1.1.1 Aplicacións multixanela

As aplicacións desenvolvidas ata o momento foron sempre aplicacións cuxa acción desenrolábase nunha única xanela. Isto adoita ocorrer en aplicacións de pouca entidade, pero unha vez que unha aplicación comenza a ter certa complexidade o habitual é que o seu contido teña que desdobrarse en varios formularios. Cando o número de formularios da aplicación comenza a ser elevado á recomendable separalos en distintas xanelas co fin de que o manexo da aplicación sexa máis sinxelo. Unha interface de usuario demasiado cargada volvese lenta e complexa de empregar. Para desenvolver aplicacións multixanela java ofrécenos dúas solucións:

- Aplicacións multixanela Dialog
- Aplicacións multixanela MDI.

1.1.1.1 Aplicacións multixanela Dialog

Para desenvolver aplicacións multixanela de tipo Dialog empregamos dous tipos de obxectos:

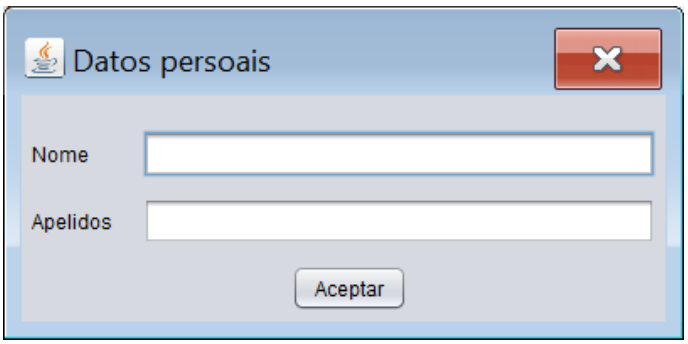
- O obxecto JFrame para representar a xanela principal (a primeira xanela que se crea).
- O obxecto JDialog para o resto de xanelas da aplicación.

Pero, ¿porqué non desenvolver as aplicacións multixanela empregando unicamente obxectos da clase JFrame?. Ben, tecnicamente pódese e ademais non presenta ningunha complicación engadida a facelo empregando JFrame e JDialogs, pero presenta algúns problemas:

Un JFrame representa a unha xanela "principal". Cando está aberta, na barra de tarefas do sistema operativo temos unha referencia gráfica (tipicamente unha icona) a esa xanela. Supoñamos que temos unha aplicación multixanela con catro JFrames. Na barra de tarefas do sistema operativo teremos catro referencias gráficas, unha por cada unha das xanelas que temos abertas na nosa aplicación. Pola contra, un JDialog representa a unha xanela "secundaria". Cando está aberta non aparece na barra de tarefas do sistema operativo ningunha referencia gráfica para poder xestionala. Polo tanto se desenvolvemos as nosas aplicacións multixanela empregando un JFrame para a xanela principal e obxectos JDialog para as xanelas secundarias, o aspecto das nosas aplicacións será o habitual, é dicir, unicamente haberá na barra de tarefas do sistemas operativo unha referencia gráfica cara á nosa aplicación multixanela.

Un JFrame, ao ser unha xanela principal, non permite ter unha xanela pai. O JDialog ao ser unha xanela secundaria si admite pai (o pai dunha xanela é o que a crea. Dise que a xanela aberta é filla de quen a creou. Os pais dun JDialog serán de tipo JFrame ou JDialog). Unha xanela filla graficamente sempre situarase por encima da súa xanela pai. A xanela filla nunca poderá quedar agochada detrás da xanela pai. Conceptualmente, unha xanela pai abre unha xanela filla para que o usuario realice algunha tarefa en ela. O non permitir que esa xanela quede agochada detrás do seu pai é un xeito de lembrar ao usuario que ten que realizar esa tarea.

A clase JDialog defínese a través da clase javax.swing.JDialog. Graficamente o seu aspecto é o seguinte:



Observa na imaxe anterior, apenas hai diferencias entre un JDialog e un JFrame. Ambas as dúas son xanelas nas cales podemos colocar os compoñentes que consideremos oportunos para desenvolver as nosas aplicacións. Visualmente a diferencia máis notable é que os JDialog non teñen botóns de redimensionamento (maximizar e minimizar).

Propiedades do contedor JDialog empregadas máis habitualmente e que se poden establecer a través do entorno de programación:

Propiedade	Función
DefaultCloseOperation	Mediante esta propiedade indícase cal será o comportamento da xanela ao pechala. Pode tomar 3 valores posibles:DISPOSE, HIDE o DO_NOTHING. Con DISPOSE péchase a xanela liberando os recursos consumidos pola mesma. Con HIDE a xanela unicamente será escondida. Con DO_NOTHING non se fará nada.
Modal	Mediante esta propiedade indícase se o comportamento da xanela será modal ou non.
Resizable	Indica se a xanela será ou non redimensionable.
Title	Título da xanela.
Type	Indica o tipo de xanela que imos crear. Pode tomar os valores NORMAL, UTILITY o POPUP. NORMAL indica unha xanela normal. Se empregamos os valores UTILITY ou POPUP, dependendo do sistema operativo sobre o que corra a nosa aplicación podería variar lixeiramente o aspecto da xanela.

Eventos do contedor JDialog empregados máis habitualmente e que pódense establecer a través do entorno de programación:

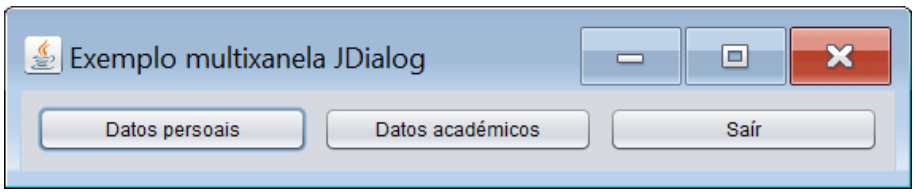
Evento	Lanzamento
WindowClosed	Este evento é disparado cando pechamos a xanela xa sexa ao premer sobre a aspa da xanela ou ao pechala a través de código.

Métodos do contedor JDialog empregados máis habitualmente:

Método	Función
public void dispose()	Mediante este método pódese pechar a xanela liberando os recursos consumidos pola mesma.
public void setVisible(boolean b)	Este método emprégase para facer visible ou invisible a xanela de diálogo.

Configuración dunha aplicación multixanela Dialog empregando NetBeans

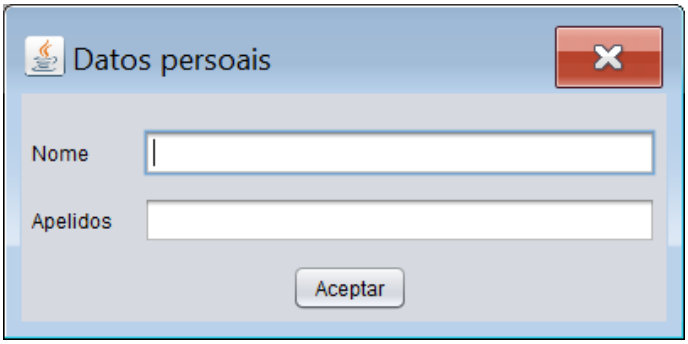
Imos desenvolver unha aplicación cuxa xanela principal (JFrame) será a seguinte:



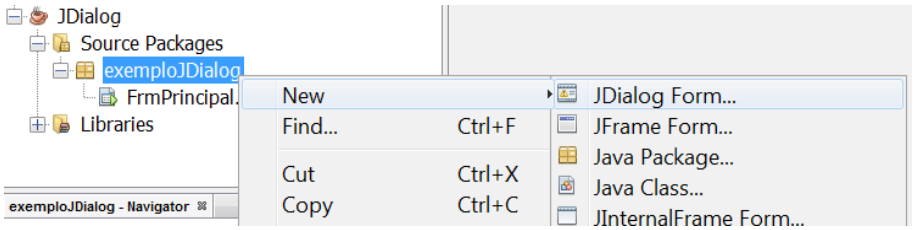
Ao premer sobre o botón Datos persoais abrirase un JDialog para introducir os datos persoais do usuario. Ao pulsar sobre o botón Datos académicos abrirase un JDialog para introducir os datos académicos do usuario. O botón Saír finalizará a execución da aplicación. A medida que avance a explicación imos ir complicando os requisitos do programa para desenvolver novas funcionalidades.

Sobre o deseño da xanela principal non hai nada que dicir. É un formulario con tres botóns.

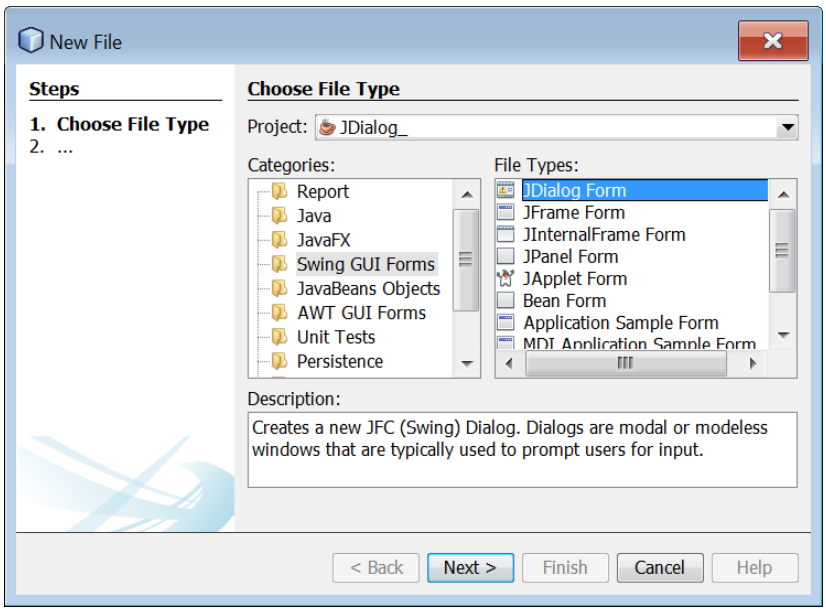
Cando pulsamos sobre o botón Datos persoais débese abrir un JDialog que ten o seguinte aspecto:



Para crear este JDialog debemos facer o seguinte: o primeiro é crear unha clase persoalizada que herde de JDialog para definir o noso contedor. Para elo, prememos co botón dereito do rato sobre o paquete no cal queremos crear a nosa clase persoalizada. Sobre o menú despregado eliximos New e sobre o novo menú despregado eliximos JDialog Form:

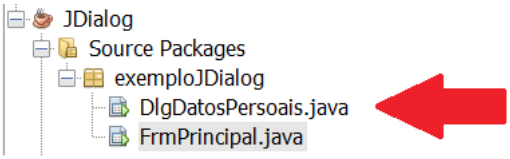


É posible que a primeira vez que creemos un JDialog, ao pulsar na opción New non apareza a opción JDialog Form. Nese caso, no mesmo menú deberemos elixir unha opción etiquetada como Other. Ao seleccionala abrirásenos a seguinte pantalla:

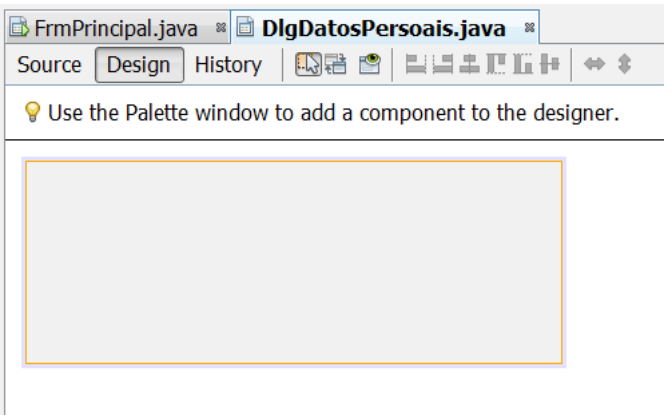


Esta pantalla emprégase para crear clases baseándose nos diferentes modelos que ofrece o entorno de programación. No caso que estamos desenvolvendo elixiremos dentro de Categories a opción Swing GUI Forms e en File Types JDialog Form.

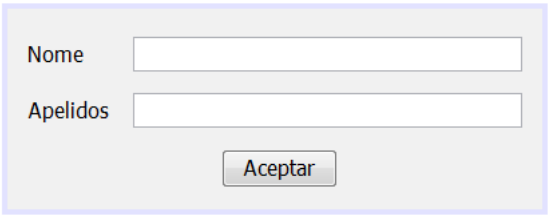
Independentemente de como cheguemos a ela, ao seleccionar a opción JDialog Form saltará un asistente similar ao de creación dun novo formulario para indicar o nome que lle imos dar á nosa nova clase de tipo JDialog (chamarémola DlgDatosPersoais) e opcionalmente para indicar en que paquete queremos almacenala. Unha vez que completamos o asistente, na nosa árbore de proxecto imos ter unha nova clase que representara ao JDialog que acabamos de crear:



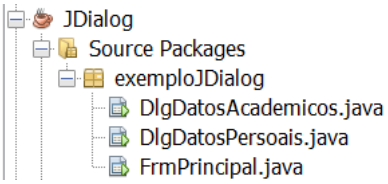
Respecto ao deseño gráfico do JDialog, lévase a cabo do mesmo xeito que para un formulario de tipo JFrame. Se seleccionamos a clase que acabamos de crear (DlgDatosPersoais), na xanela de deseño do contedor veremos o seguinte:



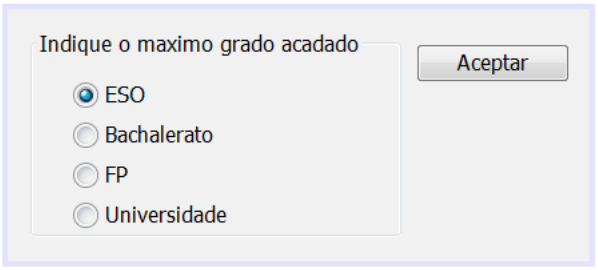
Como pódese observar é un contedor baleiro. Sobre el estableceremos as súas propiedades, inseriremos os compoñentes, estableceremos os layouts necesarios e escribiremos o código necesario para os diferentes eventos que queiramos xestionar sobre os compoñentes do JDialog. Resumindo, unha vez creado o noso JDialog persoalizado, xestionarémolo tal e como fixemos ata agora cos nosos formularios. Polo tanto, o deseño visual do dialogo DlgDatosPersoais quedaría configurado do seguinte xeito:



Una vez desenvolvido o diálogo que se amosará cando premamos o botón Datos persoais, facemos o mesmo para o dialogo que se amosará ao premer o botón Datos académicos. O resultado será o seguinte:



Como pódese observar na imaxe anterior, a clase que acabamos de crear como modelo para os obxectos que se amosaran cando premamos sobre o botón Datos académicos chámase DlgDatosAcademicos. O aspecto do seu deseño gráfico é o seguinte:



Non nos imos centrar no que facen os diálogos que acabamos de crear, senón unicamente no código relacionado coa xestión multixanela da aplicación.

De seguido imos ver que é o que temos que facer para que se amosen os diálogos ao premer sobre os botóns correspondentes. Para amosar o diálogo DlgDatosPersoais implementamos o actionPerformed do botón Datos persoais, sendo o código resultante o seguinte:

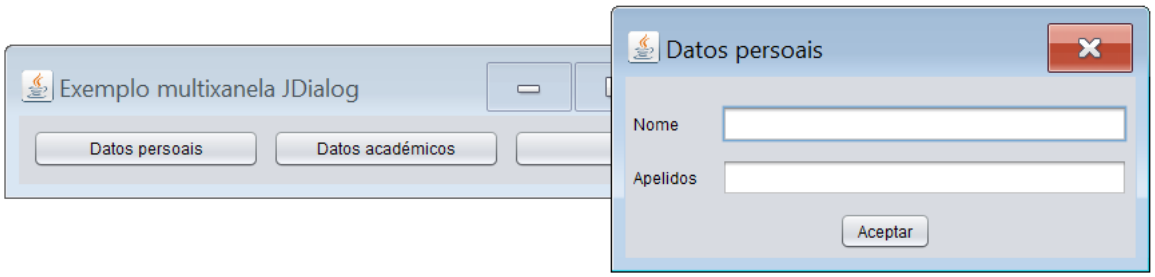
```
    private void btnDatosPersoaisActionPerformed(java.awt.event.ActionEvent evt) {
        DlgDatosPersoais dlgDatosPersoais=new DlgDatosPersoais(this, false);
        dlgDatosPersoais.setVisible(true);
    }
```

Na primeira liña o que facemos é instanciar un obxecto da clase DlgDatosPersoais empregando o seu construtor. O construtor ten dous parámetros. O primeiro parámetro emprégase para indicar quen é a xanela pai da que estamos creando (neste caso é o propio chamador, é dicir, this). O segundo parámetro indica se a xanela aberta será amosada en modo modal (true) ou non modal (false). Unha xanela aberta en modo modal bloquea ao resto da aplica-

ción. Mentres unha xanela está aberta en modo modal non se pode acceder ao resto de xanelas que compoñen a aplicación. Cando sexa pechada será posible acceder a calquera outra xanela das que compoñen a aplicación. Sóense empregar as xanelas en modo modal cando queremos chamar a atención do usuario por algunha razón determinada (datos requiridos, mensaxes de erro, …). Fixémonos que o obxecto creado (que será unha xanela de tipo diálogo non modal e filla do formulario principal) será xestionada a través da variable local dlgDatosPersoais. Neste momento o diálogo está creado en memoria (fíxose o new) pero non é visible.

Na segunda liña facemos visible ao diálogo que acabamos de crear en memoria. Empregando a súa referencia, aplicamos sobre ela o método setVisible(true) que o que fai é facer visible ao contedor. Se lle pasaramos o valor false o que fariamos sería facer invisible ao contedor.

Se executamos a aplicación e prememos sobre o botón Datos persoais este será o resultado:

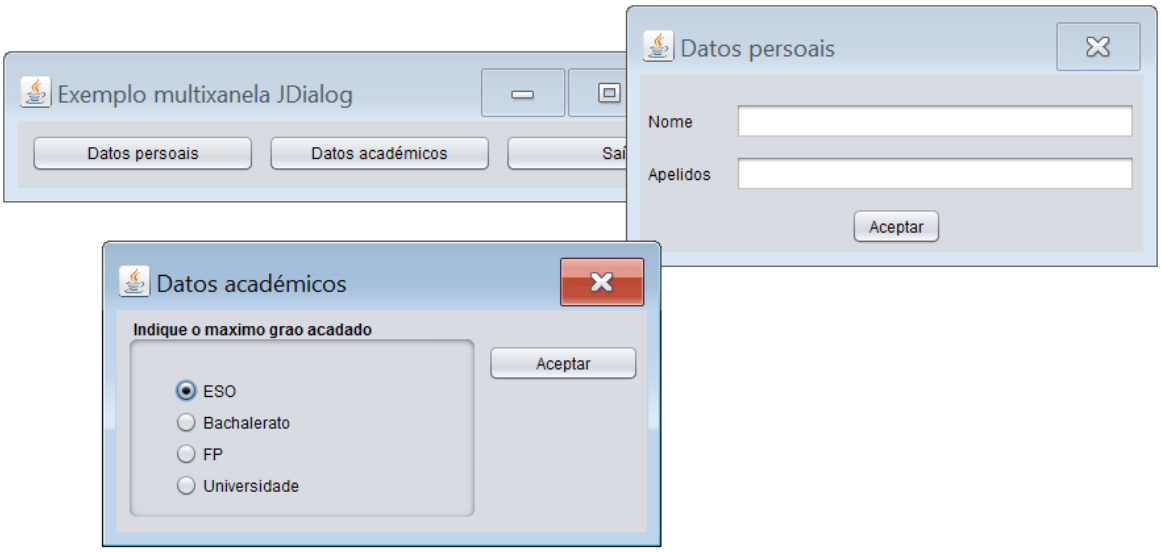


Tal e como dixemos con anterioridade, no caso de superpoñer a xanela filla sobre a pai, esta última sempre estará por debaixo da filla (aínda que a seleccionemos explicitamente).

Para a resolución do botón Datos académicos escribimos un código con una funcionalidade similar:

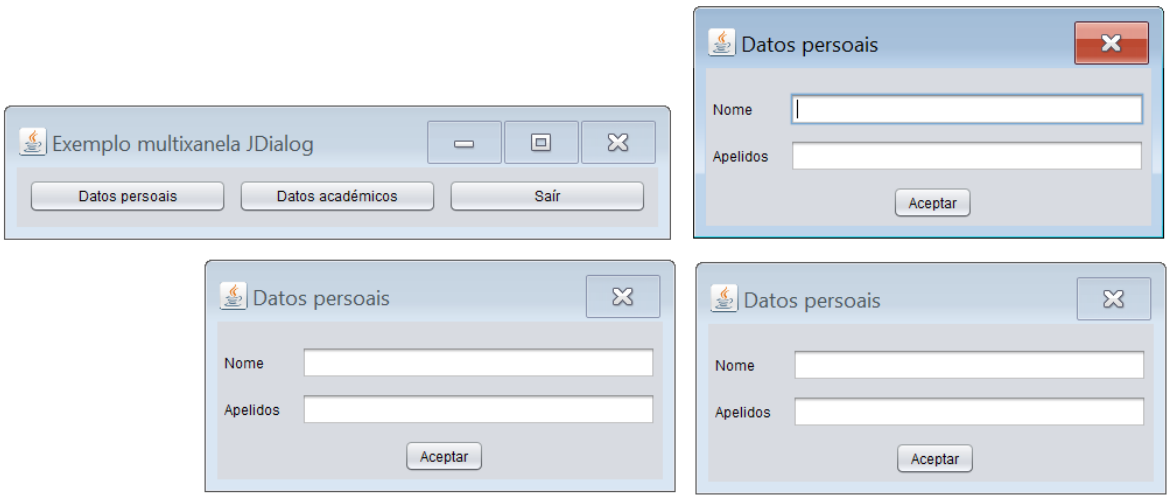
```
    private void btnDatosAcademicosActionPerformed(java.awt.event.ActionEvent evt) {
        DlgDatosAcademicos dlgDatosAcademicos=new DlgDatosAcademicos(this, false);
        dlgDatosAcademicos.setVisible(true);
    }
```

sendo o resultado de premer ambos botóns o seguinte:



Como podemos ver a xestión básica dunha aplicación multixanela en java e bastante sinxela.

Pero, co código actual, ¿que ocorre se prememos un botón varias veces?, p.e., imaxinemos que prememos tres veces o botón de Datos persoais. Ocorrerá o seguinte:



Como era de esperar, cada vez que prememos o botón ábrese un novo diálogo. Se os requirimentos do noso programa permítennos facelo entón non temos ningún problema, pero supoñamos que os requirimentos do noso programa dinnos que só podemos ter aberta á vez unha xanela de tipo dlgDatosPersoais e como moito dous de tipo DlgDatosAcademicos. Neste caso necesitaremos algún sistema a modo de xestor de xanelas para coñecer en todo momento que xanelas temos abertas e poder controlar se podemos abrir ou non podemos abrir máis. Solucións para resolver este problema hai moitas. Unha relativamente sinxela é o emprego dunha clase que almacene o estado das distintas xanelas que temos que controlar. Partindo do suposto que acabamos de propoñer (só podemos ter aberta á vez unha xanela de tipo dlgDatosPersoais e como moito dous de tipo DlgDatosAcademicos) imos desenvolver esta solución. Para elo creamos no noso proxecto unha nova clase que se chamará XestorDeXanelas. O seu código será o seguinte:

```
package exemploJDialog;

public class XestorDeXanelas {
    static private int numXanelasDatosPersoais=0;
    static private int numXanelasDatosAcademicos=0;

    public static void cerrarXanelasDatosPersoais()
    {
       numXanelasDatosPersoais--;
    }
    public static boolean abrirXanelasDatosPersoais()
    {
        if(numXanelasDatosPersoais<1)
        {
            numXanelasDatosPersoais++;
            return true;
        }
        else
        {
            return false;
        }
    }

    public static void cerrarXanelasDatosAcademicos()
    {
       numXanelasDatosAcademicos--;
    }

    public static boolean abrirXanelasDatosAcademicos()
    {
        if(numXanelasDatosAcademicos<2)
        {
            numXanelasDatosAcademicos++;
            return true;
        }
        else
        {
            return false;
        }
```

```
    }
}
```

Esta clase ten dous atributos chamados numXanelasDatosPersoais e numXanelasDatosAcademicos, ambas de tipo enteiro que empregamos para levar a conta de cantos diálogos de datos persoais e de datos académicos temos abertos en cada momento. Para o atributo numXanelasDatosPersoais hai un método abrirXanelaDatosPersoais que é chamado cada vez que prememos sobre o botón Datos persoais. Este método devolve true se é posible abrir un diálogo de tipo DlgDatosPersoais (se o contador numXanelasDatosPersoais é menor de 1) e ademais incrementa o número de xanelas abertas nunha unidade. No caso contrario, se xa hai algunha xanela de tipo DlgDatosPersoais aberta devolverá false. Para poder empregar este método debemos modificar o código do botón Datos persoais do seguinte xeito:

```
    private void btnDatosPersoaisActionPerformed(java.awt.event.ActionEvent evt) {
        if(XestorDeXanelas.abrirXanelasDatosPersoais())
        {
            DlgDatosPersoais dlgDatosPersoais=new DlgDatosPersoais(this, false);
            dlgDatosPersoais.setVisible(true);
        }
        else
        {
            JOptionPane.showMessageDialog(this, "Non é posible abrir máis xanelas deste tipo");
            return;
        }
    }
```

O código de creación do diálogo de clase DlgDatosPersoais é o mesmo de antes, pero antes de crear a xanela de diálogo compróbase se é posible creala (que non hai xa unha aberta) chamando ao método abrirXanelaDatosPersoais e só é creada se este método llo permite. Se non llo permite, emítese unha mensaxe de erro ao usuario da aplicación. É importante fixarse en que este método, ao igual que tódolos métodos da clase XestorDeXanelas é un método estático, de xeito que as chamadas son máis sinxelas ao non ter que instanciar ningún obxecto de tipo XestorDeXanelas (outra opción para facilitar e unificar o código sería introducir a xestión de xanelas no propio código da xanela que abre os diálogos, é dicir, no formulario pai).

Para poder controlar totalmente o número de xanelas abertas de clase DlgDatosPersoais, ao igual que incrementamos o seu contador cada vez que abrimos unha xanela, tamén deberemos decrementar o seu contador cada vez que pechemos unha xanela. Cando péchase un dialogo é disparado o seu evento WindowClosed. Polo tanto, este será o lugar axeitado para informar ao xestor de xanelas de que acabamos de pechar unha xanela de tipo DlgDatosPersoais. Para elo escribimos as seguintes liñas de código:

```
    private void formWindowClosed(java.awt.event.WindowEvent evt) {
        XestorDeXanelas.cerrarXanelasDatosPersoais();
    }
```

O que facemos é simplemente chamar ao método cerrarXanelasDatosPersoais, o cal decrementa o número de diálogos de tipo DlgDatosPersoais abertos nunha unidade.

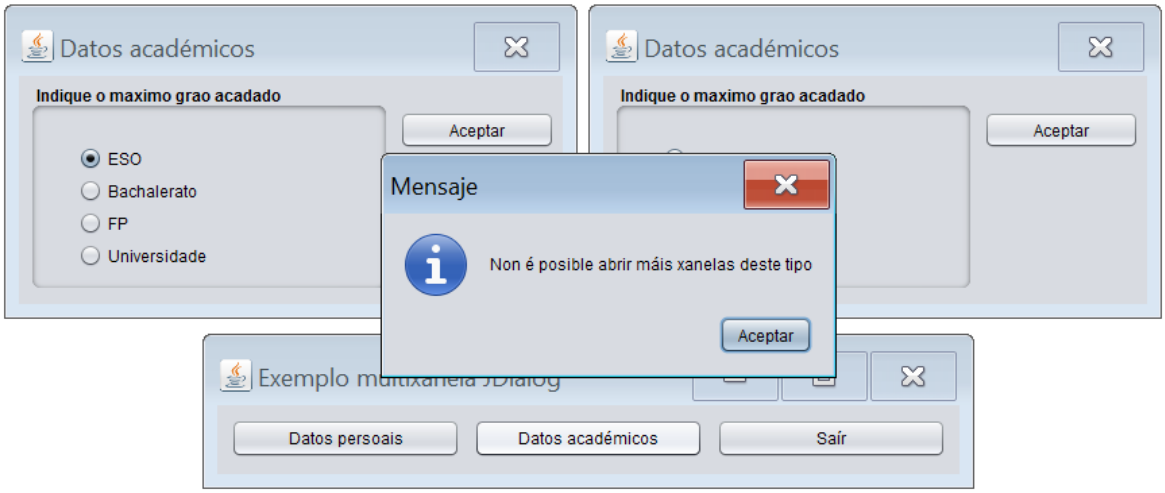
De igual xeito que acabamos de facer co botón Datos persoais, modificamos o botón Datos académicos para xestionar a creación dun novo diálogo de clase DlgDatosAcademicos:

```
    private void btnDatosAcademicosActionPerformed(java.awt.event.ActionEvent evt) {
        if(XestorDeXanelas.abrirXanelasDatosAcademicos())
        {
            DlgDatosAcademicos dlgDatosAcademicos=new DlgDatosAcademicos(this, false);
            dlgDatosAcademicos.setVisible(true);
        }
        else
        {
            JOptionPane.showMessageDialog(this, "Non é posible abrir máis xanelas deste tipo");
            return;
        }
    }
```

Por outra parte deberemos implementar o evento WindowClosed do diálogo DlgDatosAcademicos para decrementar o número de diálogos abertos cando un deles é pechado:

```
    private void formWindowClosed(java.awt.event.WindowEvent evt) {
        // TODO add your handling code here:
        XestorDeXanelas.cerrarXanelasDatosAcademicos();
    }
```

Se agora temos abertos dous diálogos de tipo DlgDatosAcademicos e intentamos abrir outro máis ocorre o seguinte:



O xestor de xanelas leva a conta do número de xanelas abertas de cada tipo e impídenos superar o límite máximo para cada uno dos tipos de xanelas implicados na nosa aplicación.

Esta é unha das utilidades que nos proporciona un xestor de xanelas. Outra utilidade que lle poderíamos dar ao xestor de xanelas sería a de ter unha referencia a cada unha das xanelas fillas dunha xanela pai. Unha vez que temos unha referencia desde unha xanela á súas fillas, a xanela pai poderá interactuar con todas elas e facer emprego dos métodos dispoñibles nas xanelas fillas. Imos desenvolver un exemplo sinxelo para practicar esta ultima utilidade que acabamos de comentar. Imos modificar a xanela pai para que o seu aspecto sexa o seguinte:

Como se podese ver hai un novo botón chamado Cascada. A utilidade deste botón é poñer en cascada as xanelas de tipo DlgDatosPersoais abertas mediante o botón Datos Persoais. Tal e como temos configurado agora mesmo o código da aplicación, cada vez que abrimos un diálogo, a variable que empregamos para crealo e unha variable local que se perde:

```
    private void btnDatosPersoaisActionPerformed(java.awt.event.ActionEvent evt) {
        if(XestorDeXanelas.abrirXanelasDatosPersoais())
        {
            DlgDatosPersoais dlgDatosPersoais=new DlgDatosPersoais(this, false);
            dlgDatosPersoais.setVisible(true);
        }
        else
        {
            JOptionPane.showMessageDialog(this, "Non é posible abrir máis xanelas deste tipo");
            return;
        }
    }
```

A variable dlgDatosPersoais é a referencia ao novo diálogo creado, pero como é unha variable local, perdese ao terminar de executarse o método e polo tanto o noso código fonte xa non pode interactuar de novo coa xanela que acabamos de crear. Unha solución é crear no noso xestor de xanelas un almacén no cal gardaremos as referencias das xanelas que imos creando. Para o caso que estamos probando engadimos o seguinte atributo:

```
static private Vector xanelasDatosPersoais=new Vector();
```

e os seguintes métodos:

```
public static void engadirXanelaDatosPersoais(DlgDatosPersoais xanela)
{
    xanelasDatosPersoais.add(xanela);
}


public static void eliminarXanelaDatosPersoais(DlgDatosPersoais xanela)
{
    for(int i=0;i<xanelasDatosPersoais.size();i++)
    {
        if(xanelasDatosPersoais.elementAt(i)==xanela)
        {
            xanelasDatosPersoais.removeElementAt(i);
            break;
        }
    }
    System.out.println(xanelasDatosPersoais.size());
}

public static Vector recuperarXanelasDatosPersoais()
{
    return xanelasDatosPersoais;
}
```

- O atributo xanelasDatosPersoais é un Vector no cal almacenaremos as referencias a tódolos novos diálogos de clase dlgDatosPersoais que imos engadindo á nosa aplicación.
- O método engadirXanelaDatosPersoais emprégase para insertar no Vector xanelasDatosPersoais unha referencia a un diálogo de clase dlgDatosPersoais. Será invocado cada vez que engadamos un diálogo de clase dlgDatosPersoais á nosa aplicación.
- O método eliminarXanelaDatosPersoais emprégase para eliminar do Vector xanelasDatosPersoais unha referencia a un diálogo de clase dlgDatosPersoais. Será invocado cada vez que eliminemos un diálogo de clase dlgDatosPersoais da nosa aplicación.
- O método recuperarXanelasDatosPersoais emprégase para devolver o contido do Vector xanelasDatosPersoais. Empregando este método pódense acadar tódalas referencias dos diálogos de clase dlgDatosPersoais da nosa aplicación.

Con estes tres elementos xestionamos tódalas referencias de calquera elemento de clase dlgDatosPersoais que teñamos na nosa aplicación.

Ademais imos modificar o método abrirXanelasDatosPersoais para que nos permita abrir ata cinco xanelas de tipo DlgDatosPersoais:

```
public static boolean abrirXanelasDatosPersoais()
{
    if(numXanelasDatosPersoais<5)
    {
        numXanelasDatosPersoais++;
        return true;
    }
    else
    {
        return false;
    }
}
```

Agora deberemos modificar o noso código de xeito que cada vez que engadamos un diálogo de tipo dlgDatosPersoais, sexa engadida a súa referencia tamén ao Vector xanelasDatosPersoais do xestor de xanelas, é dicir, chamaremos ao método engadirXanelaDatosPersonais pasándolle a xanela de diálogo que acabamos de crear:

```
private void btnDatosPersoaisActionPerformed(java.awt.event.ActionEvent evt) {
    if(XestorDeXanelas.abrirXanelasDatosPersoais())
    {
        DlgDatosPersoais dlgDatosPersoais=new DlgDatosPersoais(this, false);
        XestorDeXanelas.engadirXanelaDatosPersoais(dlgDatosPersoais);
        dlgDatosPersoais.setVisible(true);
    }
    else
    {
        JOptionPane.showMessageDialog(this, "Non é posible abrir máis xanelas deste tipo");
        return;
    }

}
```

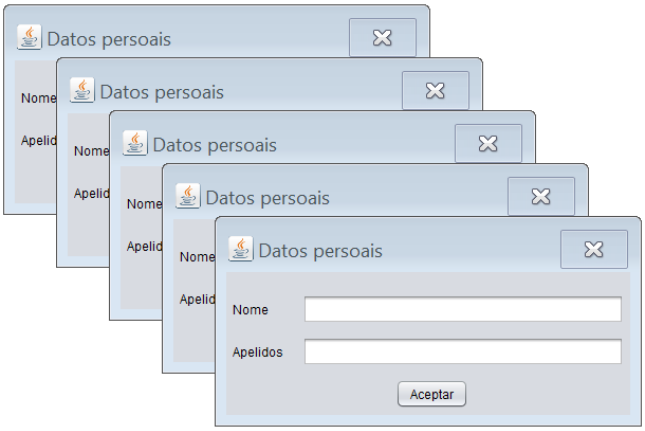
Tamén deberemos modificar o noso código de xeito que cada vez que eliminemos un diálogo de tipo dlgDatosPersoais sexa eliminada tamén a súa referencia do Vector xanelasDatosPersoais do xestor de xanelas, é dicir, chamaremos ao método eliminarXanelaDatosPersonais pasándolle a xanela de diálogo que queremos eliminar do vector:

```
private void formWindowClosed(java.awt.event.WindowEvent evt) {
    XestorDeXanelas.eliminarXanelaDatosPersoais(this);
    XestorDeXanelas.cerrarXanelasDatosPersoais();
}
```

Unha vez que temos tódalas pezas, únicamente resta por crear o método que sitúa en cascada tódalas xanelas de tipo dlgDatosPersoais. Para elo, no formulario pai creamos o seguinte método:

```
public void xanelasDatosPersoaisEnCascada()
{
    Vector xanelasDatosPersoais=XestorDeXanelas.recuperarXanelasDatosPersoais();
    int posX=10, posY=10, incremento=50;
    for(int i=0;i<xanelasDatosPersoais.size();i++)
    {
        DlgDatosPersoais xanela=(DlgDatosPersoais)xanelasDatosPersoais.elementAt(i);
        xanela.setLocation(posX, posY);
        posX+=incremento;
        posY+=incremento;
    }
}
```

Este método será chamado cando fagamos clic sobre o botón Cascada. Este método o primeiro que fai é recuperar do vector xanelasDatosPersoais do xestor de xanelas as referencias a tódalas xanelas de clase dlgDatosPersoais. Posteriormente recupera un a un os elementos dese vector, é dicir, recupera un a un os diálogos de clase dlgDatosPersoais e a cada un deles aplícalle o seu método setLocation que sirve para fixar a posición dun contedor dando a súa posición superior esquerda. Para cada uno deles recalcúlanse esas posicións de xeito que a disposición final dos diálogos sexa unha disposición en cascada. A continuación pódese ver a execución da aplicación sobre cinco diálogos de clase dlgDatosPersoais:



Se unha xanela pai ten referenciadas ás súas xanelas fillas, poderá realizar sobre elas calquera acción que estas permitan a través dos seus métodos públicos. O exemplo que acabamos de ver é unha sinxela demostración disto.

Acabamos de ver como facer que unha xanela pai teña referencias a tódalas súas fillas a través do emprego dun xestor de xanelas. Agora imos ver o caso contrario, é dicir, como facer que unha xanela filla teña unha referencia á súa xanela pai. Unha vez que unha xanela filla ten unha referencia á súa xanela pai, poderá invocar calquera método público da mesma. Para ver este concepto imos modificar a aplicación para que faga o seguinte: cando enchemos os datos dun formulario de clase dlgDatosPersoais no caso de que cometamos algún erro (non indiquemos o nome ou os apelidos) amosarase un erro ao usuario, pero o encargado de amosar o erro non vai ser o dlgDatosPersoais senón que será o formulario pai. Polo tanto, o dialogo dlgDatosPersoais terá que ter algunha referencia á xanela pai para acceder

ao método desta que amosa as mensaxes de erro. É moi sinxelo. No pai creamos un método público que se encargará de xestionar as mensaxes de erro. Este será o seu código:

```
public void xestionDeMensaxesDeErro(int numErro)
{
    switch(numErro)
    {
        case 1: JOptionPane.showMessageDialog(this, "O nome non pode estar baleiro");
                break;
        case 2: JOptionPane.showMessageDialog(this, "Os apelidos non poden estar baleiros");
                break;
    }
}
```

Agora no botón Aceptar do diálogo de clase dlgDatosPersoais escribimos as seguintes liñas:

```
private void btnAceptarActionPerformed(java.awt.event.ActionEvent evt) {
    if(txtNome.getText().trim().compareTo("")==0)
    {
        ((FrmPrincipal)getParent()).xestionDeMensaxesDeErro(1);
        return;
    }
    if(txtApelidos.getText().trim().compareTo("")==0)
    {
        ((FrmPrincipal)getParent()).xestionDeMensaxesDeErro(2);
        return;
    }
}
```

O método que se encarga de obter a referencia ao pai do diálogo é o método getParent. Este método devólvenos un obxecto de tipo java.awt.Container (un contedor xenérico). O único que temos que facer é castealo á clase que realmente é (un FrmPrincipal) e despois empregar os seus métodos públicos.

Na seguinte imaxe podemos ver a chamada ao método público na xanela pai que se encarga de xestionar as mensaxes de erro:

Un xeito de abordar o problema é o que acabamos de ver empregando getParent. Outro xeito é empregar os parámetros pasados a través do construtor da xanela filla. Cando o pai crea a xanela filla pásalle a través do construtor desta unha referencia de si mesmo, a cal poderá ser empregada pola xanela filla para acceder aos métodos públicos do seu pai. Imos aplicar esta técnica modificando a aplicación para que faga o seguinte: imos facer que cando premamos no botón aceptar do diálogo de clase DlgDatosAcademicos amósese unha mensaxe co tipo de estudios elixidos, pero quen lanzará esa mensaxe non será o diálogo senón que o diálogo chamará a un método público na xanela pai que será o encargado de amosar a mensaxe. Primeiro creamos o método público que se encargará de lanzar as mensaxes na xanela pai. Este será o seu código:

```
public void xestionDeMensaxesDeGraoAcadado(int grao)
{
    switch(grao)
    {
        case 1: JOptionPane.showMessageDialog(this, "O máximo grao que acadou vostede e ESO");
                break;
        case 2: JOptionPane.showMessageDialog(this, "O máximo grao que acadou vostede e Bachalerato");
                break;
        case 3: JOptionPane.showMessageDialog(this, "O máximo grao que acadou vostede e FP");
                break;
        case 4: JOptionPane.showMessageDialog(this, "O máximo grao que acadou vostede e Universidade");
                break;
    }
}
```

Agora imos modificar a clase DlgDatosAcademicos para que poida recibir unha referencia da súa xanela pai. Primeiro definimos un atributo na clase no cal almacenaremos a referencia á xanela pai:

```
    private FrmPrincipal xanelaPai;
```

Cando teñamos que acceder a un método público da xanela pai farémolo a través da referencia xanelaPai.

Agora imos modificar o construtor do diálogo para capturar a referencia ao seu pai que recibe a través do construtor cando é creado:

```
    public DlgDatosAcademicos(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        xanelaPai=(FrmPrincipal)parent;
        initComponents();
    }
```

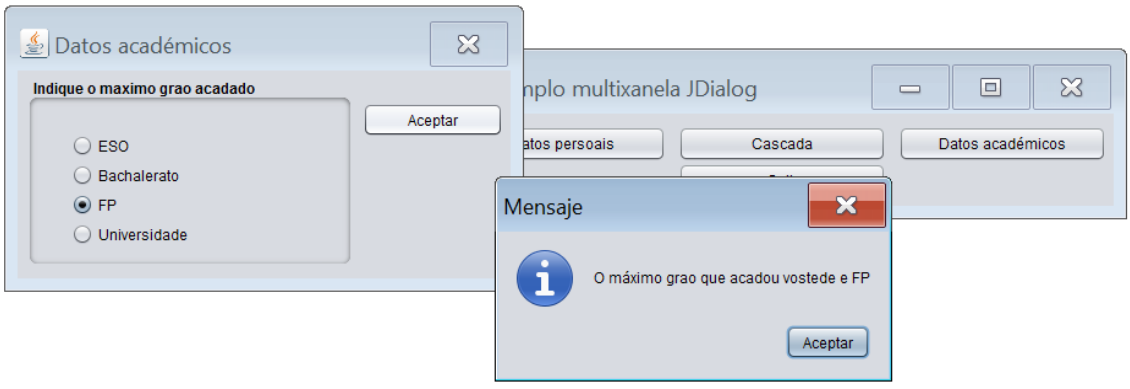
A través do parámetro parent do construtor a xanela pai pasa ao diálogo unha referencia de si mesma. No corpo do construtor asignamos esa referencia da xanela pai ao noso atributo de clase xanelaPai de xeito que agora a través de xanelaPai xa podemos acceder a calquera método público da xanela pai. Fixémonos en que para asignar a referencia da xanela pai recibida como parámetro ao noso atributo de clase xanelaPai é necesario casteala, xa que o parámetro recibido no construtor é de clase java.awt.Frame, é dicir, un formulario xenérico e non obstante o que nos recibimos realmente é un FrmPrincipal.

Agora que temos unha referencia á xanela pai a través do atributo de clase xanelaPai unicamente hai que invocar aos métodos públicos deste que necesitemos empregar. No caso que estamos tratando, ao premer sobre o botón Aceptar da xanela de clase DlgDatosAcademicos executaremos o seguinte código:

```
    private void btnAceptarActionPerformed(java.awt.event.ActionEvent evt) {
        int grao;
        if(rbtESO.isSelected()) grao=1;
        else if(rbtBachalerato.isSelected()) grao=2;
             else if(rbtFP.isSelected()) grao=3;
                  else grao=4;
        xanelaPai.xestionDeMensaxesDeGraoAcadado(grao);
    }
```

Como se pode ver no bloque de código anterior, empregando o atributo de clase xanelaPai (o cal referencia á xanela pai) podemos executar o método público da xanela pai xestionDeMensaxesDeGraoAcadado.

Na seguinte imaxe podemos ver a chamada ao método público na xanela pai que se encarga de xestionar as mensaxes de grao acadado:



Se o único que quixésemos controlar é que unha xanela se poida abrir só unha vez, ou incluso un número determinado de veces, parece que a solución para o control das xanelas aquí exposta necesita de demasiado código, ¿non cres? Propón unha mellora que nos libere do XestorDeXanelas.