2018

Tito Sánchez

Tour JavaUP – CertificaTIC

México 2018

[JAVA EN EL ESCRITORIO: FRAMEWORKS Y ALTERNATIVAS]

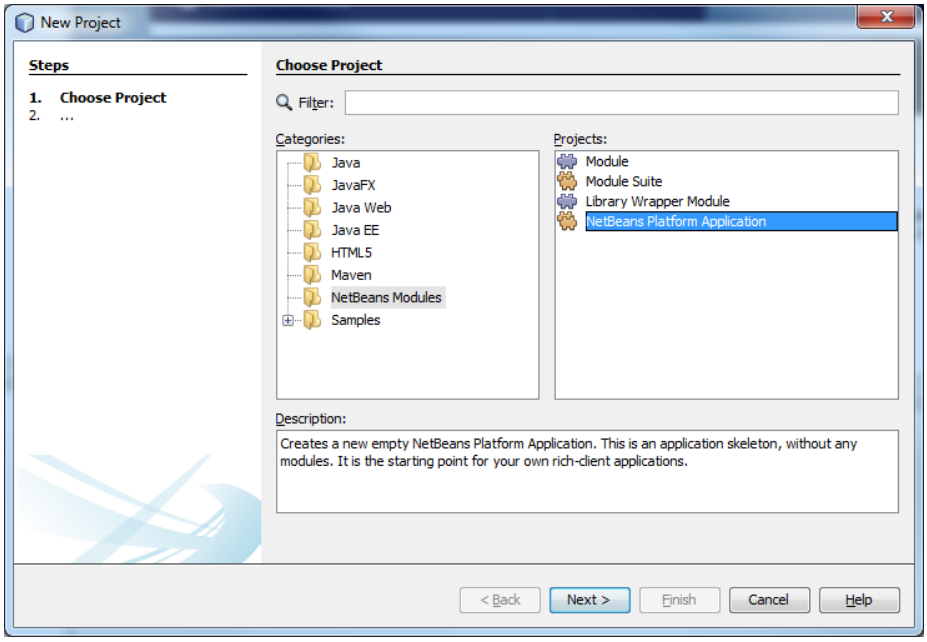
En muchos casos es necesario desarrollar una aplicación Standalone en Java para proporcionar funcionalidad que no se puede obtener en un entorno Web, NetBeans Platform proporciona un Framework en el que pueden desarrollar aplicaciones de escritorio de gran tamaño. La plataforma NetBeans contiene APIs que simplifican el manejo de ventanas, acciones, archivos y muchas otras cosas típicas en las aplicaciones swing. Netbeans Platform permite el desarrollo modular de aplicaciones (plugins) que pueden instalarse directamente en el IDE.

El objetivo del taller es guiar a los participantes para que desarrollen una aplicación que haga uso de los principales componentes del Framework de Netbeans Platform.

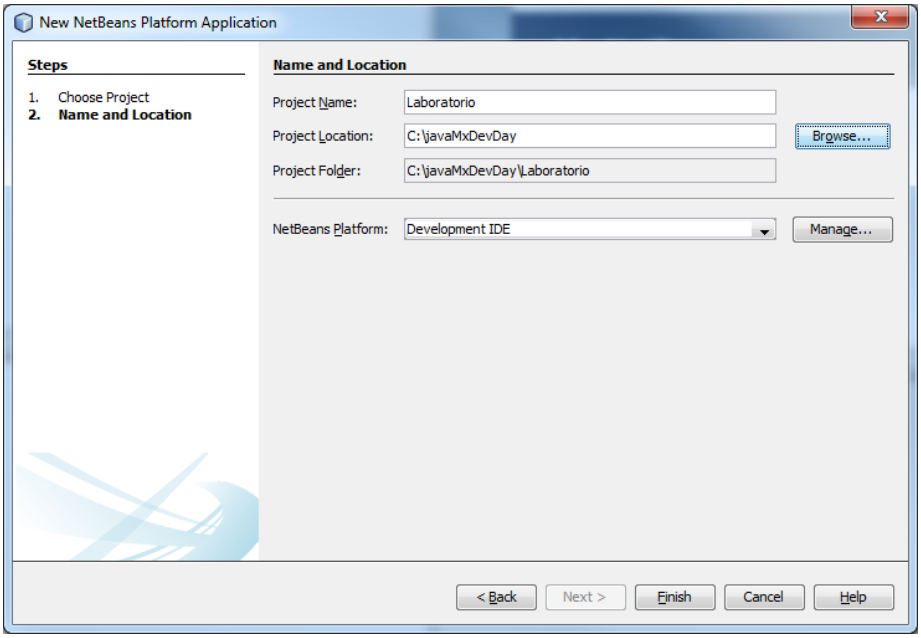
Contenido:

Crear un nuevo proyecto

a. File / New Project / NetBeans Modules / NetBeans Platform Application

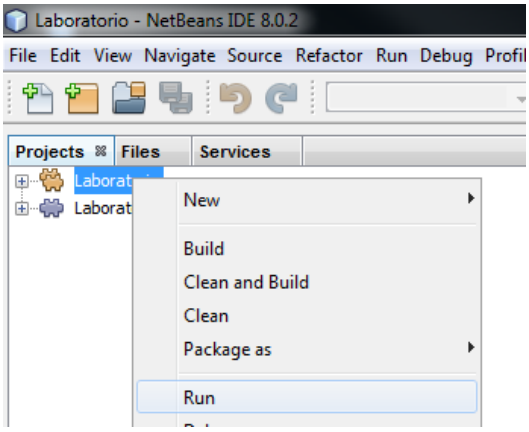


b. Name and location **Laboratorio**
   i. Project Name: Laboratorio
   ii. Project Location: Seleccionar la carpeta donde va a estar el proyecto



Ejecutar el proyecto

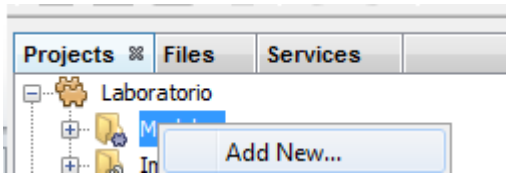
Dar clic derecho sobre el proyecto y seleccionar la opción RUN



- Si muestra el error *Warning could not install some modules* dar clic *en* ***Disable Modules and Continue***

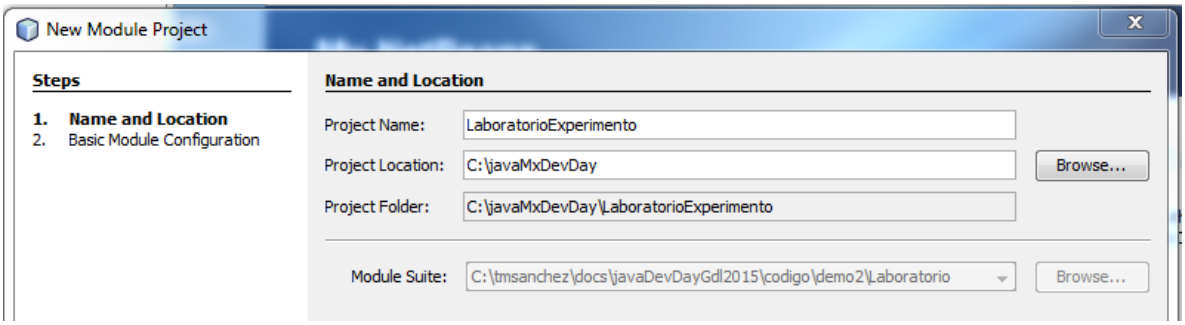
Agregar un nuevo módulo

- Dar clic derecho en *Modules* y seleccionar *Add New*

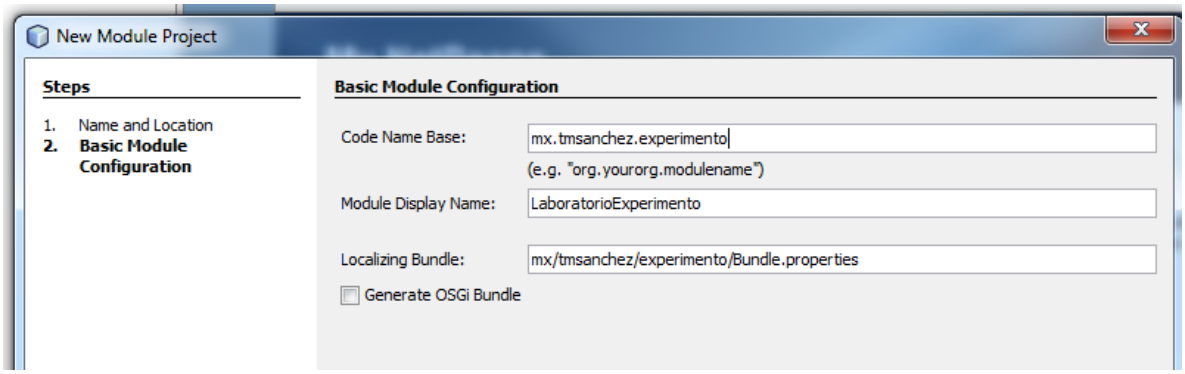


En el Asistente completar la siguiente información:

- Step 1
  - Project Name **LaboratorioExperimento**
  - Project Location: Carpeta donde va quedar el módulo
  - Clic en **Next**



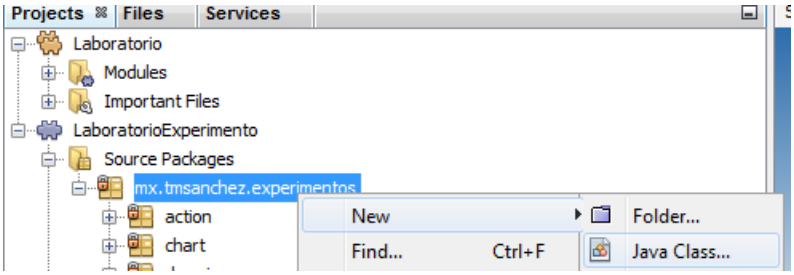
- Step 2
  - Code Name Base: **mx.tmsanchez.experimento**
  - Module Display Name: **LaboratorioExperimento**
  - Localizing Bundle: mx/tmsanchez/experimento/Bundle.properties
  - Generate OSGi Bundle (Desactivado)
  - Clic en **Finish**



Preparar un Bean Simple

Agregar un POJO para manejar la información del experimento

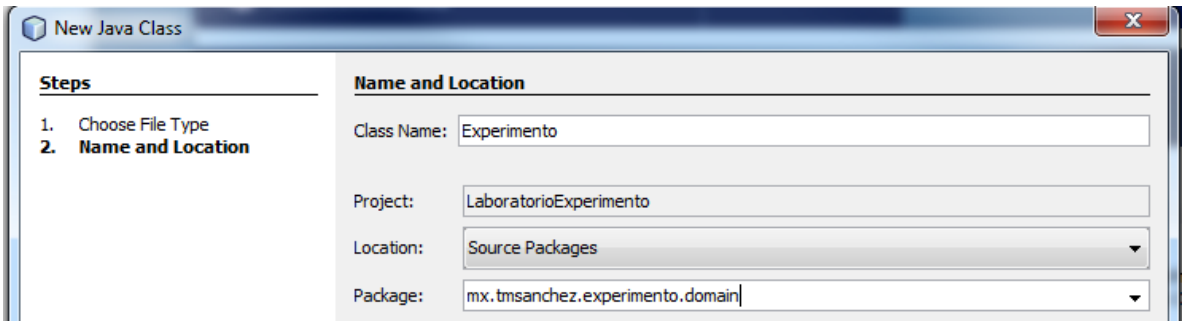
Clic derecho sobre el package **mx.tmsanchez.experimento** y seleccionar **New / Java Class…**



Completar la información de la clase

- Class Name: **Experimento**
- Package: **mx.tmsnchez.experimento.domain**

Dar clic en Finish

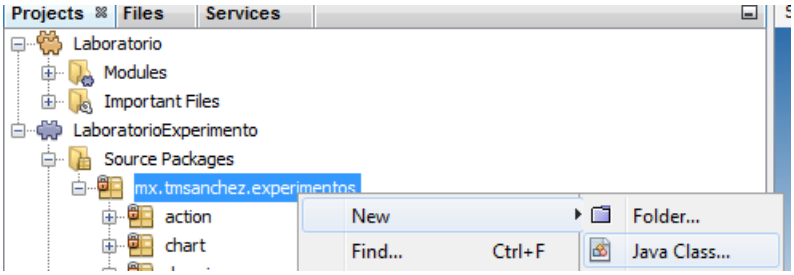


Completar la clase Experimento con el siguiente código

```java
package mx.tmsanchez.experimentos.domain;


public class Experimento {
    private Integer idExperimento;
    private String nombre;
    private String resumen;

    public Experimento(Integer idExperimento, String nombre, String resumen) {
        this.idExperimento = idExperimento;
        this.nombre = nombre;
        this.resumen = resumen;
    }

    public Integer getIdExperimento() {
        return idExperimento;
    }

    public void setIdExperimento(Integer idExperimento) {
        this.idExperimento = idExperimento;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getResumen() {
        return resumen;
    }

    public void setResumen(String resumen) {
        this.resumen = resumen;
    }



}
```

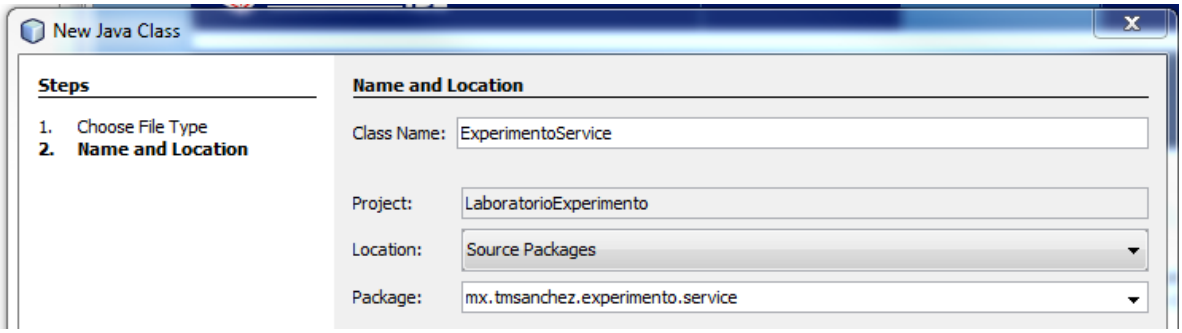
Agregar una clase para simular un servicio que proporciona la lista de experimentos

Clic derecho en el package **mx.tmsanchez.experimento** y seleccionar **New / Java Class**



Completar la información de la clase

- Class Name: **ExperimentoService**
- Package **mx.tmsanchez.experimento.service**



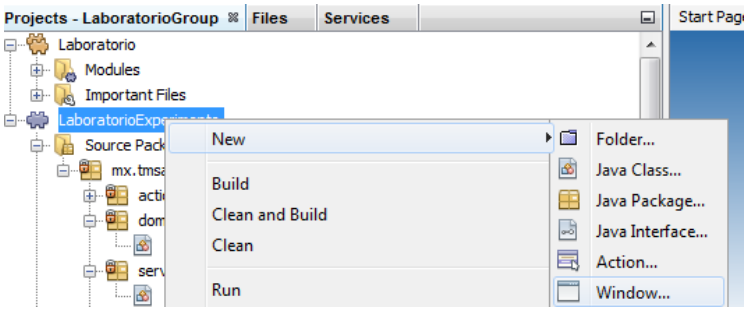
Completar la clase ExperimentoService con el siguiente código:

```
1   package mx.tmsanchez.experimentos.service;
2
3   import java.util.ArrayList;
4   import java.util.List;
5   import mx.tmsanchez.experimentos.domain.Experimento;
6
7   public class ExperimentoService {
8
9       public List<Experimento> getExperimentos() {
10          List<Experimento> lista = new ArrayList<Experimento>();
11          lista.add(new Experimento(1, "AH789", "Comportamento del cultivo en zona árida"));
12          lista.add(new Experimento(2, "HT788", "Verificar la tolerancia a la sequía del cultivo"));
13          lista.add(new Experimento(1, "T789", "Comprobar la respuesta al abono orgánico"));
14          lista.add(new Experimento(1, "TUIW", "Verificar si es posible ....."));
15          return lista;
16      }
17  }
18
```

Sistema de Ventanas

Agregar una nueva ventana en el proyecto

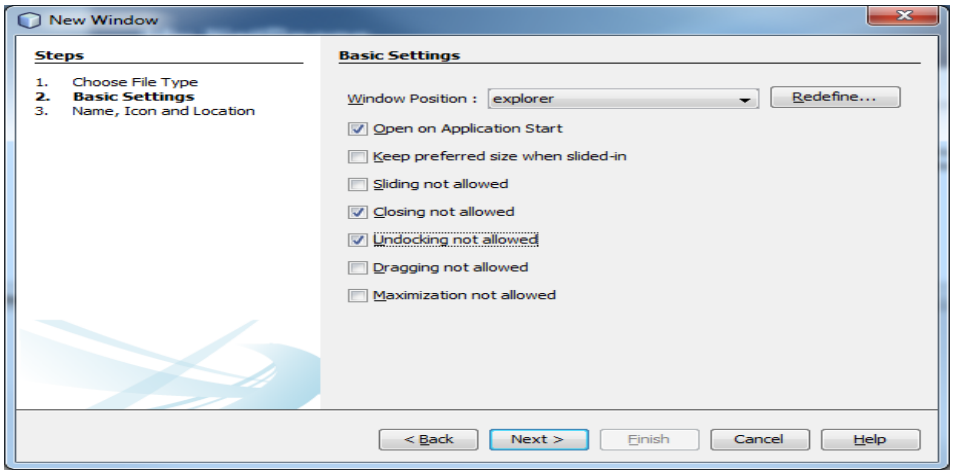
Clic derecho sobre el módulo y seleccionar **New / Window**



Completar la siguiente información

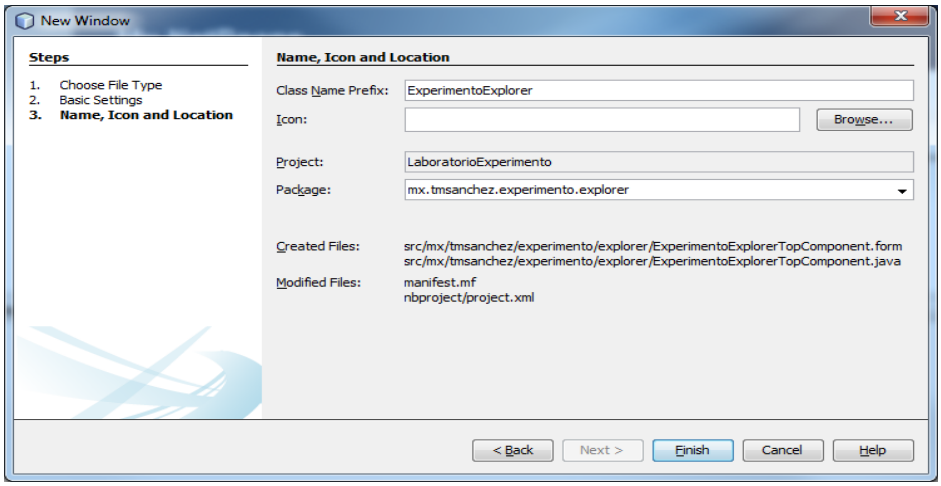
- Basic Settings
  - Window Position: **Explorer**
  - **Open on Application Start** (seleccionado)
  - **Closing not allowed** (seleccionado)
  - **Undocking not allowed**

    Dar clic en el botón **Next**



- Name, Icon, Location
  - ClassNamePrefix: ***ExperimentoExplorer***
  - Package: ***mx.tmsanchez.experimento.explorer***

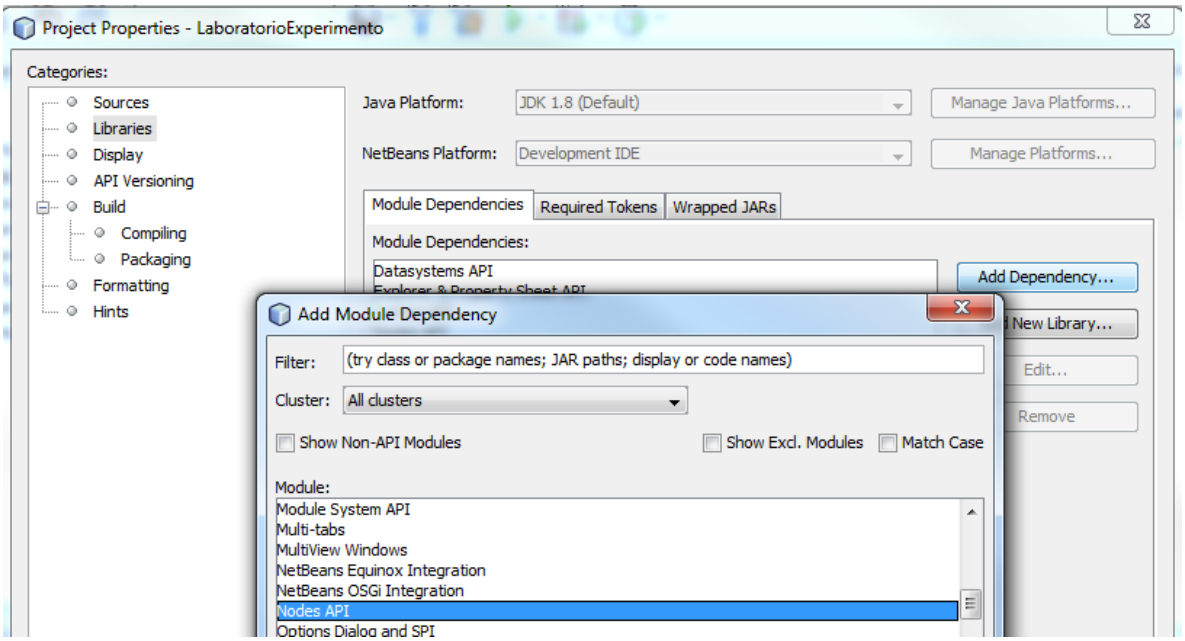
    Clic en el botón **Finish**



Uso del API Nodes

Agregar dependencia al API Nodes para poder seleccionar experimentos en el explorador

- Botón derecho / **Properties** y en ***Categories*** seleccionar / **Libraries**
- En la ventana *Project Properties* dar clic en el botón **Add Dependency**
    - En la pantalla *Add Module Depencency*
        - Seleccionar **Nodes Api**
        - Dar clic en el botón **OK**
- Al regresar al a ventana *Project properties* dar clic en el botón **OK**



Agregar una clase para representar cada elemento que se va a mostrar en el explorador

Clic derecho sobre el package **mx.tmsanchez.experimento.explorer** y seleccionar **New / Java Class** y nombrar a la clase como **ExperimentoNode**, completar la clase con el siguiente código:

```java
package mx.tmsanchez.experimento.explorer;

import java.beans.IntrospectionException;
import mx.tmsanchez.experimentos.domain.Experimento;
import org.openide.nodes.BeanNode;
import org.openide.nodes.Children;
import org.openide.util.lookup.Lookups;

/**
 *
 * @author tmsg
 */
public class ExperimentoNode extends BeanNode {

    public ExperimentoNode(Experimento experimento) throws IntrospectionException {
        super(experimento, Children.LEAF, Lookups.singleton(experimento));
        setDisplayName(experimento.getNombre());
        setShortDescription(experimento.getResumen());
    }

}
```

Agregar una clase para llenar la lista que se va a mostrar en el explorador

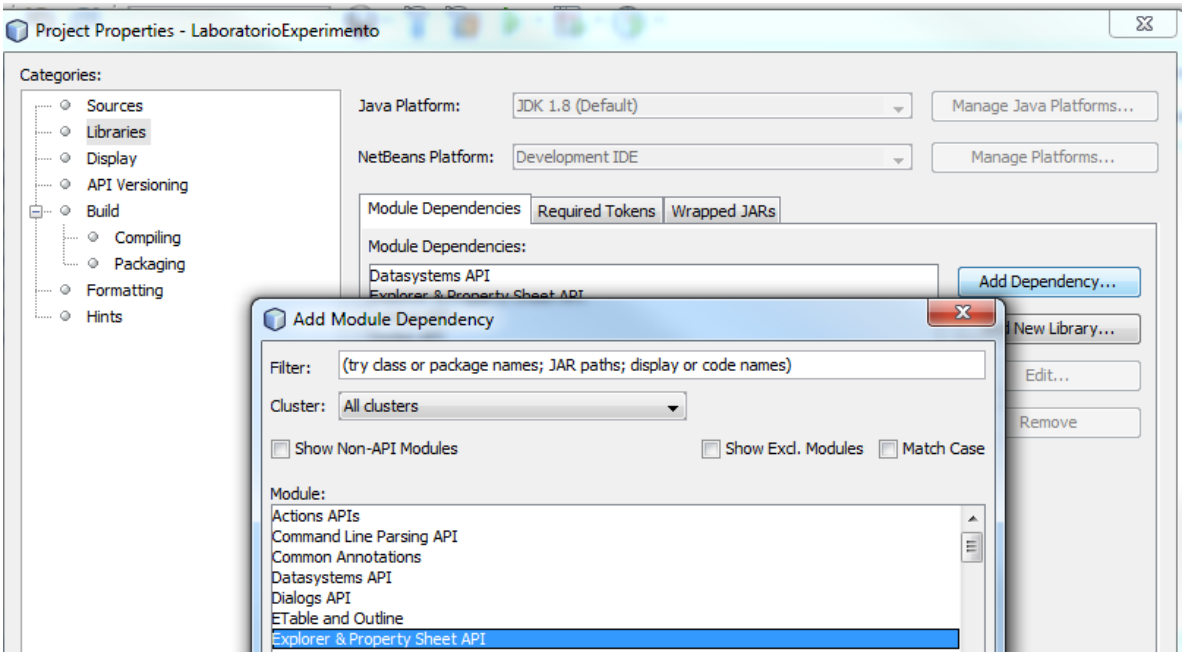
Clic derecho sobre el package **mx.tmsanchez.experimento.explorer** y seleccionar **New / Java Class** y nombrar a la clase como **ExperimentoChildFactory**, completar la clase con el siguiente código:

```java
package mx.tmsanchez.experimento.explorer;

import java.util.List;
import mx.tmsanchez.experimentos.domain.Experimento;
import mx.tmsanchez.experimentos.service.ExperimentoService;
import org.openide.nodes.ChildFactory;
import org.openide.nodes.Node;

/**
 *
 * @author tmsg
 */
public class ExperimentoChildFactory extends ChildFactory<Experimento> {

    @Override
    protected boolean createKeys(List<Experimento> lista) {
        lista.addAll(new ExperimentoService().getExperimentos());
        return true;
    }

    @Override
    protected Node createNodeForKey(Experimento key) {
        Node node = null;
        try {
            node = new mx.tmsanchez.experimentos.ExperimentoNode(key);
        } catch (Exception e) {

        }
        return node;
    }

}
```

Useo del API Explorer & Property Sheet

Agregar la dependencia al explorador

- Botón derecho / **Properties** y en ***Categories*** seleccionar / **Libraries**
- En la ventana *Project Properties* dar clic en el botón **Add Dependency**
    - En la pantalla *Add Module Depencency*
        - Seleccionar **Explorer & Property Sheet API**
        - Dar clic en el botón **OK**
- Al regresar al a ventana *Project properties* dar clic en el botón **OK**



Implementación de ExplorerManager.Provider

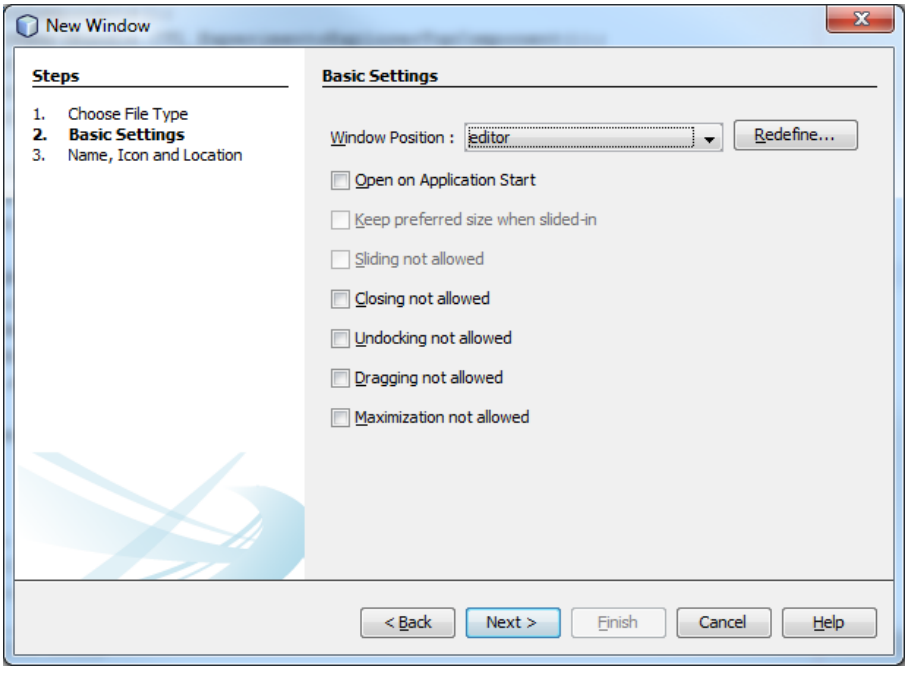
Modificar el código de la clase ExprimentoExplorerTopComponent dando clic en el botón **Source** y agregar el código encerrado en los recuadros:

```
Source  Design  History
37      "CTL_ExperimentoExplorerTopComponent=ExperimentoExplorer Window",
38      "HINT_ExperimentoExplorerTopComponent=This is a ExperimentoExplorer window"
39  })
40  public final class ExperimentoExplorerTopComponent extends TopComponent implements ExplorerManager.Provider {
41
43      private ExplorerManager em = new ExplorerManager();
44
44      @Override
46      public ExplorerManager getExplorerManager() {
47          return em;
47      }
48
49      public ExperimentoExplorerTopComponent() {
50          initComponents();
51          setName(Bundle.CTL_ExperimentoExplorerTopComponent());
52          setToolTipText(Bundle.HINT_ExperimentoExplorerTopComponent());
53          putClientProperty(TopComponent.PROP_CLOSING_DISABLED, Boolean.TRUE);
54          putClientProperty(TopComponent.PROP_UNDOCKING_DISABLED, Boolean.TRUE);
55
56          setLayout(new BorderLayout());
57          BeanTreeView btv = new BeanTreeView();
58          add(btv);
59
          ExperimentoChildFactory ecf = new ExperimentoChildFactory();
61          Children experimentoChildren = Children.create(ecf, true);
62          Node nodoRaiz = new AbstractNode(experimentoChildren);
63          nodoRaiz.setDisplayName("Experimentos");
64          em.setRootContext(nodoRaiz);
65
66          associateLookup(ExplorerUtils.createLookup(em, getActionMap()));
67      }
```

Ventana del Editor

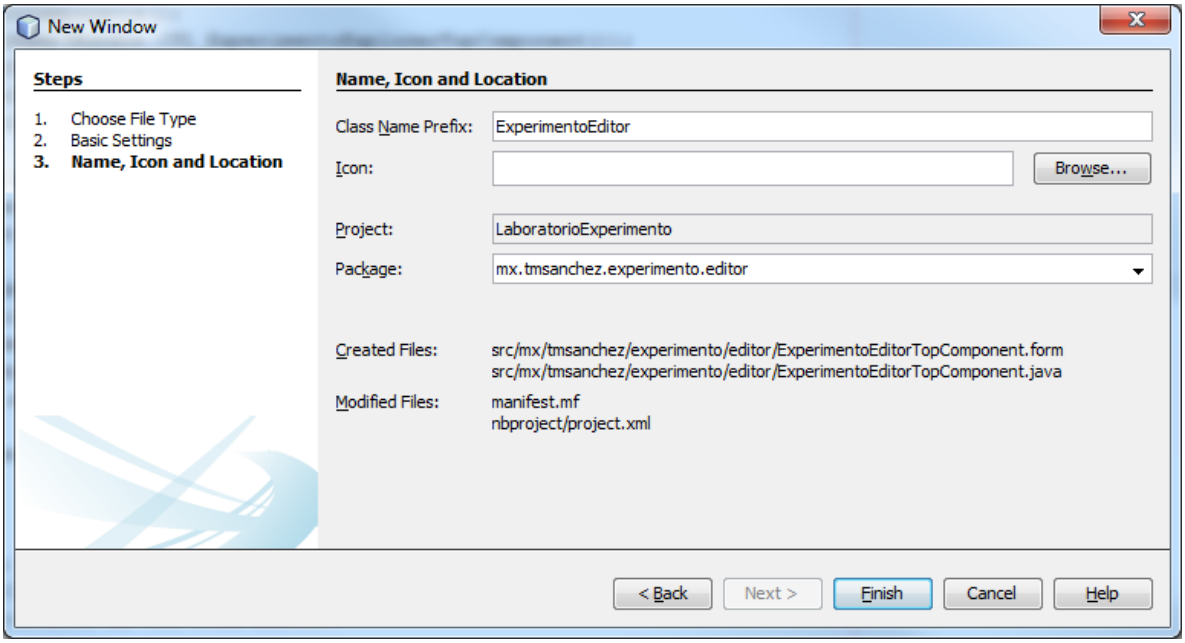
Agregar un editor para mostrar la información

- Botón Derecho sobre el módulo y seleccionar **New** / **Window**
  - En el paso *Basic Settings*
    - Window Position: *editor*
    - Dar clic en el botón **Next**



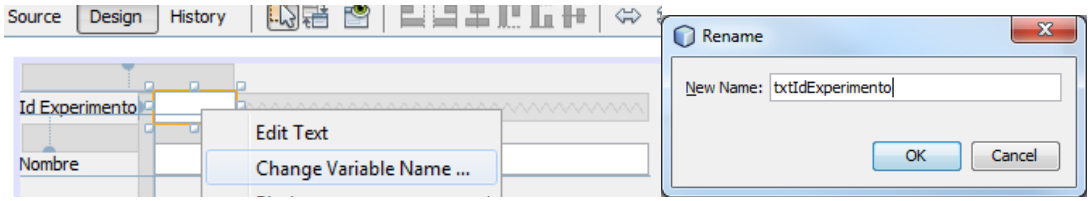
  - En el paso **Name, Icon and Location**
    - Class Name Prefix **ExperimentoEditor**

- Icon (dejar en blanco)
- Package: ***mx.tmsanchez.experimento.editor***
- Dar clic en el botón **Finish**

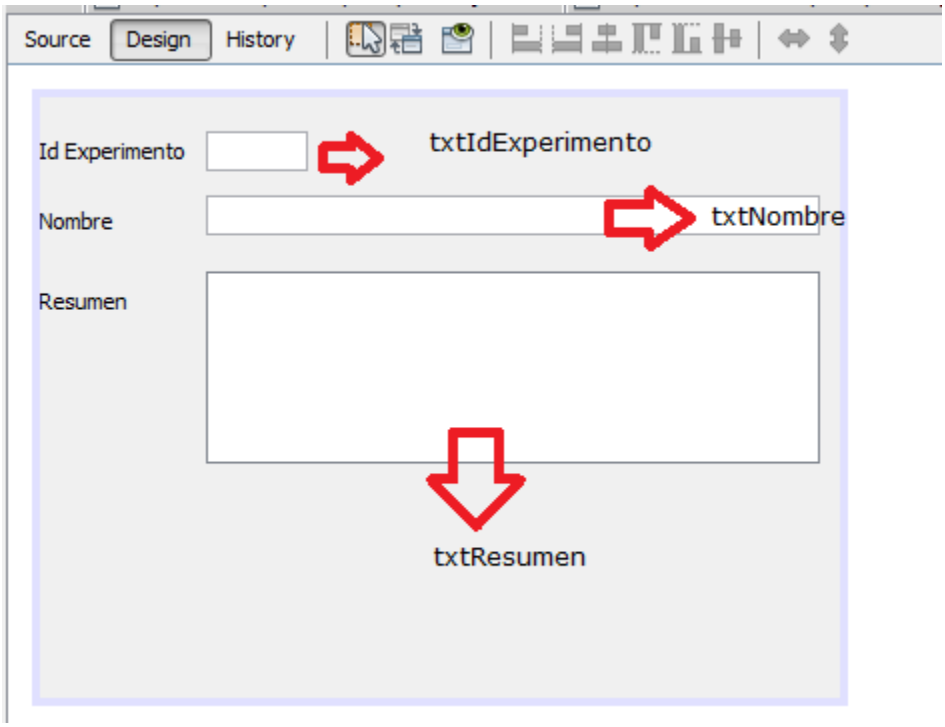


Agregar componentes para mostrar el experimento en un editor

Agregar tres **JLabel**, dos **JTextField** y un **JTextArea** de acuerdo a la siguiente imagen y cambiar el nombre del componente dando clic derecho y seleccionar



Nombrar a cada componente de acuerdo a la imagen:



Desactivar la persistencia de ExprimentoEditor para que al cerrar la aplicación la ventana no quede abierta, buscar la línea donde se encuentra **persistenceType** y dejarla a **TopComponen.PERSISTENCE_NEVER** como se muestra a continuación:

```
)
@TopComponent.Description(
        preferredID = "ExperimentoEditorTopComponent",
        iconBase = "mx/tmsanchez/experimentos/application_view_list.png",
        persistenceType = TopComponent.PERSISTENCE_NEVER
)
```

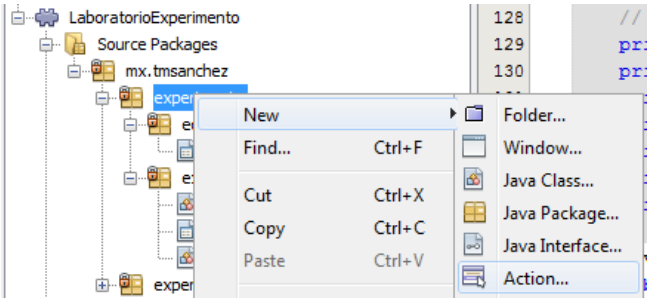
- Modificar el método **componentOpened()** con las líneas que se muestran en el recuadro
- Agregar el código que se muestra en la imagen de la línea 163 a la 171

```
137     @Override
        public void componentOpened() {
139         if (experimento != null) {
140             txtIdExperimento.setText(experimento.getIdExperimento().toString());
141             txtNombre.setText(experimento.getNombre());
142             txtResumen.setText(experimento.getResumen());
143         }
144     }
145
146     @Override
        public void componentClosed() {
148         // TODO add custom code on component closing
149     }
150
151     void writeProperties(java.util.Properties p) {
152         // better to version settings since initial version as advocated at
153         // http://wiki.apidesign.org/wiki/PropertyFiles
154         p.setProperty("version", "1.0");
155         // TODO store your settings
156     }
157
158     void readProperties(java.util.Properties p) {
159         String version = p.getProperty("version");
160         // TODO read your settings according to their version
161     }
162
163     private Experimento experimento;
164
165     public Experimento getExperimento() {
166         return experimento;
167     }
168
169     public void setExperimento(Experimento experimento) {
170         this.experimento = experimento;
171     }
172
173 }
174
```

Uso del API Actions

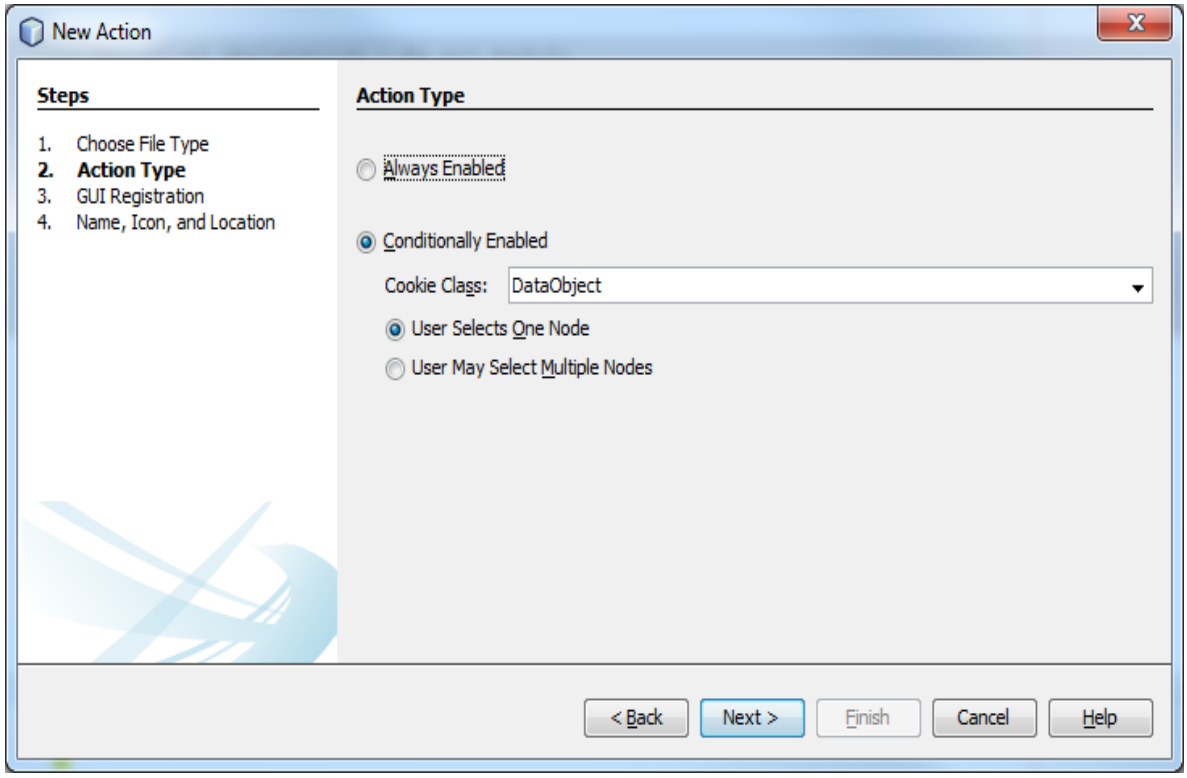
Agregar una nueva Acción que se active cuando un experimento esté seleccionado

- Botón derecho / Add New / Action



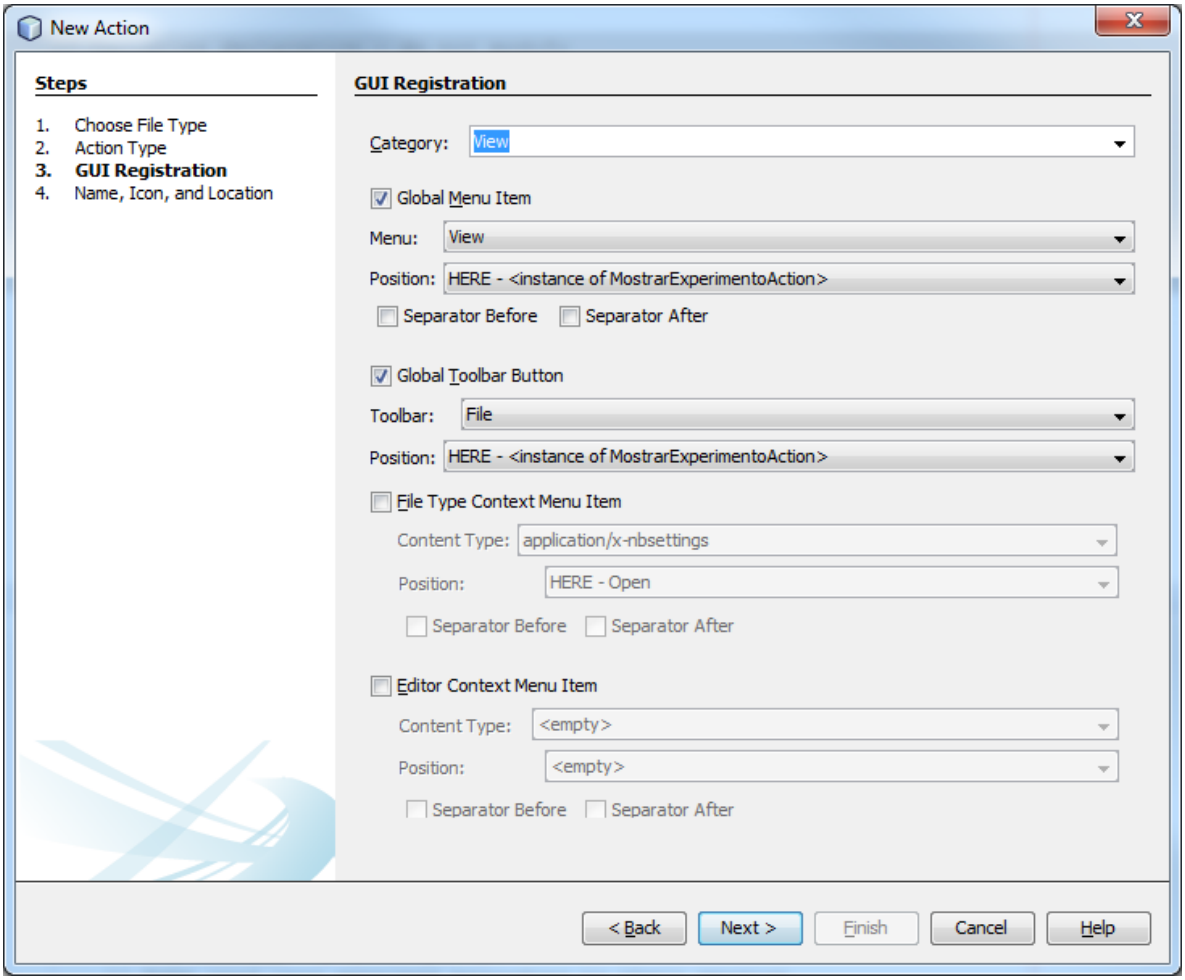
Completar la información en el Asistente

- Action Type
  - **Conditionally Enabled** (seleccionado)
    - Cookie Class: ***DataObject***
  - **Users Select One Node** (seleccionado)
  - Dar clic en el botón **Next**

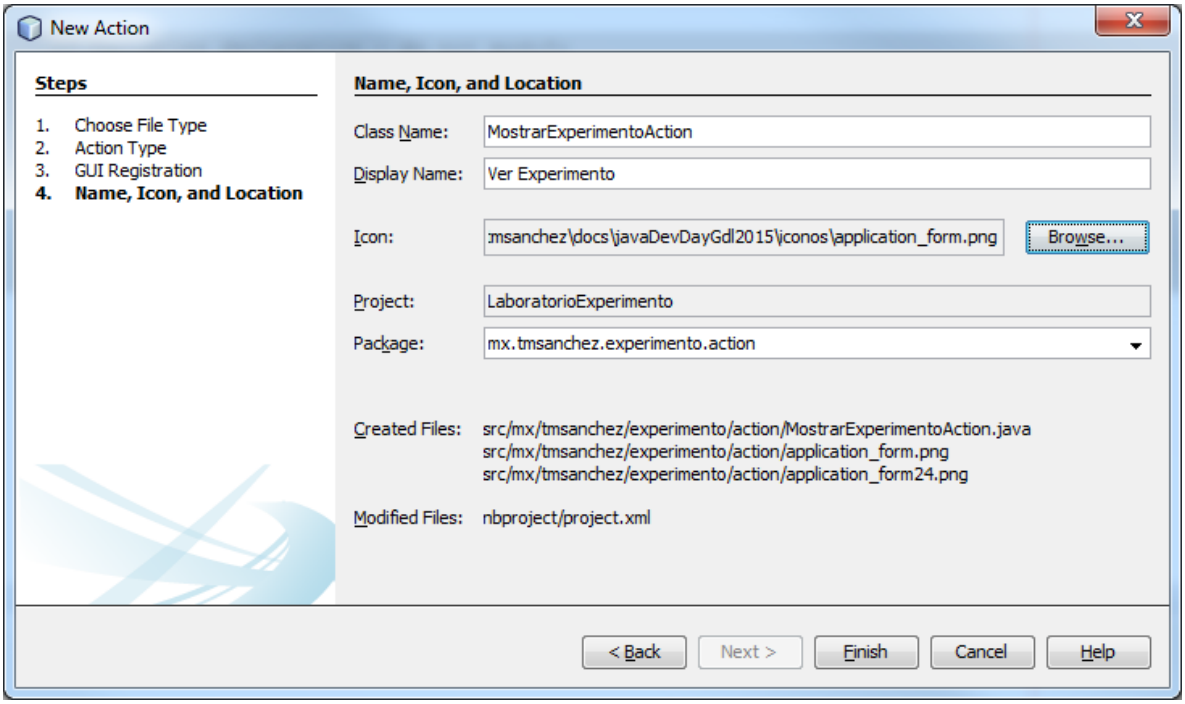


- Gui Registration

- Category: ***View***
- **Global Menu Item** (Seleccionado)
  - Menu: ***View***
- **Global Toolbar Button** (Seleccionado)
  - Toolbar: ***File***



- Name, Icon and Location
  - ClassName: ***MostrarExperimentoAction***
  - Display Name: ***Ver Experimento***
  - Icon: Seleccionar algún ícono (en realidad se requieren dos, uno de 16 pixeles y otro de 24, el de 16 es para la barra de menú y el de 24 para la barra de botones)
  - Package: ***mx.tmsanchez.experimento.action***
  - Clic en el botón Finish

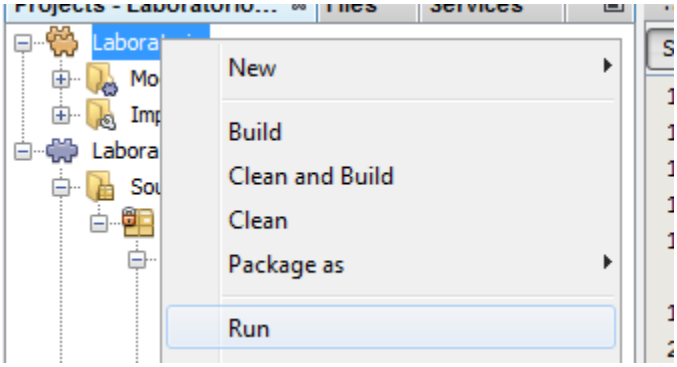


Completar el código de la clase MostrarExperimentoAction como se muestra en la imagen:

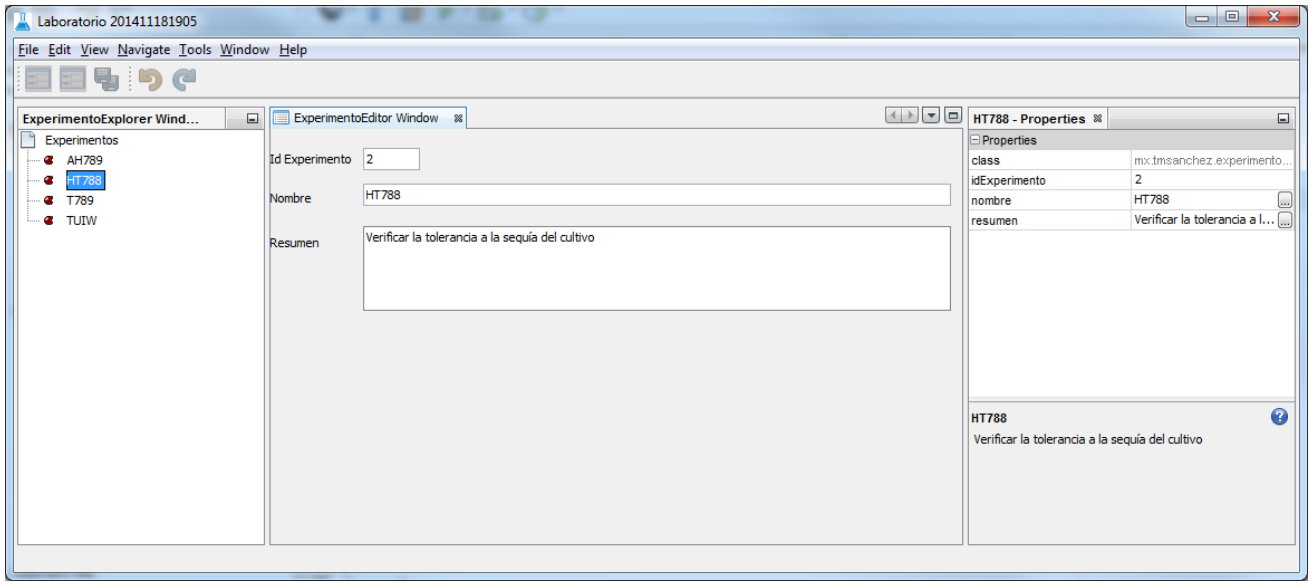
```
31  public final class MostrarExperimentoAction implements ActionListener {
32
33      private final Experimento context;
34
35      public MostrarExperimentoAction(Experimento context) {
36          this.context = context;
37      }
38
39      @Override
40      public void actionPerformed(ActionEvent ev) {
41          ExperimentoEditorTopComponent eetc = new ExperimentoEditorTopComponent();
42          eetc.setExperimento(context);
43          eetc.open();
44          eetc.requestActive();
45      }
46  }
```

Ejecutar la aplicación

Dar clic derecho sobre la aplicación “Laboratorio” y seleccionar la opción **Run**



La aplicación debe mostrarse como la siguiente imagen



- Para abrir el editor debe estar seleccionado algún experimento en el explorador y el botón se debe activar en la barra de herramientas
- Para mostrar la ventana *Properties* dar clic en el menú *Window / IDE Tools / Properties*