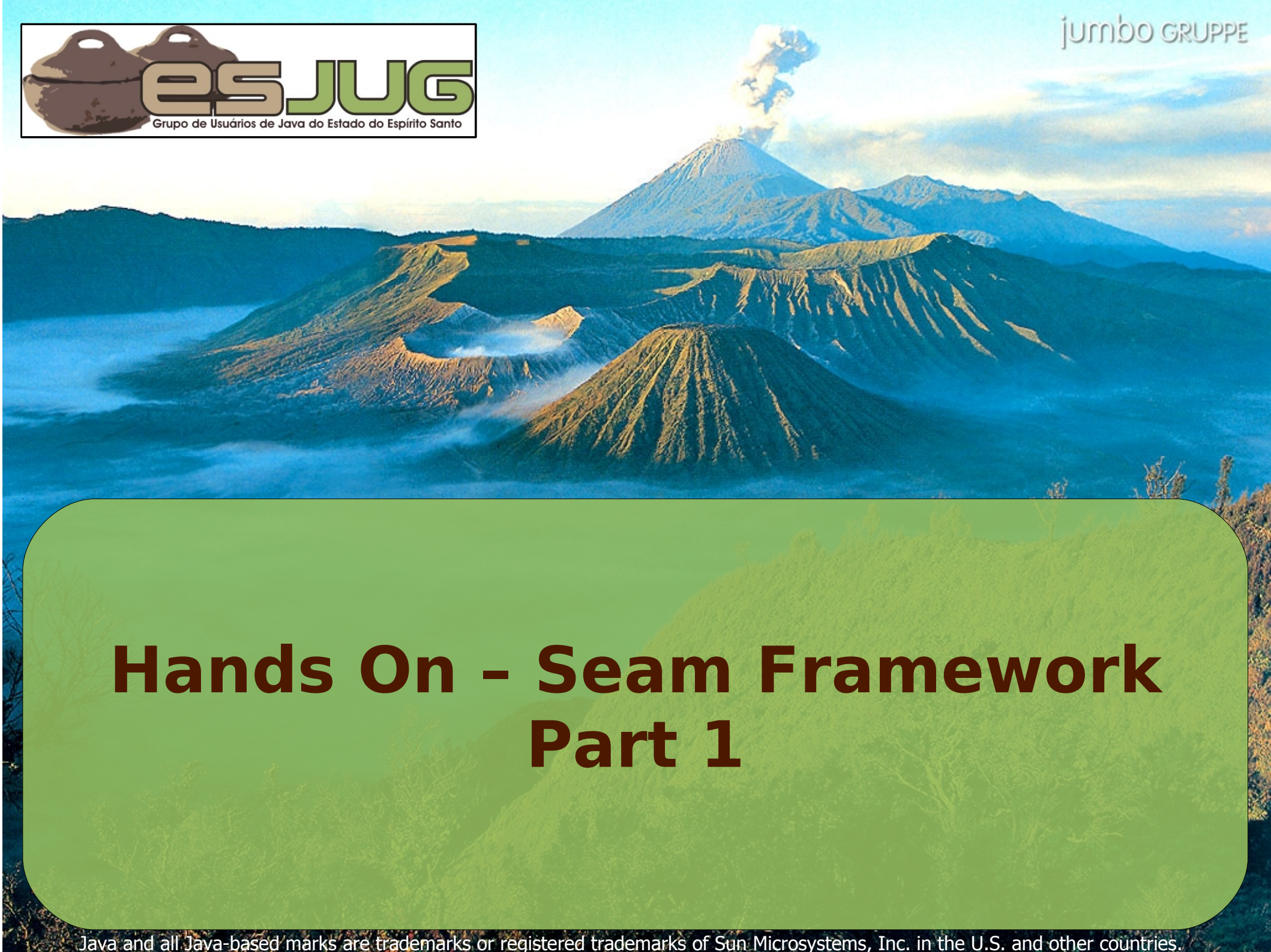


Hands On – Seam Framework Part 1

License for use and distribution

This material is available for non-commercial use and can be derived and/or redistributed, as long as it uses an equivalent license.

Attribution-Noncommercial-Share Alike 3.0 Unported

http://creativecommons.org/licenses/by-nc-sa/3.0/

You are free to share and to adapt this work under the following conditions: (a) You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work); (b) You may not use this work for commercial purposes. (c) If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

About the author – Vítor Souza

- Education:
  - Computer Science graduate, masters in Software Engineering – (UFES, Brazil), currently PhD at U. Trento.
- Java:
  - Developer since 1999;
  - Focus on Web Development;
  - Co-founder and coordinator of ESJUG (Brazil).
- Professional:
  - Substitute teacher at Federal University of ES;
  - Engenho de Software Consulting & Development.
- Contact: vitorsouza@gmail.com

Hands On Setup

- Assuming Java 6 installed beforehand;
- Procedures have been tested in an Kubuntu Linux 9.10 on Intel 32 bits. The examples are for this architecture, but can be adapted;
- `$base` = folder where we will install everything (e.g.: `~/HandsOnSeam`);
- `$base/Software` = folder containing the software to be installed (for Windows, Linux and Mac) – copy from USB drive.

Examples and demonstration

- Examples in the slides and the “hands on” demonstration are redundant...
  - ... so you can repeat it at home;
  - ... so you see it twice and learn better;
  - ... so I don't forget a step!

Ok, it's mostly because of the last one...

Part 1 – The Basics

- Install software;
- Create and deploy a blank project;
- Customize the layout with Facelets;
- Implement the domain with EJB 3;
- Generate the schema for the database;
- Implement data access via DAOs;
- Develop a simple CRUD.

Install Eclipse Galileo

- Download:
  - http://www.eclipse.org/downloads/
  - Eclipse IDE for Java EE Developers
  - Current stable version: Galileo SR1
- Install:
  - Extract the files into `$base`, creating the `$base/eclipse` folder.
- Example:

```
$ tar -zxf Software/eclipse-jee-galileo-SR1-linux-gtk.tar.gz
```

Install JBoss Tools

- Download:
  - http://www.jboss.org/tools/
  - Current development version: 3.1.0.M4
- Install:
  - Run Eclipse Galileo (create a workspace in $base);
  - Help > Install New Software... > Add... > Archive...;
  - Point to JbossTools-Update-3.1.0.v200910281724M-H247-M4.zip, click OK;
  - Select features to install, proceed, accept agreement and finish.

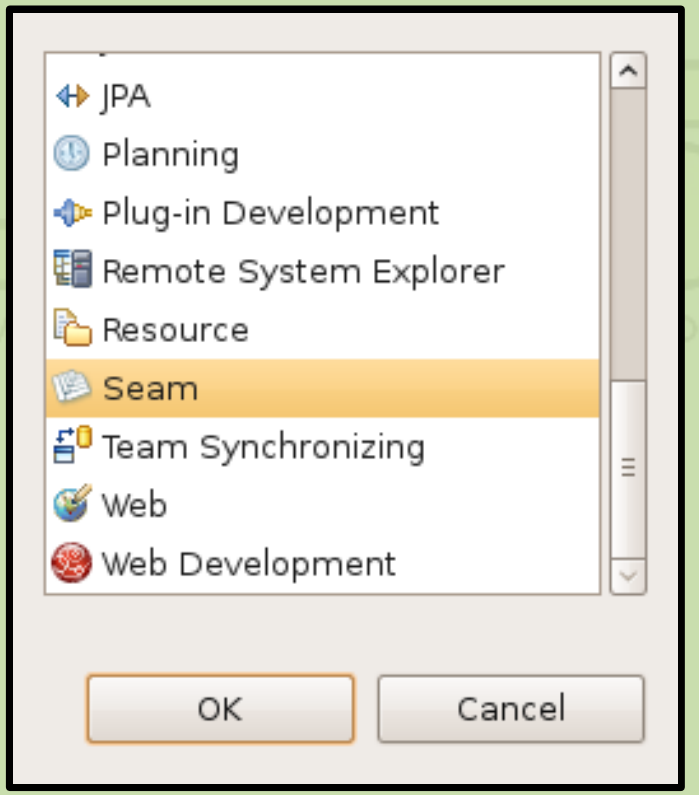
Install JBoss Application Server

- Download:
  - http://www.jboss.org/jbossas/downloads/
  - Current stable version: 5.1.0.GA
  - The 5.1 series is not yet integrated – let's use 5.0.1
- Install:
  - Extract files into `$base`. The folder `$base/jboss-5.0.1.GA` will be created.
- Example:

```
$ unzip Software/jboss-5.0.1.GA-jdk6.zip
```

Start Eclipse with JBoss Tools

- Restart eclipse;
- Check if JBoss Tools has been installed:
  - *Window > Open Perspective > Other* should show Seam perspective).



Configure JBoss AS in Eclipse

- Open the *Seam* perspective;
- Click on the *JBoss Server View*;
- Right-click in the blank area, *New > Server*;
  - Open JBoss Community (**not JBoss!**)
  - Select JBoss AS 5.0, Next >;
  - Fill in the correct Home Directory, Finish.
- Run the server and try opening `http://localhost:8080` in your browser.

Install JBoss Seam

- Download:
  - http://www.seamframework.org/Download
  - Current version: 2.2.0.GA
- Install:
  - Extract files into `$base`. The folder `$base/jboss-seam-2.2.0.GA` will be created.
- Example:

```
$ unzip Software/jboss-seam-2.2.0.GA.zip
```

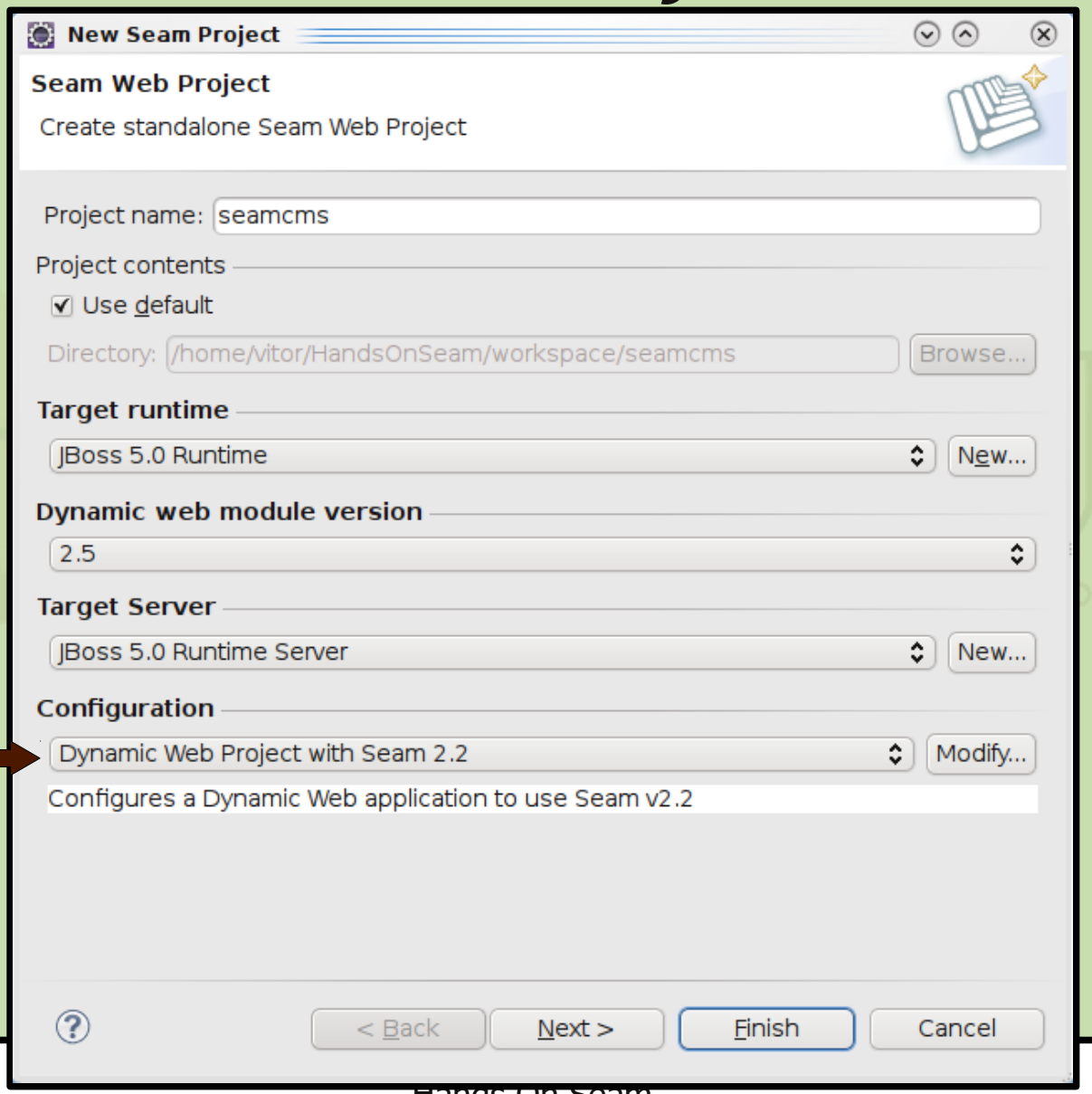
Install HSQLDB

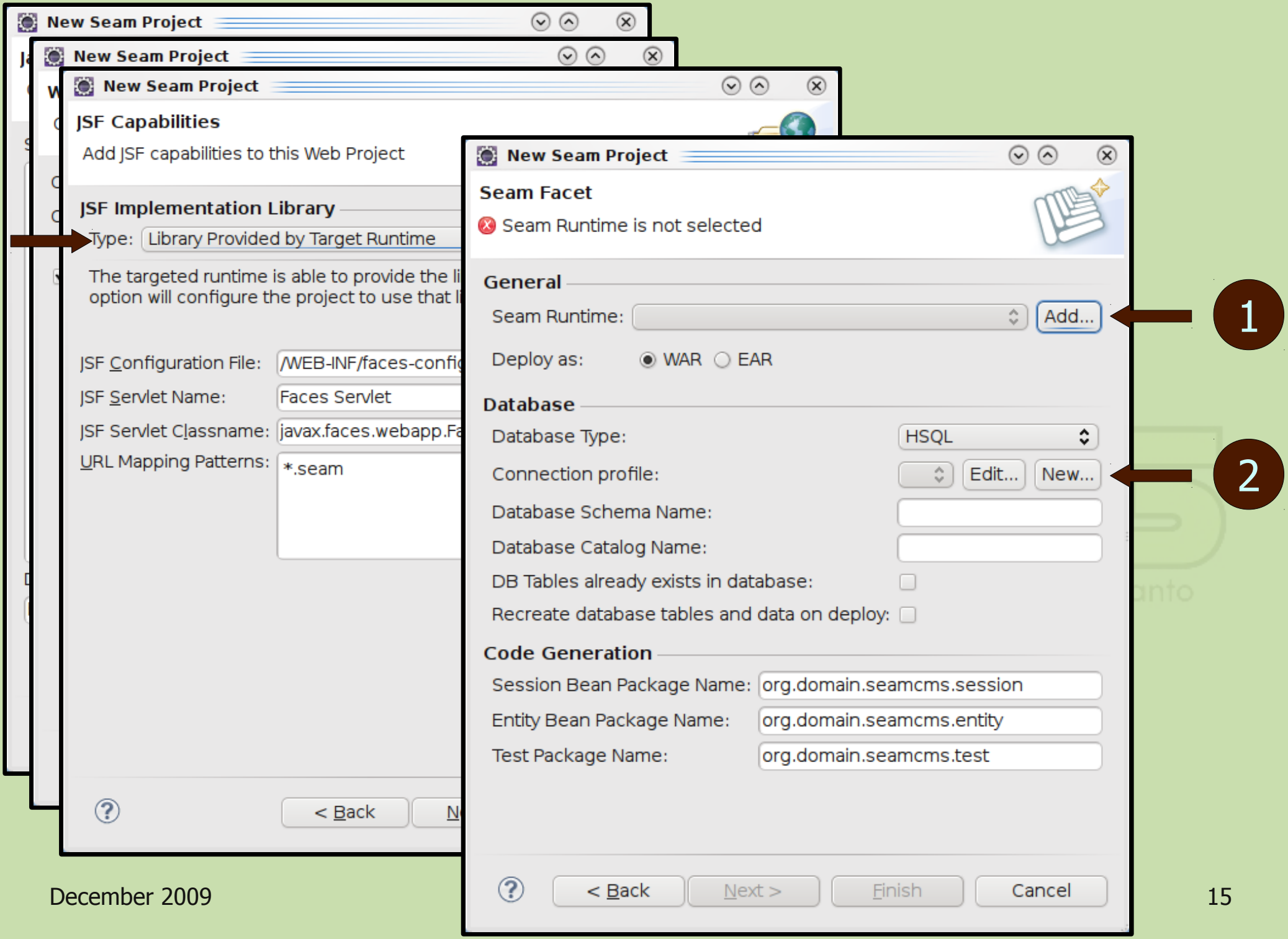
- Download:
  - http://hsqldb.org/
  - Current version: 1.8.1.1
- Install:
  - Extract files into `$base`. The folder `$base/hsqldb` will be created.
- Example:

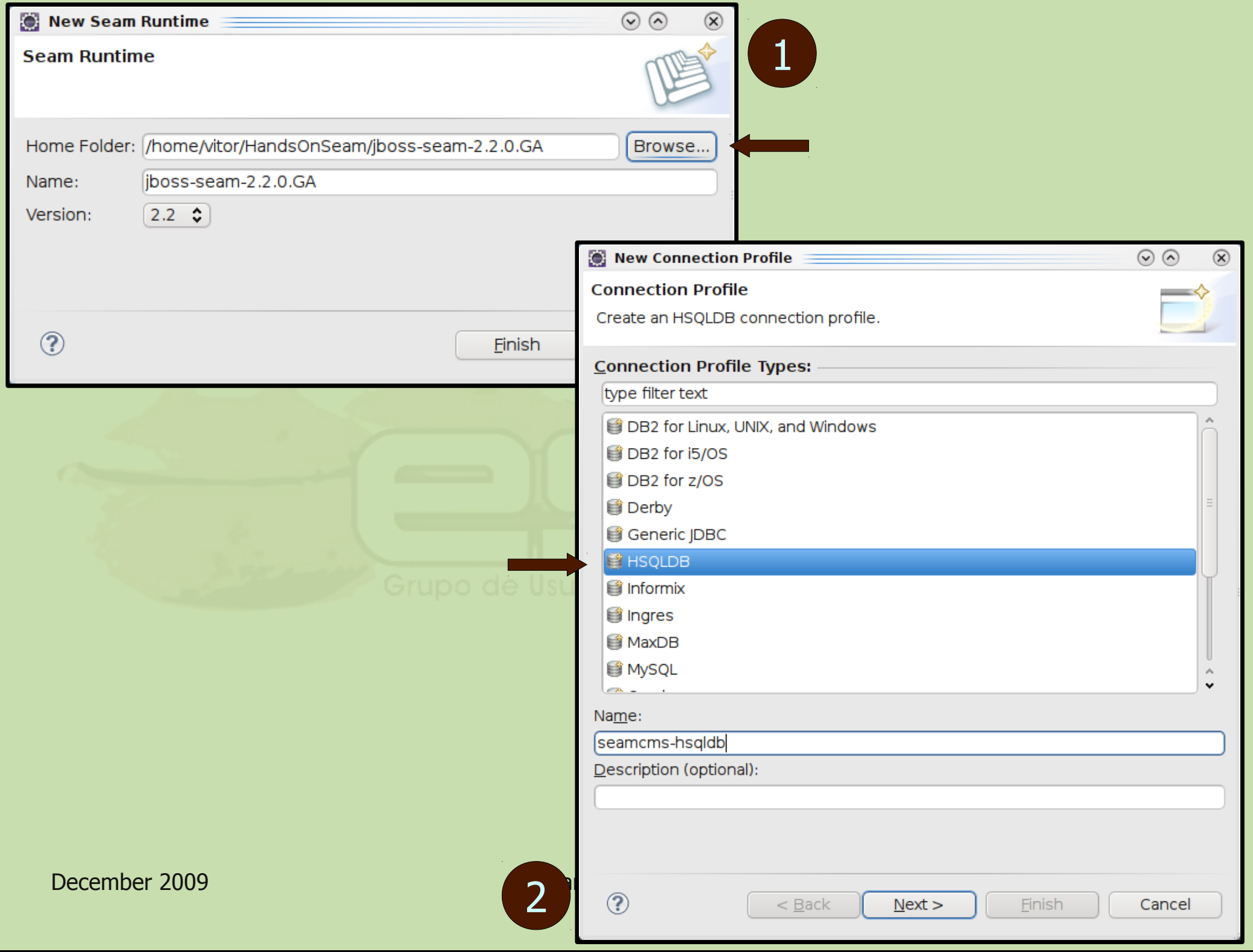
```
$ unzip Software/hsqldb_1_8_1_1.zip
```

Create a new Seam project

- *File > New > Seam Web Project...*

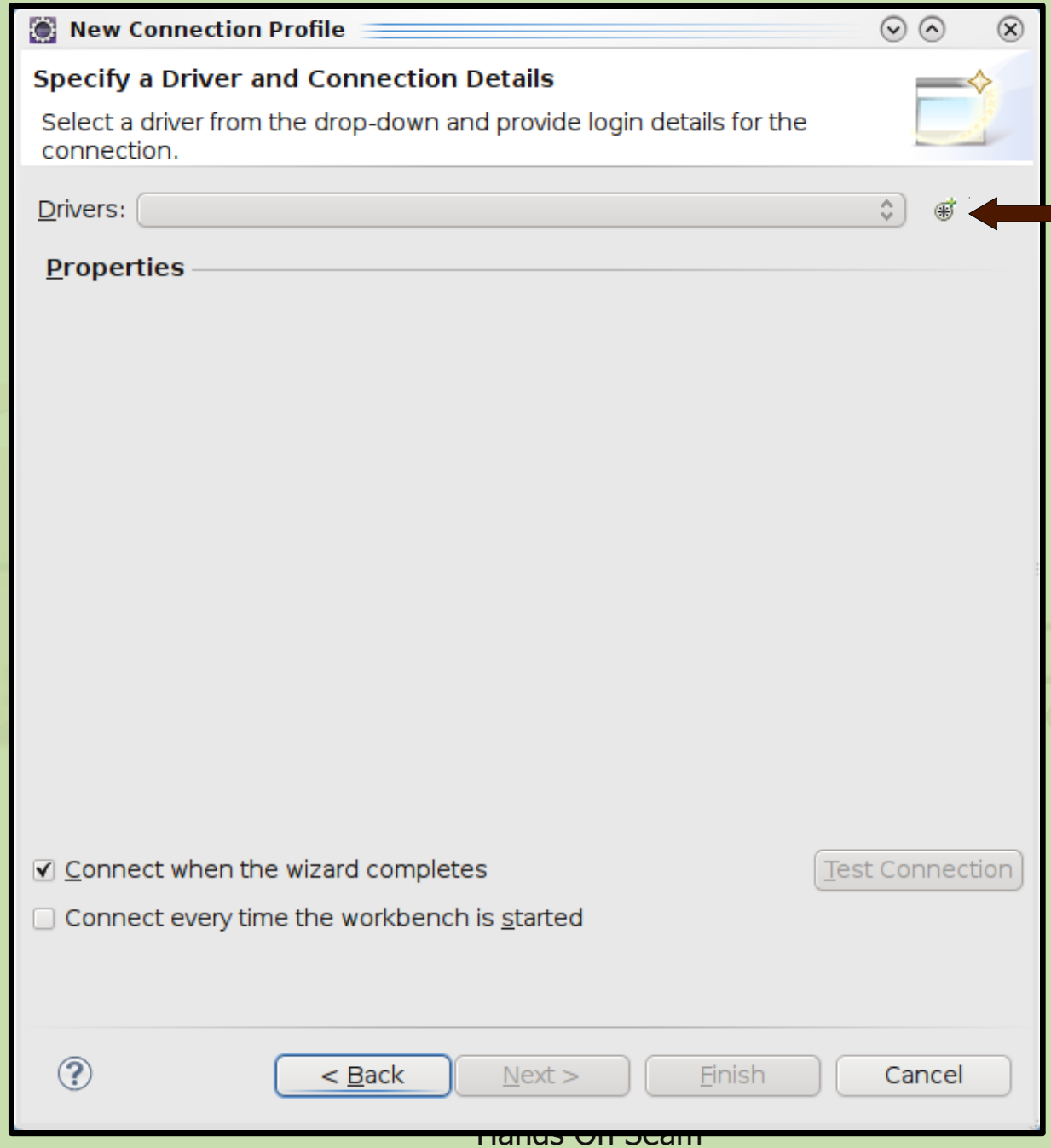


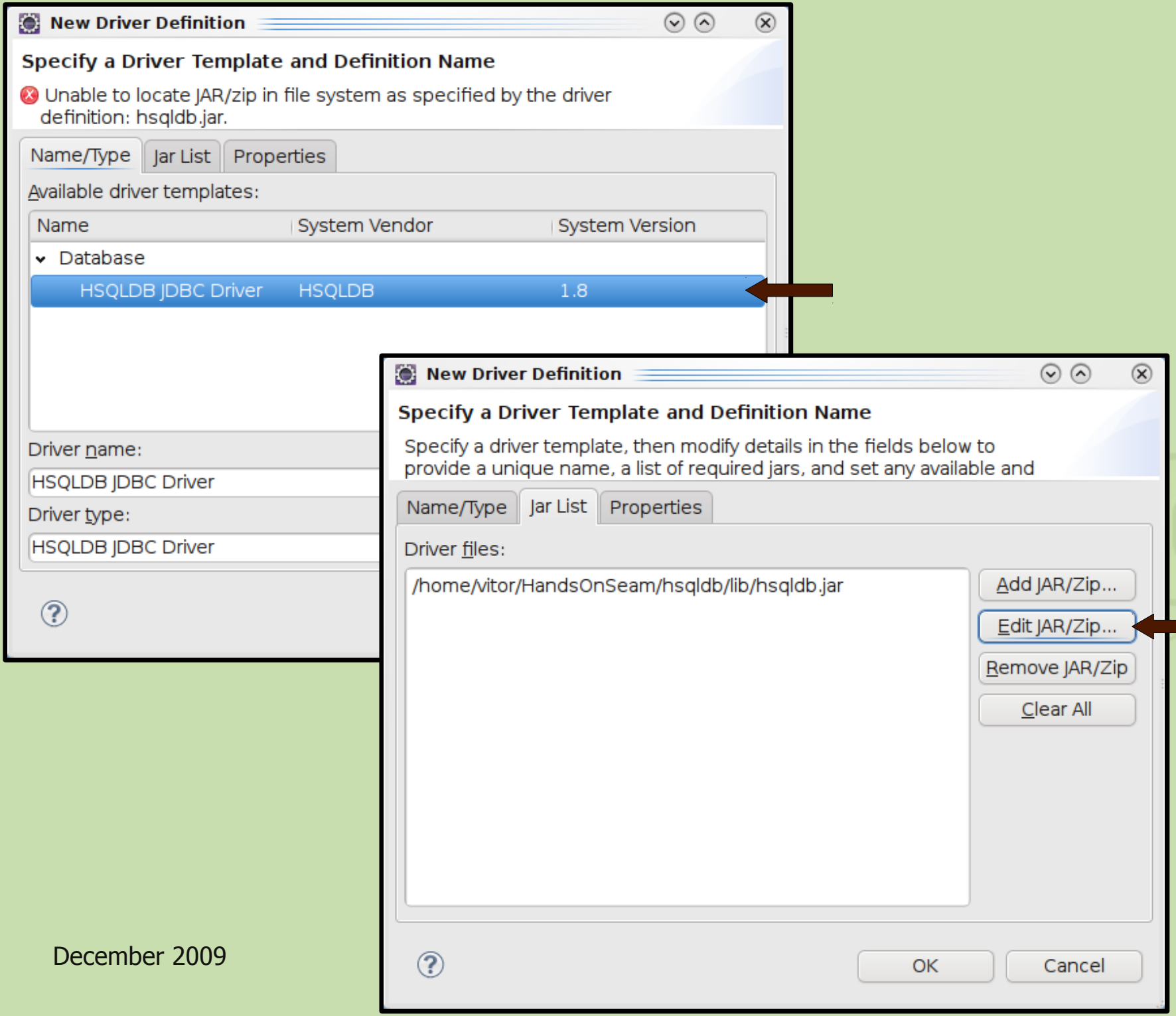

New Seam Project
New Seam Project
New Seam Project
JSF Capabilities
Add JSF capabilities to this Web Project
JSF Implementation Library
Type: Library Provided by Target Runtime
The targeted runtime is able to provide the li
option will configure the project to use that li
JSF Configuration File: /WEB-INF/faces-confi
JSF Servlet Name: Faces Servlet
JSF Servlet Classname: javax.faces.webapp.Fa
URL Mapping Patterns: *.seam
< Back
New Seam Project
Seam Facet
Seam Runtime is not selected
General
Seam Runtime:
Add...
Deploy as: WAR EAR
Database
Database Type: HSQL
Connection profile:
Edit...
New...
Database Schema Name:
Database Catalog Name:
DB Tables already exists in database:
Recreate database tables and data on deploy:
Code Generation
Session Bean Package Name: org.domain.seamcms.session
Entity Bean Package Name: org.domain.seamcms.entity
Test Package Name: org.domain.seamcms.test
< Back
Next >
Finish
Cancel
1
2

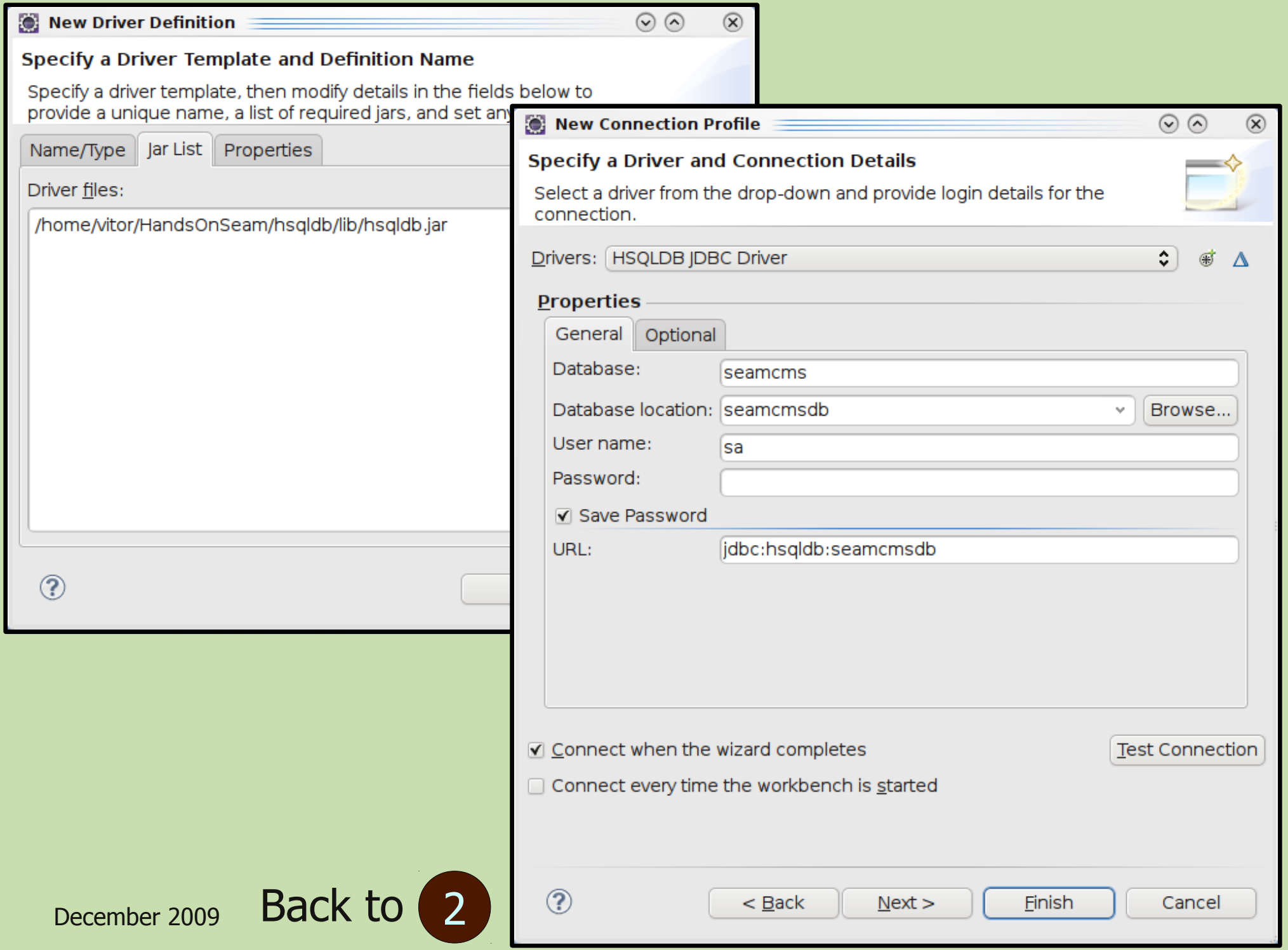


December 2009

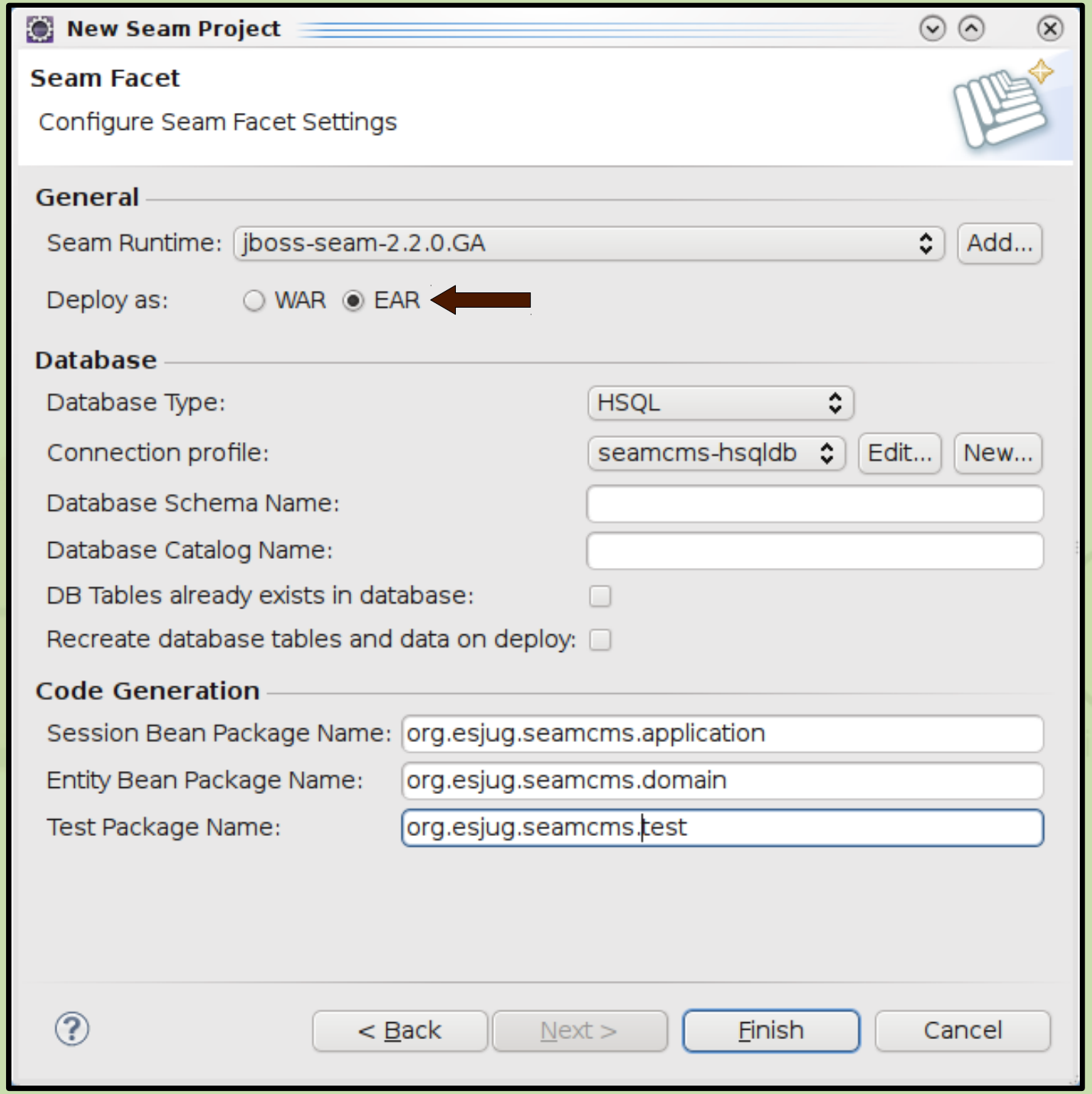
2 Continues...


New Connection Profile
Specify a Driver and Connection Details
Select a driver from the drop-down and provide login details for the connection.
Drivers:
Properties
Connect when the wizard completes
Test Connection
Connect every time the workbench is started
< Back
Next >
Finish
Cancel

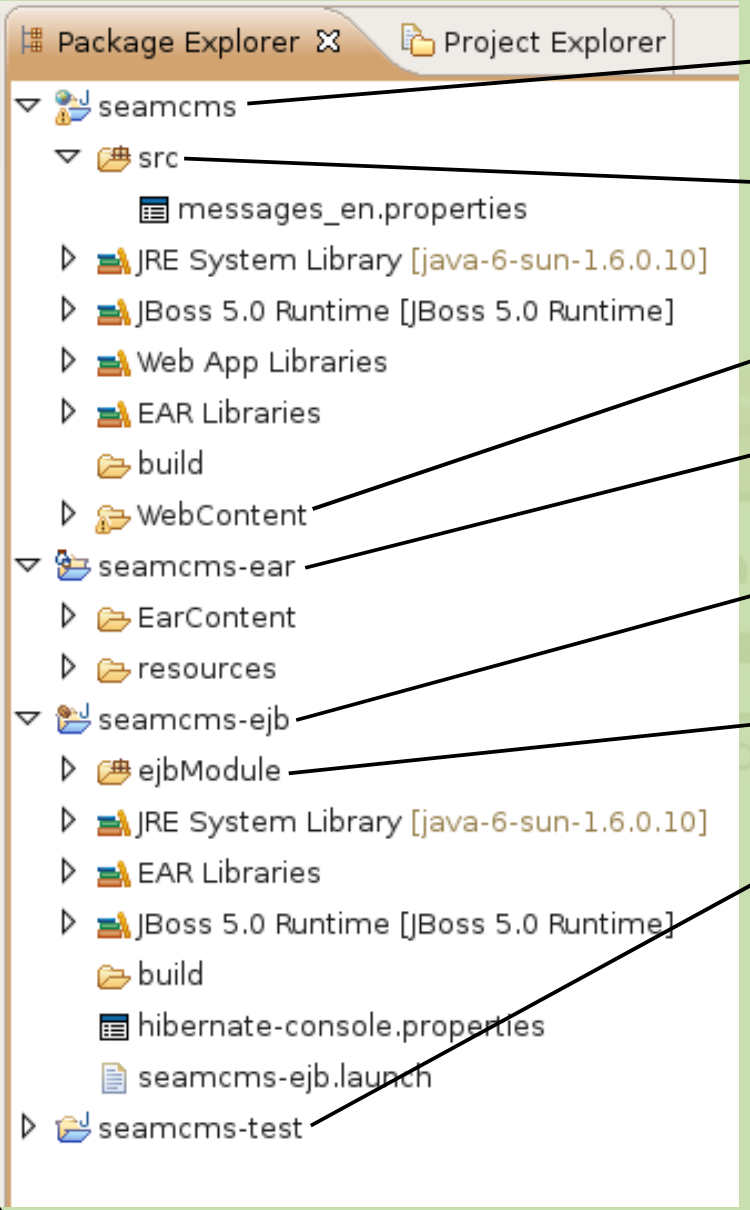

New Driver Definition
Specify a Driver Template and Definition Name
Unable to locate JAR/zip in file system as specified by the driver definition: hsqldb.jar.
Name/Type
Jar List
Properties
Available driver templates:
Name
System Vendor
System Version
Database
HSQLDB JDBC Driver
HSQLDB
1.8
Driver name:
HSQLDB JDBC Driver
Driver type:
HSQLDB JDBC Driver
New Driver Definition
Specify a Driver Template and Definition Name
Specify a driver template, then modify details in the fields below to provide a unique name, a list of required jars, and set any available and
Name/Type
Jar List
Properties
Driver files:
/home/vitor/HandsOnSeam/hsqldb/lib/hsqldb.jar
Add JAR/Zip...
Edit JAR/Zip...
Remove JAR/Zip
Clear All
OK
Cancel


New Driver Definition
Specify a Driver Template and Definition Name
Specify a driver template, then modify details in the fields below to
provide a unique name, a list of required jars, and set any
Name/Type
Jar List
Properties
Driver files:
/home/vitor/HandsOnSeam/hsqldb/lib/hsqldb.jar
New Connection Profile
Specify a Driver and Connection Details
Select a driver from the drop-down and provide login details for the
connection.
Drivers:
HSQLDB JDBC Driver
Properties
General
Optional
Database:
seamcms
Database location:
seamcmsdb
Browse...
User name:
sa
Password:
Save Password
URL:
jdbc:hsqldb:seamcmsdb
Connect when the wizard completes
Test Connection
Connect every time the workbench is started
< Back
Next >
Finish
Cancel

Back to 2


New Seam Project
Seam Facet
Configure Seam Facet Settings
General
Seam Runtime:
jboss-seam-2.2.0.GA
Add...
Deploy as:
WAR
EAR
Database
Database Type:
HSQL
Connection profile:
seamcms-hsqldb
Edit...
New...
Database Schema Name:
Database Catalog Name:
DB Tables already exists in database:
Recreate database tables and data on deploy:
Code Generation
Session Bean Package Name:
org.esjug.seamcms.application
Entity Bean Package Name:
org.esjug.seamcms.domain
Test Package Name:
org.esjug.seamcms.test
< Back
Next >
Finish
Cancel

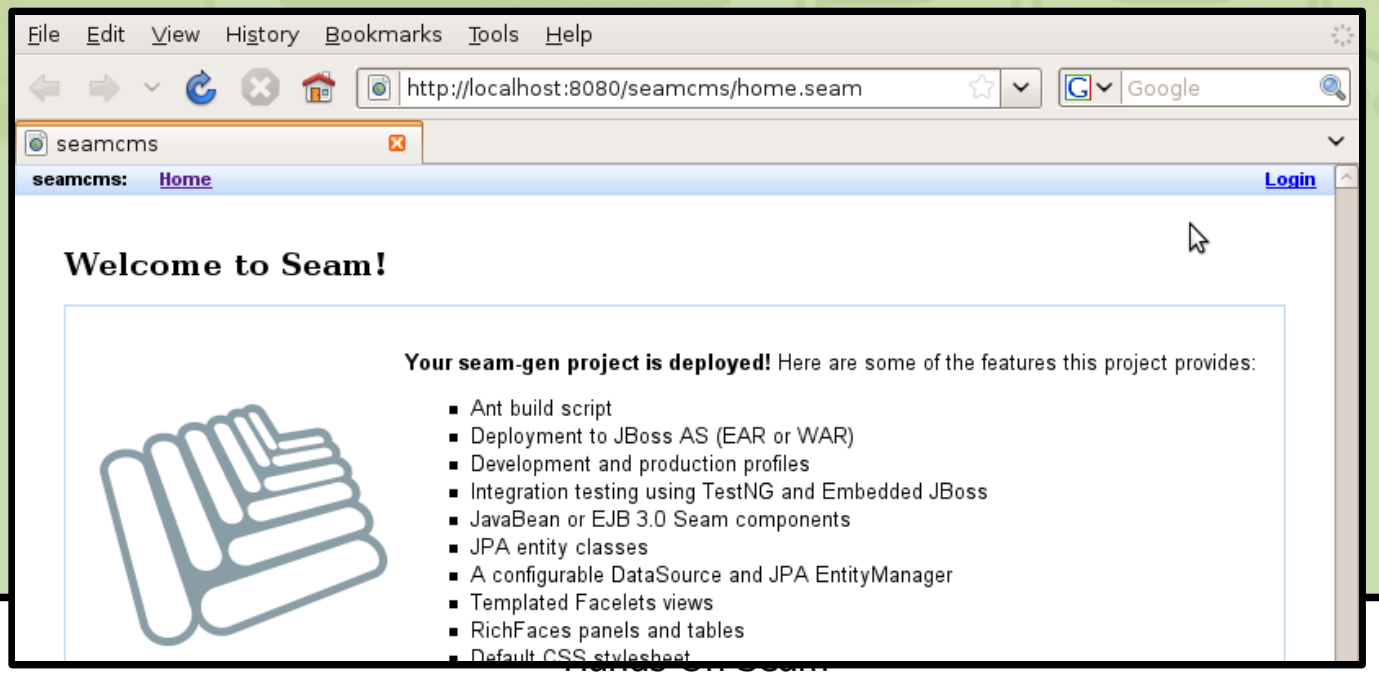
Seam Web project overview



- seamcms (Web):
  - src: resource bundles;
  - WebContent: web pages.
- seamcms-ear (EAR);
- seamcms-ejb (EJBs):
  - ejbModule: Java code;
- seamcms-test (tests).

Try your empty Seam project

- Deploy the project (if not done automatically):
  - Expand *JBoss 5.0 Server*;
  - Right-click *seamcms-ear > Publish*.
- Start the server;
- If you run into problems, try restarting the server.



Customize the layout with Facelets

```
<!-- WebContent/layout/template.xhtml -->
<f:view ...>
<html>
<head>
    ...
    <ui:insert name="head" />
</head>
<body>
  ...
    <ui:insert name="body" />
  ...
</body>
</html>
</f:view>
```

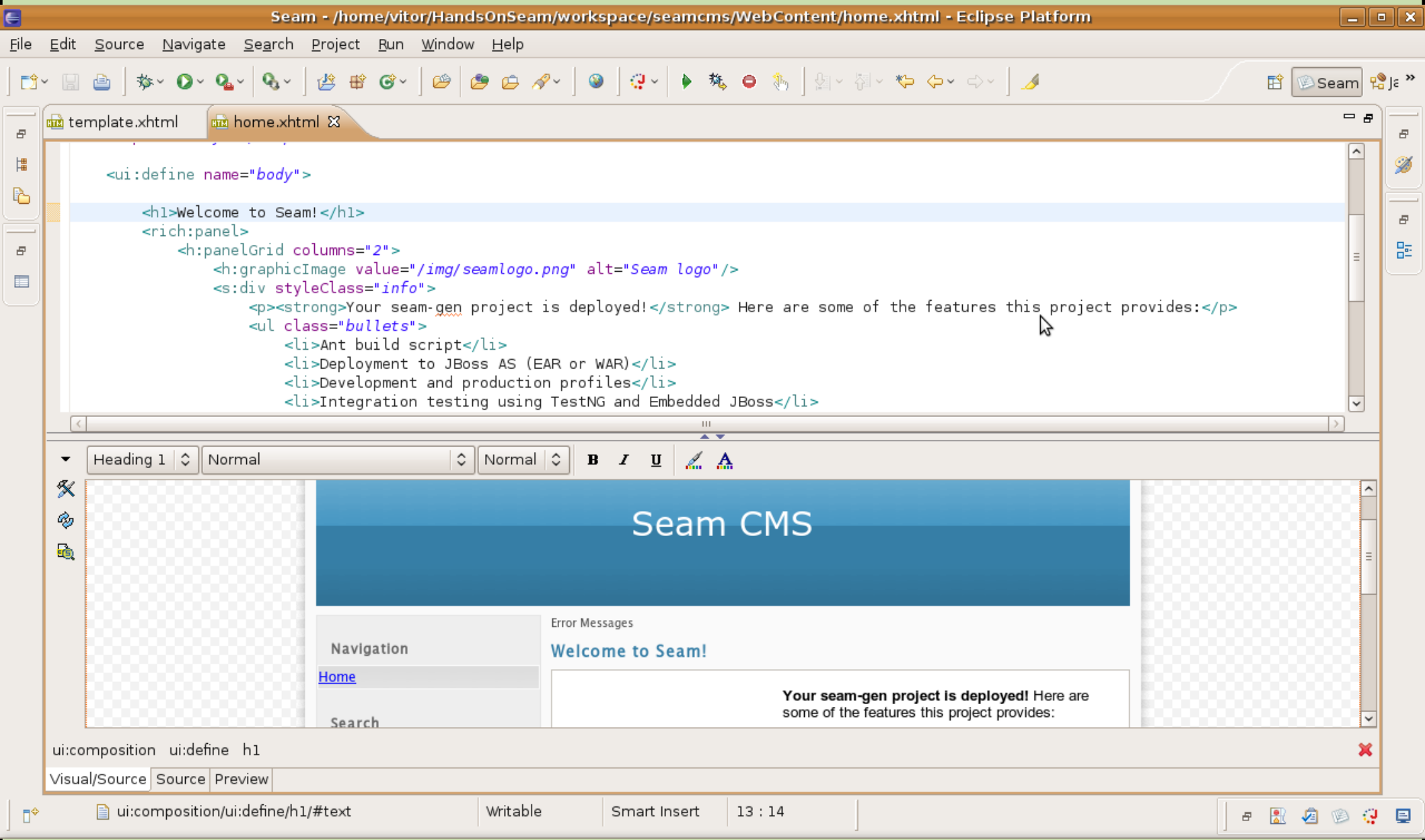
Customize the layout with Facelets

```
<!-- WebContent/home.xhtml-->
<ui:composition ...
   template="layout/template.xhtml">

   <ui:define name="body">
      <h1>Welcome to Seam!</h1>

      ...
      </ui:define>
</ui:composition>
```

Customize the layout with Facelets

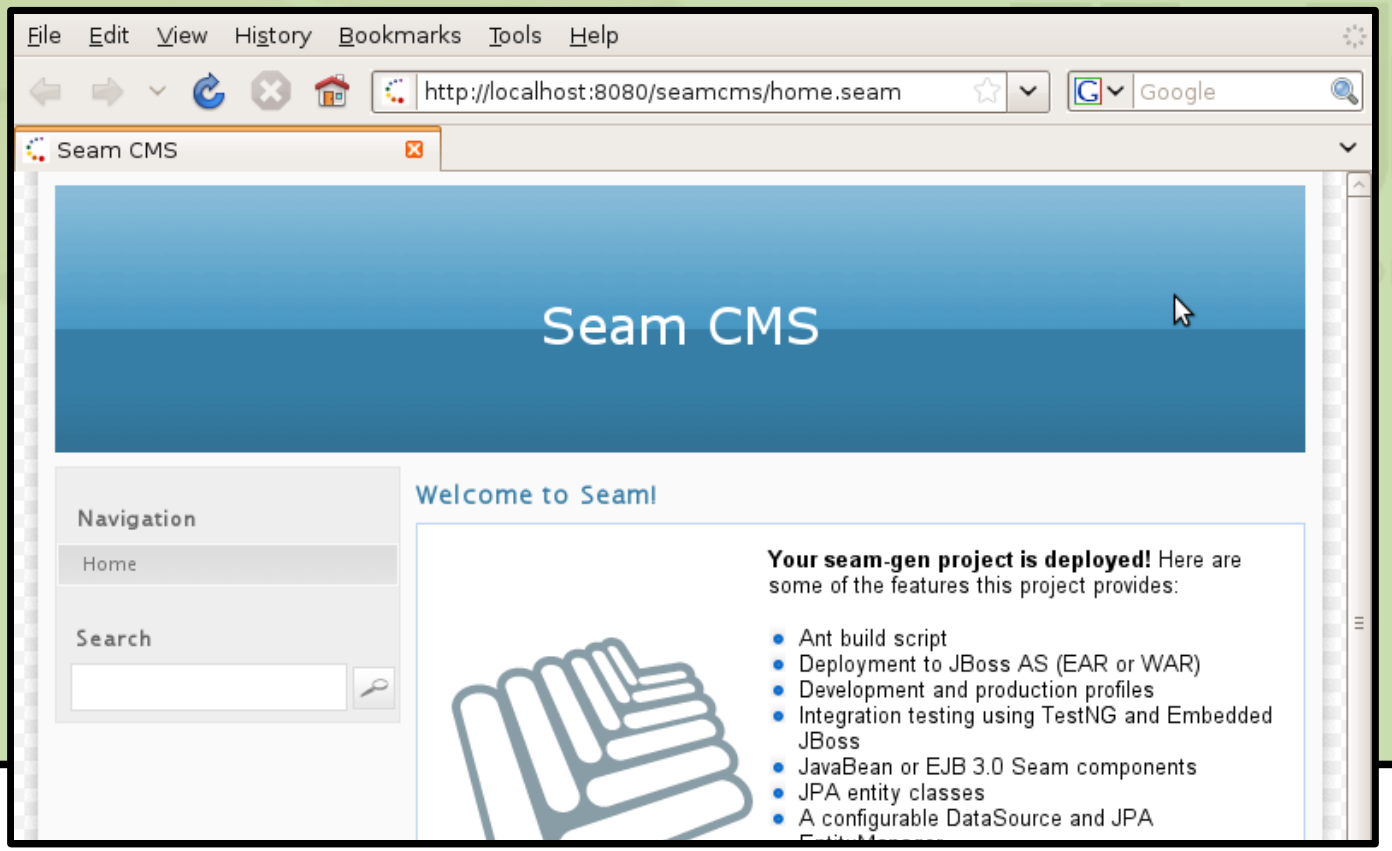
- Expand the `transparentia.zip` file under `$base/Layout;`
- Copy `default.css` and `img` to `seamcms/WebContent;`
- Adapt the template using `index.html`: `seamcms/WebContent/layout/template.xhtml`.
  - An already adapted is provided if you want to skip this step: `$base/Layout/template.xhtml`.
- Full publish of the application and test.

Try the XHTML editor



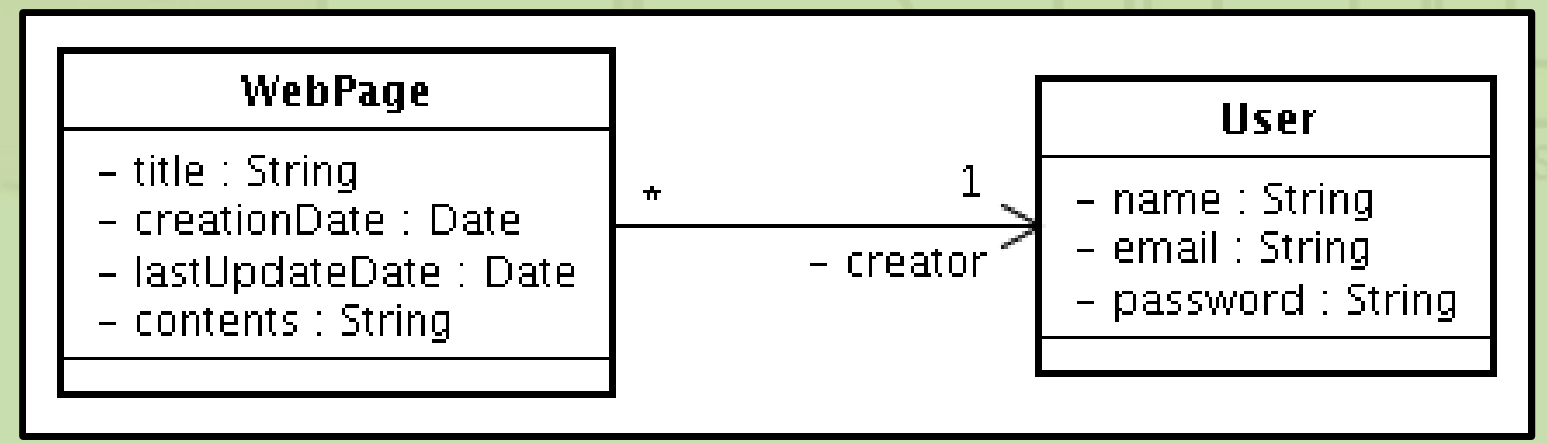
Publish Incrementally

- *JBoss Server View*, server already started:
  - Expand *JBoss 5.0 Server*;
  - Right-click *seamcms-ear* > *Full Publish*.
- Refresh the browser...



Implement the domain

- Very simple CMS;
- Users' names, e-mails and passwords are registered so they can create/edit content;
- WebPage is the only available content.



Implement the domain

- Create `domain` package;
- A superclass for domain objects might be useful:
  - All of them have IDs for persistence;
  - Optimistic locking (versioning) is a good idea;
  - Using UUIDs to implement `hashCode()` and `equals()` can save us a lot of time...
- Create the abstract class `DomainObject` as a mapped superclass with persistence annotation.

Implement DomainObject

```
@MappedSuperclass
public abstract class DomainObject
                                    implements Serializable {
   @Basic
   @Column(nullable = false, length = 40)
   protected String uuid;

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   @Version
   @Column(nullable = false)
   private Long version;
```

Implement DomainObject

```
public DomainObject() {
   uuid = UUID.randomUUID().toString();
}

/* Getters and setters. */

@Override
public boolean equals(Object obj) {
   // Checks if the class is the same.
   if ((obj == null) ||
                (!getClass().equals(obj.getClass())))
      return false;

   // Checks if the UUID is the same.
   DomainObject o = (DomainObject) obj;
   return uuid.equals(o.uuid);
}
```

Implement DomainObject

```
   @Override
   public int hashCode() {
      return uuid.hashCode();
   }

   @Override
   public String toString() {
      return "Instance of " + getClass().getName() +
         " (id: " + id + "; uuid: " + uuid + ")";
   }
}
```

Implement the domain

- In `DomainObject` we used standard persistence annotation:

```
@Basic
@Column(nullable = false, length = 40)
protected String uuid;
```

- Create the classes `User` and `WebPage`, but this time use Hibernate Validator:
  - Integrates with Seam to provide validation;
  - `@NotNull`
  - `@Length(min=9, max=99)`

Implement User

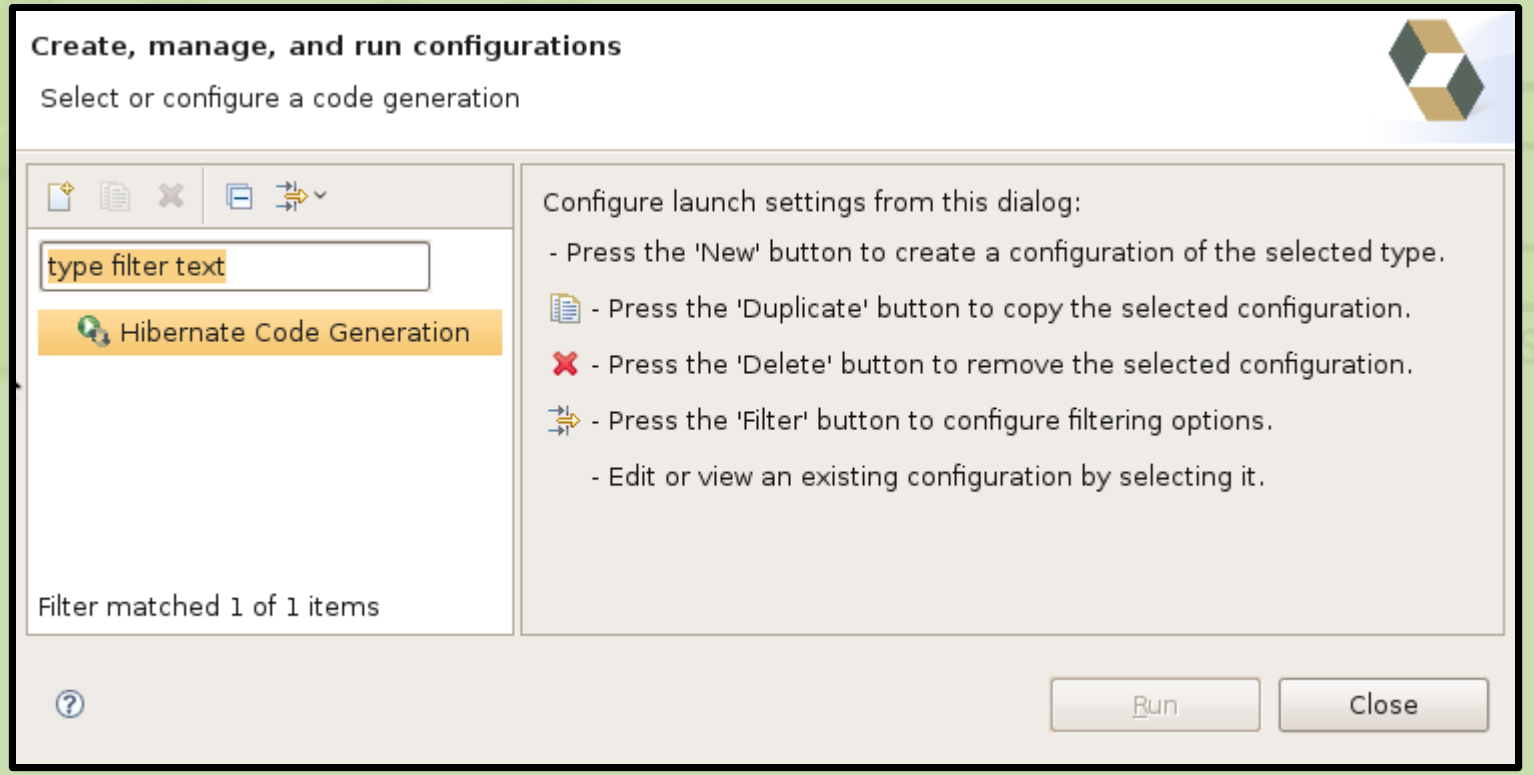
```
@Entity
public class User extends DomainObject {
   @Basic
   @NotNull
   @Length(max = 50)
   private String name;

   @Basic
   @NotNull
   @Length(max = 100)
   private String email;

   @Basic
   @NotNull
   @Length(max = 32)
   private String password;

   /* Getters and setters. */
}
```

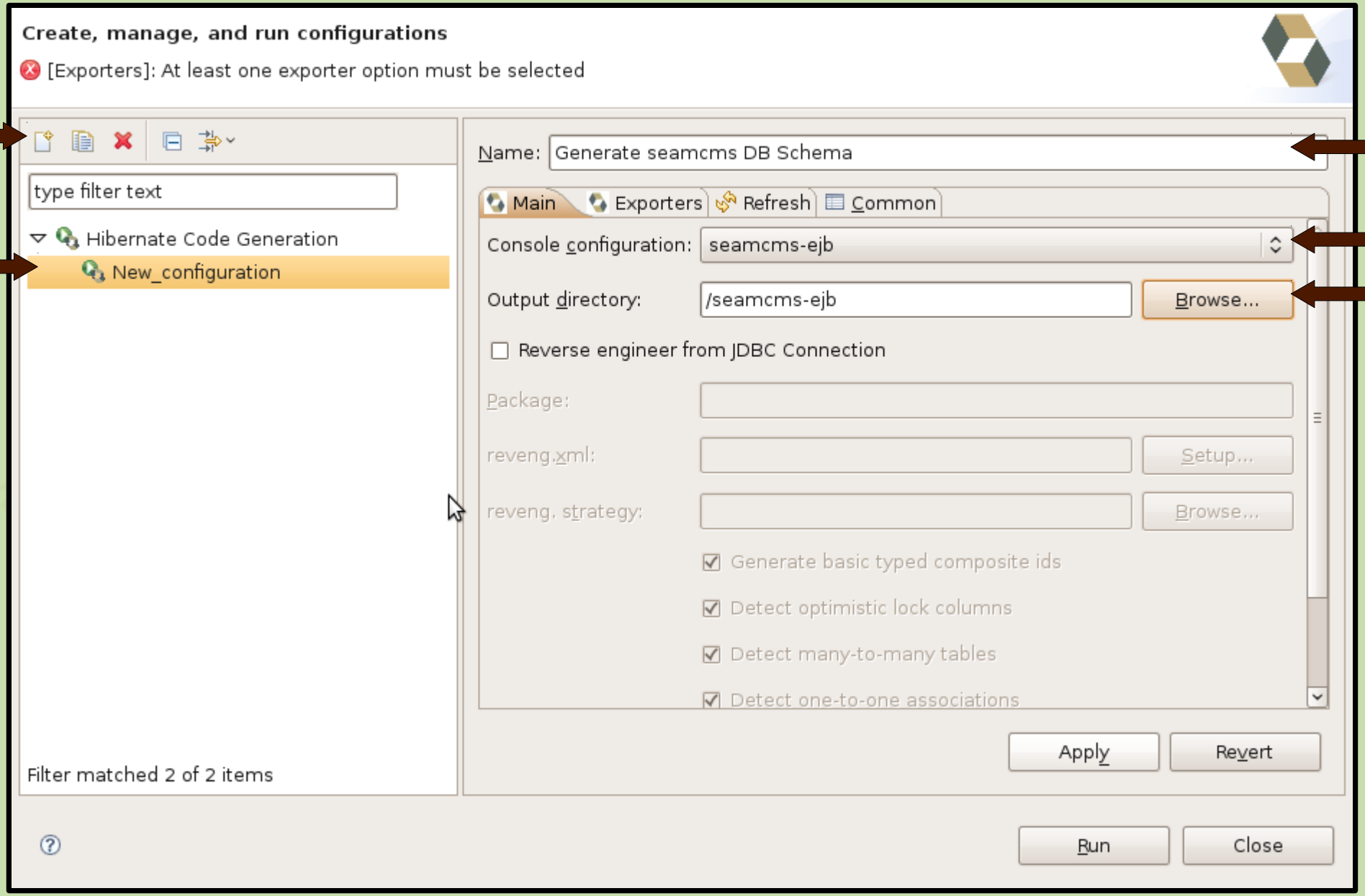
Implement WebPage

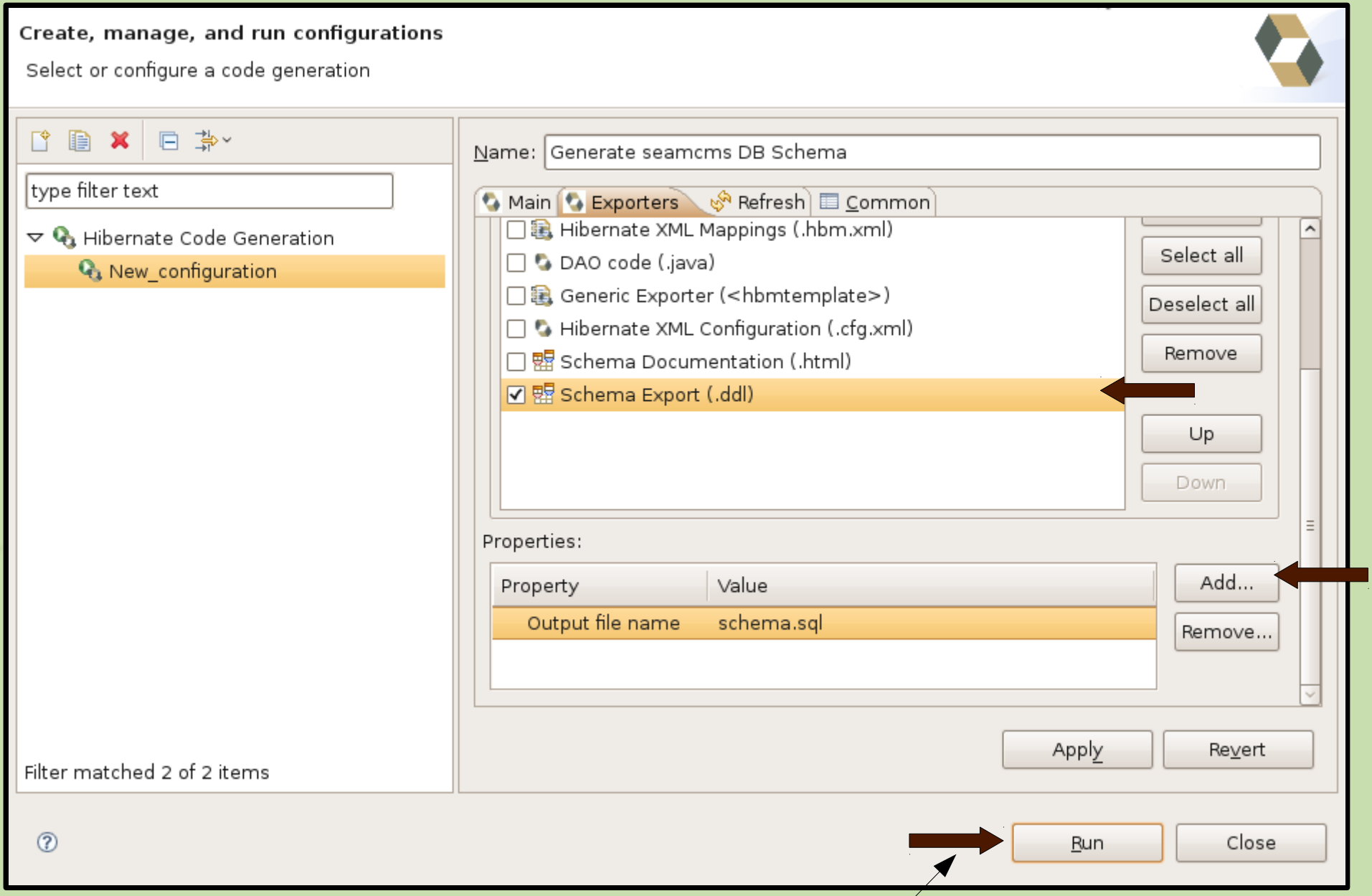
```
@Entity
public class WebPage extends DomainObject {
    @Basic
    @NotNull
    @Length(max = 200)
    private String title;

    @Temporal(TemporalType.TIMESTAMP)
    @NotNull
    private Date creationDate;

    @Temporal(TemporalType.TIMESTAMP)
    @NotNull
    private Date lastUpdateDate;

    @Lob
    @NotNull
    private String contents;

    /* Getters and setters. */
}
```

Generate DB schema from classes

- Switch to *Hibernate* perspective;
- *Run > Hibernate Code Generation... > Hibernate Code Generation Configurations...*




Create, manage, and run configurations
[Exporters]: At least one exporter option must be selected
type filter text
Hibernate Code Generation
New_configuration
Filter matched 2 of 2 items
Name: Generate seamcms DB Schema
Main
Exporters
Refresh
Common
Console configuration: seamcms-ejb
Output directory: /seamcms-ejb
Browse...
Reverse engineer from JDBC Connection
Package:
reveng.xml:
Setup...
reveng. strategy:
Browse...
Generate basic typed composite ids
Detect optimistic lock columns
Detect many-to-many tables
Detect one-to-one associations
Apply
Revert
Run
Close

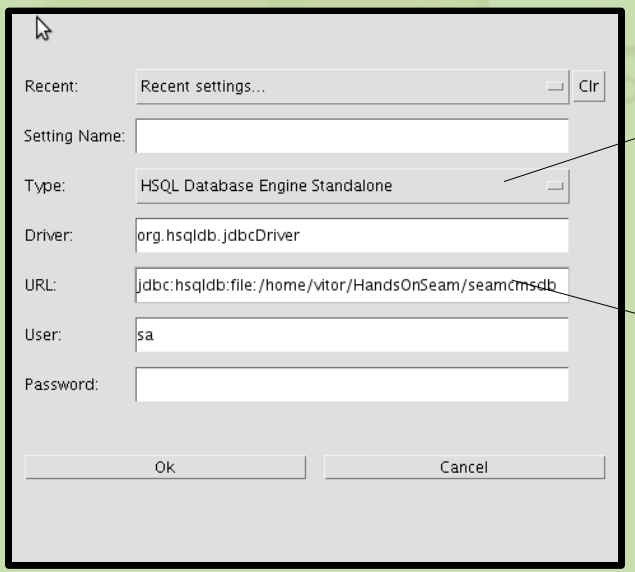


Must shutdown JBoss Server first!

Generate DB schema from classes

- The schema is written in `seamcms-ejb/schema.sql`;
- It's already executed in the database. Check it out with HSQLDB Manager.

```
$ cd hsqldb/lib
$ java -cp hsqldb.jar org.hsqldb.util.DatabaseManager
```



HSQL Database Engine Standalone

jdbc:hsqldb:file:*$path*/seamcmsdb

Check lock file first!

Use the DAO pattern for data access

- Create `persistence` package;
- A superclass for DAOs might be useful:
  - Basic operations are implemented the same way, no matter the class of the object being persisted.

```
public interface BaseDAO<T extends DomainObject> {
    long retrieveCount();

    List<T> retrieveAll();

    T retrieveById(Long id);

    void save(T object);

    void delete(T object);
}
```

Implement SeamBaseDAO

```
public abstract class SeamBaseDAO<T extends
                    DomainObject> implements BaseDAO<T> {
   @Logger
   private Log log;

   // Concrete subclasses must provide entity manager.
   protected abstract EntityManager getEntityManager();

   // Base operations need to know the class.
   protected abstract Class<T> getDomainClass();

   // Optionally, subclasses can define order.
   protected String getOrderByClause() {
      return "";
   }
```

Implement SeamBaseDAO

```
@SuppressWarnings("unchecked")
@Transactional
public List<T> retrieveAll() {
   log.info("Retrieving all #0",
                              getDomainClass().getName());
   Query query = getEntityManager().createQuery("from "
                  + getDomainClass().getName() + " obj "
                  + getOrderByClause());
   return query.getResultList();
}

@Transactional
public T retrieveById(Long id) {
   log.info("Retrieving #0 with id #1",
                            getDomainClass().getName(), id);
   return (T) getEntityManager().find(getDomainClass(),
                                                    id);
}
```

Implement SeamBaseDAO

```
@SuppressWarnings("unchecked")
@Transactional
public long retrieveCount() {
   Query query = getEntityManager().createQuery(
               "select new map (count(obj) as num) from "
               + getDomainClass().getName() + " obj");
   Map<String, Object> data = (Map<String, Object>)
                                    query.getSingleResult();
   Long count = (Long) data.get("num");
   log.info("Retrieving count for #0: #1",
                        getDomainClass().getName(), count);
   return (count == null) ? 0 : count;
}
```

Implement SeamBaseDAO

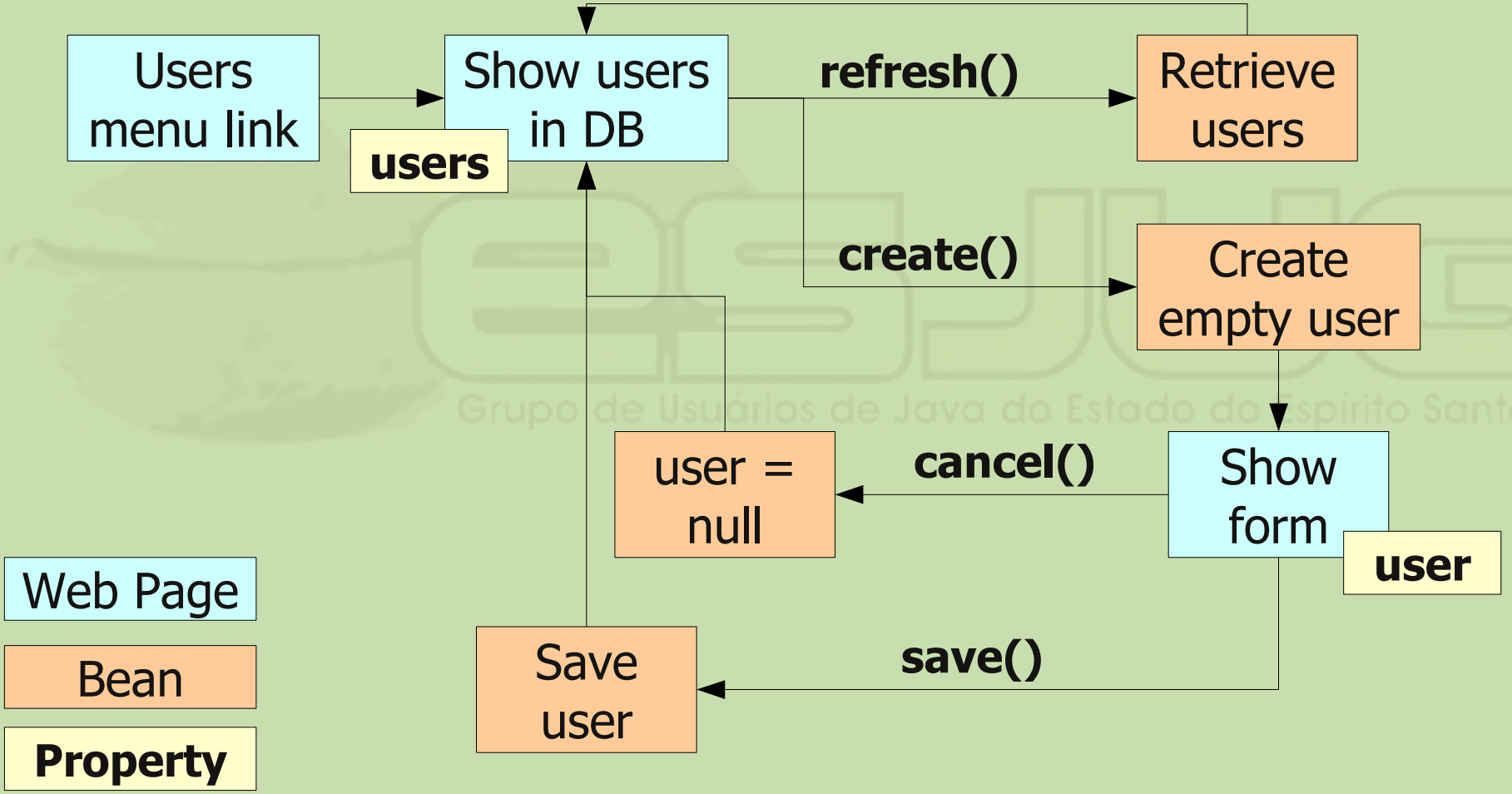
```
    @Transactional
    public void save(T object) {
        log.info("Saving #0: #1", getDomainClass().getName(),
                                                    object);
        getEntityManager().persist(object);
    }

    @Transactional
    public void delete(T object) {
        log.info("Deleting #0: #1",
                          getDomainClass().getName(), object);
        getEntityManager().remove(object);
    }
}
```

Implement UserDAO and SeamUserDAO

```
public interface UserDAO extends BaseDAO<User> { }

@AutoCreate
@Name("userDAO")
@Scope(ScopeType.APPLICATION)
@Stateless
public class SeamUserDAO extends SeamBaseDAO<User>
                                        implements UserDAO {
   @PersistenceContext
   private EntityManager entityManager;

   @Override
   protected Class<User> getDomainClass() {
      return User.class;
   }

   @Override
   protected EntityManager getEntityManager() {
      return entityManager;
   }
}
```

Implement CRUD of users

- Implement user creation first:



Implement CRUD of users

- 1st, define a local interface for a session bean:

```
@Local
public interface UserCrud {
   List<User> getUsers();

   void create();

   void save();

   void cancel();

   void refresh();
}
```

Implement CRUD of users

- Then, create the bean as a component:
  - One for each user (Session scope);
  - Keeps state.

```
@AutoCreate
@Name("userCrud")
@Scope(ScopeType.SESSION)
@Stateful
public class UserCrudService implements UserCrud {
```

- Note:
  - How stateful beans are painless in EJB3;
  - A stateless bean in Seam would have also worked.

Implement UserCrudService

```
@Logger
private Log log;

private List<User> users;

@In
private UserDAO userDAO;

@Out(required = false)
private User user;

public List<User> getUsers() {
   if (users == null) {
      log.info("Retrieving all users from the DB...");
      users = userDAO.retrieveAll();
      log.info("#0 users retrieved.", users.size());
   }
   return users;
}
```

Two kinds of *outjection*:
#{userCrud.users}
#{user}

Implement UserCrudService

```
public void setUsers(List<User> users) {
   this.users = users;
}

public void create() {
   log.info("Creating a new user...");
   user = new User();
}

public void cancel() {
   log.info("Canceling user creation...");
   user = null;
}
```

Implement UserCrudService

```
    public void save() {
        log.info("Saving user: #0 (#1)",
                                 user.getName(), user.getEmail());
        userDAO.save(user);
        users = null;
        user = null;
    }

    public void refresh() {
        log.info("Refreshing user list...");
        users = null;
    }

    @Remove @Destroy
    public void remove() { }
}
```

Mandatory for stateful beans.

Implement CRUD of users

- Now, to the web page `userCrud.xhtml`:

```
<ui:composition ... template="layout/template.xhtml">

  <ui:define name="body">
    <h:panelGroup rendered="#{userCrud.users.size == 0}">
      <p>No users were registered yet.</p>
    </h:panelGroup>

    <h:panelGroup rendered="#{(user == null) and
                              (userCrud.users.size != 0)}">
      <h:dataTable value="#{userCrud.users}" var="user"
                                             border="1">
        <h:column>
          <f:facet name="header">
            <h:outputText value="Name" />
          </f:facet>
          <h:outputText value="#{user.name}" />
        </h:column>
```

Implement userCrud.xhtml

```
      <h:column>
        <f:facet name="header">
          <h:outputText value="E-mail" />
        </f:facet>
        <h:outputText value="#{user.email}" />
      </h:column>
    </h:dataTable>
    <h:form>
      <h:commandButton action="#{userCrud.refresh}"
                                        value="Refresh" />
    </h:form>
  </h:panelGroup>

  <h:panelGroup rendered="#{user == null}">
    <h:form>
      <h:commandButton action="#{userCrud.create}"
                                      value="New User" />
    </h:form>
  </h:panelGroup>
```

Implement userCrud.xhtml

```
    <h:panelGroup rendered="#{user != null}">
      <h:form>
        <h:panelGrid columns="2">
          <h:outputText value="Name:" />
          <h:inputText value="#{user.name}" size="30" />
          <h:outputText value="Email:" />
          <h:inputText value="#{user.email}" size="30" />
          <h:outputText value="Password:" />
          <h:inputSecret value="#{user.password}" size="15"
                                                      />
        </h:panelGrid>
        <h:commandButton action="#{userCrud.cancel}"
                                          value="Cancel" />
        <h:commandButton action="#{userCrud.save}"
                                          value="Save" />
      </h:form>
    </h:panelGroup>
  </ui:define>
</ui:composition>
```

End of Part 1

- We've experimented with the basics of Seam;
- There is a lot more to see:
  - Security (authentication & authorization);
  - Internationalization;
  - Conversations / workspace management;
  - AJAX with Richfaces;
  - Sending email;
  - Testing;
  - And much more...

Hands On – Seam Framework