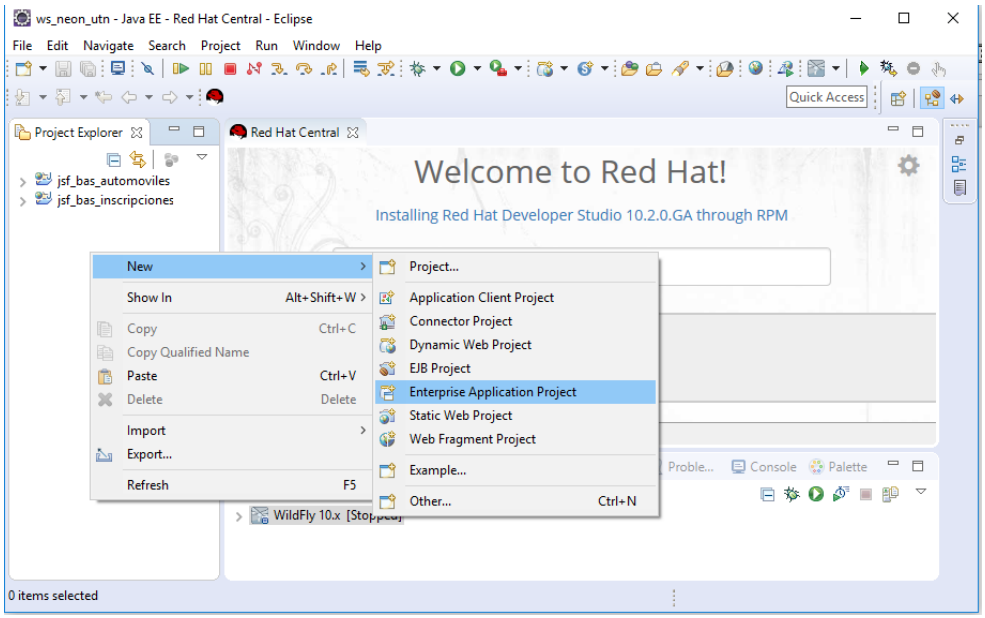
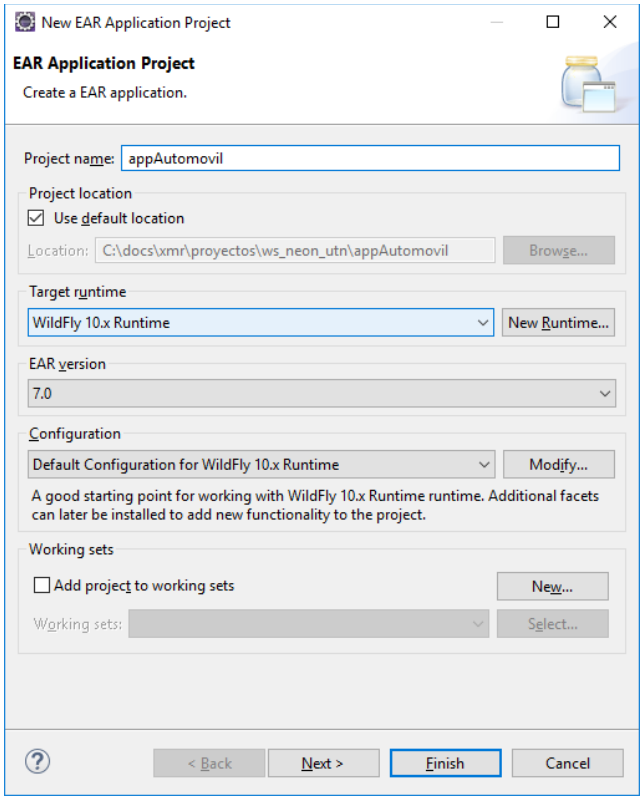
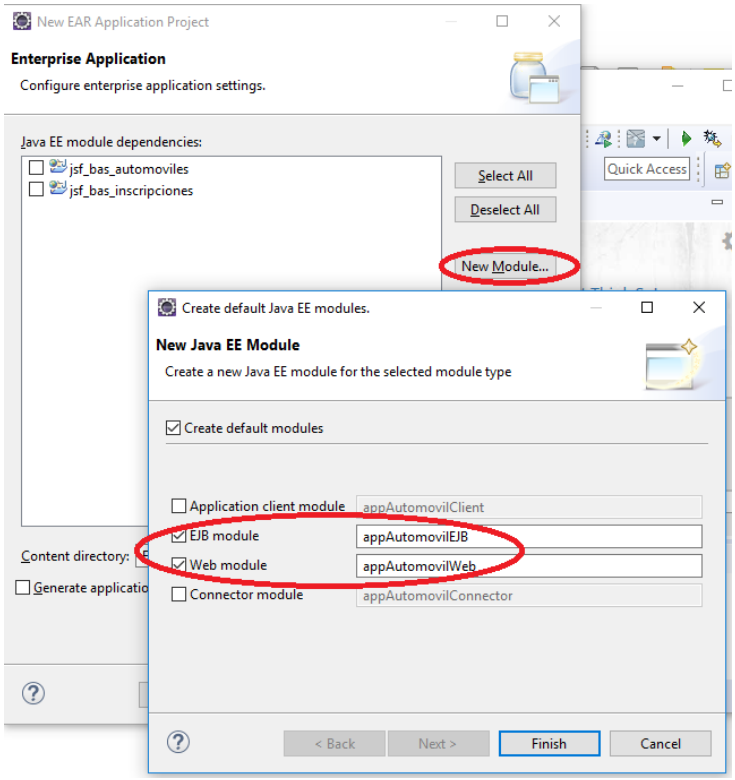
1. En la 2da parte de esta práctica creamos la estructura de proyectos de una aplicación Java Enterprise y realizamos la configuración inicial.
2. Desde el IDE Eclipse creamos una nueva aplicación empresarial de nombre **appAutomovil**:

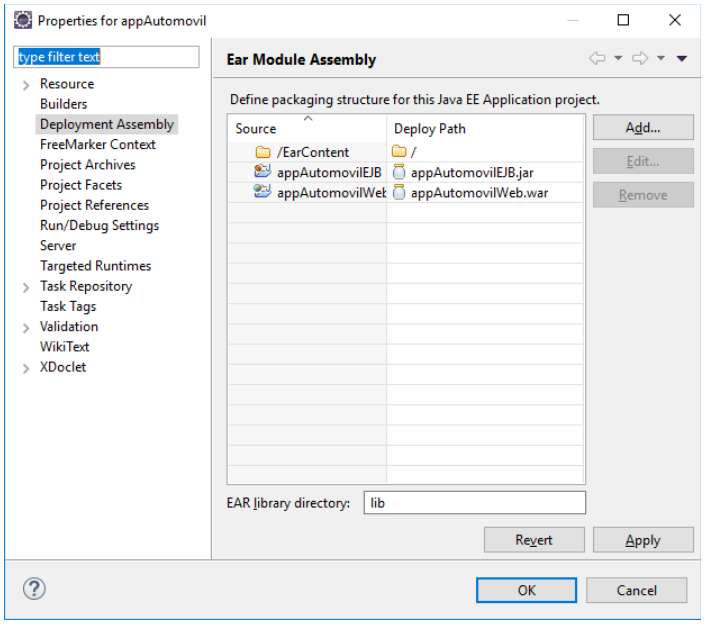




3. En la siguiente pantalla presionamos el botón **New module** y seleccionamos únicamente los módulos **EJB** y **Web**:

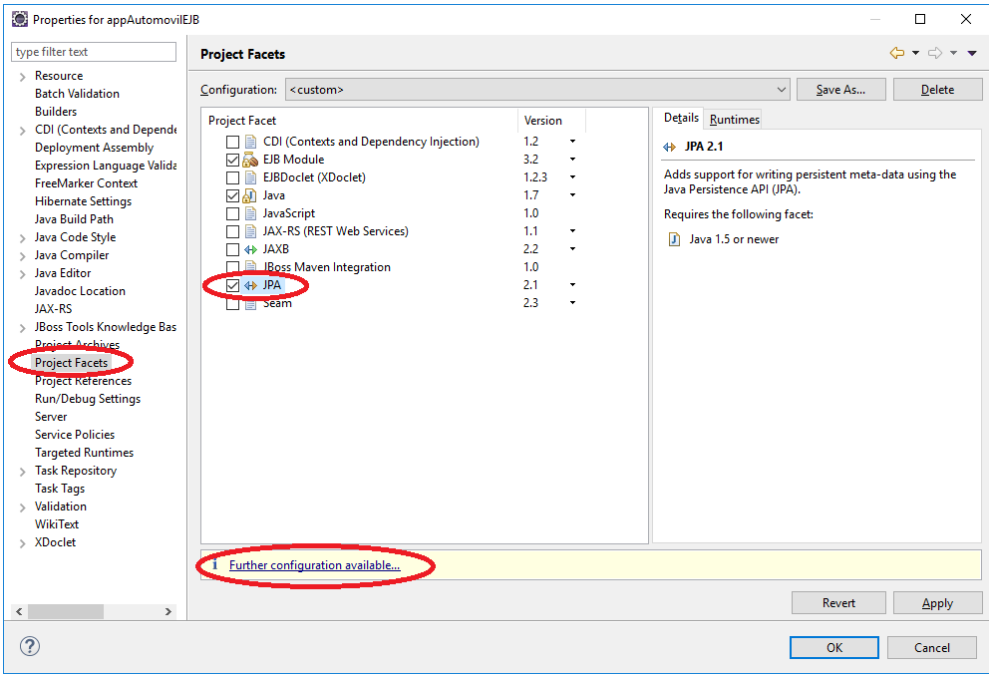


4. Finalizamos el asistente con el resto de opciones por defecto. Se ha creado un proyecto principal (appAutomovil) y dos subproyectos. Esto lo comprobamos mirando las propiedades del proyecto principal **appAutomovil** la opción **Deployment Assembly**:

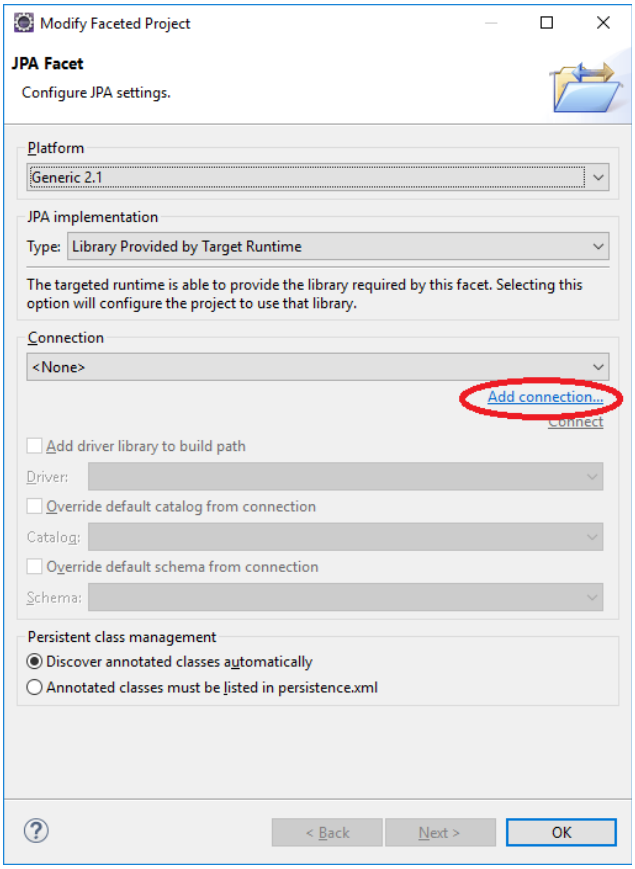


El proyecto appAutomovilEJB implementará la lógica de la aplicación (EJB + JPA) y appAutomovilWeb implementará la vista y el controlador (JSF + Primefaces).

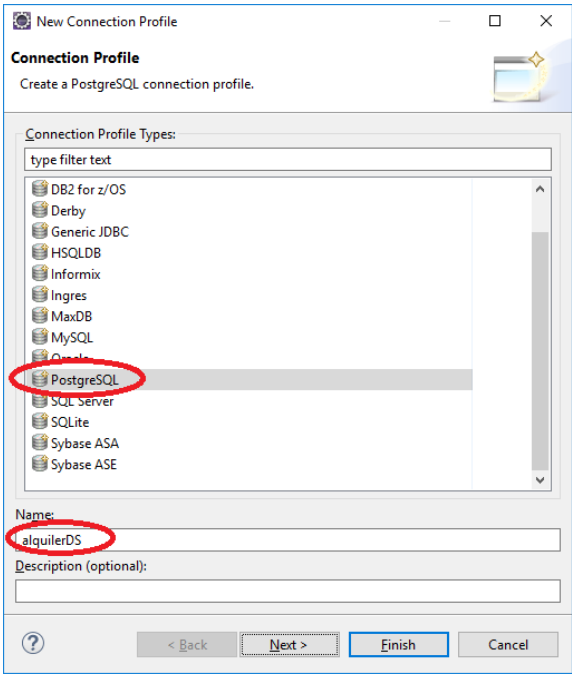
5. A continuación configuramos la librería JPA en el proyecto **appAutomovilEJB**, para ello accedemos a las propiedades del proyecto y activamos JPA:



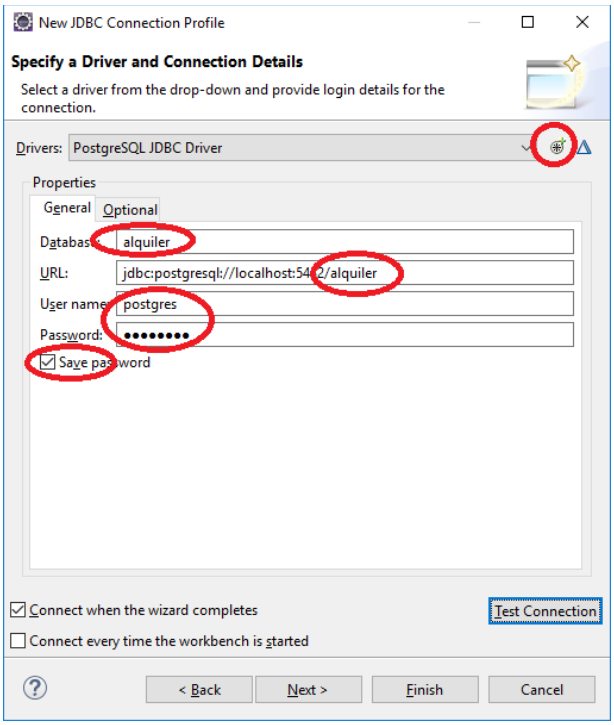
6. Seleccionamos el enlace **Further configuration available** y en el siguiente cuadro de diálogo configuramos una nueva conexión:



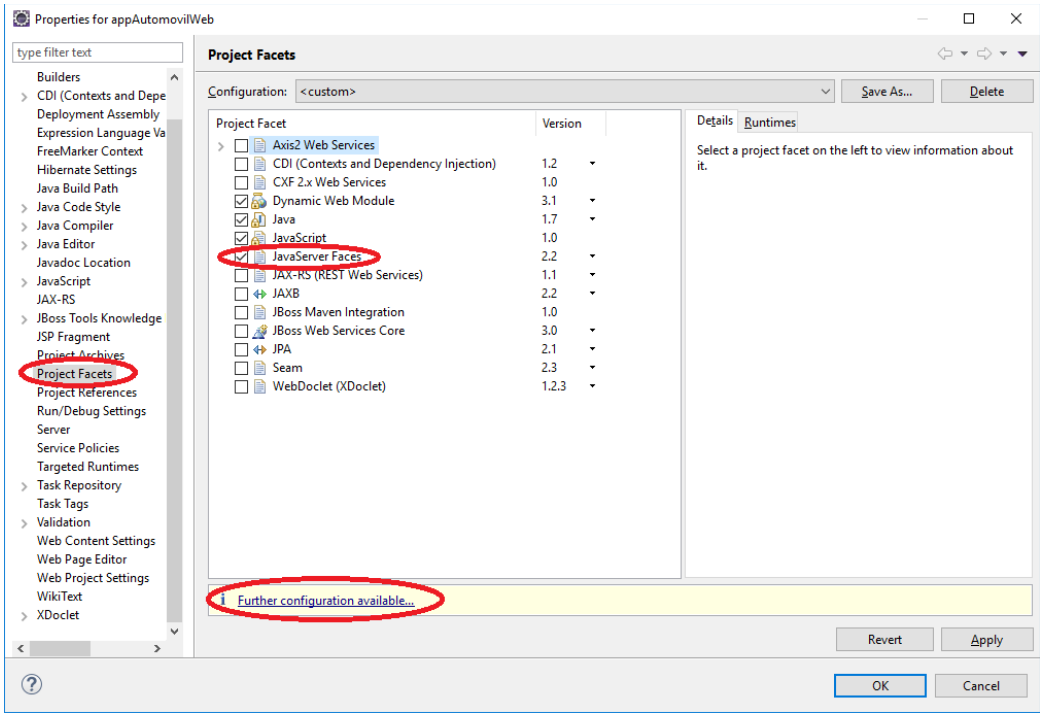
7. Seleccionamos Postgresql, que es la base de datos seleccionada para esta práctica, y configuramos el nombre de la conexión como **alquilerDS,** haciendo coincidir con el nombre del pool de conexiones de Wildfly. Esta coincidencia de nombres no es obligatoria pero se recomienda para evitar confusiones:



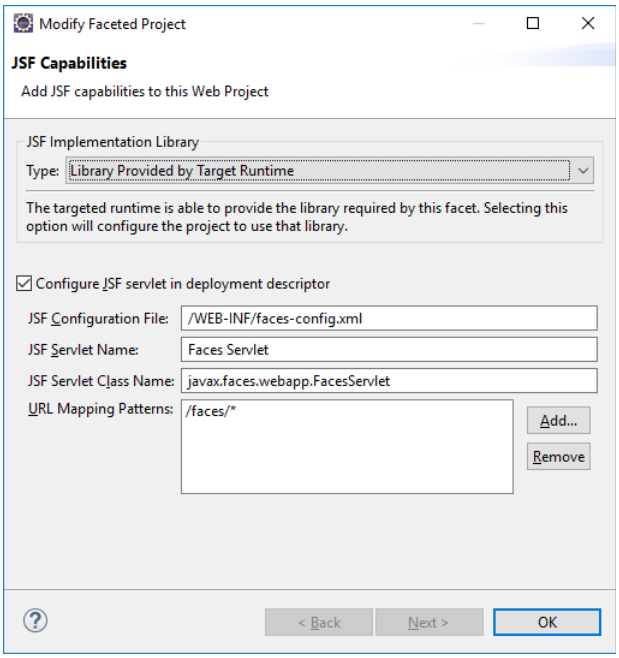
8. En la siguiente pantalla, configuramos el driver JDBC de Postgresql (si fuera necesario se crea un nuevo driver) y llenamos las opciones de conexión a la base de datos:



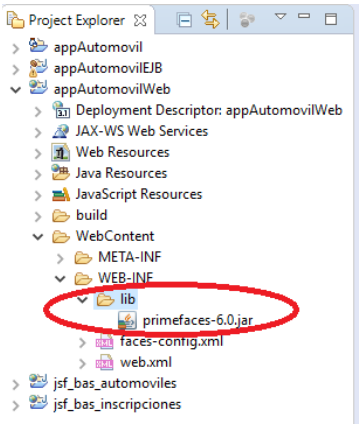
9. Presionar el botón **Test connection** para verificar la conexión. Finalizar el asistente seleccionando la nueva conexión. El resto de opciones se dejan por defecto.
10. Aceptar todos los cuadros de diálogo y cerrarlos.
11. Ahora pasamos a configurar el proyecto appAutomovilWeb, para ello accedemos a sus propiedades y activamos Java Server Faces:



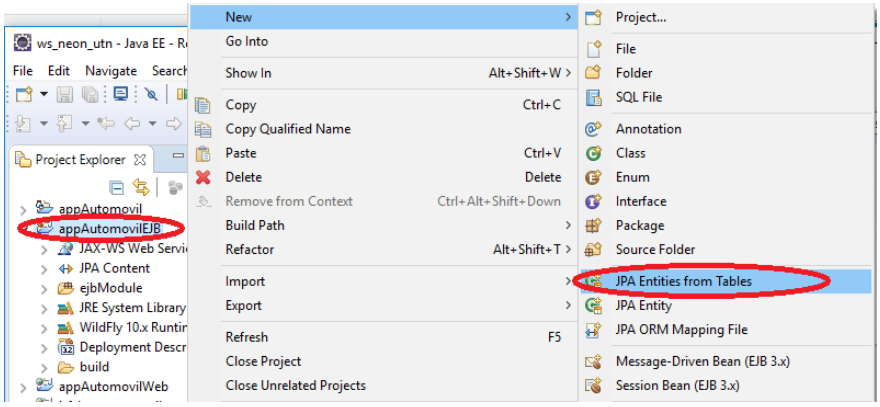
12. Al seleccionar **Further configuration available** se muestra un cuadro de diálogo de configuración del framework JSF. Se dejan todas las opciones por defecto:



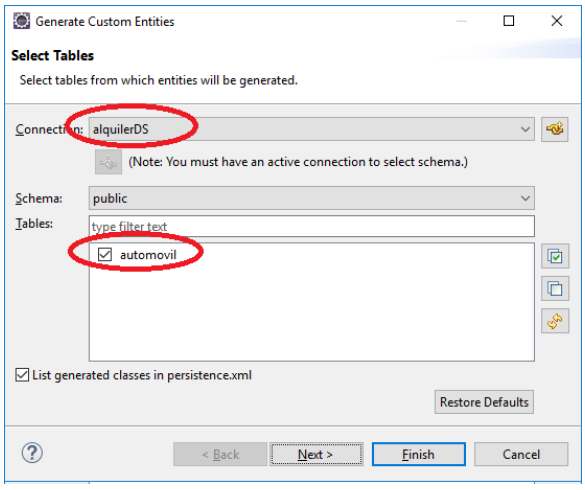
13. Aceptamos todos los cuadros de diálogo.
14. Para terminar la configuración adicionamos la librería de Primefaces:



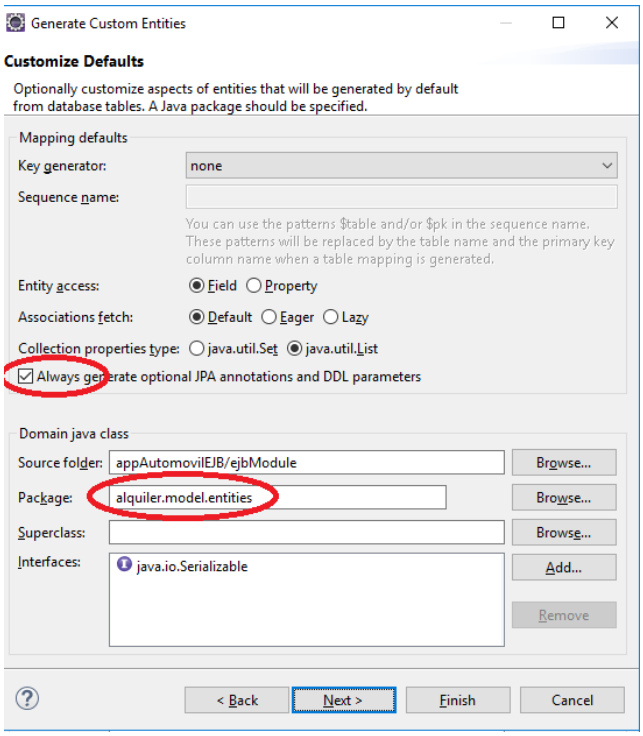
15. **Mapeo de las entidades:** el siguiente paso es crear las entidades a partir de las tablas de la base de datos. Para ello seleccionamos la opción respectiva en el proyecto **appAutomovilEJB**:



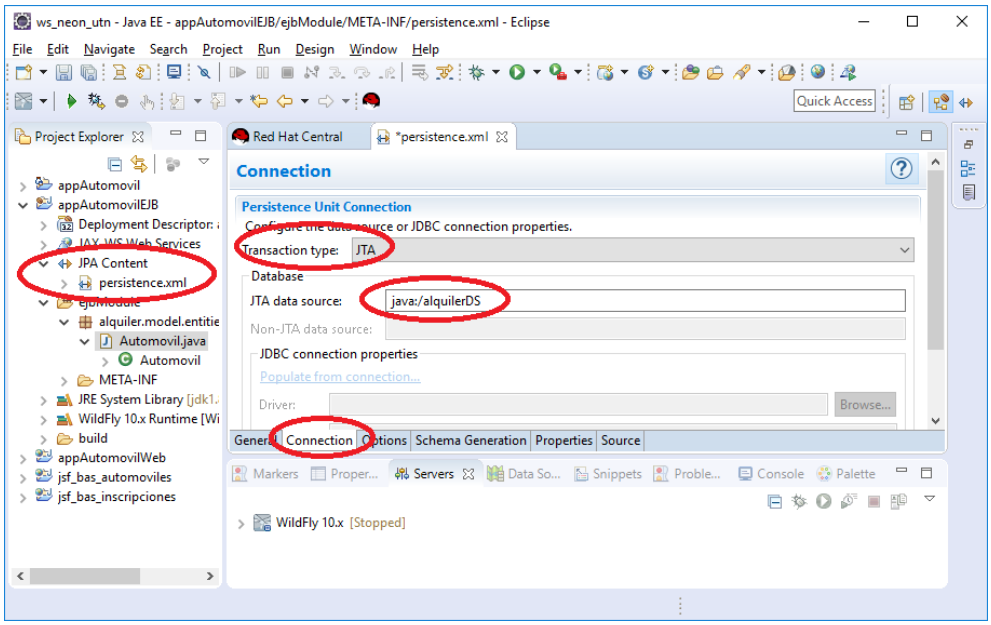
16. Mediante el asistente, seleccionamos la única conexión configurada al momento (alquilerDS) y la tabla **automovil**:



17. Pulsamos el botón siguiente. La nueva pantalla no tiene información ya que no existen relaciones con otras tablas. Pulsamos siguiente y tenemos la pantalla de personalización, donde se recomienda activar la casilla **Always generate optional JPA annotations…** y se especifica también el nombre del paquete (alquiler.model.entities):



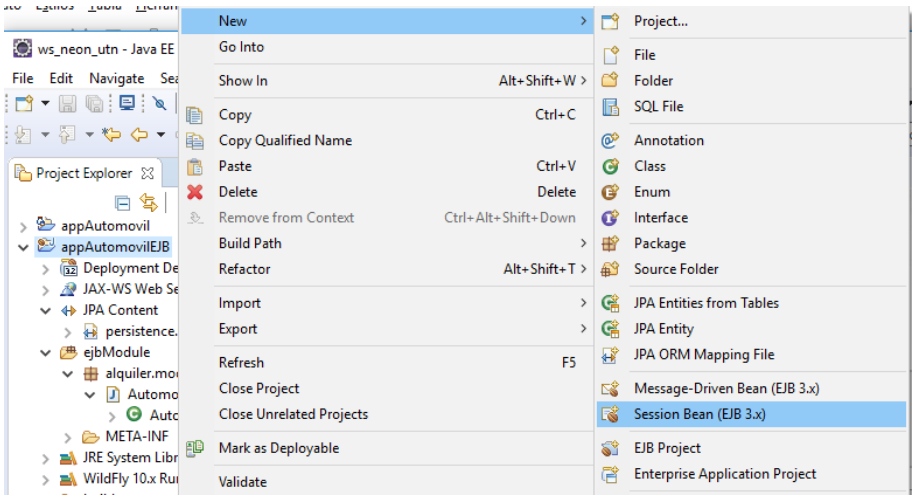
18. En la siguiente pantalla dejamos las opciones por defecto y finalizamos el asistente. Observe que se ha creado la nueva clase java **Automovil**.
19. Dentro del proyecto seleccionamos el archivo persistence.xml y editamos el datasource a utilizar (java:/alquilerDS), el tipo de transacción (JTA) y el nombre como **alquilerDS**:



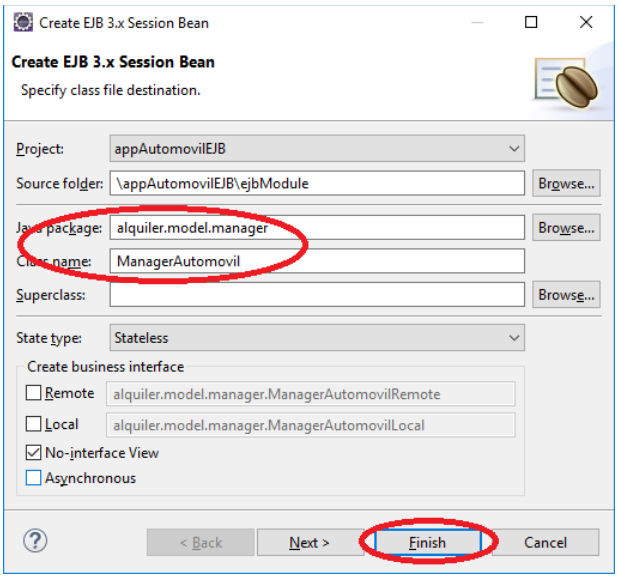
20. El código de **persistence.xml** queda así:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="alquilerDS" transaction-type="JTA">
    <jta-data-source>java:/alquilerDS</jta-data-source>
    <class>alquiler.model.entities.Automovil</class>
  </persistence-unit>
</persistence>
```

21. En el mismo proyecto appAutomovilEJB creamos un EJB que se encargará de la lógica de negocio:



22. Especificamos el nombre del paquete, el nombre de la nueva clase Java, dejamos el resto de opciones por defecto y finalizamos el asistente:



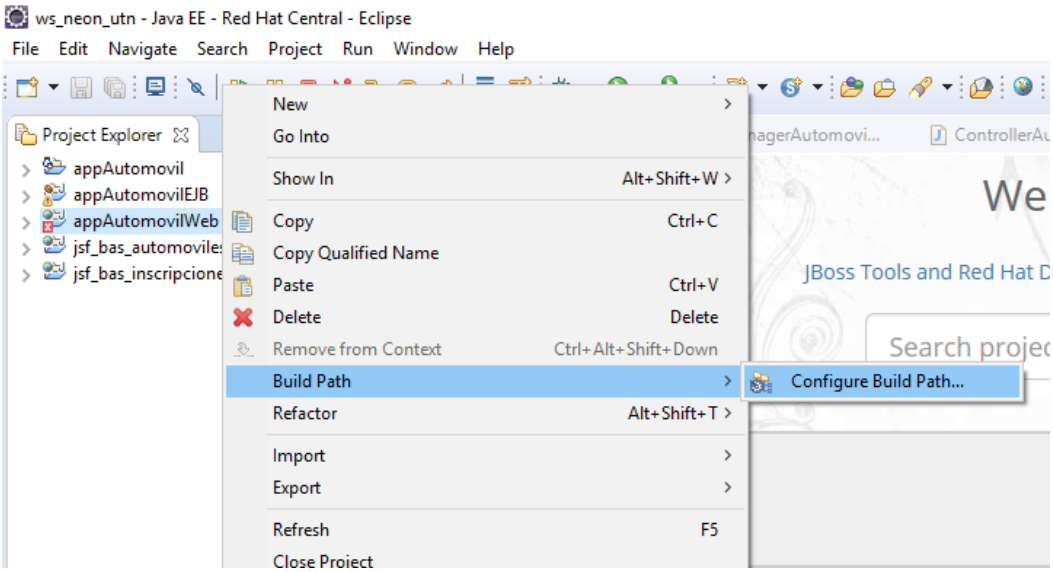
23. Editamos el nuevo EJB y colocamos el siguiente código:

```
package alquiler.model.manager;

import java.util.List;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

import alquiler.model.entities.Automovil;

/**
 * Session Bean implementation class ManagerAutomovil
 */
@Stateless
@LocalBean
public class ManagerAutomovil {
  @PersistenceContext(unitName = "alquilerDS")
  private EntityManager em;
  /**
   * Default constructor.
   */
  public ManagerAutomovil() {
  }

  public void crearLista(){
    Automovil a=new Automovil();
    a.setAlquilado(false);
    a.setAnio(2012);
    a.setColor("BLANCO");
    a.setGasolina(4);
    a.setMarca("TOYOTA");
    a.setPlaca("IBA3244");
    em.persist(a);//guardamos el nuevo automovil
    Automovil b=new Automovil();
    b.setAlquilado(true);
    b.setAnio(2010);
    b.setColor("GRIS");
    b.setGasolina(8);
    b.setMarca("CHEVROLET");
    b.setPlaca("PHQ1276");
    em.persist(b);
  }
  public void agregarAutomovil(String placa,int anio,
                  String color,String marca) throws Exception{
    if(marca==null||marca.length()==0)
      throw new Exception("Debe especificar la marca.");
    Automovil a=new Automovil();
    a.setAlquilado(false);
    a.setAnio(anio);
    a.setColor(color);
    a.setGasolina(0);
    a.setMarca(marca);
    a.setPlaca(placa);
    em.persist(a);
  }
  public Automovil findAutomovil(String placa) throws Exception{
```

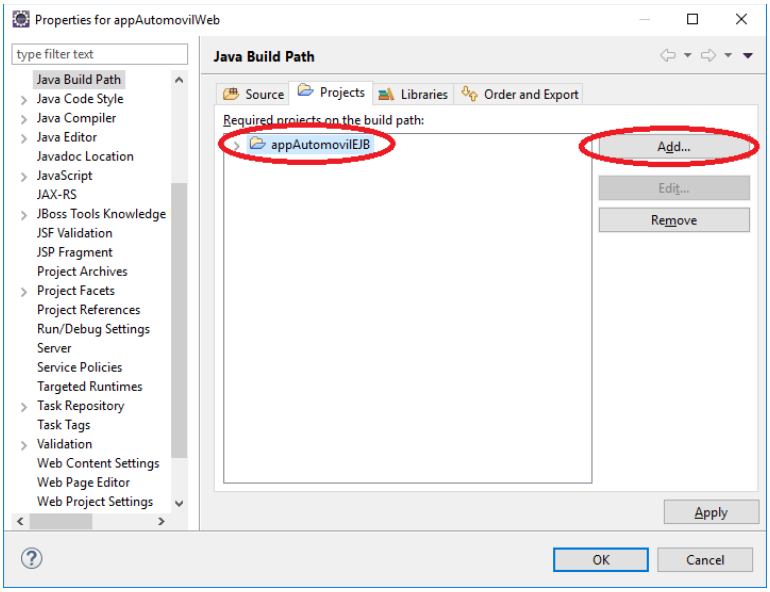
```
    Automovil a=em.find(Automovil.class, placa);
    return a;
  }
  public void alquilarAutomovil(String placa) throws Exception{
    //buscamos el automovil:
    Automovil a=findAutomovil(placa);
    if(a==null)
      throw new Exception("No existe el automovil especificado.");
    //verificamos si aun no ha sido alquilado:
    if(a.getAlquilado())
      throw new Exception("El automovil ya fue alquilado.");
    //caso contrario lo alquilamos y actualizamos la informacion:
    a.setAlquilado(true);
    em.merge(a);
  }
  public void eliminarAutomovil(String placa) throws Exception{
    //buscamos el automovil:
    Automovil a=findAutomovil(placa);
    if(a==null)
      throw new Exception("No existe el automovil especificado.");
    //lo eliminamos:
    em.remove(a);
  }
  public void editarAutomovil(String placa,String color,int gasolina)
                  throws Exception
  {
    //buscamos el automovil:
    Automovil a=findAutomovil(placa);
    if(a==null)
      throw new Exception("No existe el automovil especificado.");
    //actualizamos ciertos campos especificados en los parametros del metodo:
    a.setColor(color);
    a.setGasolina(gasolina);
    em.merge(a);
  }
  public List<Automovil> findAllAutomoviles(){
    Query q;
    List<Automovil> listado;
    String sentenciaSQL;
    sentenciaSQL = "SELECT o FROM Automovil o ORDER BY o.placa";
    q = em.createQuery(sentenciaSQL);
    listado = q.getResultList();
    return listado;
  }
  public int sumaGasolina(){
    List<Automovil> lista=findAllAutomoviles();
    int suma=0;
    for(Automovil a:lista){
      suma+=a.getGasolina();
    }
    return suma;
  }
}
```

24. Fíjese que, con respecto a la práctica anterior que sólo manejaba los datos en sesión, se han adicionado los métodos de búsqueda **findAllAutomoviles** y **findAutomovil**. El resto mantiene el mismo esquema de los métodos pero puede apreciarse que se han simplificado.

25. Ahora vamos a implementar la parte de la vista y controlador en el proyecto **appAutomovilWeb**. Debemos iniciar configurando el proyecto para que los componentes del model estén disponibles. Para ello configuramos las dependencias del proyecto con la opción Build Path:

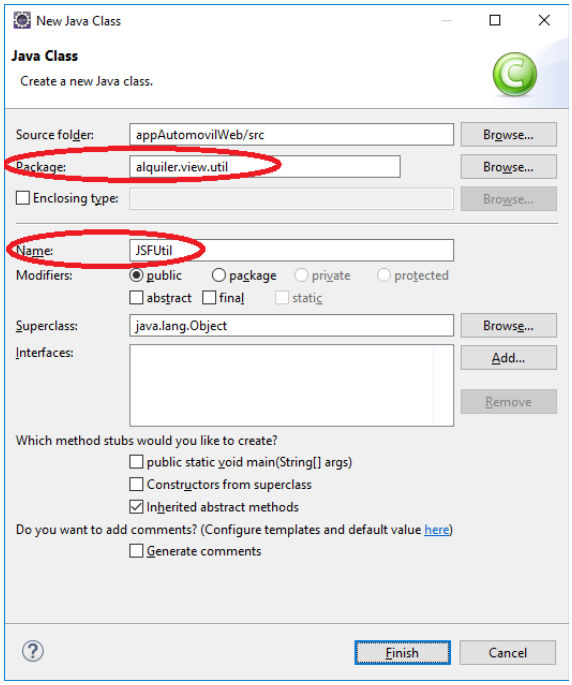


26. Adicionamos como proyecto requerido a **appAutomovilEJB**:



27. Aceptamos el cuadro de diálogo.

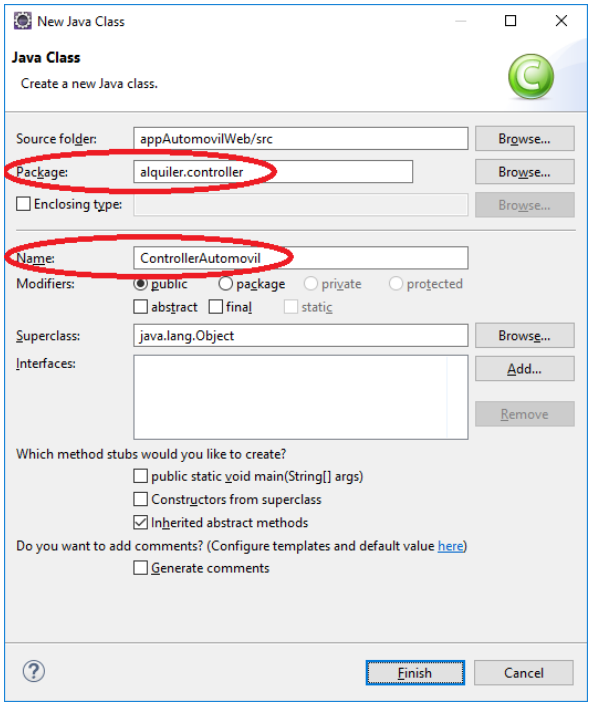
28. A continuación creamos una nueva clase Java utilitaria llamada **JSFUtil** que tendrá varios métodos que permitirán de manera simple crear mensajes personalizados para visualizar en las páginas web. Creamos la nueva clase en el paquete **alquiler.view.util**:



29. Editamos la nueva clase:

```
package alquiler.view.util;

import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;

public class JSFUtil {
  public static void crearMensajeInfo(String mensaje){
    FacesMessage msg = new FacesMessage();
    msg.setSeverity(FacesMessage.SEVERITY_INFO);
    msg.setSummary(mensaje);
    FacesContext.getCurrentInstance().addMessage(null, msg);
  }
  public static void crearMensajeWarning(String mensaje){
    FacesMessage msg = new FacesMessage();
    msg.setSeverity(FacesMessage.SEVERITY_WARN);
    msg.setSummary(mensaje);
    FacesContext.getCurrentInstance().addMessage(null, msg);
  }
  public static void crearMensajeError(String mensaje){
    FacesMessage msg = new FacesMessage();
    msg.setSeverity(FacesMessage.SEVERITY_ERROR);
    msg.setSummary(mensaje);
    FacesContext.getCurrentInstance().addMessage(null, msg);
  }
}
```

30. A continuación creamos una nueva clase Java llamada **ControllerAutomovil** que mantendrá una estructura muy parecida al controlador de la práctica anterior:



31. Editamos la nueva clase:

```
package alquiler.controller;

import java.util.List;

import javax.annotation.PostConstruct;
import javax.ejb.EJB;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

import alquiler.model.entities.Automovil;
import alquiler.model.manager.ManagerAutomovil;
import alquiler.view.util.JSFUtil;

@ManagedBean
@SessionScoped
public class ControllerAutomovil {
  private String placa;
  private int anio;
  private String color;
  private String marca;
  private List<Automovil> lista;
  private int gasolina;
  private int totalGasolina;
  @EJB
  private ManagerAutomovil managerAutomovil;
  @PostConstruct
  public void iniciar(){
    lista=managerAutomovil.findAllAutomoviles();
    totalGasolina=managerAutomovil.sumaGasolina();
```

```
  anio=2016;
}
public void actionListenerAgregar(){
  try {
    managerAutomovil.agregarAutomovil(placa, anio, color, marca);
    lista=managerAutomovil.findAllAutomoviles();
    totalGasolina=managerAutomovil.sumaGasolina();
    JSFUtil.crearMensajeInfo("Automóvil registrado.");
  } catch (Exception e) {
    JSFUtil.crearMensajeError(e.getMessage());
    e.printStackTrace();
  }
  placa="";
  anio=2016;
  color="";
  marca="";
}
public void actionListenerReset(){
  managerAutomovil.crearLista();
  lista=managerAutomovil.findAllAutomoviles();
  totalGasolina=managerAutomovil.sumaGasolina();
}
public void actionListenerAlquilar(){
  try {
    managerAutomovil.alquilarAutomovil(placa);
    lista=managerAutomovil.findAllAutomoviles();
    JSFUtil.crearMensajeInfo("Alquiler realizado.");
  } catch (Exception e) {
    JSFUtil.crearMensajeError(e.getMessage());
    e.printStackTrace();
  }
}
public void actionListenerAlquilarFila(Automovil automovil){
  try {
    managerAutomovil.alquilarAutomovil(automovil.getPlaca());
    lista=managerAutomovil.findAllAutomoviles();
    JSFUtil.crearMensajeInfo("Alquiler realizado con éxito.");
  } catch (Exception e) {
    JSFUtil.crearMensajeError(e.getMessage());
    e.printStackTrace();
  }
}
public void actionListenerEliminar(String placa){
  try {
    managerAutomovil.eliminarAutomovil(placa);
    lista=managerAutomovil.findAllAutomoviles();
    totalGasolina=managerAutomovil.sumaGasolina();
    JSFUtil.crearMensajeInfo("Automóvil de placa "+placa+" eliminado.");
  } catch (Exception e) {
    JSFUtil.crearMensajeError(e.getMessage());
    e.printStackTrace();
  }
}
public void actionListenerCargar(Automovil automovil){
  placa=automovil.getPlaca();
  color=automovil.getColor();
  gasolina=automovil.getGasolina();
}
public void actionListenerActualizar(){
```

```
    try {
      managerAutomovil.editarAutomovil(placa, color, gasolina);
      lista=managerAutomovil.findAllAutomoviles();
      totalGasolina=managerAutomovil.sumaGasolina();
      JSFUtil.crearMensajeInfo("Actualización correcta.");
    } catch (Exception e) {
      JSFUtil.crearMensajeError(e.getMessage());
      e.printStackTrace();
    }
  }
  public String getPlaca() {
    return placa;
  }
  public void setPlaca(String placa) {
    this.placa = placa;
  }
  public int getAnio() {
    return anio;
  }
  public void setAnio(int anio) {
    this.anio = anio;
  }
  public String getColor() {
    return color;
  }
  public void setColor(String color) {
    this.color = color;
  }
  public String getMarca() {
    return marca;
  }
  public void setMarca(String marca) {
    this.marca = marca;
  }
  public List<Automovil> getLista() {
    return lista;
  }
  public void setLista(List<Automovil> lista) {
    this.lista = lista;
  }
  public int getGasolina() {
    return gasolina;
  }
  public void setGasolina(int gasolina) {
    this.gasolina = gasolina;
  }
  public int getTotalGasolina() {
    return totalGasolina;
  }
  public void setTotalGasolina(int totalGasolina) {
    this.totalGasolina = totalGasolina;
  }
}
```

32. Ahora implementamos la página **index.xhtml**:

```
<DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://primefaces.org/ui">

<h:head></h:head>
<body>
    <p:messages autoUpdate="true"></p:messages>
    <h:panelGrid columns="2">
        <h:form id="form1">
            <p:panel header="MANEJO DE AUTOMOVILES">
                <p:panelGrid columns="2">
                    <h:outputText value="PLACA:" />
                    <p:inputText value="#{controllerAutomovil.placa}" required="true"
                        requiredMessage="Especifique la placa" />
                    <h:outputText value="AÑO:" />
                    <p:spinner value="#{controllerAutomovil.anio}" max="2016"
                        min="2010" />
                    <h:outputText value="COLOR:" />
                    <p:inputText value="#{controllerAutomovil.color}" required="true"
                        requiredMessage="Indique el color" />
                    <h:outputText value="MARCA:" />
                    <p:inputText value="#{controllerAutomovil.marca}" />
                    <p:commandButton value="Agregar" icon="ui-icon-disk"
update="@form,:form2:tabla1"
                        actionListener="#{controllerAutomovil.actionListenerAgregar()}">
                    </p:commandButton>
                </p:panelGrid>
            </p:panel>
        </h:form>
        <h:form id="form2">
            <p:panel header="LISTA DE REGISTROS">
                <p:dataTable value="#{controllerAutomovil.lista}" var="a"
                    id="tabla1">
                    <p:column headerText="PLACA">
                        <h:outputText value="#{a.placa}" />
                    </p:column>
                    <p:column headerText="AÑO">
                        <h:outputText value="#{a.anio}" />
                    </p:column>
                    <p:column headerText="COLOR">
                        <h:outputText value="#{a.color}" />
                    </p:column>
                    <p:column headerText="MARCA">
                        <h:outputText value="#{a.marca}" />
                    </p:column>
                    <p:column headerText="GASOLINA" style="text-align:right">
                        <h:outputText value="#{a.gasolina}"/>
                        <f:facet name="footer">
                            <h:outputText value="#{controllerAutomovil.totalGasolina}" />
                        </f:facet>
                    </p:column>
                    <p:column headerText="ALQUILADO">
                        <h:outputText value="#{a.alquilado}" />
                    </p:column>
                    <p:column headerText="ALQUILAR">
                        <p:commandButton icon="ui-icon-check" update="@form"

actionListener="#{controllerAutomovil.actionListenerAlquilarFila(a)}"></p:commandButton>
```

```
                    </p:column>
                    <p:column headerText="ELIMINAR">
                        <p:commandButton icon="ui-icon-close" update="@form"
        actionListener="#{controllerAutomovil.actionListenerEliminar(a.placa)}"></p:commandButton>
                    </p:column>
                    <p:column headerText="EDITAR">
                        <p:commandButton icon="ui-icon-pencil" update="form4"
                            onclick="PF('dialogo1').show();"
                actionListener="#{controllerAutomovil.actionListenerCargar(a)}"></p:commandButton>
                    </p:column>
                </p:dataTable>
                <p:commandButton value="RESET" icon="ui-icon-refresh"
                    update="tabla1"
                    actionListener="#{controllerAutomovil.actionListenerReset()}"></p:commandButton>
            </p:panel>
        </h:form>
        <h:form id="form3">
            <p:panel header="ALQUILER">
                <h:outputText value="Placa:" />
                <h:inputText value="#{controllerAutomovil.placa}" required="true" />
                <p:commandButton icon="ui-icon-check" update=":form2:tabla1"
                    actionListener="#{controllerAutomovil.actionListenerAlquilar()}"
                    value="Alquilar" />
            </p:panel>
        </h:form>
    </h:panelGrid>
    <p:dialog header="Edición de automovil" widgetVar="dialogo1"
        id="dialogo1" modal="true" height="200">
        <h:form id="form4">
            <p:messages autoUpdate="true"></p:messages>
            <p:panelGrid columns="2">
                <h:outputText value="PLACA:" />
                <h:outputText value="#{controllerAutomovil.placa}" />
                <h:outputText value="COLOR:" />
                <p:inputText value="#{controllerAutomovil.color}" required="true" />
                <h:outputText value="GASOLINA:" />
                <p:inputText value="#{controllerAutomovil.gasolina}" required="true" />
                <p:commandButton value="Actualizar" update="@form,:form2:tabla1"

actionListener="#{controllerAutomovil.actionListenerActualizar()}"></p:commandButton>
            </p:panelGrid>
        </h:form>
    </p:dialog>
</body>
</html>
```

33. Finalmente ejecutamos el archivo **index.xhtml**: