CRUD framework tutorial

Göran Stäck

Table of Contents

Introduction

This tutorial will guide you through the process of creating a Java Client Server Application based on the CRUD application framework. Step by step you will learn how to set up your development environment, define a SQL database, implement entity classes, write user interface classes and finally how glue those parts together using some other frameworks. The tutorial tries to cover the full range of programming a Client Server Application including testing and deployment.

By following this tutorial you will learn how to develop a fully functional multiuser client server application that stores album, artists and songs in a SQL database. The user interface in the clients will be able to create, read, update and delete those entities. The Album, Artist Song data model is widely spread in the education community. Here is one example. [http://staff.science.uva.nl/~netpeer/teaching/webdb2002/practicum/P2.htm]

Setting up your environment

1. Create a new workspace and checkout the following projects from `svn://hulk/iprobe-dev` repository:

   ```
   svn://hulk/iprobe-dev/Crud/trunk
   svn://hulk/iprobe-dev/CrudClient/trunk
   svn://hulk/iprobe-dev/CrudServer/trunk
   svn://hulk/iprobe-dev/CrudExample/trunk
   ```

   The CrudExample contains a working example that can be useful if you run into problems with the tutorial. Its based on a small part of the Isac system and its tested and should work. If it don't you have probably some problems with your environment.

2. Download .NET framework 2.0 from Microsoft and install

3. Download SQL Server 2005 Express™ from Microsoft and install

   - install server and client connectivity pack

   - use mixed authentication mode

4. Enable remote connections for SQL server

   - see http://blogs.msdn.com/sqlexpress/archive/2005/05/05/415084.aspx

   - remember to set a specific port number to use, e.g. 1433

5. Download Microsoft SQL Server Management Studio Express™

   - Install Microsoft SQL Server Management Studio Express

   - Start Microsoft SQL Server Management Studio Express with Sql Server Authentication, user name = sa and password = the one you entered at installation

Create projects in your workspace for the tutorial

Create three Java projects in your workspace. The reason for three projects is to reflect the client server tiers in the project structure. By using this project structure its impossible to make references between server and client a mistake that otherwise is discovered first when building.

CrudTutorial	This project contains code used on both server and client. Make the project dependent on the Crud project and export the Crud project.
CrudTutorialServer	This project contains code used on the server side. Make the project dependent on the CrudServer project.
CrudTutorialClient	This project contains code used on the client side. Make the project dependent on the CrudClient project.

Create database table

Use SQL Server Management Studio Express to create a new database called `crud-tutorial`. Create a new table called `Artist` with the columns: Id, Name, Biography. Make the Id column an auto sequence primary key. You can design the database with the graphical user interface of SQL Server Management Studio Express.

One single table will do for now. More tables will be added further on in the tutorial.

Save the sql script that you created in a file called `crud-tutorial.sql` and put the file int the location: `CrudTutorialServer/runtime/install/sql/crud-example.sql`. The script should look something like this:

```
USE [crud-tutorial]
GO
/****** Object:  Table [dbo].[Artist]    Script Date: 07/16/2008 15:36:57 *****
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Artist](
 [Id] [int] IDENTITY(1,1) NOT NULL,
 [Name] [varchar](50) NULL,
 [Biography] [varchar](1000) NULL,
 CONSTRAINT [PK_Artist] PRIMARY KEY CLUSTERED
```

```
(
 [Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, A
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
```

Create your first entity class

Create a class in CrudTutorial project called: `se.iprobe.crud.tutorial.Artist`

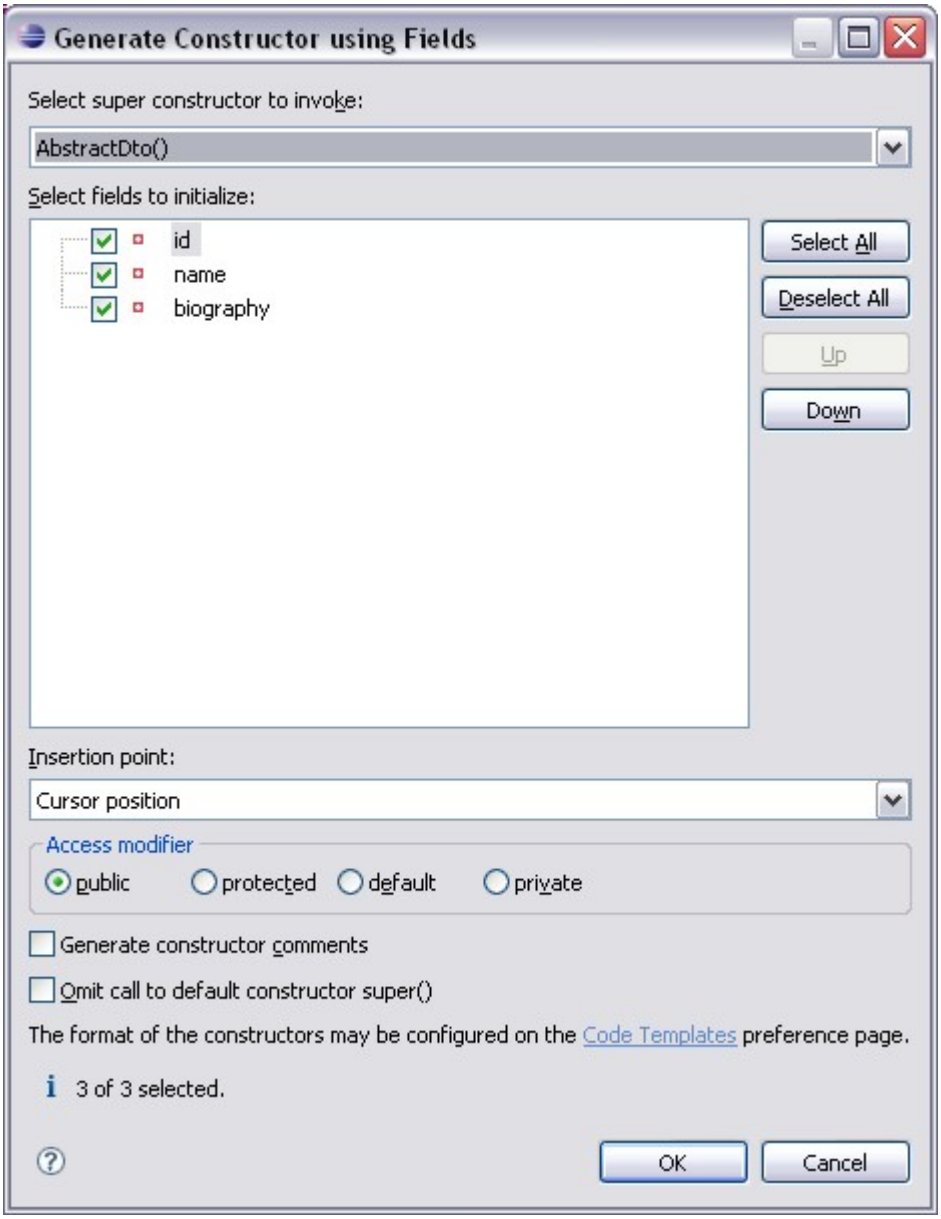
The Artist class is used on both server and client and will be used as a data transfer object (DTO) and is therefore implementing the `Serializable` interface. By adding

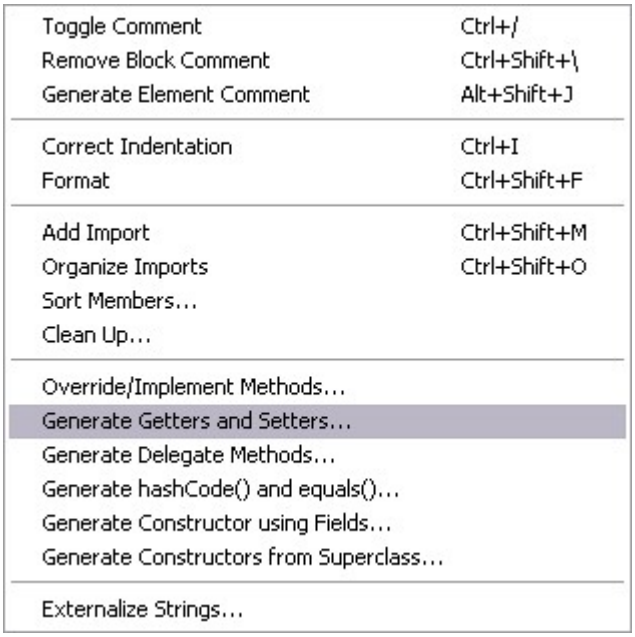
```
private static final long serialVersionUID = 1L;
```

you get rid of the “The serializable class Artist does not declare a static final serialVersionUID field of type long” warning.

Without the `serialVersionUID` field the compiler will generate one for you. The problem with that is that the Eclipse built in compiler use a different algorithm from the `javac` compiler in the JDK. That makes it impossible to run a client inside Eclipse against a server built with `javac`

Add instance variables for id, name and biography and generate constructor and getters and setters using Eclipse.





All entity classes in a CRUD application must implement the Observable pattern. Most of it is inherited from super class but every set method except the `setId` method must be implemented in a certain way. This could of course be done by AOP but its not that hard to to by hand. Here is an example of a set method:

```
public void setName(String name)
{
 if (!equals(name, this.name)) ❶
 {
  this.name = name;
  fireValueChanged("name", this.name); ❷
 }
}
```

❶ The not equals test is important to stop infinite loops that can arise when you are dealing with intense listener programming.

❷ By calling `fireValueChanged` listeners of Artists gets notified when the Artists name is changed.

Implement an equals method

The superclass `AbstractDto` throws an exception in the `equals` method to point out the importance of implementing an `equals` method. In a distributed environment its possible to end up with two instances of `Artist` that represent the same artist. By implementing an `equals` method based on the database id they will be considered as equal which is nessecary for several vital functions in the CRUD framework

Here is an example of an `equals` method for your `Artist` class:

```
@Override
public boolean equals(Object obj)
{
 return obj == null ? false : id == ((Artist) obj).id;
}
```

Implement the ArtistDao

You should implement an ArtistDao that consist of one interface and two implementations. One implementation works against the database and the other implementation is a stub implementation that only stores Artists in the memory and can be used for testing. You can skip the stub implementation for now since we will get back to it further on in the tutorial.

Create the interface `se.iprobe.crud.tutorial.remote.ArtistDao` in the `CrudTutorial` project and create methods for getting a list of artists, adding a new artist, changing an artist and removing an artist. Thanks to the Spring framework we can publish the interface as a Java RMI remote interface without extend the `Remote` interface which is a good thing since we are not forced to throw checked exception from our interface methods. Your interface should look something like this:

```
package se.iprobe.crud.tutorial.remote;

import java.util.List;

import se.iprobe.crud.tutorial.Artist;

public interface ArtistDao
{
 public List<Artist> getArtists();
 public int addArtist(Artist artist);
 public void changeArtist(Artist artist);
 public void removeArtist(int artistId);
}
```

Create a class called `se.iprobe.crud.tutorial.server.ArtistDaoImpl` in the `CrudTutorialServer` project. This class is a pure server class never exposed outside the server. The class should implement the `ArtistDao` interface and do extend `AbstractJdbcDao`.

Study the Spring JDBC framework and try whats going on in the programlisting below. Copy the code to your `ArtistDaoImpl`

```
package se.iprobe.crud.tutorial.server;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

import org.springframework.jdbc.core.simple.ParameterizedRowMapper;
import org.springframework.transaction.annotation.Transactional;

import se.iprobe.crud.server.AbstractJdbcDao;
import se.iprobe.crud.tutorial.Artist;
import se.iprobe.crud.tutorial.remote.ArtistDao;

public class ArtistDaoImpl extends AbstractJdbcDao implements ArtistDao
{

 private ParameterizedRowMapper<Artist> artistMapper =
     new ParameterizedRowMapper<Artist>()
 {
```

```
 @Override
 @Transactional(rollbackFor=Exception.class)
 public Artist mapRow(ResultSet rs, int rowNum) throws SQLException
 {
  Artist artist = new Artist(
   rs.getInt("Id"),
   rs.getString("Name"),
   rs.getString("Biography")
   );
  return artist;
 }
};

@Override
@Transactional(rollbackFor=Exception.class)
public List<Artist> getArtists()
{
 String sql = "SELECT * FROM Artist";
 return jdbcTemplate.query(sql, artistMapper);
}

@Override
@Transactional(rollbackFor=Exception.class)
public int addArtist(Artist artist)
{
 int artistId = 0;
 String sql = "INSERT INTO Artist" +
 " (Name, Biography)" +
 "  VALUES(?, ?) ";
 jdbcTemplate.update(sql, artist.getName(),
         artist.getBiography()
         );
 //Now get the maximum value of the Id
 sql = "SELECT MAX(Id) FROM Artist";
 artistId =  jdbcTemplate.queryForInt(sql);
 return artistId;
}

@Override
@Transactional(rollbackFor=Exception.class)
public void changeArtist(Artist artist)
{
 String sql = "UPDATE Artist " +
              " SET Name = ?, Biography = ?" +
              " WHERE  Id = ?  ";
 jdbcTemplate.update(sql, artist.getName(),
   artist.getBiography()
   );
}

@Override
@Transactional(rollbackFor=Exception.class)
public void removeArtist(int artistId)
{
 String sql = "DELETE FROM Artist WHERE  Id = ?";
 jdbcTemplate.update(sql, artistId);
}
```

```
 @Override
 protected int getMaxNumberOfRowsFromDb()
 {
  return 20000;
 }
}
```

Configuration files

The Crud framework use a minimum of configuration from files. We belive in expressing as much as possible as Java code. The main reason for the configuration files we do have is testability. Without changing is single line of code or configuration file you can run a Crud application inside Eclipse with your personal local database or as a deployed application connecting to a remote production database.

You should create three configuration files:

`server.xml` — Is the configuration file of the Spring framework. It specifies several things:

- Information of the database connection. Database URL, driver to use, login information etc
- Configuration of transactional behavior based on annotations as you can see in the `ArtisDaoImpl`
- Definition of an interceptor that catch exceptions in remote calls before they reach the client
- A list of all interfaces that should be exported as RMI interfaces
- A list of the implementing of the interfaces that are requested from the Spring IoC

Some values are expressed as placeholder and will be replaced with values from the `server.properties` file

`server.properties` — Properties read by the server application. Some properties contains values for the place holders in `server.xml` file.

`log4j.properties` — Contains the logging settings used in a production environment.

Copy the files from the CrudExample project and change accordingly.

All configuration files are read using a class loader that is they must be in the class path. But this makes it possible to implement a simple override mechanism. Its possible to override the properties in the `log4j.properties` file by have an additional `log4j.properties` file in a directory that precedes the runtime directory in the class path. The `server.properties` works the same way but with a little twist.

To enable individual properties for each developer the server.properties can be overridden by a file that is called: `server.[username].properties`. Individual properties is handy for connecting to individual test databases and for defining mail addresses for applications that sends mail etc

Since the tutorial is based on a local personal database there is no need for overriding the `server.properties`. But you should override some properties in `log4j.properties` file to get the logging in the console instead of in logging files. You could also increase the logging level if you like.

See how its done in the CrudExample projects.

Implement the server side Service Locator

Create a subclass to `ServiceLocatorTemplate` called `ServiceLocator` and implement one static get method for each DAO class you have in your application. Put the class in the CrudTutorial project to enable usage from both server and client. In this tutorial we only have one DAO so far which means that the `ServiceLocator` will look like this:

```
package se.iprobe.crud.tutorial;

import se.iprobe.crud.ServiceLocatorTemplate;
import se.iprobe.crud.tutorial.remote.ArtistDao;

public class ServiceLocator extends ServiceLocatorTemplate
{
 public static ArtistDao getArtistDao()
 {
  return (ArtistDao)ctx.getBean("artistDao"); ❶
 }
}
```

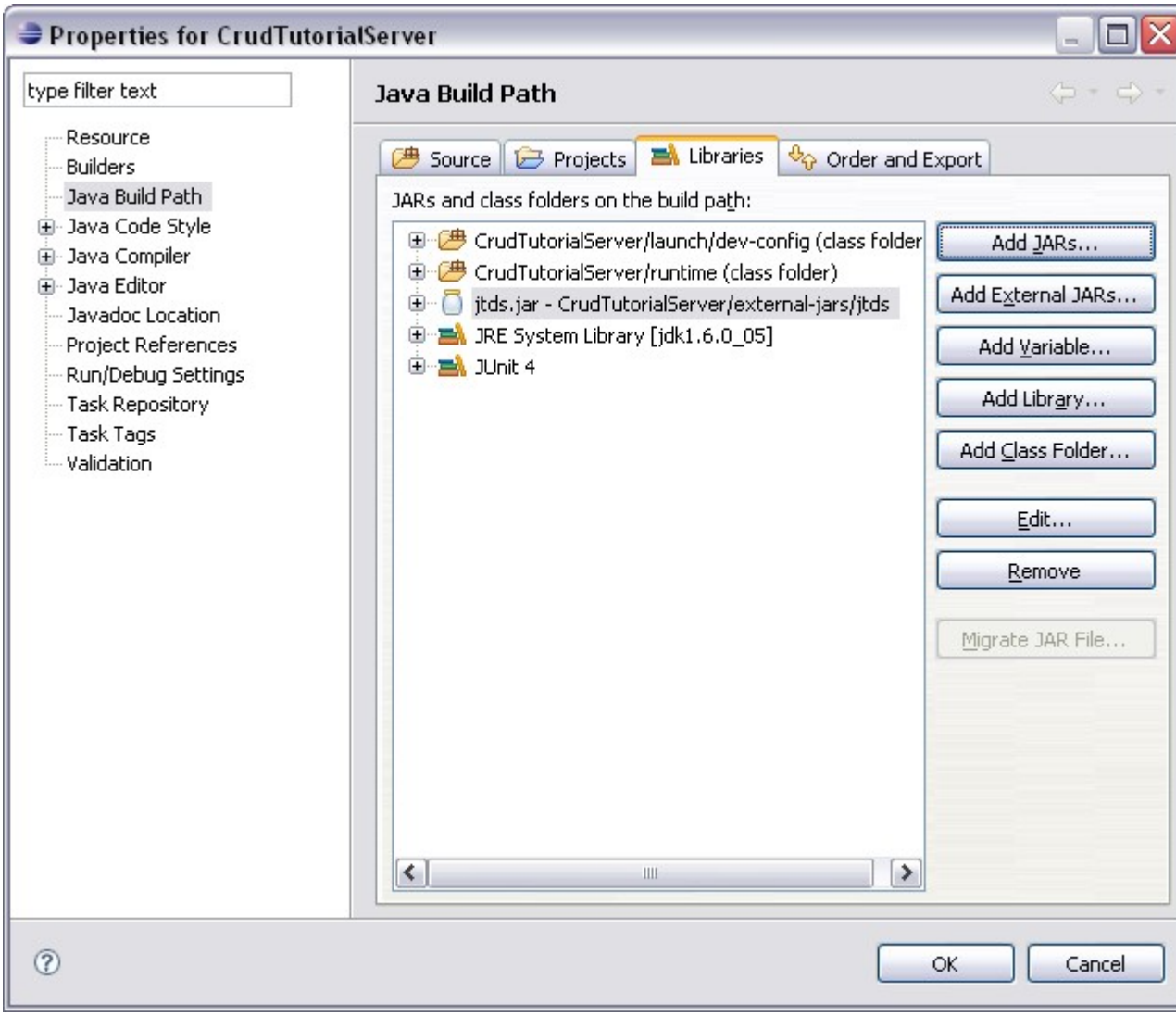
❶ The name you enter here should be found in the corresponding bean element in the `server.xml` file

Test your DAO class with a unit test class

Create a new source folder in the CrudTutorialServer project called `src-test`. Add the JUnit4 library to the class path of the project:

Make sure the `runtime` folder and the `launch/dev-config` folder are included in the class path and in the correct order.

The Crud framework has no binding to any specific SQL database and do not include any JDBC driver jar. That has to be included in every Crud based server application. Copy the `jtds.jar` from the CrudExample project and include it in the class path. The property dialog of the Eclipse project should look now like this:



Create a `se.iprobe.crud.tutorial.server.ArtistDaoTest` in the scr-test folder. Copy the code from `UserDaoTest` in the CrudExampleProject and make nessecary changes. Run it as a JUnit test. The logging should appear in the console window thanks to the additional `log4j.properties` file you put in `launch/dev-config` folder. Add more test methods to cover all methods in your DAO class.

Implement the main class of the server application

Create a class called `se.iprobe.crud.tutorial.server.CrudTutorialServer` Copy the code from `CrudExampleServer` in the CrudExampleProject and make nessecary changes. The `configureAndWatch` call starts a very nice feature that makes it possible to change logging properties in runtime. If your server behaves strange you can increase the logging level and hopefully see what's going on.

Create your first model panel

Now its time to start with the client side of your application. Panels are very central things in Crud client application. A panel is responsible for displaying and editing the properties of a models of a certain type. Panels contains a set of Swing components and each component has an adapter that defines which property in the model the component should be bound to. The Crud framework uses the Esox binding framework for this purpose. The components and the adapters are instantiated in the constructor of the panel. One nice feature of panels in the Crud framework is that you can instantiate them without having a model. Instead you provide a `ModelOwnerIF` an object that is assigned with models and is observable in that aspect.

In your TutorialClient project create a class called `se.iprobe.crud.tutorial.client.ArtistPanel` that extends DirtyPredicatePanel. There is no abstract methods that must be overridden but you will benefit from following some convention for organizing the code.

Create a default constructor and add the following method calls:

```
initComponents();
arrangeLayout();
bindComponents();
setRenderers();
```

Let Eclipse generate stubs for you.

Create an instance variable for each Swing component that will be bound to a property in the Artist object. In this case there will be only two components:

```
private JTextField name;
private JTextPane biography;
```

Create the Swing components in the `initComponents` by calling convenience methods in the superclass:

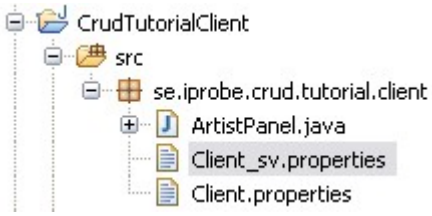
```
name= createInputLimitedTextField(50); ❶
biography = createInputLimitedTextPane(1000); ❷
```

❶❷ The limitation you enter here should correspond to ones in the database

To handle multiple languages you must add a `ResourceBundle` to your class. Declare a static variable as follows:

```
private final static ResourceBundle bundle = ResourceBundle.getBundle("se.iprob
```

In same folder as your class resides create a property file for each language you like to support.



Put the following code lines in the `arrangeLayout` method:

```
setLayout(new BorderLayout()); ❶
FormPanel form = new FormPanel(); ❷
form.addFormRow(new JLabel(bundle.getString("artist.name")), name); ❸
form.addFormRow(new JLabel(bundle.getString("artist.biography")), biography); ❹
form.adjustLabelWidthsToLargest(); ❺
add(form, BorderLayout.CENTER); ❻
```

❶ All panels that are going to be used as a detail panel in a master detail panel must have a `BorderLayout` as layout manager since a Save button will be injected in the south region of the `BorderLayout`

❷ The `FormPanel` is just a convenient way to arrange components to a form. The `FormPanel` only handles layout and has no other features.

❸❹ Add artist.name and artist.biography as entries in your language properties files

❺ By making this final call to the `FormPanel` you get nice columns i the form.

In the `createBindings` methods you should put code that connects the Swing components with the properties of the panel model in this case the properties of the `Artist` class:

```
new TextFieldAdapter(name, this,❶ Artist.class, "getName", "setName", String.cl
TextFieldFocusHandler.add( name );❸
new TextPaneAdapter(biography, this, Artist.class, "getBiography", "setBiography
```

❶ The two first arguments are swing component and a `ModelOwnerIF`. In this case the model owner is the panel it self. The adapter listen to the model owner and gets notified when a model is assigned to the panel.

❷ The remaining arguments are used to construct accessors to the property. The adaptor uses reflection for that purpose

❸ The adaptor listen to the swing component and gets notified when the enters data in the component. But a focus lost in text field don't won't generate that notification. This line fix that problem.

Leave the `setRenderers` method empty for the time being.

Test the ArtistPanel