id="paging">
        <h:commandLink value="#{menu.paging}" action="paging" />
      </li>
    </ul>
  </div>
```

The bottom fragment, named 'footer', contains the following code:

```
<f:loadBundle basename="menu.Messages" var="foot" />

<div id="footer">
  ${foot.application}
</div>
```

User selection errors

> *home.jsp* triggers *user* bean methods that compare input by user with data stored in the user table of the underlying database and return navigation actions accordingly:

- if username exists and password digest matches the stored one the user is allowed to navigate to *main.jsp* page
- if username does not exist or password is not correct the user receives suitable error pages (as mapped in *WEB-INF/path-config.xml* descriptor)



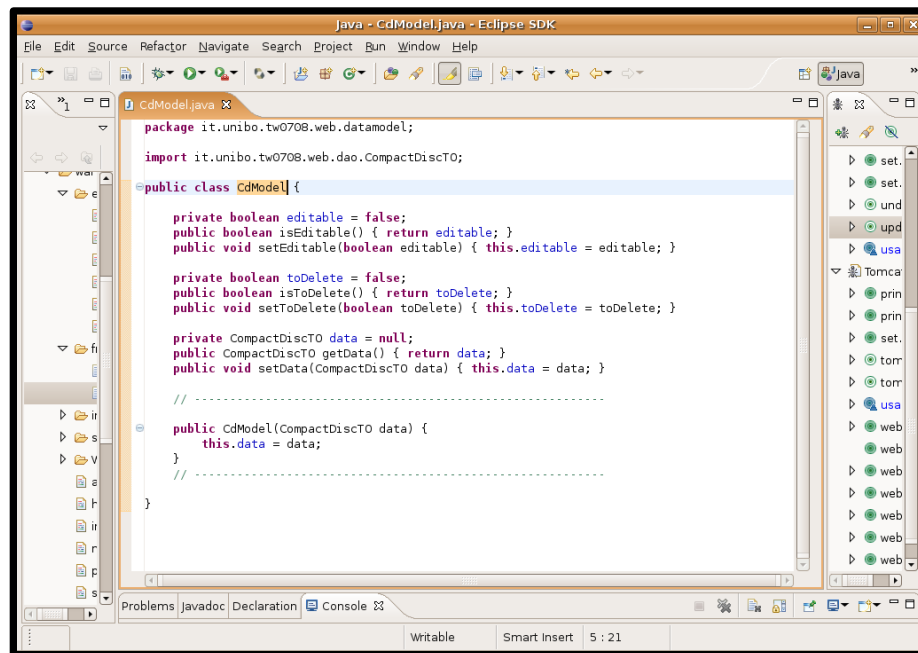
- ...*errorpages/nouser_error.jsp* does not embed *styles/default.css* !
 - you can add it yourself (for instance by copying the corresponding line of code from one of the other *errorpages/xxx_error.jsp* pages)
 - then you can use the *update.modified.pages* target in *ant/build.xml* to update only the modified page on the server!
 - **it is not an actual redeployment!** just a file copy / replacement!
 - **it works only with static resources** (i.e., not with message bundles, deployment descriptors and Java classes)
 - but it is a lot faster than completely redeploy your project (and you ain't losing your session data, this way!)

main.jsp page (1/3)

- > *main.jsp* is the first page that gets shown after the 'login-alike' process succeeds
 - an `<h:dataTable>` tag iterates over a `javax.faces.model.DataModel` element that wraps the actual list of cd items
- > *manager* bean provides
 - a getter method for retrieving the datamodel object (by invoking initialization methods and DAO functionalities, in case)
 - additional methods for handling editing/deleting or modification discarding
 - all these methods return to the same current view
 - we change the page being shown just by selecting links from the tabs in the upper part of the screen (and not by clicking any page button, this time)
- > `javax.faces.model.DataModel` is an abstract class: we build up the actual datamodel for `<h:dataTable>` by using out-of-the-box utility classes (i.e., `javax.faces.model.ListDataModel`) to wrap the cd list
 - see `it.unibo.tw0708.beans.CdManager.getCompactDiscs()` method for details

main.jsp page (2/3)

- > to realize the cd items of the datamodel, we don't directly use `it.unibo.tw0708.dao.CompactDiscTO` objects, as we need further features:
- transfer objects are wrapped by `it.unibo.tw0708.datamodel.CdModel` objects
 - the latter ones support the extended logic we need
 - some boolean flags to indicate that the cd is being edited or deleted



```
Java - CdModel.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help

CdModel.java
package it.unibo.tw0708.web.datamodel;
import it.unibo.tw0708.web.dao.CompactDiscTO;

public class CdModel {

    private boolean editable = false;
    public boolean isEditable() { return editable; }
    public void setEditable(boolean editable) { this.editable = editable; }

    private boolean toDelete = false;
    public boolean isToDelete() { return toDelete; }
    public void setToDelete(boolean toDelete) { this.toDelete = toDelete; }

    private CompactDiscTO data = null;
    public CompactDiscTO getData() { return data; }
    public void setData(CompactDiscTO data) { this.data = data; }

    // .....

    public CdModel(CompactDiscTO data) {
        this.data = data;
    }

    // .....
}
```

main.jsp page (2/2)

> columns in `<h:dataTable>` display one of two different controls (respectively of input and output type) for each cd field, depending on the status of the boolean flags:

- since flag controls present the `onChange="submit()"` attribute, page view is immediately recalculated after flag modification (by triggering the JSF life-cycle you already know)
- the *manager* bean holds the cd items referenced by the `dataTable` UI component, so that modifications can be applied to the page view, before the user chooses to persist them to the database

Pagina principale per la visualizzazione della collezione di CD - Mozilla Firefox

Elle Modifica Visualizza Cronologia deLicio.us Segnalibri Strumenti Guida

http://localhost:8080/CdCollectionManager/home.faces

Somaro DeisMail phpMyAdmin Tomcat@Localhost JMX Console Repository Swimm Configurazione Mas...

Autenticazione Pagina principale Aggiungi Ordina Visualizza a gruppi

Possessore = pist
Numero di CD posseduti = 15

Cancellazione	Modifica	Artista	Titolo	Tracce	Anno	Raccolta
<input type="checkbox"/>	<input type="checkbox"/>	ABBA	Gold	19	1992	true
<input type="checkbox"/>	<input type="checkbox"/>	Cat Stevens	The very best of	18	1990	true
<input type="checkbox"/>	<input type="checkbox"/>	The Cranberries	Bury the hatchet	14	1999	false
<input type="checkbox"/>	<input type="checkbox"/>	Ani DiFranco	Not a pretty girl	14	1997	false
<input type="checkbox"/>	<input type="checkbox"/>	Ani DiFranco	Little plastic castle	12	1998	false
<input type="checkbox"/>	<input type="checkbox"/>	K's Choice	Cocoon Crash	14	1998	false
<input type="checkbox"/>	<input type="checkbox"/>	K's Choice	Paradise in me	14	1995	false
<input type="checkbox"/>	<input type="checkbox"/>	Guns n' Roses	Live in Paris	22	1992	false
<input type="checkbox"/>	<input type="checkbox"/>	Madonna	American life	11	2003	false
<input type="checkbox"/>	<input type="checkbox"/>	R.E.M.	Around the sun	13	2004	false
<input type="checkbox"/>	<input type="checkbox"/>	R.E.M.	Out of time	11	1991	false
<input type="checkbox"/>	<input type="checkbox"/>	R.E.M.	Automatic for the people	12	1992	false
<input type="checkbox"/>	<input type="checkbox"/>	Queen	News of the world	11	1977	false
<input type="checkbox"/>	<input type="checkbox"/>	Queen	Greatest Hits	17	1981	true
<input type="checkbox"/>	<input type="checkbox"/>	Queen	Innuendo	12	1991	false

Salva Annulla

Gestore della collezione di CD

Completato

Pagina principale per la visualizzazione della collezione di CD - Mozilla Firefox

Elle Modifica Visualizza Cronologia deLicio.us Segnalibri Strumenti Guida

http://localhost:8080/CdCollectionManager/main.faces

Somaro DeisMail phpMyAdmin Tomcat@Localhost JMX Console Repository Swimm Configurazione Mas...

Autenticazione Pagina principale Aggiungi Ordina Visualizza a gruppi

Possessore = pist
Numero di CD posseduti = 15

Cancellazione	Modifica	Artista	Titolo	Tracce	Anno	Raccolta
<input type="checkbox"/>	<input type="checkbox"/>	ABBA	Gold	19	1992	true
<input type="checkbox"/>	<input type="checkbox"/>	Cat Stevens	The very best of	18	1990	true
<input type="checkbox"/>	<input type="checkbox"/>	The Cranberries	Bury the hatchet	14	1999	false
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Ani DiFranco	Not a pretty girl	14	1997	false
<input type="checkbox"/>	<input type="checkbox"/>	Ani DiFranco	Little plastic castle	12	1998	false
<input type="checkbox"/>	<input checked="" type="checkbox"/>	K's Choice	Cocoon crash	14	1998	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	K's Choice	Paradise in me	14	1995	false
<input type="checkbox"/>	<input type="checkbox"/>	Guns n' Roses	Live in Paris	22	1992	false
<input type="checkbox"/>	<input type="checkbox"/>	Madonna	American life	11	2003	false
<input type="checkbox"/>	<input type="checkbox"/>	R.E.M.	Around the sun	13	2004	false
<input type="checkbox"/>	<input type="checkbox"/>	R.E.M.	Out of time	11	1991	false
<input type="checkbox"/>	<input type="checkbox"/>	R.E.M.	Automatic for the people	12	1992	false
<input type="checkbox"/>	<input type="checkbox"/>	Queen	News of the world	11	1977	false
<input type="checkbox"/>	<input type="checkbox"/>	Queen	Greatest Hits	17	1981	true
<input type="checkbox"/>	<input type="checkbox"/>	Queen	Innuendo	12	1991	false

Salva Annulla

Gestore della collezione di CD

Completato

CdManager: a deeper look (1/2)

- > it keeps track of the current user id, in order to re-act to changes
 - for instance, the physical user could re-login with different credentials

- > **it holds data for the *main.jsp* view** and it provides methods for resetting and refreshing it when needed
 - it is necessary to have a member variable where to store modifications provided by the end-user, before and after writing them to the database
 - **page UI components** such as *dataTable* **refer to this data** (in this case, the cd list) to read what to show to the user and to store input from the user herself

CdManager: a deeper look (2/2)

> update methods provide business logic and return suitable navigation actions

- `it.unibo.tw0708.beans.CdManager.updateCompactDiscs()`:
 - **write** modifications/deletions **to the database**, if needed
 - refresh data for the current view (by resetting it = resyncing it to the db)
 - redisplay the current view (by returning *null*)
- `it.unibo.tw0708.beans.CdManager.updateCompactDiscs()`:
 - reset data for the current view (= resyncing it to the db status)
 - redisplay the current view (by returning *null*)

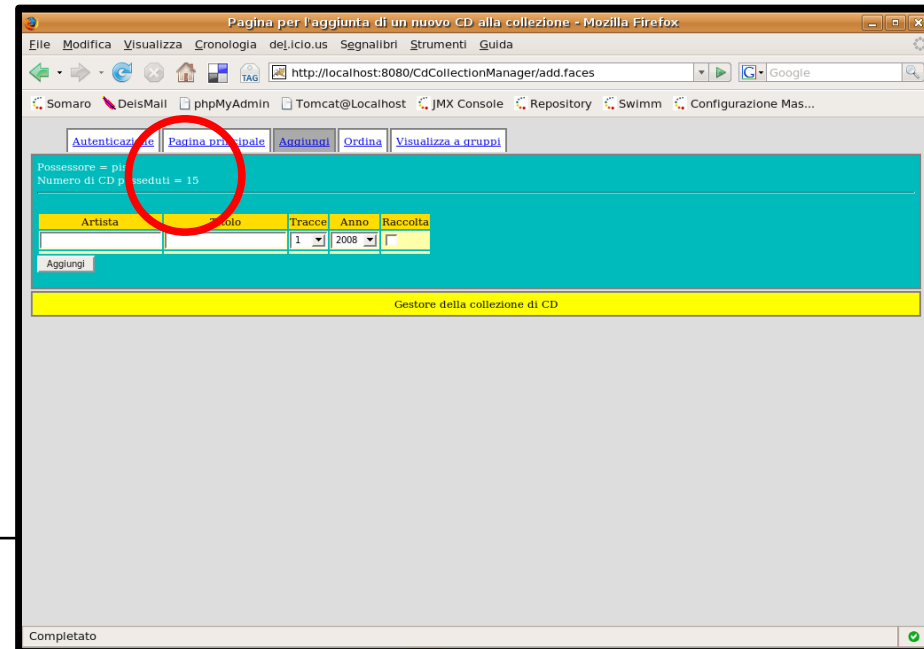
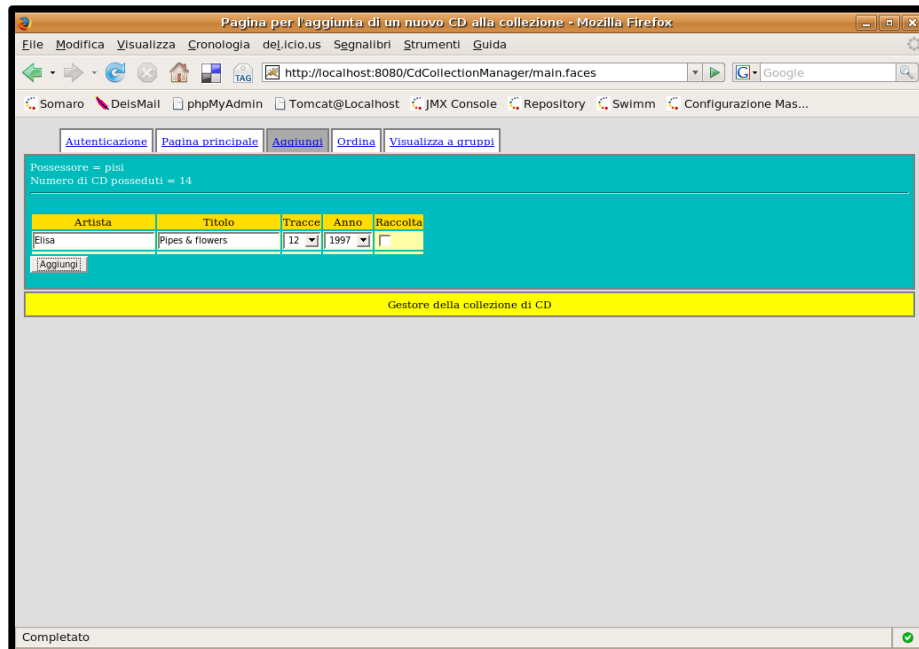
> **the same schema repeats itself** almost un-modified **in all the pages** that are backed by the manager

- **to hold the data referred by the page UI components**

- **to provide the methods that perform business logic on that data** (e.g., updating the database) **and return a suitable navigation action**

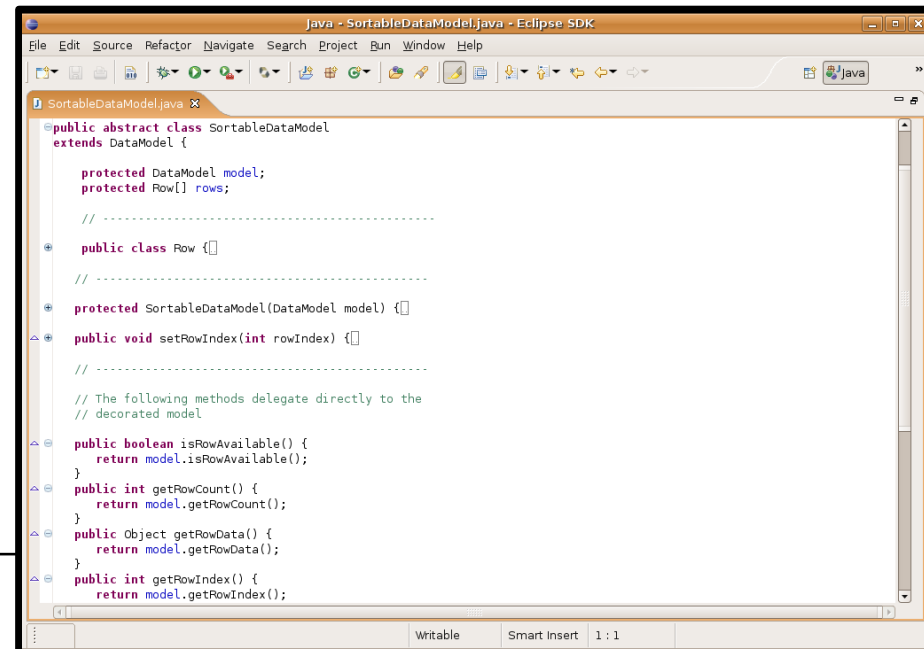
add.jsp page

- > no `<h:dataTable>` tags and `javax.faces.model.DataModel` objects this time
 - just a **member variable** of type `it.unibo.tw0708.dato.CompactDiscTO` in the backing bean and an `<h:panelGrid>` tag to show the cd fields to edit
 - page button invokes `manager` method `it.unibo.tw0708.beans.CdManager.insertCompactDisc()` to perform the business logic that entails:
 - writing the new entry to the database (by invoking DAO functionalities)
 - updating the cd list in the `manager` (by re-syncing it to the database)
 - redisplaying the current view (by returning a null action)



sort.jsp page (1/2)

- > Just some tricks you can play on the datamodel before displaying it
 - *manager* bean holds **another member variable** (different from the cd list we have already seen) **where to store the sorted datamodel list**
 - the abstract `it.unibo.tw0708.datamodel.sorting.SortableDataModel` class defines a private subclass **Row** to
 - **store row index for sorting** (**Row** has just an `int` field)
 - **provide an accessor method for returning the index-corresponding row from the datamodel object being wrapped by `SortableDataModel` class** (this is why **Row** is an inner class of it: *for the sake of visibility scope!*)
 - **SortableDataModel** then:
 - **wraps the traditional `javax.faces.DataModel` and holds the index array** (= array of **Row** objects) that reflects the current sorting
 - **bridges the rest of `DataModel` methods to the wrapped object**



```
public abstract class SortableDataModel
extends DataModel {
    protected DataModel model;
    protected Row[] rows;
    // .....
    public class Row {}
    // .....
    protected SortableDataModel(DataModel model) {}
    public void setRowIndex(int rowIndex) {}
    // .....
    // The following methods delegate directly to the
    // decorated model
    public boolean isRowAvailable() {
        return model.isRowAvailable();
    }
    public int getRowCount() {
        return model.getRowCount();
    }
    public Object getRowData() {
        return model.getRowData();
    }
    public int getRowIndex() {
        return model.getRowIndex();
    }
}
```

sort.jsp page (2/2)

> Just some tricks you can play on the datamodel before displaying it

- concrete class `it.unibo.tw0708.datamodel.sorting.SortableCdDataModel` extends `SortableDataModel` and provides methods to perform the actual sorting:

→ these methods just leverage `java.util.Array.sort()` function by providing suitable comparator objects...

→ if you want, you can have a look at the comparator classes in `it.unibo.tw0708.datamodel.sorting.comparators` package (they are just object implementing the

`java.util.Comparator` interface by providing a suitable `compare()` method that deals with `CdModel` objects)

- `<h:commandLink>`s in the page force resorting the model (by invoking `sort_xxx()` methods) and redisplaying the `<h:dataTable>` content accordingly

Artista su' ntu''	Titolo su' ntu''	Tracce su' ntu''	Anno su' ntu''	Raccolta
Madonna	American life	11	2003	false
R.E.M.	Around the sun	13	2004	false
R.E.M.	Automatic for the people	12	1992	false
The Cranberries	Bury the hatchet	14	1999	false
K's Choice	Cocoon crash	14	1998	false
ABBA	Gold	19	1992	true
Queen	Greatest Hits	17	1981	true
Queen	Innuendo	12	1991	false
Ani DiFranco	Little plastic castle	12	1998	false
Guns n' Roses	Live in Paris	22	1992	false
Queen	News of the world	11	1977	false
R.E.M.	Out of time	11	1991	false
K's Choice	Paradise in me	14	1995	false
Elsa	Pipes & flowers	12	1997	false
Cat Stevens	The very best of	18	1990	true

paging.jsp page (1/3)

> **paginating** data is a bit more complex (but it **avoids long-lasting queries** when the database holds huge tables → `<h:dataTable>` would load everything at once!!)

- again, we wrap the traditional `javax.faces.DataModel` into an enhanced datamodel object that provides additional features and we use it in the page:

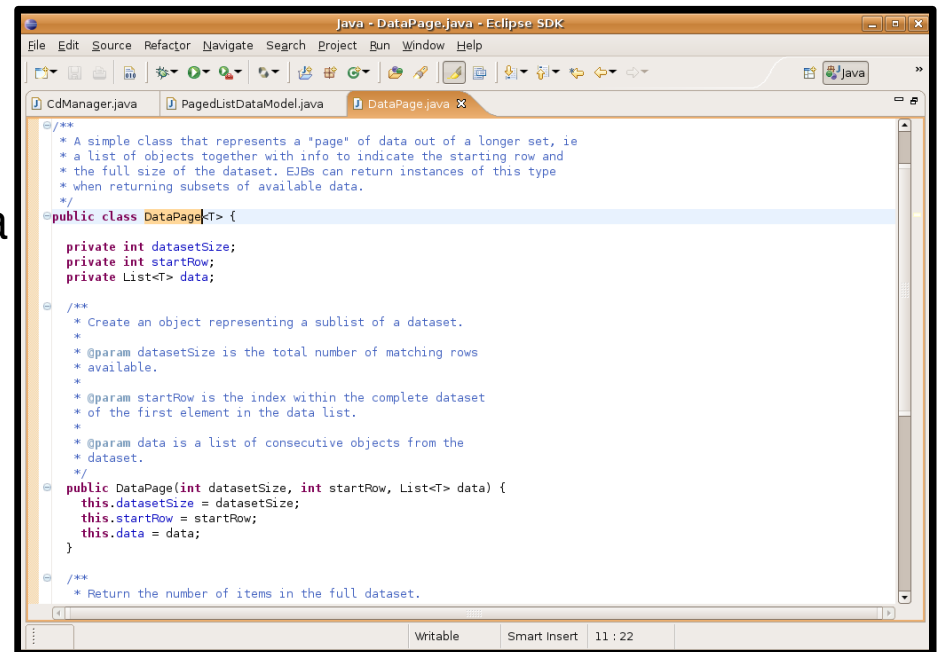
`it.unibo.tw0708.datamodel.paging.PagedCdDataModel`

- again, we gather non-datatype-dependent methods into an abstract superclass (→ **you can re-use this kind of classes for your own project data!**):

`it.unibo.tw0708.datamodel.paging.PagedListDataModel`

- this time we also need a generic helper class to store the current data page and additional related information such as starting row, page number, ecc:

`it.unibo.tw0708.datamodel.paging.DataPage<T>`



```
java - DataPage.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
CdManager.java PagedListDataModel.java DataPage.java
/**
 * A simple class that represents a "page" of data out of a longer set, i.e.
 * a list of objects together with info to indicate the starting row and
 * the full size of the dataset. EJBs can return instances of this type
 * when returning subsets of available data.
 */
public class DataPage<T> {
    private int datasetSize;
    private int startRow;
    private List<T> data;

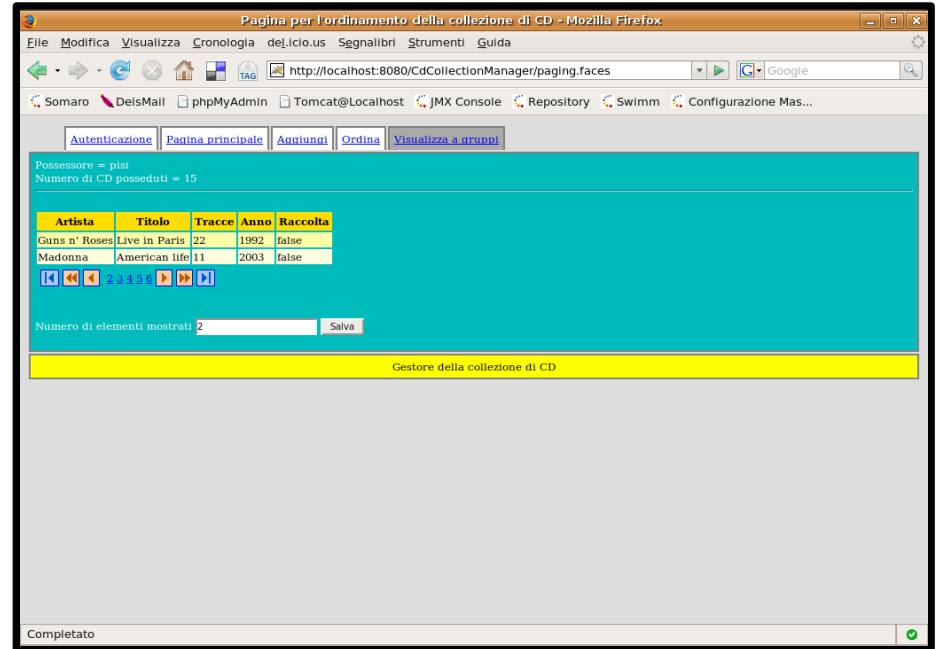
    /**
     * Create an object representing a sublist of a dataset.
     *
     * @param datasetSize is the total number of matching rows
     * available.
     *
     * @param startRow is the index within the complete dataset
     * of the first element in the data list.
     *
     * @param data is a list of consecutive objects from the
     * dataset.
     */
    public DataPage(int datasetSize, int startRow, List<T> data) {
        this.datasetSize = datasetSize;
        this.startRow = startRow;
        this.data = data;
    }

    /**
     * Return the number of items in the full dataset.
     */
}
```


paging.jsp page (2/3)

> furthermore, this time we are not going to create everything on our own: scroller controls **UI components** are provided by a third-party library: *Tomahawk* (from the *Apache group*)

- you can find *tomahawk-1.1.5-SNAPSHOT.jar* library under *WEB-INF/lib* path



- the JSF page declares and uses it just like the other tag libraries
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t" %>

paging.jsp page (3/3)

> *manager* bean holds the current page of the datamodel (= subset of datamodel rows) in another member variable: *pagedCompactDiscs*

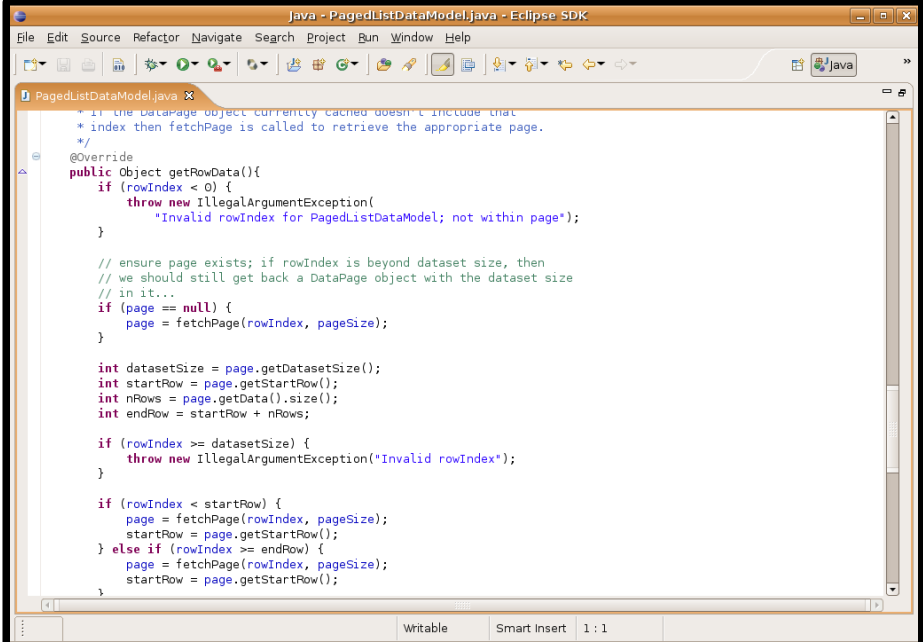
- `<h:dataTable>` iterates over data from this object
- controls from the `<t:dataScroller>` UI component do not invoke any *cd manager* method, but directly modify the datamodel object represented by *pagedCompactDiscs*, invoking its *getRowData()* and similar methods

- **it is the enhanced datamodel object that performs all the logic to return results page by page and to retrieve new data rows only when needed**

→ this is achieved by overriding the default behaviour of *getRowData()* method and alike ones

→ just try to reuse this features

into your projects: it's by far simpler than understanding things theoretically



```
Java - PagedListDataModel.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
PagedListDataModel.java
* If the DataPage object currently cached doesn't include that
* index then fetchPage is called to retrieve the appropriate page.
*/
@Override
public Object getRowData(){
    if (rowIndex < 0) {
        throw new IllegalArgumentException(
            "Invalid rowIndex for PagedListDataModel; not within page*");
    }

    // ensure page exists; if rowIndex is beyond dataset size, then
    // we should still get back a DataPage object with the dataset size
    // in it...
    if (page == null) {
        page = fetchPage(rowIndex, pageSize);
    }

    int datasetSize = page.getDatasetSize();
    int startRow = page.getStartRow();
    int nRows = page.getData().size();
    int endRow = startRow + nRows;

    if (rowIndex >= datasetSize) {
        throw new IllegalArgumentException("Invalid rowIndex*");
    }

    if (rowIndex < startRow) {
        page = fetchPage(rowIndex, pageSize);
        startRow = page.getStartRow();
    } else if (rowIndex >= endRow) {
        page = fetchPage(rowIndex, pageSize);
        startRow = page.getStartRow();
    }
}
```