INTRO

Hello and welcome to this fifteen minute presentation showing how to build a JViews Maps web application using Java Server Faces.
Before to watch this presentation, it's preferable to have a look at the Map Tool Chain video to understand the JViews tools the usage of the Map Builder allowing to build a map background, and the Diagrammer Designer to build a diagrammer application.
The Diagram Display Desktop video will provide more details on the JViews Maps API. And, it's preferable to watch the Java Server Faces Support video if you want more details on JSF technology.
Obviously, JSF and Java development knowledge is preferable. Concerning the development plateform, Eclipse will be used.

The step by step demonstration of a JSF JViews maps Application development will be done in 4 phases.
The first step is the construction of a standard JSF application using Eclipse.
Then, in a second step, we will add a JSF map view loading a diagrammer project with a map background
The third step will show how to add interactivity.
And, finally we will realize a binding of the JSF component to a managed bean in order to replace the JSF diagrammer project configuration by a pure java development.

STEP 1

First, create a new project using the template "Dynamic Web Project" in the "Web" part.
We name this project "JSF Map Sample" and in the configuration part, we choose "JavaServer Faces v1.2 Project".
Now, we create a new JSP page that we name "map.jsp". We choose the template "New JavaServer Faces Page (html)"
As you can see, Eclipse shows 2 errors concerning the common sun JSF taglib. It's equivalent to the import task in java.
So, we have to add the needed jsf libraries. Go, in the jviews-framework86\lib\external directory and copy paste jsf-api-1.2, jsf-impl-1.2 and jstl in the WebContent\WEB-INF\lib directory.
Now, we just add a simple text in this empty JSP page to test it.
Just write : "Create a simple JSF JViews Maps Application"
Now, we test this page. Right Click on the JSP page and select Run As: Run on Server and choose the Apache Tomcat 6 Server.
The web application only shows the text that we have written proving the application is well deployed.

STEP 2

The target of this step is to load a diagrammer project with a map background using a JSF map view. We will use a sample provided by the JViews Maps distribution.
Go in the directory jviews-maps\samples\loaddiag\resources and drag drop the data directory in the directory WebContent of the JSF project.

By double-clicking on the world.idpr project file, it launches the Diagrammer Designer. We will have a look at what is contained in this project.
First, you can find in Background Map options the IVL map file that has been generated by the Map Builder.
By selecting the node leaf, you have access to the rendering options. We use a specific palette for nodes allowing to render node with different vehicules according to the type property.
In the edition mode, you can test and edit the project. For example, by selecting a node and modifying the type property to 1, the picture is changed in a plane.
You can also add new node using the edition toolbar.
For more details on the Diagrammer Designer, please watch the video "Map Tool Chain".

Back to Eclipse.
The first step to build a JViews JSF application is to adapt the web.xml file to configure the JViews servlet. The easiest way is to copy the text coming from an existing sample.
Go in the directory jviews-diagrammer\sample\diagrammer\jsf-diagrammer-dashboard and open the web.xml file in a text editor.
Copy the JSF and servlet configuration part in the web.xml file of the JSF project.
Now, we have to add the JViews libraries in the lib project. These libraries are jviews-maps-all.jar, jviews-diagrammer-all.jar,
jviews-palette-shared-symbols.jar for the symbol rendering and jviews-framework-all.jar for all the core JViews API.
Finally, don't forget to manage the licensing. In development, the easiest way is to build a jar file with the keys.jlm file and to add it to the lib directory.

Now, we can edit the JSP page.
The first step is to add the JViews Maps JSF taglib using the prefix jvmf.
Between the view tag part, add the mapView JSF component, we give map as id, we will load the diagrammer project world.idpr file using the project property and
we manage the width and height of the generated picture using the style property.
The first JViews JSF application is ready.
We can test it using the Run AS> Run on Server menu.
Now, below the text, you can see a picture of the diagrammer view, build using the Diagrammer designer.
We still can't interact with this image. It's the next step target.

STEP 3

First, we had the jviews framework taglib, with the prefix jvf.
Then, we had the zoom interactor faces with the id zoom.
Now, we just have to connect the zoom interactor to the mapView using the property interactorId.
In order to manage precisely the zoom levels, we can had this property with the adapted value, 2, 4, 8 and so on, meaning double the zoom factor each time.
We test the result with the classical action on the JSP page: Run AS>Run on Server
The result is the cursor is a cross and you can draw a rectangle to select the zoom area.

Now, we had an other interactor named pan interactor. We give the id pan.

And, we will create a panelGrid component that will contain several buttons. These buttons will be used to change the interactor set on the mapView.
These buttons are provided by the jviews taglib with the prefix jv.
We first create an imageButton to set the zoom interactor on the mapView
With the onclick property, we can set the javascript action allowing to set the zoom interactor on the mapView using the MapView and zoomInteractor ids.
We give the image size using the style property and we give the image displayed in this rectangle. We copy paste this code 4 times to create the other buttons. The second button is used to set the pan interactor. We just have to update the onclick and image properties.
And we do the same for the Fit, zoomIn and Zoomout actions.

Now, we can test this toolbar.
As before, the default interactor allows to zoom.
If you click on the second button, you can pan on the view.
The 2 last buttons, allow to respectively zoom in and out on the map view.
Finally, the button in the middle of the toolbar fit the view.

STEP 4

In this last step, we will use a Managed Bean in order to feed the view using java code.
The first step is to update the faces configuration with the faces-config.xml file.
You select the ManagedBean tab, and the session element.
Click on Add. Select: "Create a new Java class". We give "faces" as package name. and MapBean for the name of the class. Obviously, you can choose others package and class name.

Now, in the source directory, we will edit the mapbean class in order to read the diagrammer project by code.
We create a diagrammer parameter and we generate the getter. If the diagrammer object doesn't exist, we instantiate it and we use the FacesContext in order to access a file resource
by using getCurrentInstance().getExternalContext().getResource() and we give the path data to access the world.idpr diagrammer project file.
It returns an URL. So, we have to catch a possible exception, for example if the file is not found.
We can add this data to diagrammer using the method setDataFile() on diagrammer.

In order to be sure this code will be accessible by the web application, we edit the project property. Select the "Java Build Path", and choose WebContent>Web-inf>Classes as output folder.
Now, we edit the JSP page to replace the project property by the diagrammer property. This property will be directed to MapBean.diagrammer using a specific syntax.

As a final exercise, we will create a new node and add it to the view.
We will add a plane on the top middle of the US Map.
First, you have to create a node by calling diagrammer.createNode. The parameter provides the tag value
It returns an object.

We set some properties using setObjectProperty. The first property is the object type. If we use 1, the node image is a plane.
We set the node position using latitude and longitude properties.
First the latitude information give the north south position. With, 48DN, we are near of the top of the united states.
Then, the longitude information provides the east west position. With 105DW, we are ath center of the united states.
Now, we just have to add this node to the model using addObject. As, there is no parent hierarchy, the second parameter is null.

Thank you.
I hope this quick video helped you to understand the JViews JSF Maps API.
I invite you to download the sample source code and realize some modification to start your first JViews JSF Maps project.