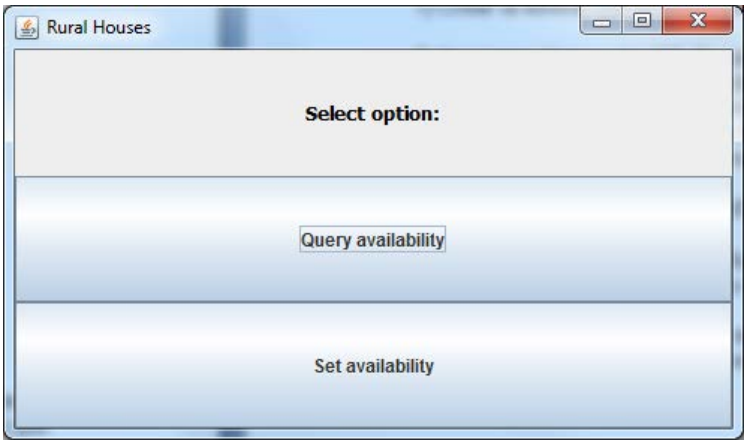
Hay que crear un JSF inicial equivalente al siguiente, que navegue a un JSF distinto, dependiendo del caso de uso de que se trate: QueryAvailibility.xhtml y SetAvailability.xhtml, por ejemplo.



Consejo: en todos los JSFs conviene añadir un componente <h:messages> al final, para que se muestren posibles mensajes inesperados que puedan generarse en tiempo de ejecución. Identificarlos es una ayuda importante para un desarrollo correcto del software.

Ayuda: para crear un calendario que se asemeje al jCalendar del proyecto anterior, se puede utilizar el componente p:calendar de Prime Faces, que es una implementación de Faces más completa que las que se ofrecen con Eclipse (MyFaces o Mojarra).

Ejemplo:

```
<p:calendar id="inline" value="#{queryAvailabilityBean.arrivalDay}"
                        navigator="true" mode="inline" />
```

Cómo se usa: https://www.primefaces.org/showcase/ui/input/calendar.xhtml

- Para que funcione p:calendar hay que descargar la librería .jar de Prime Faces y añadirla tanto al proyecto, como al directorio donde se despliegan las clases en el servidor web (Project Properties => Deployment Assembly)

- Hay que añadir el espacio de nombres en el JSF. Usar http://primefaces.org/ui en vez de http://primefaces.prime.com.tr/ui porque este último es específico para trabajar con la versión Prime Faces 2.X

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
...
```

- calendar de Prime Faces no funciona bien si se utilizan las etiquetas HTML <head> y <body>. Hay que usar las etiquetas propias de JSF <h:head> y <h:body>

```
<h:head>
      <title>Query Availability</title>
</h:head>
```

Ayudas: para la lista desplegable

Para que los elementos de una lista desplegable se obtengan de manera dinámica invocando a un bean gestionado que los devuelva hay que utilizar f:selectItems del componente h:selectOneMenu, en vez de <f:selectItem itemValue=…> (ya que esto último implica proporcionarlos uno a uno de manera explícita y estática)

```
<h:selectOneMenu value="#{queryAvailabilityBean.casa}">
<f:selectItems value="#{queryAvailabilityBean.casas}" />
</h:selectOneMenu>
```

Ayuda: invocar a la lógica del negocio desde el bean

En la implementación del bean, hay que invocar a la lógica del negocio del proyecto anterior, para obtener las casas rurales, por ejemplo.

Para ello, el bean gestionado deberá llamar al método getAllRuralHouses() de la lógica del negocio implementada en businessLogic.FacadeImplementationWS.

```
ApplicationFacadeInterfaceWS facadeBL; // Atributo que guarda lógica negocio

facadeBL=new FacadeImplementationWS(); // Obtención objeto lógica negocio

List<RuralHouse> casas=facadeBL.getAllRuralHouses(); // Invocación a log. neg.
```

Ayuda: uso de conversores

- En la lista desplegable

que se ha definido en el JSF correspondiente así:

```
<h:selectOneMenu value="#{queryAvailabilityBean.casa}">
<f:selectItems value="#{queryAvailabilityBean.casas}" />
</h:selectOneMenu>
```

se muestran las casas rurales que el método getCasas() del bean asociado al nombre *queryAvailabilityBean* devuelve. Además, ya se ha dicho que en la implementación del método getCasas() se invocará al método *facadeBL*.`getAllRuralHouses()` de la lógica del negocio, que devuelve una lista de objetos: `List<RuralHouse>`.

Sin embargo, en el navegador web donde finalmente se visualiza el JSF no se muestran objetos Java: se muestra código HTML con formato de texto. Por lo tanto, lo que el controlador JSF enviará al navegador web para que lo coloque en la lista desplegable serán los strings devueltos por el método toString() de RuralHouse para cada casa rural.

Por otro lado, cuando el usuario seleccione un elemento de la lista desplegable se necesita convertirlo a un objeto de tipo RuralHouse para pedirle que ejecute el método getOffers(firstDay,lastDay). Con JSF se pueden crear conversores propios, pero en este caso se recomienda utilizar que utilizar conversores de Omnifaces (http://showcase.omnifaces.org/converters/SelectItemsConverter). Su uso es muy sencillo:

```
<h:selectOneMenu value="#{queryAvailabilityBean.casa}"
                 converter="omnifaces.SelectItemsConverter">
<f:selectItems value="#{queryAvailabilityBean.casas}" />
</h:selectOneMenu>
```

- No hay que olvidarse de añadir el .jar de omnifaces al proyecto y al despliegue (deployment) del servidor. Nota: cuidado con las versiones del jar. Si el proyecto JSF es v2.0, hay que descargar el omnifaces-1.8.1.jar.

Ayuda: para mostrar los datos de las ofertas

- Se puede usar un componente JSF h:dataTable, el cual permite ir mostrando los datos de las ofertas por columnas.

Ofertas disponibles en esas fechas				
Num oferta	**Primer día**	**Último día**	**Precio**	**Casa rural**
#{item.offerNumber}	#{item.firstDay}	#{item.lastDay}	#{item.price}	#{item.ruralHouse.houseNumber}

```
<h:dataTable id="table1" value="#{queryAvailabilityBean.offers}" var="item" border="1">
        <f:facet name="header">
                <h:outputText value="Ofertas disponibles en esas fechas" />
        </f:facet>

        <h:column>
                <f:facet name="header">
                        <h:outputText value="Num oferta" />
                </f:facet>
                <h:outputText value="#{item.offerNumber}"/>
        </h:column>
        <h:column>
                <f:facet name="header">
                        <h:outputText value="Primer día" />
                </f:facet>
                <h:outputText value="#{item.firstDay}">
                    <f:convertDateTime pattern="dd/MM/yyyy" timeZone="CET"/>
                </h:outputText>
        </h:column> ...
</h:dataTable>
```

Ayuda: usar validadores para la entrada

- Hay que identificar errores en la entrada cuando, por ejemplo, no se introduce un número y se espera (por ejemplo: f:validateLongRange)

Ayuda: navegar entre páginas usando "action" de "h:commandButton"

- Usar botones h:commandButton porque permiten definir un valor para el atributo "action" que es el que servirá para la navegación entre JSFs. Ese valor para el action deberá ser un string definido en el faces-config.xml. Además, si se desea que se navegue a otra vista y no se está interesado en los valores que se hayan podido introducir, entonces hay que añadir `immediate="true"`

```
<h:commandButton value="Volver" action="inicio" immediate="true">
```

Ayuda: obtención de la lógica del negocio (FacadeImplementationWS) desde distintos beans

Se puede trabajar con un Bean diferente para cada caso de uso (que se encargue de manejar la interacción de datos con cada JSF). Pero como todos ellos necesitarán invocar a la misma lógica del negocio, en vez de que todos ellos creen su propia instancia de la misma, lo mejor es que definir otro Bean que haga de fachada (un FacadeBean) que proporcione la única instancia de FacadeImplementationWS. La implementación sería parecida a un patrón Singleton, pero en vez de una única instancia de FacadeBean, getInstance() devolvería la instancia singleton de FacadeImplementationWS.

Nota: si el bean FacadeBean no se usa directamente en los JSF, entonces no hace falta declararlo en el faces-config.xml. Se trata de una clase Java utilizada en los beans gestionados usados en los JSFs.

```
public class FacadeBean {

        private static FacadeBean singleton = new FacadeBean( );

        private static ApplicationFacadeInterfaceWS facadeInterface;

        private FacadeBean(){
        try { facadeInterface=new FacadeImplementationWS(); }
        catch (Exception e) {
         System.out.println("FacadeBean: error creando la lógica del negocio: "+e.getMessage());
        }}
        public static ApplicationFacadeInterfaceWS getBusinessLogic( ) {
                    return facadeInterface;
        }
}
```

Y desde el resto de beans, se puede invocar a la lógica del negocio así:

```
        facadeBL=FacadeBean.getBusinessLogic();
        casas=facadeBL.getAllRuralHouses();
```

Ayuda para resolver un posible problema: si al seleccionar un elemento de la lista desplegable (SelectOneMenu) da un error de "valor no válido"

-El conversor SelectItemsConverter de omnifaces necesita, por un lado, transformar a string los objetos que se introducen en la lista desplegable y, por otro lado, encontrar correctamente el objeto que corresponda al string seleccionado en la lista desplegable.

Para que funcione correctamente hay que proporcionar implementaciones adecuadas de "toString" y de "equals" (esta última se usa para encontrar si el objeto seleccionado se encuentra entre los que están en la lista desplegable, ya que si no lo encuentra da un error de "valor no válido").

Podrían ser equivalentes a las siguientes (que son de la clase RuralHouse):

```
@Override
public String toString() {
	return this.houseNumber + ": " + this.city;
}

@Override
public boolean equals(Object obj) {
	RuralHouse other = (RuralHouse) obj;
	if (this == obj)
	  return true;
	if (obj == null)
	  return false;
	if (getClass() != obj.getClass())
	  return false;
	if (!houseNumber.equals(other.houseNumber))
	  return false;
    return true;
}
```