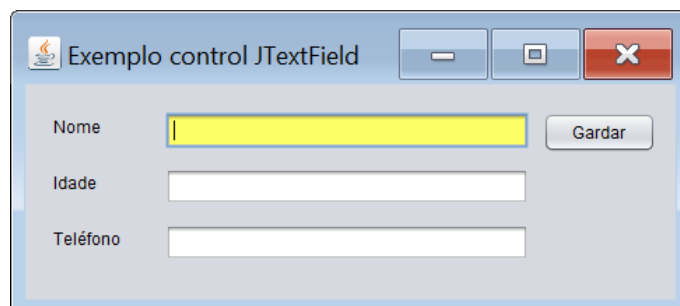


# 1. Compoñente Caixa de texto (JTextField)

---

Cando desenvolvemos un programa, habitualmente témoslle que solicitar ao usuario que nos proporcione unha serie de datos de entrada para poder procesalos e xerar un resultado. Cando os datos de entrada que queremos recoller son cadeas alfanuméricas, o xeito de recollelas é a traves de controis de tipo caixa de texto. Unha caixa de texto é un compoñente dentro do cal o usuario pode escribir valores alfanuméricos a través do teclado. Adicionalmente, pode comportarse como as etiquetas, é dicir, podemos facer que as nosas aplicacións escriban texto dentro delas. Habitualmente unha caixa de texto vai precedida por unha etiqueta que se encarga de informar ao usuario da información que se espera recoller na caixa de texto que acompaña. Típicamente nos formularios nos que hai caixas de texto existe algún control (normalmente un botón) sobre o cal o usuario pode actuar de xeito que esa actuación desencadee a recollida dos datos das caixas de texto para o seu posterior procesamento. Aínda que non é habitual que as caixas de texto desencadeen eventos en función de accións realizadas sobre elas, cabe destacar que poden disparar eventos cada vez que realizamos algunha acción de teclado sobre elas, de xeito que poderemos en todo momento realizar validacións sobre o texto que o usuario está introducindo nun momento dado. O compoñente caixa de texto defínese a través da clase `javax.swing.JTextField`. Gráficamente o seu aspecto é o seguinte:



Na imaxe anterior podemos ver tres caixas de texto acompañadas por tres etiquetas e un botón.

As caixas de texto, ao igual que vimos que ocurría coas etiquetas, tamén teñen un forte compoñente de persoalización que nos permite adaptalas ás necesidades gráficas dos nosos interfaces de usuario.

Propiedades do control `JTextField` empregadas máis habitualmente e que pódense establecer a través do entorno de programación:

Propiedade	Función
Background	Cor de fondo do compoñente
Border	Borde do compoñente
Cursor	Mediante esta propiedade indícase a imaxe que amosará o punteiro do rato ao navegar sobre o compoñente

Editable	Emprégase para indicar se podemos ou non escribir na caixa de texto
Enabled	Se esta propiedade está activada o compoñente será funcional. No caso de que esta propiedade estea desactivada o compoñente será visualizable, pero o usuario non poderá interaccionar con el.
Focusable	Se esta propiedade está activada o compoñente entrará na roda de reparto do foco de xeito que ao premer a tecla de cambio de foco (habitualmente o tabulador) nalgún momento tomará o foco da aplicación (cando sexa a súa quenda na roda de reparto do foco). Pola contra, se esta propiedade está desactivada o compoñente non entrará na roda de reparto do foco e soamente será posible acceder a el mediante o rato.
Font	Características da fonte do compoñente (tipo de fonte, tamaño, ... )
Foreground	Cor do texto do compoñente
HorizontalAlignment	Aliñación horizontal do texto do compoñente
SelectedTextColor	Cor do texto seleccionado na caixa de texto
SelectionColor	Cor de fondo do texto seleccionado na caixa de texto
Text	Emprégase para establecer o texto que aparece na caixa de texto
ToolTipText	Mediante esta propiedade establécese unha mensaxe (tooltip) que será amosado ao deixar o punteiro do rato sobre o compoñente. É habitual empregar esta mensaxe para indicar cal é a utilidade do compoñente

Eventos do control JTextField empregados máis habitualmente e que pódense establecer a través do entorno de programación:

Evento	Lanzamento
KeyPressed	Este evento é disparado cando pulsamos unha tecla sobre a caixa de texto. Se deixamos unha tecla pulsada, o evento dispararase tantas veces como caracteres sexan escritos na caixa de texto
KeyReleased	Este evento é disparado cando deixamos de pulsar unha tecla sobre a caixa de texto. Se deixamos unha tecla pulsada, o evento dispararase úicamente cando deixemos de pulsar a tecla.
KeyTyped	Este evento é disparado cando escribimos un carácter na caixa de texto.

Métodos do control JTextField empregados máis habitualmente:

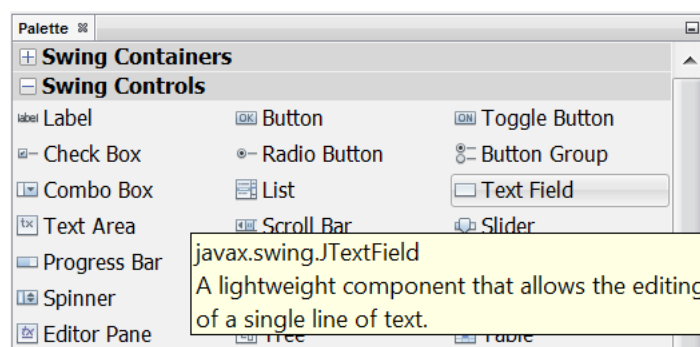
Método	Función
public String getText()	Emprégase para recuperar o texto dunha caixa de texto
public void setText(String text)	Emprégase para establecer o texto dunha caixa de texto

## Xestión de caixas de texto empregando NetBeans

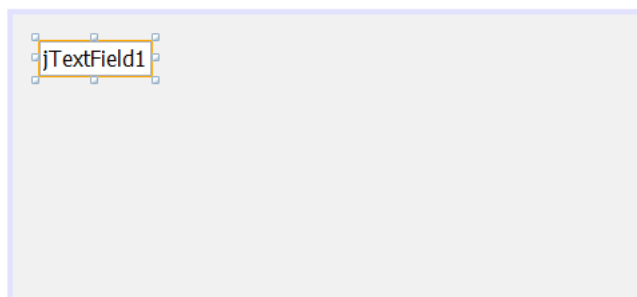
A continuación imos desenvolver o seguinte formulario:

O formulario consta de catro caixas de texto. As tres primeiras empregaremos para que o usuario realice entradas de texto indicando o nome, a idade e o teléfono dalgunha persoa. Ao pulsar sobre o botón Gardar, recolleranse os datos das caixas de texto, validarase a súa corrección e finalmente amosaranse na caixa de texto asociada á etiqueta Resultado o resultado da validación dos datos de entrada indicados polo usuario (esta caixa de texto non será editable).

Para engadir un compoñente de tipo JTextField no noso formulario primeiramente seleccionáremolo da paleta de compoñentes do entorno de programación:

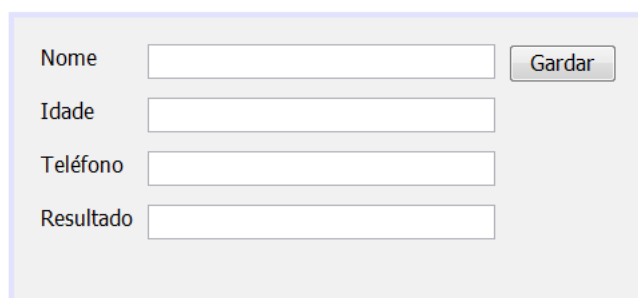


e arrastrámolo ata o formulario:

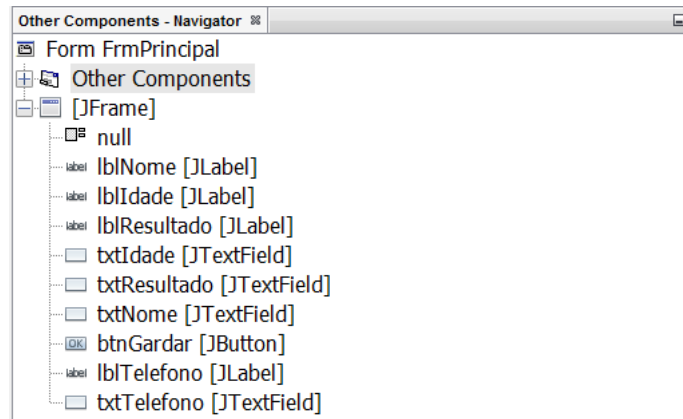


Unha vez situado dentro do formulario, e como sempre, adaptámolo para que teña o aspecto visual que desexamos que teña.

No caso que estamos desenvolvendo necesitamos engadir catro etiquetas, catro caixas de texto e un botón. Unha vez engadidos e colocados no seu lugar correspondiente, este será o resultado do deseño:



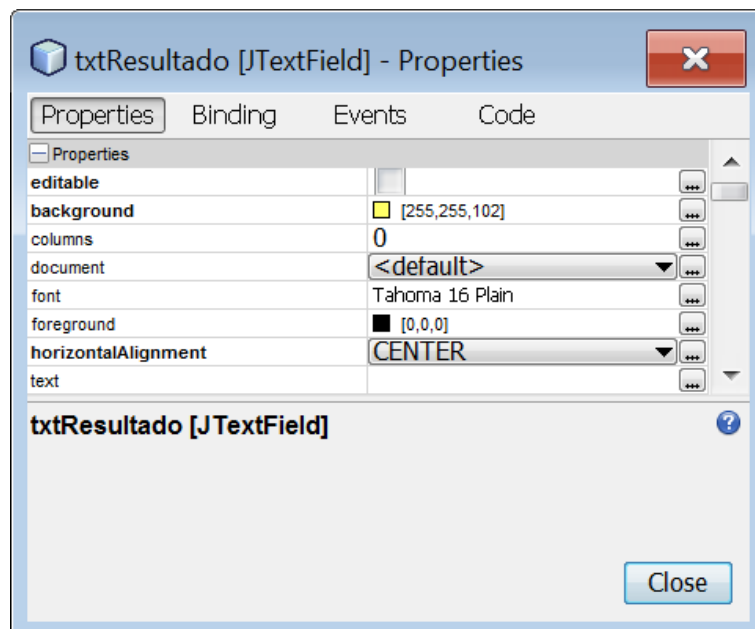
O noso formulario quedará configurado do seguinte xeito:



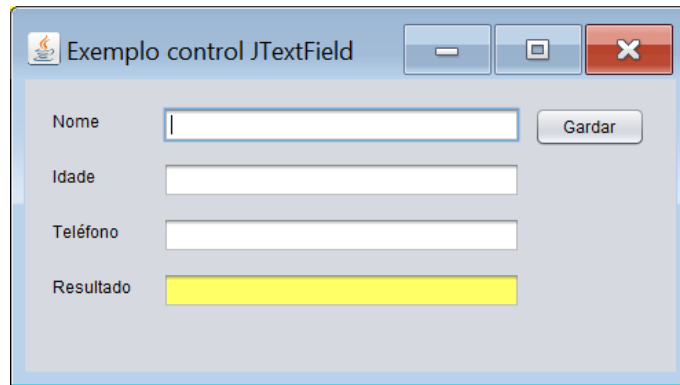
Unha vez que temos colocados os compoñentes que necesitamos hai que configurar o seu aspecto. Neste caso necesitamos facer as seguintes modificacións:

A caixa de texto resultado debe ser non editable e ter como cor de fondo o amarelo. Ademáis imos facer que todo o texto que conteña sexa centrado. Para realizar estas accións accedemos ás propiedades da caixa de texto txtResultado e facemos as seguintes modificacións:

- Desmarcamos a casilla de verificación da propiedade editable para indicar que esta caixa de texto non será editable.
- Seleccionamos unha cor de fondo amarelo para a súa propiedade background.
- Modificamos a propiedade horizontalAlignment e dámoslle o valor CENTER para indicar que o contido da caixa de texto estará centrado.



Se executamos a aplicación este será o resultado:



Como pódese observar o comportamento visual da aplicación xa está resolto.

Imos realizar unha pequena modificación sobre a caixa de texto txtIdade co fin de limitar a súa lonxitude a tres caracteres (non ten sentido introducir unha idade de máis de tres cifras). Lamentablemente para limitar a lonxitude dunha caixa de texto non existe ningunha propiedade, senón que temos que realizar unha tarefa un pouco máis complexa. Imos explicar o que temos que facer. Para elo, antes de nada imos ver que ocorre cada vez que escribimos un carácter nunha caixa de texto: canda escribimos un carácter nunha caixa de texto, realmente ese carácter nin sequera é pintado na caixa de texto, senón que é enviado a un buffer de memoria. O buffer de memoria terá a capacidade de decidir se acepta ao carácter ou se non o acepta. Se non o acepta (p.e.: non queremos aceptar números no buffer), o carácter premido perderase e nunca vaise pintar na caixa de texto. Polo contrario, se o carácter é aceptado, será almacenado no buffer. Unha vez almacenado, o buffer de memoria vaille indicar á caixa de texto que é o que hai almacenado nel, para que finalmente a caixa de texto amose o contido do buffer por pantalla. Agora é cando nos imos ver aparecer ao carácter escrito. O proceso é practicamente instantáneo, pero en realidade ocorre todo isto (e algunha cousa máis). É dicir, realmente a información que amosa unha caixa de texto non está almacenada na caixa de texto, senón que se almacena nun buffer e o único que fai a caixa de texto é amosar o contido do buffer. Este xeito de traballar chámase paradigma vista controlador e o veremos máis detalladamente cando vexamos compoñentes gráficos avanzados.

Volvendo á limitación da lonxitude da caixa de texto txtIdade, o primeiro que temos que facer é crear unha clase que herde de `javax.swing.text.PlainDocument`. A clase `javax.swing.text.PlainDocument` emprégase para representar o buffer de memoria do que acabamos de falar. Non ten limitación ningunha respecto ao número de caracteres que admite, nin ao seu tipo (letras, números, ...). Por elo, imos crear unha clase que herde de `javax.swing.text.PlainDocument` e imola modificar para introducir a limitación do número de caracteres. A esta clase ímola chamar `LimiteLonxitudeJTextField`, e será o noso buffer personalizado (neste caso a personalización consiste na limitación no número de caracteres que admite). Este vai ser o seu código:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package exemploJTextField;

/**
 *
```

```

* @author German DR
*/
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.PlainDocument;

public class LimiteLonxitudeJTextField extends PlainDocument
{
    private int lonxitude;

    public LimiteLonxitudeJTextField(int lonxitude)
    {
        super();
        this.lonxitude = lonxitude;
    }

    @Override
    public void insertString( int offset, String str, AttributeSet attr ) throws BadLocationException
    {
        if (str == null) return;

        if ((getLength() + str.length()) <= lonxitude)
        {
            super.insertString(offset, str, attr);
        }
    }
}

```

Como pódese observar no código anterior, creamos un novo atributo (que se sumará aos herdados) chamado lonxitude. Empregarémolo para indicar o número máximo de caracteres almacenables no buffer e polo tanto a lonxitude máxima da caixa de texto que empregue este buffer. A continuación reescribimos o método insertString. Este método é invocado cada vez que se intenta inserir información no buffer, é dicir, cada vez que escribimos na caixa de texto asociada ao buffer. O método insertString encargarase de comprobar se a lonxitude do que xa ten almacenado mais a lonxitude do que recibe (o que escribimos na caixa de texto asociada) supera o límite (atributo lonxitude) establecido para o buffer. No caso de que o supere non fai nada, co cal non se almacenará a nova información no buffer e polo tanto non se mostrará na caixa de texto. Non obstante, no caso de que non o supere, chamará ao método insertString da súa superclase e este encargarase de almacenar a información no buffer para finalmente amosala na caixa de texto adxunta ao buffer.

O último que nos falta por facer é ligar a caixa de texto ao buffer de clase LimiteLonxitudeJTextField. Isto facémolo mediante a seguinte liña de código:

```
txtIdade.setDocument(new LimiteLonxitudeJTextField(3));
```

Neste caso, mediante o método setDocument asociamos a caixa de texto txtIdade cun buffer de almacenamento de clase LimiteLonxitudeJTextField que poderá almacenar un máximo de tres caracteres. É importante destacar que non estamos validando o tipo de información gardada no buffer (numérica, alfanumérica,...). Unicamente a súa lonxitude (sería posible facer as validacións de tipo neste buffer que acabamos de crear, pero por razóns didácticas ímolo facer doutro xeito).

Por certo, o lugar escollido para ligar a caixa de texto co buffer non é casual:

```

public FrmPrincipal() {
    initComponents();
    txtIdade.setDocument(new LimiteLonxitudeJTextField(3));
}

```

Faise no construtor do formulario e xusto despois do método initComponents. O método initComponents encargase de crear tódolos obxectos do noso formulario deseñados mediante o entorno gráfico. Polo tanto, unha vez que se executa este método podemos estar seguros de que existen tódolos compoñentes gráficos deseñados no entorno gráfico e que as súas propiedades iniciais están establecidas. A partir de agora é o momento de establecer por código calquera

cambio que queiramos realizar sobre os compoñentes do noso entorno gráfico tal e como facemos p.e. coa liña que liga a txtIdade co seu buffer.

Ben, unha vez establecida a lonxitude da caixa de texto txtIdade a tres caracteres, imos desenvolver o resto da funcionalidade do formulario.

A acción desenvolverase no momento que fagamos clic sobre o botón Gardar. Por tanto imos escribir o seu actionPerformed.

O primeiro que temos que facer é recoller o texto das caixas de texto do formulario:

```
String nome=txtNome.getText().trim();
String idade=txtIdade.getText().trim();
String telefono=txtTelefono.getText().trim();
```

Como pódese observar, recollemos o contido da cada caixa de texto aplicando o seu método getText e almacenámolo nunha variables de tipo String para o seu posterior tratamento.

Unha vez recollidos os valores das caixas de texto pasamos a validalos. Por exemplo, a validación para a caixa de texto nome sería a seguinte:

```
if(nome.compareTo("")==0)
{
    txtResultado.setText("Debe indicar o nome");
    return;
}
```

Neste caso o que facemos é comprobar se o usuario escribiu ou non algo dentro da caixa de texto nome. Para elo o que facemos é comprobar se a variable nome esta baleira ou non. Recordemos que a variable nome contén o valor da caixa de texto txtNome xa que foi asignado a través do método getText da caixa de texto. No caso de que a variable nome esté baleira, amosaremos na caixa de texto txtResultado unha mensaxe de erro. Para amosar unha mensaxe dentro da caixa txtResultado facemos emprego do seu método setText pasándolle a mensaxe que queremos amosar.

Finalmente, este sería o código completo do actionPerformed do botón Gardar:

```
private void btnGardarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String nome=txtNome.getText().trim();
    String idade=txtIdade.getText().trim();
    String telefono=txtTelefono.getText().trim();

    //Validación do formulario

    if(nome.compareTo("")==0)
    {
        txtResultado.setText("Debe indicar o nome");
        return;
    }

    if(idade.compareTo("")==0)
    {
        txtResultado.setText("Debe indicar a idade");
        return;
    }

    if(!Utilidades.eNumeroEnteiro(idade))
    {
        this.txtResultado.setText("A idade debe ter un valor numérico enteiro");
        return;
    }

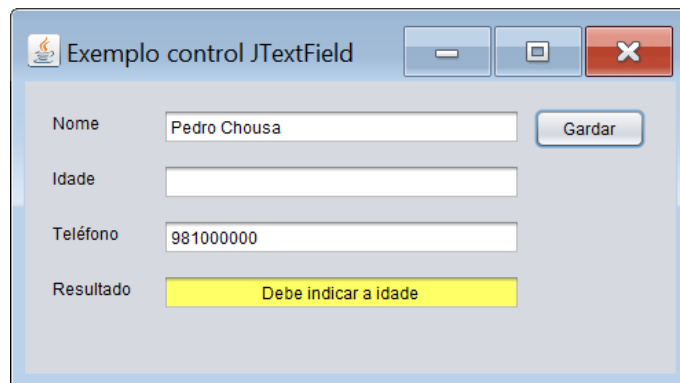
    int idadeI=new Integer(idade).intValue();

    if(idadeI < 0 || idadeI > 130)
    // A comprobacion de idade.length e para evitar enteiros maiores que (2^31)-1 (Integer.MAX_VALUE)
    {
        txtResultado.setText("A idade debe estar entre 0 e 130");
        return;
    }

    if(telefono.compareTo("")==0)
    {
        txtResultado.setText("Debe indicar o teléfono");
        return;
    }
}
```

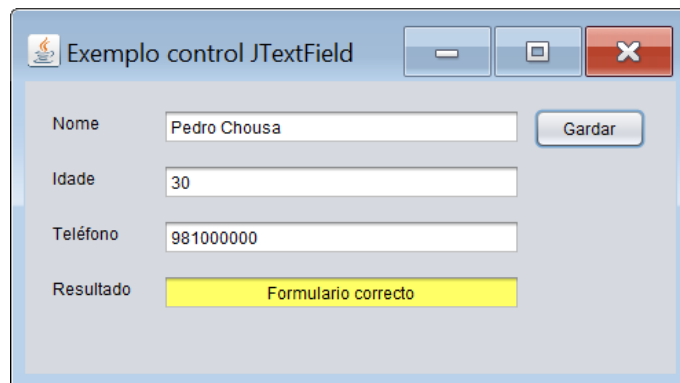
```
txtResultado.setText("Formulario correcto");  
}
```

Se executamos a aplicación introducindo algún erro este será o resultado:



The screenshot shows a Java Swing window titled "Exemplo control JTextField". It contains four text input fields: "Nome" (containing "Pedro Chousa"), "Idade" (empty), "Teléfono" (containing "981000000"), and "Resultado" (containing "Debe indicar a idade" in a yellow box). A "Gardar" button is located to the right of the "Nome" field. The window has standard Mac OS X window controls (minimize, maximize, close) in the title bar.

Se executamos a aplicación sin introducir erros este será o resultado:



The screenshot shows the same Java Swing window "Exemplo control JTextField". In this state, the "Idade" field contains the value "30". The "Resultado" field now displays "Formulario correcto" in a yellow box. The "Nome" field still contains "Pedro Chousa" and the "Teléfono" field contains "981000000". The "Gardar" button remains visible.

## Xestión de eventos de teclado

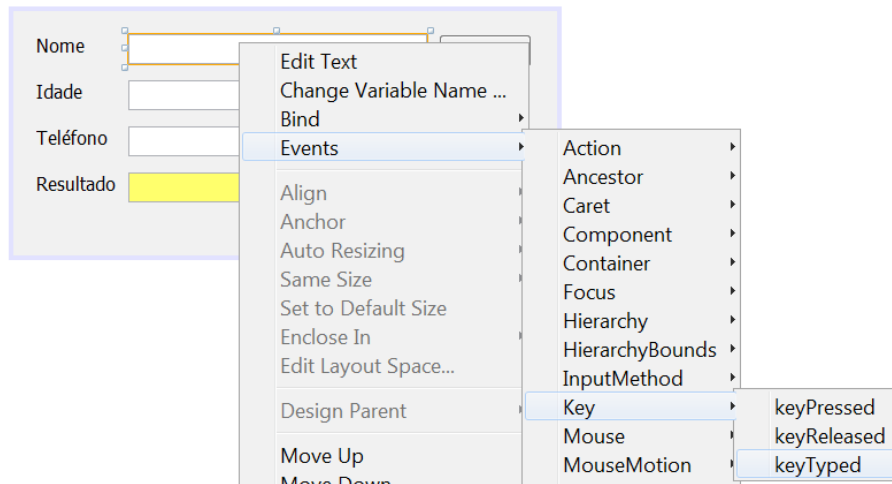
Na aplicación que acabamos de desenvolver permitimos que na caixa de texto que se emprega para escribir o nome poidamos escribir caracteres distintos das letras. Do mesmo modo permitimos que nas caixas de texto empregadas para recuperar a idade e o teléfono poidamos escribir caracteres que non sexan números. Imos modificar a aplicación para introducir as seguintes restricións:

- Na caixa de texto txtNome só será posible escribir letras (incluídas as vocais acentuadas), espazos e retrocesos.
- Na caixa de texto txtIdade e na caixa de texto txtTelefono só será posible escribir números e retrocesos.

O xeito de evitar que escribamos caracteres non permitidos nunha caixa de texto é mediante o emprego dos seus eventos. O primeiro que imos facer é detectar cando escribimos un carácter nunha caixa de texto mediante o seu evento `KeyTyped`. Nese momento o carácter está en memoria (aínda nin sequera chegou ao buffer asociado á caixa de texto). Polo tanto o que faremos será ver se o carácter que acabamos de escribir e que aínda está en memoria é un carácter permitido ou non. No caso de que o sexa non facemos nada, pero no caso de que non sexa un carácter permitido evitaremos que sexa debuxado na caixa de texto.



A continuación imos ver como facemos isto para a caixa de texto txtNome. O primeiro debemos implementar o seu evento KeyTyped. Para implementar o método que se disparará cando ocorra o evento KeyTyped da caixa de texto txtNome, pulsamos co botón dereito sobre a caixa de texto txtNome e seleccionamos Events no menú despregable amosado. Dentro do submenú despregable que nos amosa Events seleccionamos Key e finalmente dentro do menú despregable que nos amosa Key seleccionamos a opción keyTyped:



Ao realizar esta acción o entorno de programación creará automaticamente o seguinte método:

```
private void txtNomeKeyTyped(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
}
```

Agora o único que resta é implementar o seu código para que cando o evento KeyTyped da caixa de texto txtNome sexa disparado lévense a cabo as accións indicadas. Neste caso imos escribir as seguintes liñas de código:

```
private void txtNomeKeyTyped(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if (!isLetra(evt))
    {
        evt.consume();
    }
}
```

Lembremos que neste momento o carácter tecleado está en memoria, pero aínda non foi debuxado. O primeiro que facemos é pasar a información asociada ao evento KeyTyped (a cal inclúe entre outros datos o carácter tecleado) a un método chamado isLetra. Este método comprobará se o carácter tecleado é un carácter permitido (letras, espazos e retrocesos) ou non devolvendo true ou false respectivamente. No caso de non ser un carácter permitido consúmese o evento mediante o evento consume(). Consumir o evento quere dicir terminalo, polo tanto non se seguirá co resto do proceso. O carácter non pasará ao buffer e polo tanto non será debuxado na caixa de texto.

Vexamos a continuación como é o código do método isLetra:

```
private boolean isLetra(java.awt.event.KeyEvent evt)
{
    // Comprobar se a pulsacion de teclado pasada
    // en evt é unha letra
```

```

char caracter=evt.getKeyChar();
if((
    (caracter<'A' || caracter >'z')&&
    (caracter!='á') &&
    (caracter!='é') &&
    (caracter!='í') &&
    (caracter!='ó') &&
    (caracter!='ú') &&
    (caracter!='À') &&
    (caracter!='É') &&
    (caracter!='Í') &&
    (caracter!='Ó') &&
    (caracter!='Û') &&
    (caracter!='ñ') &&
    (caracter!='Ñ') &&
    (caracter!=KeyEvent.VK_SPACE) &&
    (caracter!=KeyEvent.VK_BACK_SPACE)
) ||
    (evt.isAltDown() ||
    evt.isAltGraphDown() ||
    evt.isControlDown() ||
    evt.isMetaDown()
))
{
    return false;
}
return true;
}

```

O primeiro que fai este método é recuperar o carácter tecleado mediante o método `getKeyChar`. A continuación simplemente realiza unha serie de comparacións para comprobar se ese carácter é válido ou non. A comparación que fai con `VK_SPACE` está comprobando se o carácter é un espazo. A comparación feita con `VK_BACK_SPACE` está comprobando se o carácter é a tecla de retroceso. A clase `KeyEvent` ten definidas constantes para un gran número de teclas. Todas comencian por `VK_`. Ademais, o método tamén encárgase de comprobar se temos pulsada algunha tecla especial como `Alt` (`evt.isAltDown`), `AltGrp` (`evt.isAltGrpDown`), `Ctrl` (`evt.isControlDown`), ou la `Apple Key` (`isMetaDown`). A `Apple Key` é unha tecla de los Macintosh que non ten equivalencia actualmente baixo sistemas non Apple. No caso da nosa aplicación se temos pulsada algunha tecla especial consumiremos o evento.

Respecto ao control de caracteres para a caixa de texto `txtIdade` faremos algo parecido. O primeiro debemos implementar o seu evento `KeyTyped`. Para elo escribimos as seguintes liñas de código para o seu evento `KeyTyped`:

```

private void txtIdadeKeyTyped(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if(!isNumero(evt))
    {
        evt.consume();
    }
}

```

Recordemos de novo que neste momento o carácter tecleado está en memoria, pero aínda non foi debuxado. O primeiro que facemos é pasar a información asociada ao evento `KeyTyped` (a cal inclúe entre outros datos o carácter tecleado) a un método chamado `isNumero`. Este método comprobará se o carácter tecleado é un carácter permitido (números e retrocesos) ou non devolvendo `true` ou `false` respectivamente. No caso de no ser un carácter permitido consúmese o evento mediante o evento `consume()`.

Vexamos a continuación como é o código do método `isNumero`:

```

private boolean isNumero(java.awt.event.KeyEvent evt)
{
    // Comprobar se a pulsacion de teclado pasada
    // en evt é un número
    char caracter=evt.getKeyChar();
    if((
        (caracter<'0' || caracter >'9')&&
        (caracter!=KeyEvent.VK_BACK_SPACE)
    ) ||
        (evt.isAltDown() ||
        evt.isAltGraphDown() ||
        evt.isControlDown() ||
        evt.isMetaDown()
    ))
    {

```

```
        return false;
    }
    return true;
}
```

O primeiro que fai este método é recuperar o carácter tecleado mediante o método `getKeyChar`. A continuación simplemente realiza unha serie de comparacións para comprobar se ese carácter é válido ou non.

Para o control de caracteres da caixa de texto `txtTelefono` faremos unha xestión de eventos similar á realizada para a caixa de texto `txtIdade`.