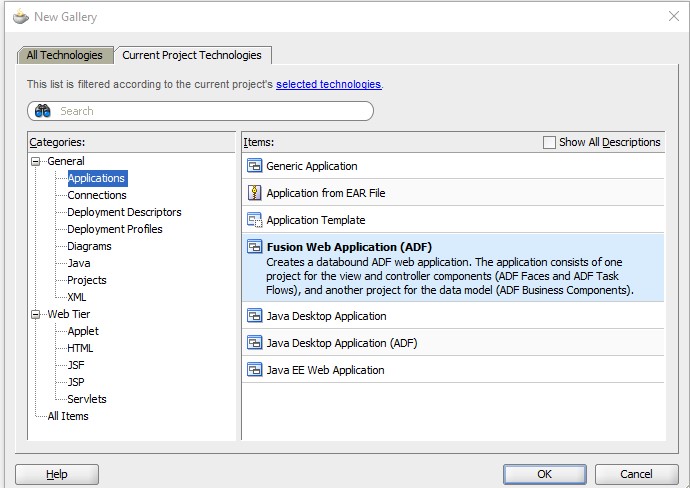
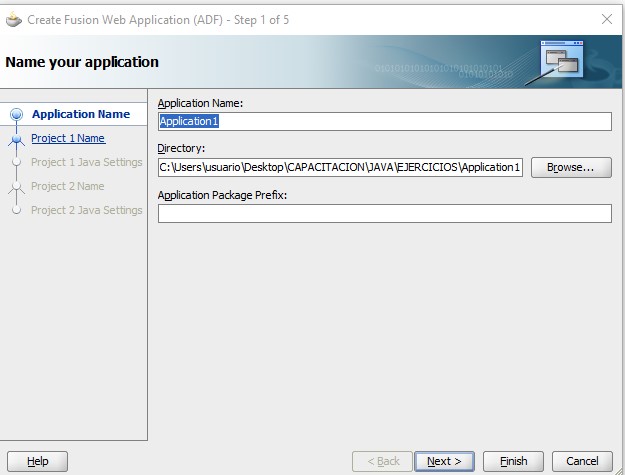
**APLICACIONES ADF**

PASO

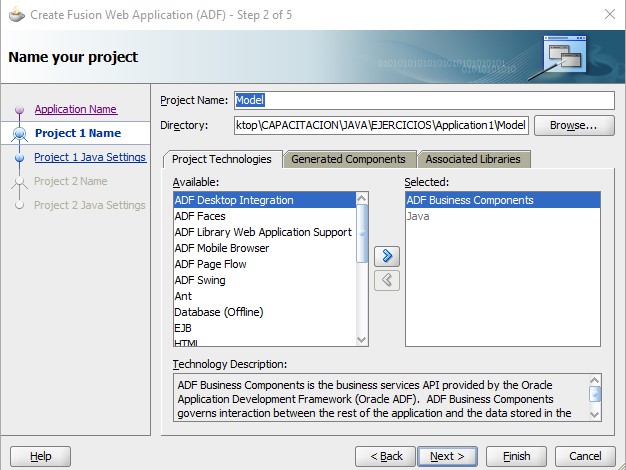
1. Creamos un un proyecto File - - > new ... y luego una **Fusión Web Application** (Esta Web Application tendrá un Aplication module y un View Controller)



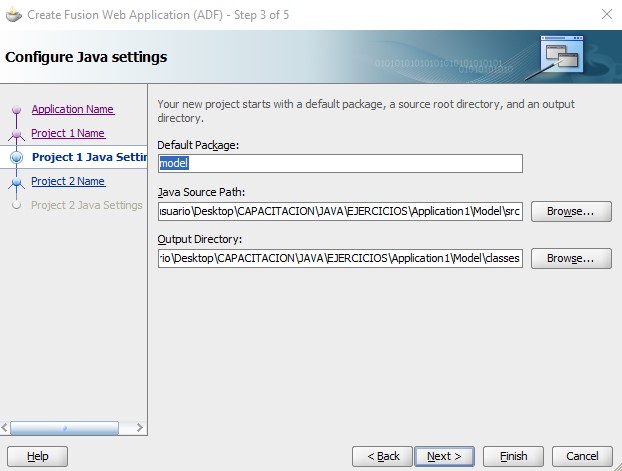
1. Seleccionamos el nombre de la aplicación



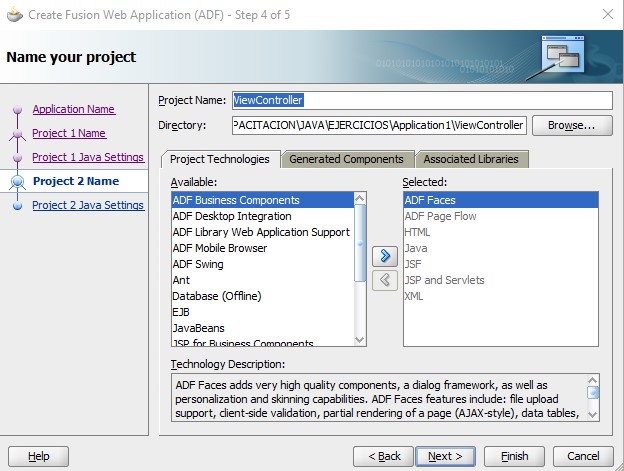
1. Selecciono el nombre del APPMODULE



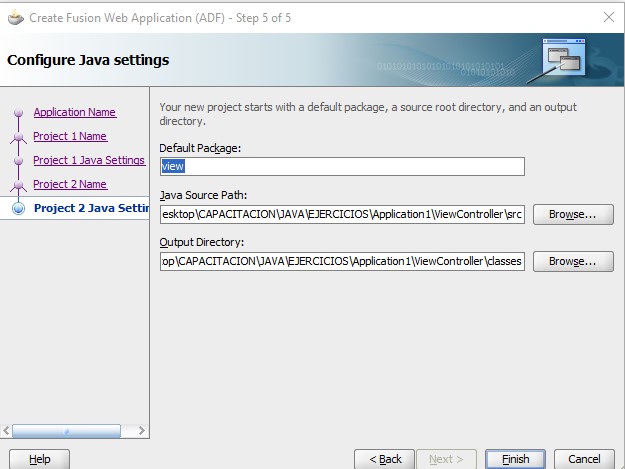
1. Definimos los directorios del proyecto y el directorio del paquete que contendrá el model



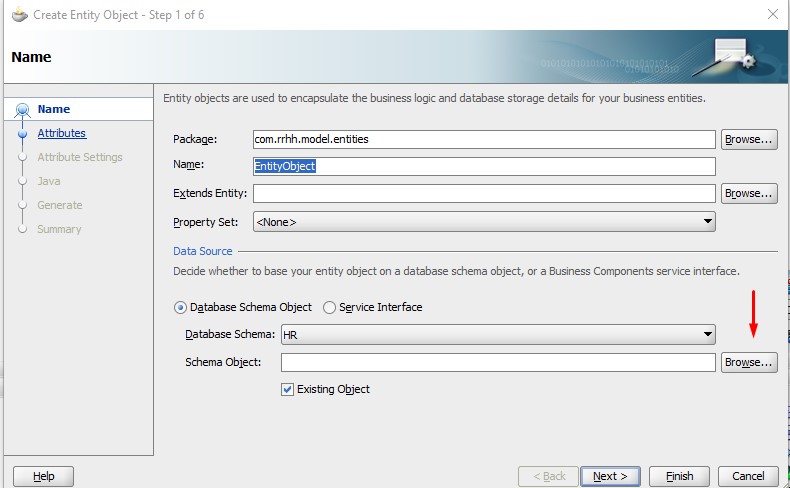
1. Nombramos el nombre del View controller



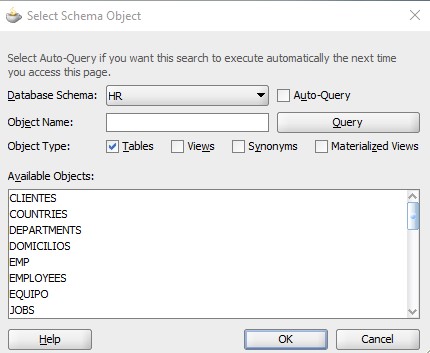
1. Definimos el directorio del paquete que contendrá el View Controller



1. Tendremos el modelo listo para programarlo, lo siguiente es crear las entidades con las cuales se trabajaran.

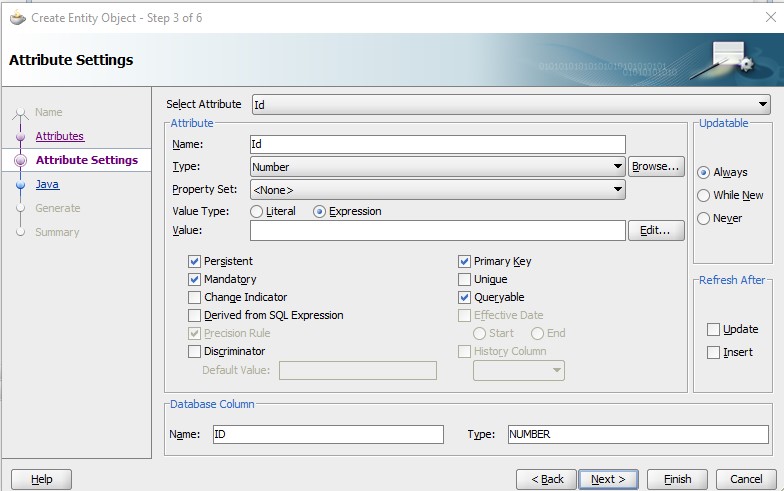


1. Nombramos la entidad, luego iremos a la flecha roja donde indicaremos cual la tabla del esquema que vamos a utilizar



IMPORTANTE: las entidades se pueden crear también de otra manera.

Boton derecho sobre model -> new -> Business component -> Entity Object

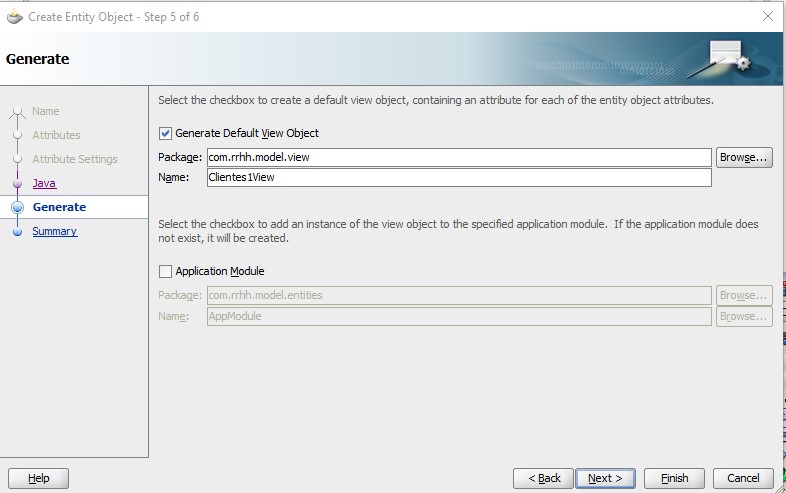


1. Luego Pasamos al Step 3 de 6 y renombramos los campos en caso que tiendan a confundir y declaramos como van a ser los campos de las entidades
2. En caso de trabajar con Secuencias de las Base de datos , debemos aplicárselo al id de la tabla específica y poner el tipo de valor del ID como Expression

Y luego ponemos :

(new oracle.jbo.server.SequenceImpl("TABLA\_SEQ", adf.object.getDBTransaction())).getSequenceNumber()

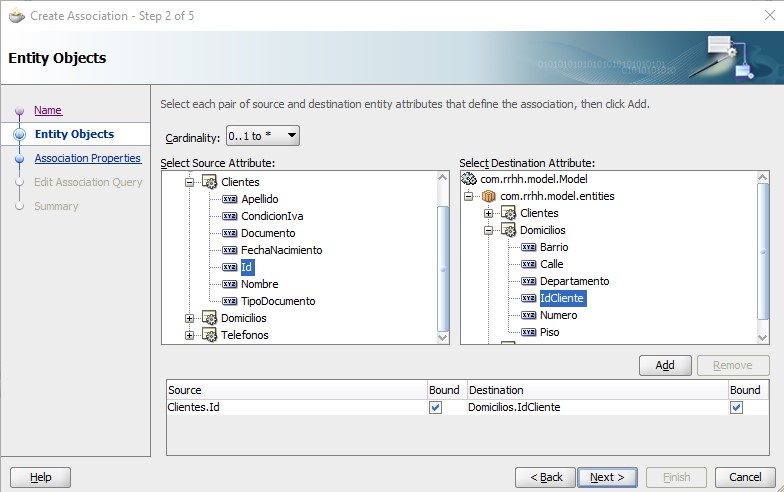
En los Id declaramos que sean “While New” y que sea “Insert”.



1. Podemos crear la vista para la entidad en el mismo paso y definirle un directorio
2. Creamos una Asociacion , le damos nombre y luego la relación en la que se van a Persistir ambas tablas

-Las Asociaciones trabajan con Entidades en conjunto para PERSISTIR DATOS y los LINK con Vistas.

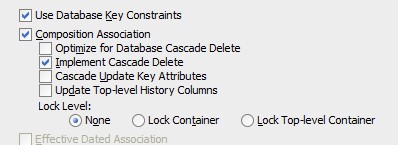
- La cardinalidad es depende la relación que queremos hacer. (1 a \*, 1 a 1 , \* a \* , 0..1 a \*)



Luego voy a la solapa Relationship-> Behavior:

Chequear:

* “Use database Key constraints”
* Composition Asociations
* Delete Cascade (en caso de que el viewController hagamos un ABM)

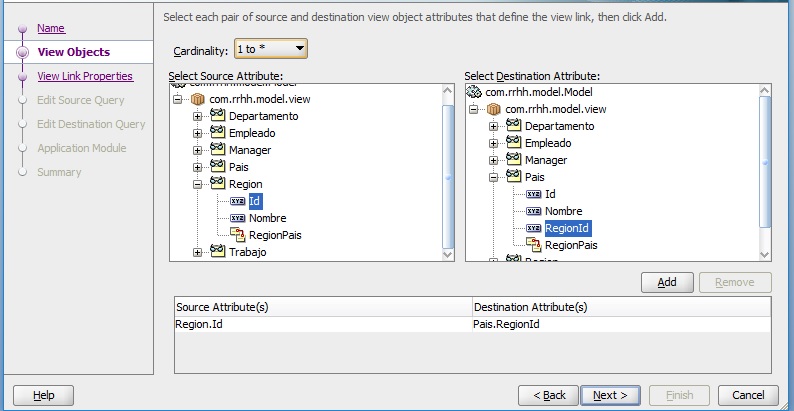


IMPORTANTE: crear las vistas de todas las entidades con las cual se trabajen.

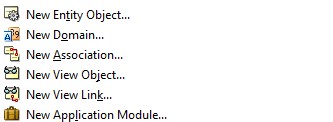
Se puede crear vistas sin la entidad pero no se puede **PERSISTIR LOS DATOS**.

Las reglas de la entidad se crean a partir de la base y se cargan antes de la base de datos.

1. Crear los View Link Para Trabajar con las vistas

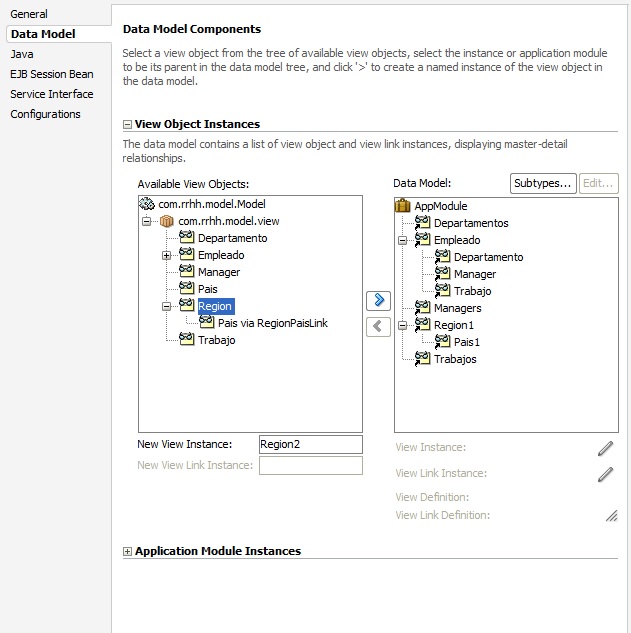


1. Crear el App Module



1. En el AppModule seleccionamos la solapa “Data Model”

Seleccionamos lo que esta anidado del lado izq hacia al lado derecho dependiendo lo que queremos mostrar.

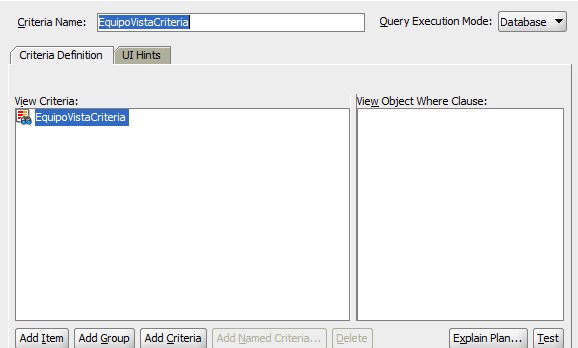


**CREACION DE VIEW CRITERIA**

1. Selecciono la vista que le voy a dar el criterio , voy a la solapa “Query” de las vistas
2. Creo los bind variables para las variables con la letra P\_
3. Creo la view criteria (utilizo las variables)
4. Utilizo Add Group luego Add Ítem(para agregar la cantidad de criterio que quiera)
5. - En Attribute: pongo la entidad

- En Operator: la condición

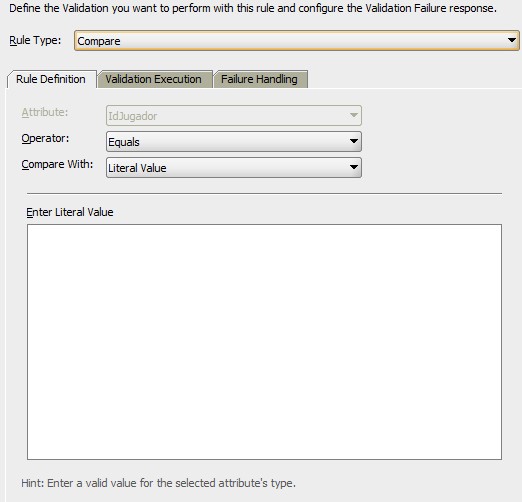
- En Operand: Bind Variable (en caso de utilizar los bind )



**CREACION DE RULES VALIDATION**

Creo validación en la entidad Business rules/new Validator

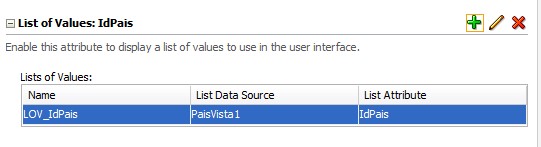
Ej :RULE TYPE:COMPARE , RULE DEFINITION:NOT EQUALS , EXPRESSION:!NULL



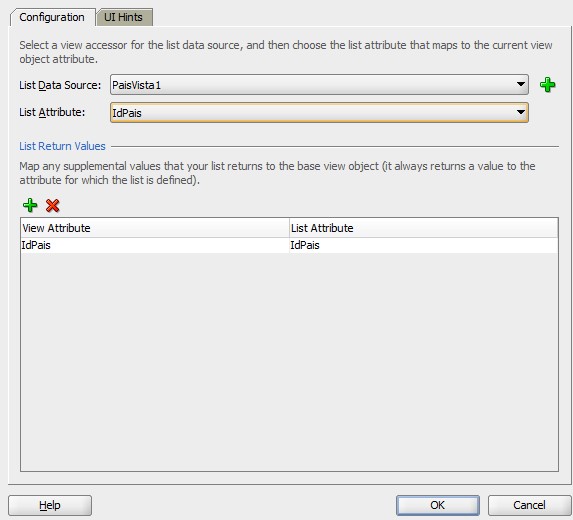
**LISTA DE VALORES EN APPMODULE**

1) Selecciono la Vista de la entidad, luego la solapa “Attributes”

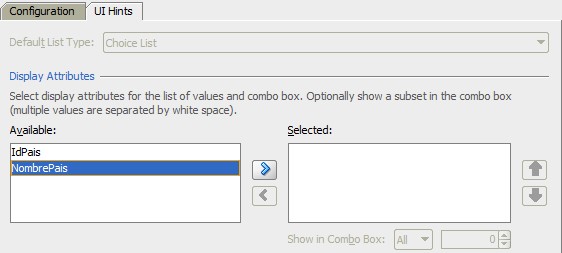
2) Selecciono el campo que quiero que sea una lista



3) en List Data Source elijo la vista en el list Attribute elijo el id.



4) En la solapa “UI hints” elijo lo que quiero que aparezca como lista de valores



**DATOS IMPORTANTES**

Definiciones:

Propiedad refactor: renombra clases y cambia todo lo que tenga el ese nombre.

JVM CLASS LOADER pone las clases en memoria

GARBAGE COLLECTOR - sacar los objetos en memoria que no se usen

JDK: Tiene un compilador, interpretador y un generador de documentación

MODEL BUNDLE: Sirve para modificar los label del proyecto

**GENERIC APLICATION**

ERRORES:

- debe insertar la librería ORACLE.JDBC

- entrada y salida, revisar la sentencia de la conexión establecida.

SI ELEGIMOS GENERIC APLICATION LUEGO DEBEMOS ELEGIR ADF VIEW CONTROLLER

**LIBRERIAS IMPORTANTES PARA LA CONEXIÓN A LA BASE DE DATOS**

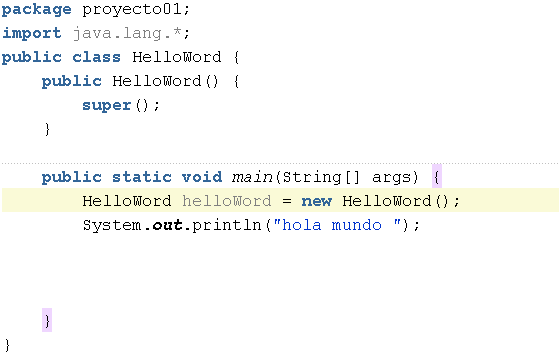
Selecciono el proyecto -> Project properties -> libraries and classpath y agregar la librería manualmente

Import java.sql.Connection;

Import java.sql.DriverManager;

Import java.sql.SQLException;

**CLASE JAVA TEORIA:**



SOP , SISTEM OUTPUT PRINTLN, crtl + enter

*Generate accessors*: crea los métodos GET Y SET para el encapsulamiento de los atributos

*FOREACH* : for( clase objeto : lista)

**VIEW CONTROLLER**

Definiciones:

**DATA CONTROLS**: encapsula lo que hay por detrás

Bind**:** son clases de java de la cual tiene requisitos para ser utilizados.

Invocamos el constructor de un bind, el constructor no puede recibir parámetros

Las validaciones se pueden hacer en las vistas y en las entidades

REQUESTS – Scope que mapea, le damos un valor y después nos devuelve un valor , dura un ciclo de vida de una página.

SESSION – Scope que MAPEA, dura la sesión http del usuario.

SCOPE: Es una estructura en memoria que tiene determinada duración , todos los objetos tienen que vivir en algún lado.

Put (key, valor), get (key)

Aplicattion: dura desde que se inicia la app hasta que termina, cada página está asociada a un componente.

BEAN: es una CLASE JAVA, tiene constructor atributos privados, los get y set.

ValueChangeListener: cuando un componente cambia de valor, se lo pasan a este manejador (AutoSubmit = true)

ActionListener: va a refrescar ese evento en particular del componente, usando también el parcial submit.

Action: Posteo de todos los datos y refrescos de la página Completa, le da una action al componente.

Managed bean: es donde se declara los backing bean para ser usado con el JSF managed facility

Backing bean: es una clase java que tiene asociados componentes y vive en el REQUEST.

Creo un backing en el cual se crea una Clase JAVA y pueda definir los componentes y bindearlos.

Cada componente se bindea con un backing bean y un método(el método le dará nombre al campo).

Auto Submit: setea los datos del componente inmediatamente y cambia el valor

PARCIAL SUBMIT: submitea algunos componentes

BINDING:

-Tipo

- A nivel componentes: por medio de java accedo a las propiedades Enlazando a algo,

- Evento: (Action, ActionListener, ValueChangeListener) a la espera del cambio del componente.

- A nivel Valor: el valor lo gestiona el framework. No necesita código extra.

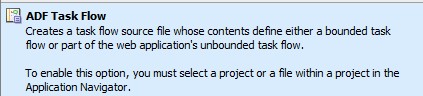
View: solo para componentes, NO USAR JAMAS

PARCIAL TRIGERS: dispara varias acciones, un componente escuche a otro/s ej.: botones

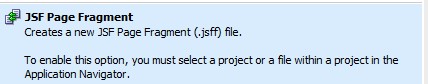
INMEDIATE: saltea las validaciones

**ACLARACION: las propiedades se ejecutan por prioridad (1ºValuechangelistener, 2ºactionListener, 3º Action)**

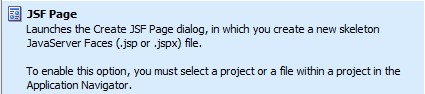
1. Creo un Task Flow



1. Luego Los Fragmentos de las paginas de las cuales quiero crear (.jsff)



1. Luego Creo una pagina (.jpsx)

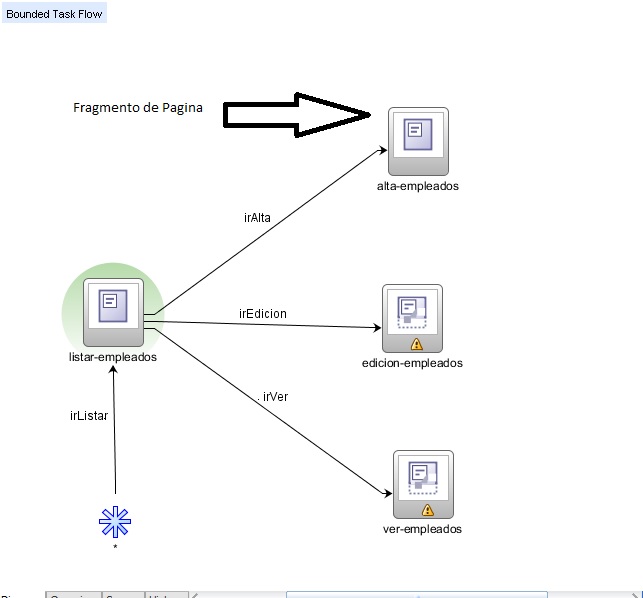


1. Arrastro el Task Flow a la página y lo denomino Región

En El Task Flow

Utilizo la herramienta **Control Flow Case** que son las conexiones entre Fragmentos de Pagina y nombro esa conexión.

El **wildcard Flow rule** es una conexión en la cual todos los fragmentos pueden utilizar.



**FORMULARIO**

PANEL GROUP LAYOUT: es un componente que agrupa otros componentes.

PANEL FORM LAYOUT: es un componente que a agrupa componentes como formulario común.

PANEL HEADER: es un componente que crea un panel con etiquetas para inserción de componentes.

PANEL ACCORDION: es un componente que panel que se minimiza.

PANEL BOX: es un componente que crea un panel con título y es interactivo.

PANEL SPLITTER: es un componente que crea un panel que divide la pantalla en 2 partes.

PANEL TABBED: es un componente que crea un panel con solapas.

PANEL WINDOW: es un componente que crea un panel genérico.

POPUP: es un componente que genera un panel invisible en el cual se puede interactuar.

ROUTER:es como si fuera una funcion IF

Expandir Tabla: **STYLE** -> styleClass: AFStretchWidth, Luego Behavior-> FetchSize: #{bindings.TablaEjemploIterator.estimatedRowCount}

AutoHeightsRows: 0

1. Utilizar la solapa Structure de Jdeveloper para analizar el árbol de los componentes
2. Crear un “Panel Header” y utilizar las herramientas Panel Form Layout o Panel Group Layout
3. Arrastrar las entidades encapsuladas del Data Control hacia el form.
4. En el Form Principal creo un botón en la solapa Data Controls->Operations->CreateInsert y lo arrastro al form para crear un objeto.
5. En Properties utilizo la propiedad Action para redirigirlo hacia un Fragmento de página.
6. Los botones del formulario deben volver al formulario principal, en caso de error y no poder volver, utilizar la propiedad Inmediate.
7. Para guardar los cambios en la base ya sea Alta, Baja o Modificación debe ser guardados con la Operations Commit.

ACLARACION: **Para realizar un ABM y utilizar operaciones como CreateInserts, Delete, etc. que llamen al PopUp y ejecuten la acción, se deberá realizar en una serie de pasos.**

**CREACION DE GET Y SET DE LOS COMPONENTES**

Selecciono la herramienta -> propiedades-> binding-> edit -> lo asocio al backing bean, y pongo nombre al método

Aclaración:

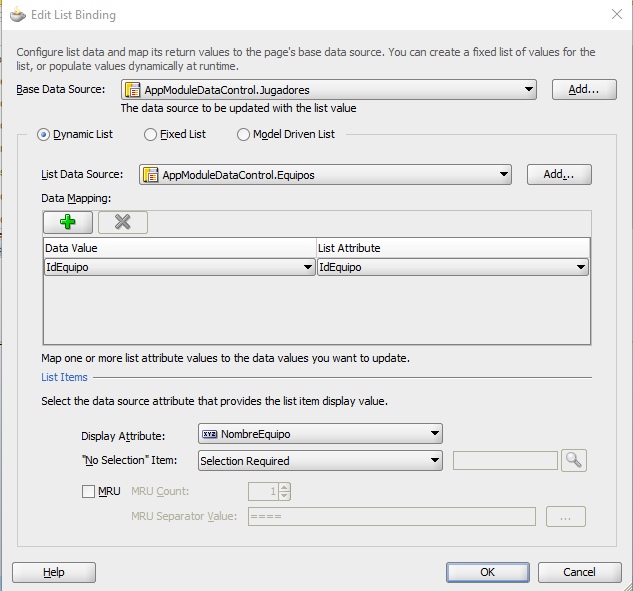
**Para que los componentes perduren en el ciclo de vida debemos crear una clase JAVA para que cuando se cierre un formulario perdure en el tiempo y morirá cuando se termina la clase**.

PLANTILLA DE ORACLE

Utilizar plantillas para el refresco de la tablas.

**HACER UNA LISTA DE VALORES EN VIEW CONTROLLER**

Arrastro del data control el campo, selecciono “select one choice” , Luego conecto con la tabla externa al cual pertenece la lista.



**CREACION DE UN ABM CON POPUP**

Link: <https://www.youtube.com/watch?v=-HK8nvnnxyk&feature=em-share_video_user>

Herramientas utilizadas

1) Task Flow

2) Pagina con Plantilla de Oracle

3) View (Task Flow)

4) Table

5) Command Button

6) PopUp

7) Dialog

8) el Managed Bean del Task Flow

9) Clase Java

10) Uso de Page Definition

11) Propiedad PopUpFetchListener

12) BindingContainer y OperationsBindings

**WEB SERVICE**

De una manera más clara se podría decir que un web service es una función que diferentes servicios o equipos utilizan; es decir, solo se envían parámetros al servidor (lugar donde está alojado el web service) y éste responderá la petición. Entre algunas que se manejan de utilizar servicios webs en las aplicaciones destacan las siguientes:

Link:

<http://docs.oracle.com/cd/E18941_01/tutorials/jdtut_11r2_14/jdtut_11r2_14.html> ( WS from Page)

<http://www.oracle.com/technetwork/developer-tools/jdev/overview/index-100269.html> (TUTORIAL ADF)

<http://docs.oracle.com/cd/E18941_01/tutorials/jdtut_11r2_55/jdtut_11r2_55_2.html> (RICH ADF)

<http://docs.oracle.com/cd/E18941_01/tutorials/jdtut_11r2_52/jdtut_11r2_52_1.html> (USING WS)

<http://oraclejuniors.blogspot.com.ar/2013/07/como-crear-un-webservice-desde-una.html> (ORACLE JUNIOR)

<http://docs.oracle.com/cd/E18941_01/tutorials/jdtut_11r2_52/jdtut_11r2_52_5.html> (WS PROXY)

VIDEO

<https://www.youtube.com/watch?v=QP20pmd9Hq8> (CREA UN WEB SERVICE FROM JAVA CLASS)

<https://www.youtube.com/watch?v=wNZJbE4pc_E> (LLAMA AL WEB SERVICE)

<https://www.youtube.com/watch?v=Cos1qSn4EvU> (BUSINESS COMPONENTS AS SERVICE)

 ningun componente que esta dentro de un for each se puede bindear

USO DE EXPRESIONES REGULARES

<af:validateRegExp pattern="[0-9]{1,10}"  
 noMatchMessageDetail="Only Numbers Please"/>