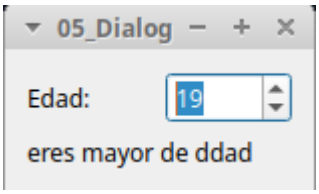
# Documentación en construcción

# Diálogos

## Diálogo básico




| main.cpp | |
|---|---|

```
#include <QApplication>
#include "dedad.h"

int main(int argc, char *argv[]) {

   QApplication app(argc,argv);

   DEdad * dEdad = new DEdad();

   dEdad->crear();

   dEdad->show();


   return app.exec();
}
```

Declarar la clase:

| dedad.h | |
|---|---|

```
#include <QWidget>
#include <QSpinBox>
```

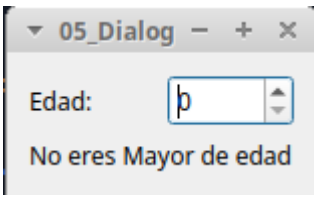
```
#include <QLabel>

class DEdad : public QWidget {
public:

   QLabel *etiqSup;
   QLabel *eticInfo;

   QSpinBox *spinBox;




};
```

| dedad.cpp | ¡¡ vacío !! |
|---|---|
| | |

compila y funciona

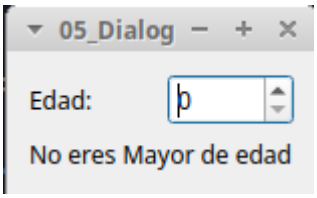
| dedad.h | |
|---|---|

```
#include <QWidget>
#include <QSpinBox>
#include <QLabel>

class DEdad : public QWidget {
public:

   QLabel *etiqSup;
   QLabel *eticInfo;

   QSpinBox *spinBox;

   void crear(void);



};
```




| dtransferencia.h | |
|---|---|

```
#include "dedad.h"
#include <QVBoxLayout>
#include <QHBoxLayout>

void DEdad::crear(void){

   QVBoxLayout * layout;
   QHBoxLayout * layoutSuperior;

   layout = new QVBoxLayout();
   layoutSuperior = new QHBoxLayout;

   spinBox = new QSpinBox();
   etiqSup = new QLabel("Edad: " );
   eticInfo = new QLabel("No erers Mayor de edad");

   layoutSuperior->addWidget(etiqSup);
   layoutSuperior->addWidget(spinBox);

   layout->addLayout(layoutSuperior);
   layout->addWidget(eticInfo);

   this->setLayout(layout);
}
```




| dtransferencia.h | |
|---|---|
| | |

## comportamiento

Creamos nuestro propio slot (explicar por qué, hacer dibujo)

dtransferencia.h

```
class DEdad : public QWidget {

public:

    DEdad(QWidget *parent = nullptr, Qt::WindowFlags f =
Qt::WindowFlags());

    QLabel *etiqSup;
    QLabel *eticInfo;

    QSpinBox *spinBox;

public slots:

    void slotIndicar(int edad);

} ;
```

implementar el slot:

dtransferencia.h

```
void DEdad::slotIndica(int e) {

      if (e >= 18 )
            etiqInfo->setText("eres mayor de ddad");
      else
            etiqInfo->setText("eres menor de edad");


}
```

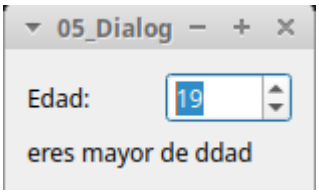
necsitamos declarar Q_OBJECT

dtransferencia.h

```
class DEdad : public QWidget {

Q_OBJECT

public:

   DEdad(QWid
```

Ahora en el constructor, hacemos la conexión (fijarsus que es this)

| dtransferencia.h | |
|---|---|
| | |




| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|

# Diálogo Transferencia

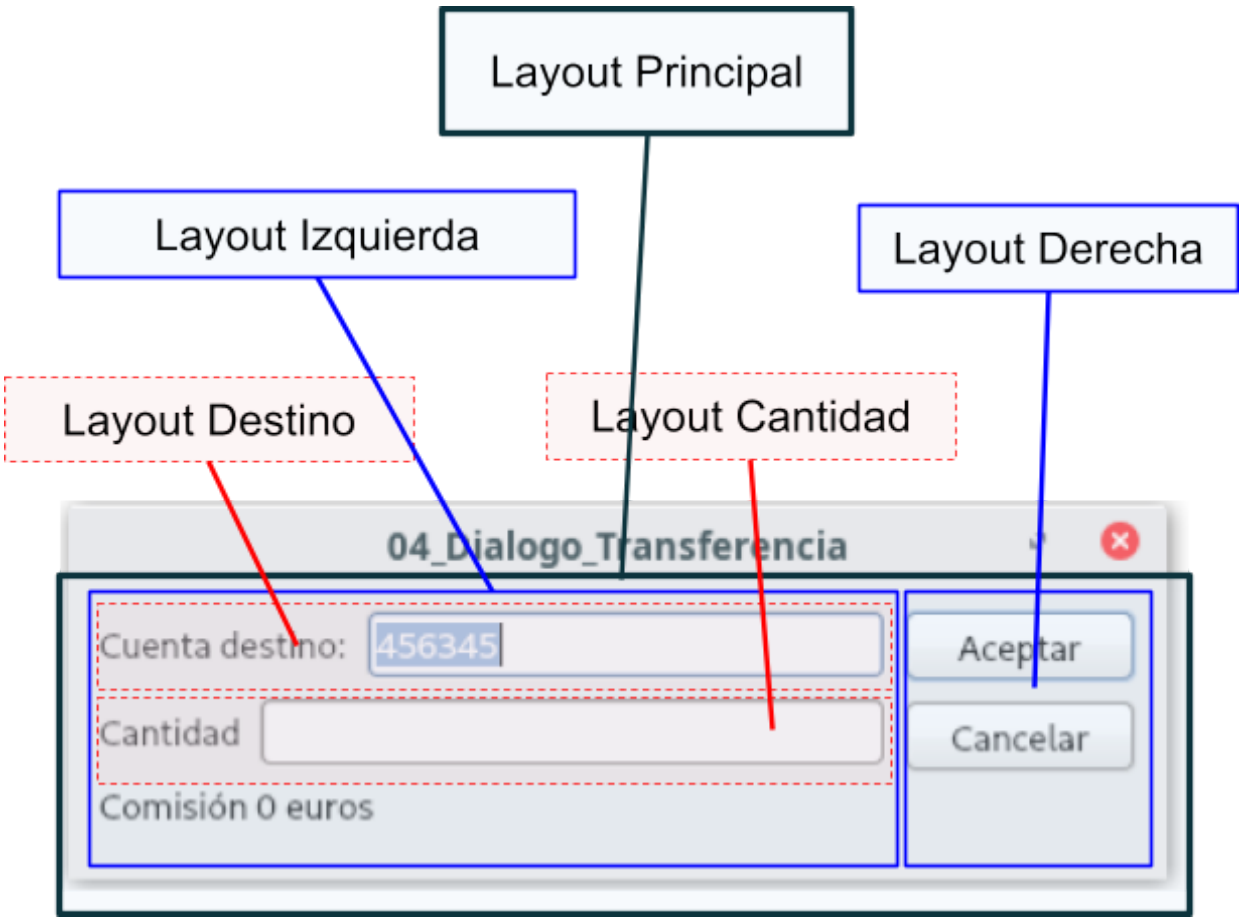
En este importante documento vamos a ampliar el comportamiento que obtenemos de un Diálogo. A partir de aquí, como ejemplo conductor, vamos a crear un diálogo con el siguiente aspecto:


Este diálogo recoge datos para realizar una transferencia bancaria a alguna cuenta. En él introducimos el número de cuenta y la cantidad a transferir (observa las entradas de texto correspondientes). El comportamiento que deseamos es el siguiente:

- Cuando se introduzca un número de cuenta correcto y una cantidad monetaria (entera), el botón "Aceptar" se habilitará. Una cuenta válida es aquella que tiene 6 dígitos.
- Cuando se cambie la cantidad, aparecerá el valor de la comisión en la parte inferior. La comisión es el 1% de la cantidad introducida.
- Al pulsar el botón aceptar el diálogo se cerrará. Con el botón de Cancelar también se cerrará, pero el diálogo debe indicar si se ha aceptado o rechazado (ver la solución para entender mejor este requisito)

Aviso: Sólo vamos a trabajar con el diálogo, no se van a manipular cuentas como en las actividades del curso.

# Disposición de componentes




| dtransferencia.h | Declaramos componentes y layouts |
|---|---|

```
class DTransferencia : public QDialog {

Q_OBJECT

public:
/* layouts */
    QVBoxLayout *lyDer, *lyIzq;
    QHBoxLayout *lyPrincipal, *lyDestino, *lyCantidad;
/* Componentes */
    QPushButton *bAceptar, *bCancelar;
    QLabel *lDestino, *lCantidad, *lComision;
    QLineEdit *leDestino ,  *leCantidad;
```

## ¿Por qué es un QDialog?

Anteriormente usamos QWidget

Vamos a añadir la declaración del constructor... imitando el constructor del padre:

| dtransferencia.h | |
|---|---|

```
#include <QDialog>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QLabel>
#include <QLineEdit>


class DTransferencia : public QDialog {

Q_OBJECT

public:
...

    DTransferencia(QWidget * parent = 0) ;
....
};
```

Seguimos con la implementación en dtransferencia.cpp

Inicialmente, damos aspecto a la ventana en el constructor. Aunque veas muchas líneas de código, observa que se hace lo siguiente:

1. Crear los widgets (botones, etiquetas y entradas de texto)
2. Crear los layouts
3. Meter los widgets dentro de los layouts
4. Meter unos layouts dentro de otros
5. Asignar el Layout principal como layout del diálogo

| dtransferencia.cpp | Creamos los componentes, los layouts y los disponemos unos dentro de otros |
|---|---|

```
DTransferencia::DTransferencia(QWidget * parent) : QDialog(parent)
{

 lyDer = new       QVBoxLayout;
 lyIzq = new      QVBoxLayout;
 lyPrincipal = new QHBoxLayout;

 lyDestino = new     QHBoxLayout;
 lyCantidad = new     QHBoxLayout;
/* Componentes */
```

```
bAceptar = new QPushButton("Aceptar");
bAceptar->setEnabled(false);

bCancelar = new QPushButton("Cancelar");
lDestino = new QLabel("Cuenta destino:");
lCantidad = new QLabel ("Cantidad");
lComision = new QLabel ("Comisión 0 euros");

leDestino = new QLineEdit;
leCantidad = new QLineEdit;

lyDer->addWidget(bAceptar);
lyDer->addWidget(bCancelar);
lyDer->addStretch();

lyDestino->addWidget(lDestino);
lyDestino->addWidget(leDestino);

lyCantidad->addWidget(lCantidad);
lyCantidad->addWidget(leCantidad);

lyIzq->addLayout(lyDestino);
lyIzq->addLayout(lyCantidad);
lyIzq->addWidget(lComision);
lyIzq->addStretch();

lyPrincipal->addLayout(lyIzq);
lyPrincipal->addLayout(lyDer);

   setLayout(lyPrincipal);
```

**addStretch()** añade espacio en ese lugar del layout para evitar que el resto de elementos del layout tenga que ensancharse hasta abarcar todo el área del layout. Esto permite a los botones mantener un tamaño natural.

## Comportamiento

El comportamiento del diálogo no es más que la reacción a Dos eventos:

1. El cambio de texto en la entrada de texto de la cuenta bancaria
2. El cambio de texto en la entrada de texto de la cantidad a transferir

Por otra parte, el efecto de los eventos es:

1. Activar o desactivar el botón
2. Actualizar la cantidad reflejada como comisión

Estos dos efectos vamos a programarlos mediante dos slots diferentes:

| dtransferencia.h | Declaramos los slots |
|---|---|

```
public slots:
   void slotActualizarBoton(const QString &);
   void slotActualizarComision(const QString &);
```

Ahora asociamos cada evento con el o los slots implicados. Observa el código donde se realiza la conexión evento - slot para deducir cómo estamos estableciendo la reacción

1. El botón se debe actualizar con cualquier evento de cambio de texto
2. El texto de la comisión se actualiza con el cambio de texto de la etiqueta cantidad:

La señal que refleja un cambio de texto es `textChanged(const QString &)` .

| dtransferencia.cpp | Realizamos las conexiones |
|---|---|

```
DTransferencia::DTransferencia(QWidget * parent) : QDialog(parent)
{
...
 connect(leDestino,SIGNAL(textChanged(const QString &)),
       this, SLOT(slotActualizarBoton(const QString &)));

 connect(leCantidad,SIGNAL(textChanged(const QString &)),
       this, SLOT(slotActualizarBoton(const QString &)));

 connect(leCantidad,SIGNAL(textChanged(const QString &)),
       this, SLOT(slotActualizarComision(const QString &)));
```

La implementación del slot es compleja porque hemos de validar las cadenas de texto introducidas asegurándonos de que son de seis dígitos en el caso del número de cuenta y de una cantidad numérica entera en el caso de la cantidad. En este punto del curso esto es algo difícil , <u>limítate a copiar el código.</u>

| dtransferencia.cpp | Implementamos el slot  para actualizar el botón |
|---|---|

```
void DTransferencia::slotActualizarBoton(const QString & cad){

    QString cadDestino = leDestino->text();
    QRegularExpression reDestino("^\\d{6}$");
```

```
    QRegularExpressionMatch mDestino =
reDestino.match(cadDestino);


    QString cadCantidad = leCantidad->text();
    QRegularExpression reCantidad("^\\d+$");

    QRegularExpressionMatch mCantidad =
reCantidad.match(cadCantidad);

    bool destinoOK = mDestino.hasMatch();
    bool cantidadOK = mCantidad.hasMatch();

    if (destinoOK && cantidadOK)
       bAceptar->setEnabled(true);
    else
       bAceptar->setEnabled(false);
   }
```

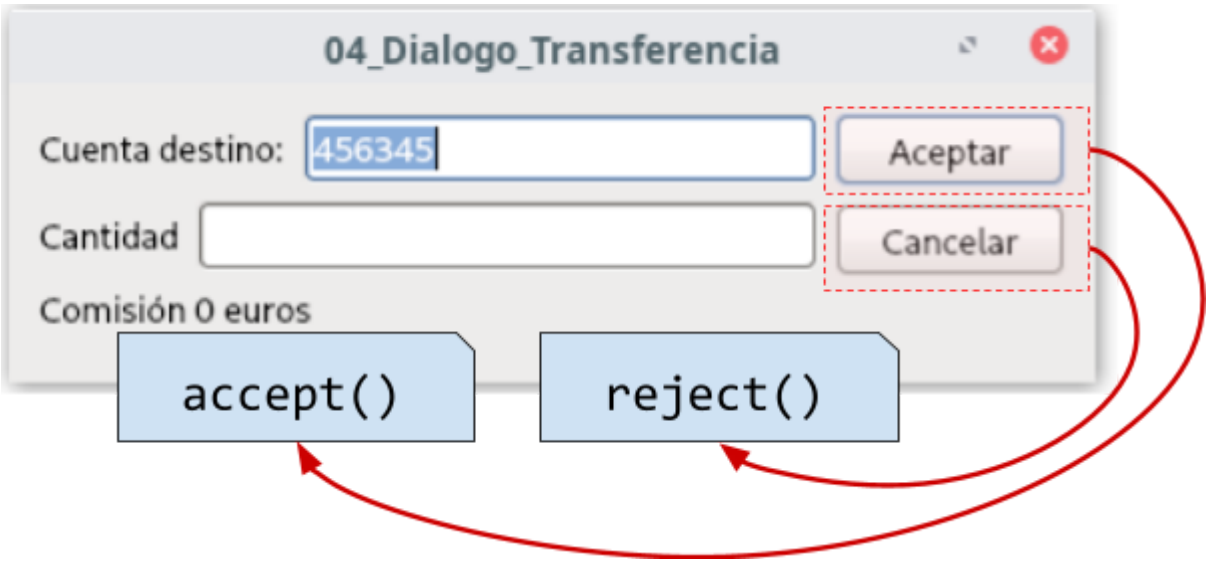
| dtransferencia.cpp | Implementamos el slot de actualizar la comisión |
|---|---|

```
void DTransferencia::slotActualizarComision(const QString &cad){

    QString cadena = leCantidad->text();
    float cantidad = cadena.toFloat();

    QString resultado("La comision es de ");
    resultado += QString::number(cantidad*0.01);
    resultado += " euros";

    lComision->setText(resultado);

}
```

Ahora vamos a añadir la funcionalidad de los botones "aceptar" y "cancelar". Para aprender, es conveniente que vayas a la documentación "assistant" y busques qué slots tiene la clase `QDialog` para el hecho de cerrase aceptando y cerrarse rechazando.

Desubrirás dos slots "accept()" y "reject()" . Estos slots deben activarse al hacer click en el botón aceptar y botón "cancelar" respectivamente.

Debemos conectar las señales de los botones con los slots indicados. Ojo! los Slots son del diálogo... del diálogo que estamos programando ahora mismo. Es decir, "this" es el objeto que recibe la señal y tiene el slot




| dtransferencia.cpp | Realizamos la conexión necesaria en lso botones canclear y aceptar |
|---|---|

```
connect( bAceptar  , SIGNAL (clicked() ),
      this , SLOT( accept()));

 connect( bCancelar  , SIGNAL (clicked() ),
      this , SLOT( reject()));
```

De esta forma cuando se pulse el botón "Aceptar", se dispara la señal "clicked()", que está conectada al slot "accept()" del diálogo, que desencadena que el diálogo se cierre. Con el botón "Cancelar" ocurre lo mismo, pero el diálogo se rechaza. En ambos casos, se cierra el diálogo pero no desaparece, sigue existiendo aunque no se vé.  Más adelante se

| dtransferencia.h | |
|---|---|

Código completo:

dtransferencia.h | Fichero completo

```
#include <QDialog>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QLabel>
#include <QLineEdit>


class DTransferencia : public QDialog {

Q_OBJECT

public:
/* layouts */
    QVBoxLayout *lyDer, *lyIzq;
    QHBoxLayout *lyPrincipal, *lyDestino, *lyCantidad;
/* Componentes */
    QPushButton *bAceptar, *bCancelar;
    QLabel *lDestino, *lCantidad, *lComision;
    QLineEdit *leDestino ,  *leCantidad;

    DTransferencia(QWidget * parent = 0) ;

public slots:
   void slotActualizarBoton(const QString &);
   void slotActualizarComision(const QString &);

};
```

dtransferencia.cpp | fichero completo

```
#include "dtransferencia.h"
#include <QLineEdit>
#include <QRegularExpression>
#include <QRegularExpressionMatch>

#include <QDebug>

DTransferencia::DTransferencia(QWidget * parent) : QDialog(parent)
{

 lyDer = new QVBoxLayout;
 lyIzq = new QVBoxLayout;
```

```
 lyPrincipal = new     QHBoxLayout;

 lyDestino = new     QHBoxLayout;
 lyCantidad = new     QHBoxLayout;
/* Componentes */
 bAceptar = new QPushButton("Aceptar");
 bAceptar->setEnabled(false);

 bCancelar = new QPushButton("Cancelar");
 lDestino = new QLabel("Cuenta destino:");
 lCantidad = new QLabel ("Cantidad");
 lComision = new QLabel ("Comisión 0 euros");

    leDestino = new QLineEdit;
    leCantidad = new QLineEdit;

 lyDer->addWidget(bAceptar);
 lyDer->addWidget(bCancelar);
 lyDer->addStretch();

 lyDestino->addWidget(lDestino);
 lyDestino->addWidget(leDestino);

 lyCantidad->addWidget(lCantidad);
 lyCantidad->addWidget(leCantidad);

 lyIzq->addLayout(lyDestino);
 lyIzq->addLayout(lyCantidad);
 lyIzq->addWidget(lComision);
 lyIzq->addStretch();

 lyPrincipal->addLayout(lyIzq);
 lyPrincipal->addLayout(lyDer);

    setLayout(lyPrincipal);

 connect(leDestino,SIGNAL(textChanged(const QString &)),
       this, SLOT(slotActualizarBoton(const QString &)));

 connect(leCantidad,SIGNAL(textChanged(const QString &)),
       this, SLOT(slotActualizarBoton(const QString &)));

 connect(leCantidad,SIGNAL(textChanged(const QString &)),
       this, SLOT(slotActualizarComision(const QString &)));

connect( bAceptar  , SIGNAL (clicked() ),
       this , SLOT( accept()));
```

```
 connect( bCancelar  , SIGNAL (clicked() ),
       this , SLOT( reject()));

    resize(400,100);
}


void DTransferencia::slotActualizarBoton(const QString & cad){

    QString cadDestino = leDestino->text();
    QRegularExpression reDestino("^\\d{6}$");

    QRegularExpressionMatch mDestino =
reDestino.match(cadDestino);


    QString cadCantidad = leCantidad->text();
    QRegularExpression reCantidad("^\\d+$");

    QRegularExpressionMatch mCantidad =
reCantidad.match(cadCantidad);

    bool destinoOK = mDestino.hasMatch();
    bool cantidadOK = mCantidad.hasMatch();

    if (destinoOK && cantidadOK)
      bAceptar->setEnabled(true);
    else
      bAceptar->setEnabled(false);
    qDebug() << "slotActualizarBoton ejecutándose " << cad;
}

void DTransferencia::slotActualizarComision(const QString &cad){

    QString cadena = leCantidad->text();
    float cantidad = cadena.toFloat();

    QString resultado("La comision es de ");
    resultado += QString::number(cantidad*0.01);
    resultado += " euros";

    lComision->setText(resultado);


    resultado = "La cantidad es de " + cadena ;

   qDebug() << resultado;
}
```

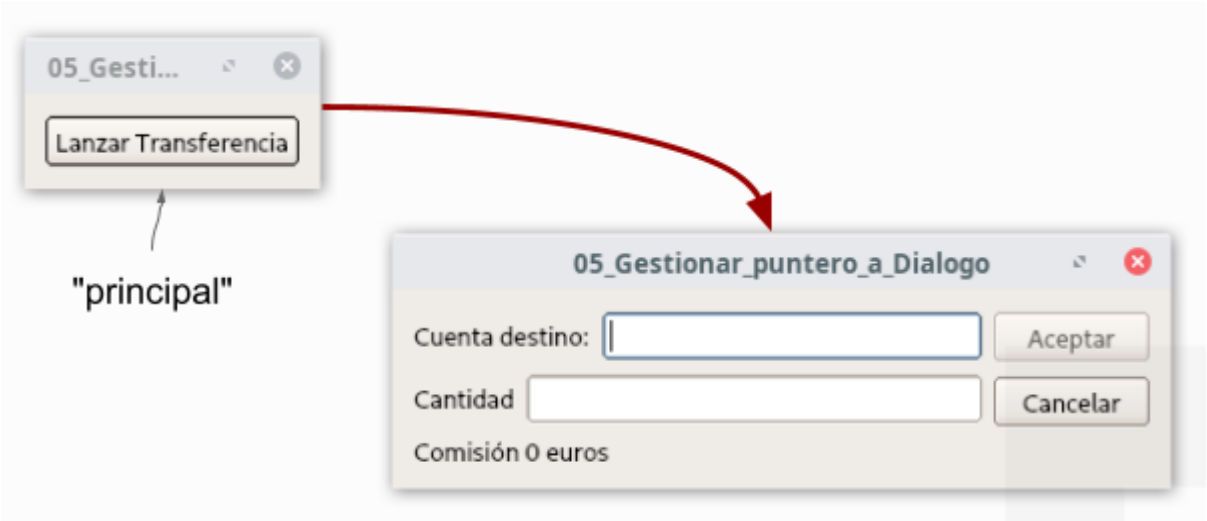
```
/*
connect(leCantidad,SIGNAL(textChanged(const QString &)),
       this, SLOT(slotActualizarBoton(const QString &)));

connect(leCantidad,SIGNAL(textChanged(const QString &)),
       this, SLOT(slotActualizarComision(const QString &)));

    */
```

main.cpp

```
#include <QApplication>

 #include "dtransferencia.h"

 int main(int argc, char *argv[])
 {
      QApplication app(argc, argv);
      DTransferencia * dTransferencia = new DTransferencia();
      dTransferencia->show();
      return app.exec();
 }
```

# Segundo diálogo

En este ejercicio vamos a aprovechar el diálogo creado anteriormente. El objetivo es lanzar ese diálogo desde otro diálogo anterior. Llamaremos a este nuevo diálogo "principal" porque aparece inicialmente y desde él se lanza el diálogo DTransferencia anterior.




Se pide que si el diálogo ya se ha lanzado desde el botón, al volver a ser pulsado no se lance un nuevo diálogo, y si el diálogo DTransferencia ha sido ocultado, entonces vuelva a hacerse visible al pulsar el botón, pero no se cree internamente uno nuevo.

## Solución.

Lo más novedoso de esta situación, es que el diálogo principal, debe conocer y controlar un objeto de tipo DTransferencia. El primer paso es hacer el include en el fichero de cabecera:

dprincipal.h

```
#ifndef DPRINCIPAL_H
#define DPRINCIPAL_H

#include <QDialog>
#include <QPushButton>
#include "dtransferencia.h"

class DPrincipal : public QDialog {
```

Seguidamente hemos de controlar el diálogo que se creará con un puntero. Ese puntero lo declaramos aquí porque va a ser accedido en distintos métodos de esta clase a lo largo de los ejercicios.

dprincipal.h

```
#ifndef DPRINCIPAL_H
#define DPRINCIPAL_H
```

```
#include <QDialog>
#include <QPushButton>
#include "dtransferencia.h"

class DPrincipal : public QDialog {
Q_OBJECT
public:
  DPrincipal(QWidget * parent = 0) ;

  DTransferencia *dTransferencia;

};
#endif
```

Seguidamente creamos el puntero al botón y la declaración del slot al que conectaremos el botón

dprincipal.h

```
#ifndef DPRINCIPAL_H
#define DPRINCIPAL_H

#include <QDialog>
#include <QPushButton>
#include "dtransferencia.h"

class DPrincipal : public QDialog {
Q_OBJECT
public:
  DPrincipal(QWidget * parent = 0) ;

  DTransferencia *dTransferencia;
  QPushButton *bLanzar;

public slots:
  void slotLanzarDialogoTransferencia();
};
#endif
```

El botón (y un layout que se hace necesario, como siempre) se crean en el constructor, se disponen mediante addWidget y se usan para mostrarse en el diálogo con setLayout()

(recuerda que setLayout() es un método del presente diálogo que permite colocar contenido dentro del área visible del diálogo)

| dprincipal.cpp | Creamos lo necesario para mostrar el botón |
|---|---|

```
DPrincipal::DPrincipal(QWidget * parent ) : QDialog(parent) {

    QHBoxLayout *lyPrincipal = new QHBoxLayout();
    bLanzar = new QPushButton("Lanzar Transferencia");
    lyPrincipal->addWidget(bLanzar);
    setLayout(lyPrincipal);
```

El botón lo conectamos con el slot que hemos creado

| dprincipal.cpp | creamos la conexión para lanzar el slot al pulsar el botón |
|---|---|

```
DPrincipal::DPrincipal(QWidget * parent ) : QDialog(parent) {

    QHBoxLayout *lyPrincipal = new QHBoxLayout();
    bLanzar = new QPushButton("Lanzar Transferencia");
    lyPrincipal->addWidget(bLanzar);
       setLayout(lyPrincipal);

    connect(bLanzar,SIGNAL(clicked()),this,
                 SLOT(slotLanzarDialogoTransferencia()));
```

Finalmente, en el constructor, hemos de poner a NULL el puntero al diálogo de tipo DTransferencia que se creará. Esto es más importante de lo que puede parecer, ya que más tarde aprovechamos que el puntero apunta a NULL para detectar que no se ha creado todavía el diálogo

| dprincipal.cpp | creamos la conexión para lanzar el slot al pulsar el botón |
|---|---|

```
DPrincipal::DPrincipal(QWidget * parent ) : QDialog(parent) {

    QHBoxLayout *lyPrincipal = new QHBoxLayout();
    bLanzar = new QPushButton("Lanzar Transferencia");
    lyPrincipal->addWidget(bLanzar);

    connect(bLanzar,SIGNAL(clicked()),this,
                 SLOT(slotLanzarDialogoTransferencia()));

    dTransferencia = NULL;
    setLayout(lyPrincipal);
```

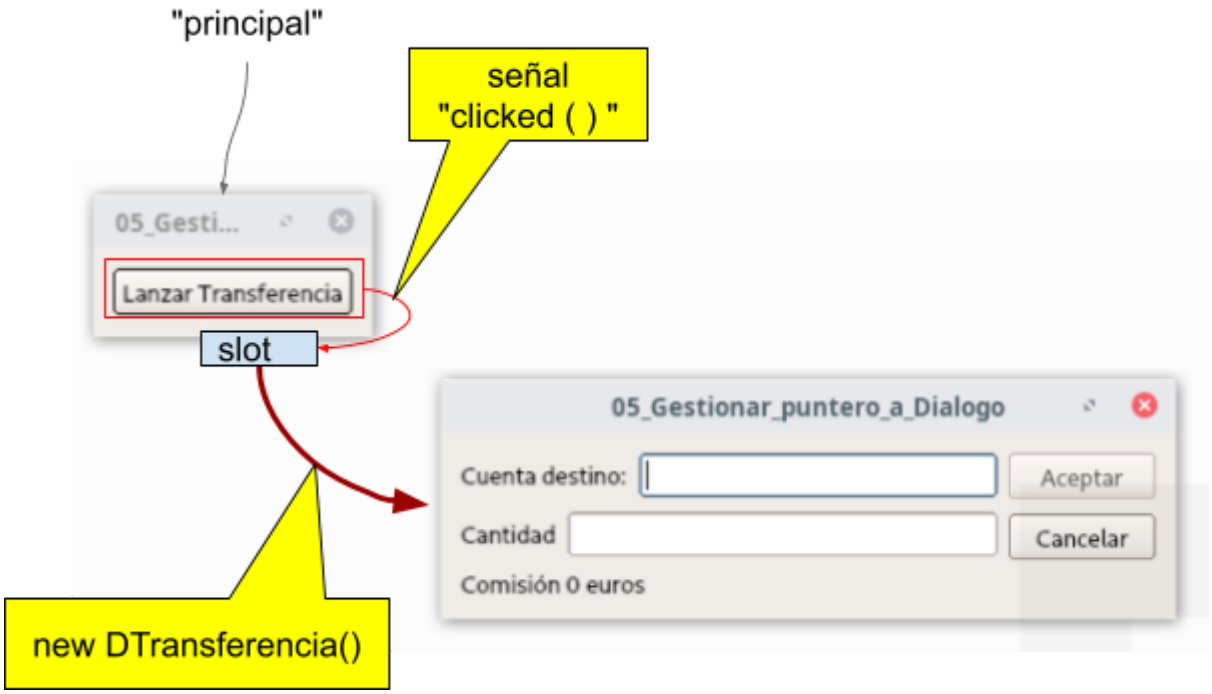
```
}
```

Vamos con el slot. Recuerda que esto se ejecuta cuando se pulsa el botón y lo que deseamos en ese caso, es crear y mostrar un diálogo de transfernecia:

| dprincipal.cpp | En principio deberíamos crerar el diálogo y mostrarlo |
|---|---|

```
void DPrincipal::slotLanzarDialogoTransferencia(){
    dTransferencia = new DTransferencia();
    dTransferencia->show();
}
```

El esquema aproximado de lo que está ocurriendo puedes verlo en el siguiente diagrama




Ten en cuenta que el botón puede ser pulsado muchas veces y tal cual hemos implementado el slot, se crea un nuevo diálogo cada vez. Esto no es deseable, no debemos crear un nuevo diálogo si ya lo hemos creado antes. Por suerte, al crear el díalgoo la primera vez, hemos almaceando un valor diferente a NULL en el puntero, y esto nos permite descubrir que ya existe el diálogo:

| dprincipal.cpp | |
|---|---|
| | |

```
void DPrincipal::slotLanzarDialogoTransferencia(){

    if ( dTransferencia == NULL  )
       dTransferencia = new DTransferencia();

    dTransferencia->show();

}
```

Fíjate en el slot anterior, que la primera vez que se ejecute, dTransferencia vale NULL y por tanto se ejecuta el if y con ello se crea un diálogo, pero las siguientes veces sólo se ejecuta ->show()

Ya está todo... o casi todo, ¡¡ Queda lo más importante pero fácil !! Lanzar este diálogo principal desde el main... esto ya deberías saberlo hacer, tan sólo deberás cambiar el diálogo que allí se lanza, antes era DTransferencia y ahora DPrincipal

main.cpp

```
#include <QApplication>
#include "dprincipal.h"
#include "dtransferencia.h"

 int main(int argc, char *argv[])
 {
     QApplication app(argc, argv);
          DPrincipal * dPrincipal = new DPrincipal();
          dPrincipal->show();

     return app.exec();
 }
```

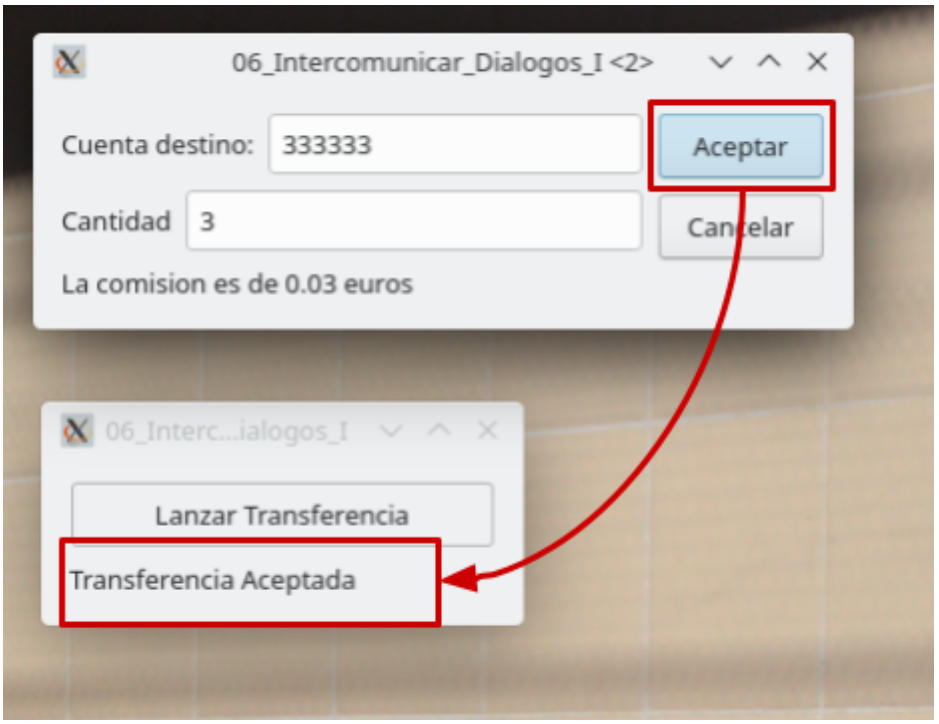
## Intercomunicar dos diálogos

El diálogo principal sólo tiene un botón y una vez éste lanza el diálogo de la transferencia se desentiende bastante de él. Sin embargo, habitualmente, cuando un diálogo se cierra, el programa principal o el diálogo que lo lanza toma nota y actúa en consecuencia. en el caso de este diálogo de transferencia, podríamos pensar que si:

- El diálogo se cierra mediante el botón "aceptar", la transferencia debe hacerse y debe informarse al usuario
- El diálogo se cancela usando el botón de cancelar, el programa debe informar de que se ha cancelado o rechazado la transferencia.

En la siguiente imagen se muestra el efecto perseguido y también el nuevo aspecto del diálogo principal.

Cuando se pulse Aceptar, el diálogo inicial debe mostrar (en una QLabel) el texto "Transferencia Aceptada"




Vamos a modificar el diálogo principal (y también el diálogo de transferencias)

Empezamos por lo más fácil: dotar de un nuevo aspecto al diálogo principal, que ahora tiene un botón y una etiqueta:

dprincipal.cpp

```
DPrincipal::DPrincipal(QWidget * parent ) : QDialog(parent) {

    QHBoxLayout *lyPrincipal = new QHBoxLayout();
    bLanzar = new QPushButton("Lanzar Transferencia");
    lEstadoTransferencia = new QLabel("Listo para lanzar una
transferencia");

    QVBoxLayout *lyLanzar = new QVBoxLayout();
    lyLanzar->addWidget(bLanzar);
    lyLanzar->addWidget(lEstadoTransferencia);

    lyPrincipal->addLayout(lyLanzar);

    connect(bLanzar,SIGNAL(clicked()),this,
                SLOT(slotLanzarDialogoTransferencia()));

    dTransferencia = NULL;
    setLayout(lyPrincipal);
```

```
}
```

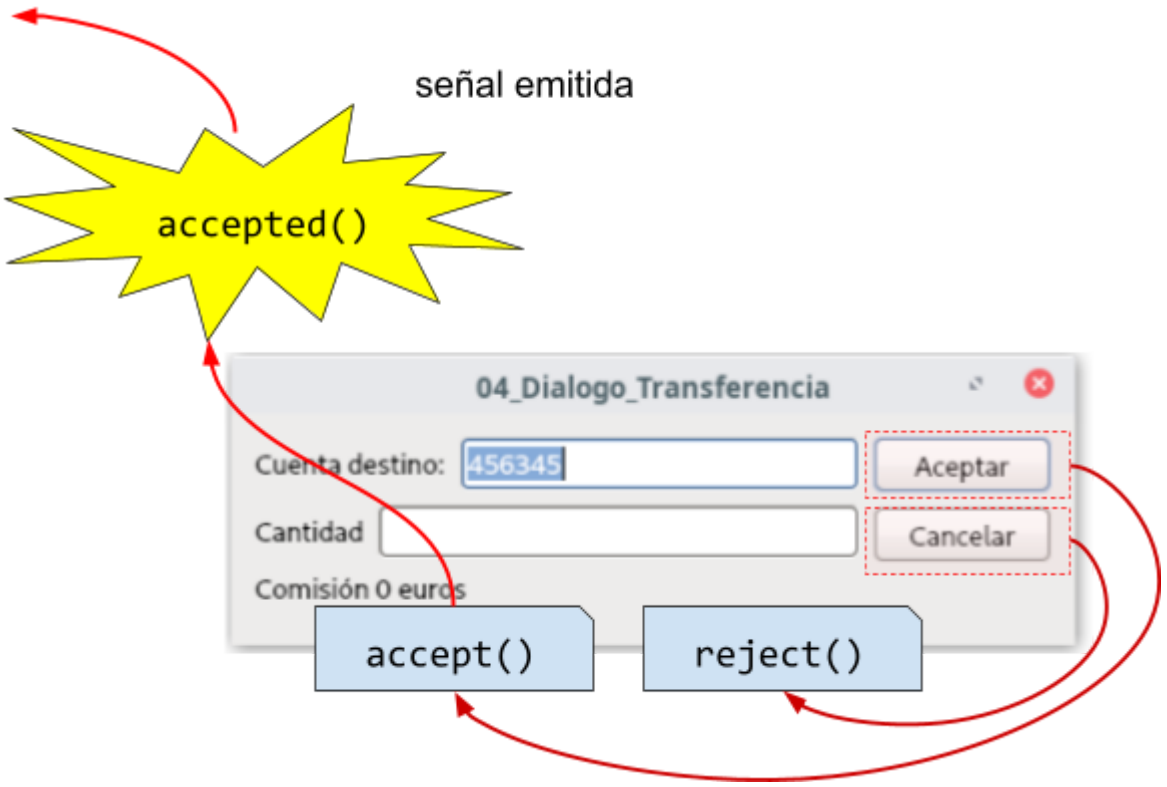
El siguiente paso, por ser fácil, va  a ser crear un slot que actualice el texto de la etiqueta. Observa que el slot no recoge ningún argumento, más tarde verás por qué cuando lo conectemos. De momento asume que no hay que recoger nada, el enunciado dice que se escriba "Transferencia aceptada" y ya está

dprincipal.cpp

```
void DPrincipal::slotAceptadaTransferencia(){
     lEstadoTransferencia->setText("Transferencia Aceptada");

}
```

La declaración la haces tú en el .h

Para seguir, es necesario comprender mejor cómo funciona un diálogo. si vas a la documentación, observarás que también tiene señales, y entre otras, hay dos señales lamadas "accepted()" y "rejected()". Estas señales se emiten cuando el diálogo es aceptado o rechazado (no cuando se pulsa el botón "Aceptar", sino, en nuestro caso, como consecuencia final de pulsar el botón "Aceptar", que está conectado con el slot "Aceptar"...¡¡ Huuuuy !! ¡¡ Qué Lío !!



Ahora vamos a combinar dos ideas:

- Hay algo en lo que se debe insistir: En todo momento hay que conocer qué elementos dominan o conocen a qué otros elementos. En nuestro caso , el diálogo principal es el que crea y por tanto conoce al diálogo de la transferencia.
- El diálogo de la transferencia emite señales y una de ellas interesa al diálogo principal.

Estas dos ideas se materializan en que el diálogo princpal conectará la señal "accepted()" del diálogo de la transferencia con un slot propio . De esta forma , el díalogo principla, logra reaccionar al hecho de que el diálogo de la transferencia sea aceptado.

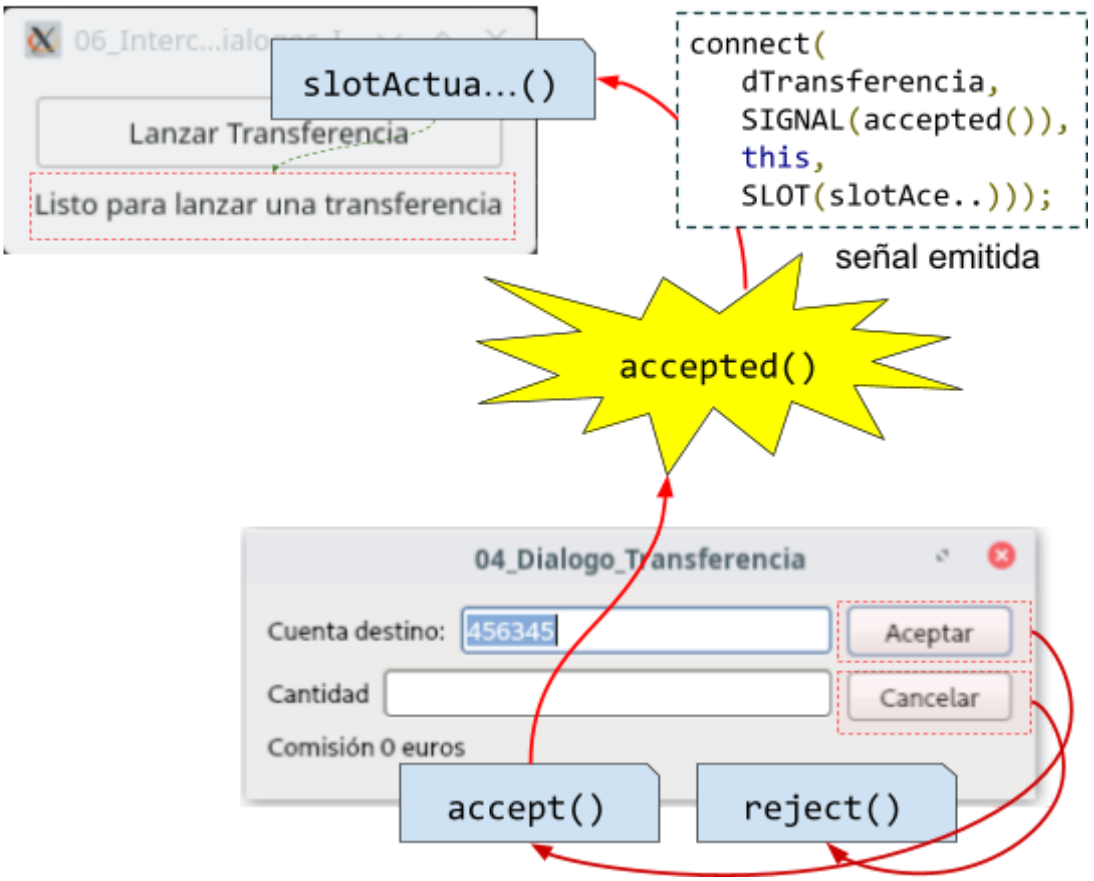
Esto sólo se puede hacer en el momento en el que se crea el diálogo de transfernecia (y no antes , en el constructor de dprincipal, en ese momento no existe todavía el diálogo de la transfernecia)

dprincipal.cpp

```
void DPrincipal::slotLanzarDialogoTransferencia(){

      if ( dTransferencia == NULL  ) {
            dTransferencia = new DTransferencia();
            connect(dTransferencia,SIGNAL(accepted()),
                        this,SLOT(slotAceptadaTransferencia()));
            }

      dTransferencia->show();
}
```

Es decir, la señal acept del diálogo dTrans, activará el slot slotAceptadaTransfernecia() del diálogo principal. Dado que la señal no lleva argumentos, el slot tampoco los debe llevar.

Al final tenemos un montaje algo complejo, que desede el botón de "Aceptar" logra cambiar el texto de la etiqueta Resultado. La siguiente imagen esquematiza esto




| | |
|---|---|
| | |

| | |
|---|---|
| | |

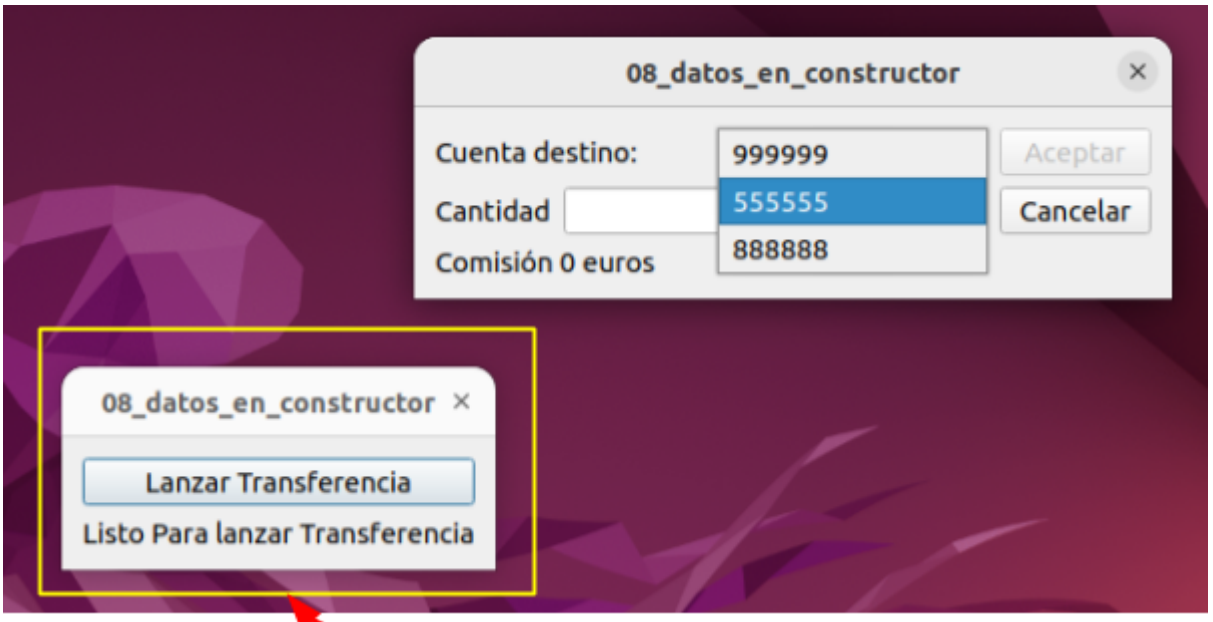
## Pasar datos entre diálogos.

Para el siguiente paso es necesario imaginar un escenario concreto, algo forzado y artificial, pero simple, y necesario para seguir el aprendizaje en orden.

Vamos a suponer que sólo se puede elegir una cuenta concreta entre unas posibles. Estas cuentas se desplegarán en un desplegable que ocupará el lugar de la anterior entrada de texto que habíamos hecho hasta ahora. Todo esto es visible en la captura del diálogo de transferencia de la captura siguiente.

Sin embargo, estas cuentas posibles no las sabe el diálogo DTransferencias, éste tan sólo recibe esa información desde el diálogo principal. Es decir, las cuentas están registradas inicialmente en el diálogo principal, pero de alguna forma, terminan siendo conocidas desde el diálogo de la transferencia. Esto supone el objetivo en este ejercicio.

Inicialmente las cuentas estarán almacenadas como una QStringList, creada en la clase DPrincipal:




```
QStringList listacuentas = {"999999",
                            "555555",
                            "888888"};

// da error pero vale para seguir
```

Empecemos modificando la clase DPrincipal para añadir el almacén de cuentas registradas. Por una parte declaramos el atributo en el fichero de cabecera, dentro de la clase

| dprincipal.h | |
|---|---|

```
#ifndef DPRINCIPAL_H
...

class DPrincipal : public QDialog {
...
      QStringList listaCuentas ;
...
```

```
};
#endif
```

No es un puntero , porque no hace falta que sea un puntero. Son punteros las variables que necesitan serlo por algún motivo, pero ahora no lo tenemos. Recuerda: En Java TODO son punteros.

Esta lista se debe inicializar con los valores de ejemplo que queramos. Esto lo vamos a hacer en el constructor de la clase y no en la declaración de la misma (es decir, no en el .h). Y además lo vamos a hacer de una forma compacta aprovechando una especie de licencia que permite el compilador para inicializar objetos en el momento de declararlos

dprincipal.cpp

```
DPrincipal::DPrincipal(QWidget * parent ) : QDialog(parent) {

    ...

    QStringList l = {"999999", "555555","888888"};
    listaCuentas = l;
...
```

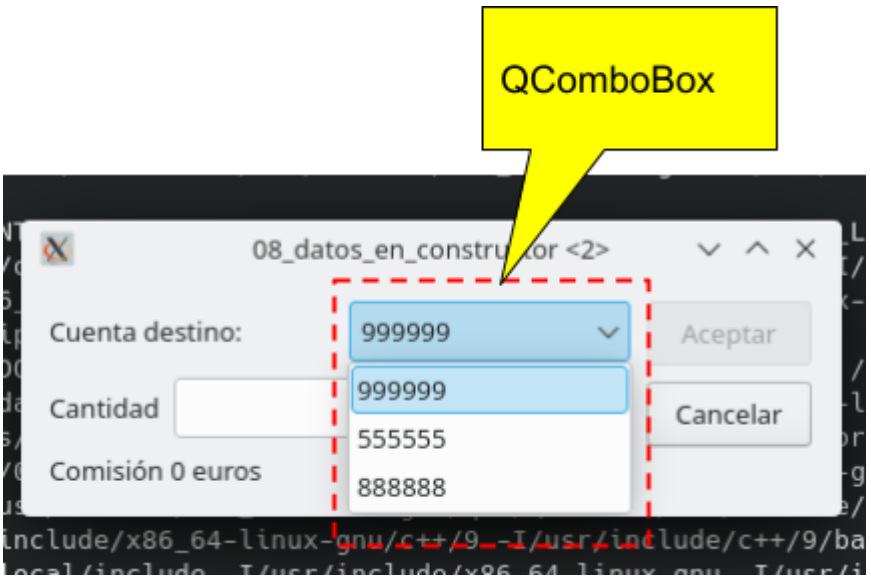
Ya no hay más modificaciones en la clase DPrincipal (en la clase propiamente, sí en el código que crea un objeto DTransferencia)

La clave que debemos responder es "¿Cómo le pasa el diálogo principal este QStringList al diálogo de la transferencia?"

Esta pregunta tiene varias respuestas que iremos viendo a lo largo del curso en distintos ejercicios. Ahora vamos a darle solución de una forma:  pasando en el constructor la información necesaria.

El constructor o constructores de una clase puede cambiarse y decidirse según nuestras necesidades, como es el caso ahora: necesitamos que la clase DTransferencia reciba de alguna forma un QStringList y esa forma es en el constructor.

Por ello cambiamos o añadimos (no aclaramos esta cuestión) el constructor. También añadimos un QComboBox en la clase




| DTransferencia.h | |
|---|---|

```
#ifndef DTRANSFERENCIA_H
...
class DTransferencia : public QDialog {

Q_OBJECT

public:
    QComboBox *cmbDestino;
    DTransferencia(QStringList, QWidget * parent = 0) ;
...
};
#endif
```

Nota: Dado que el parámetro QWidget * parent es opcional (tiene inicialización por defecto), el compilador obliga a que cualquier parámetro obligatorio esté delante de cualquier opcional. En definitiva: no podríamos cambiar el orden de los argumentos.

Nota: Podríamos también crear un atributo en esta clase para almacenar la lista de cadenas de forma similar a como hicimos con DPrincipal. Pero no es necesario en este caso

Vamos a implementar el constructor y poblar el comboBox con las cadenas procedentes del QStringList que es pasado al constructor

| DTransferencia.cpp | |
|---|---|

```
DTransferencia::DTransferencia(QStringList listaCuentas,
                               QWidget * parent) : QDialog(parent) {
...

 cmbDestino = new QComboBox();

 for (int i=0; i < listaCuentas.size();i++)
      cmbDestino->addItem(listaCuentas.at(i));
...
 lyDestino->addWidget(cmbDestino);
...
```

Ese bucle for se puede sustituir por otro método de la clase QComboBox, lo sé, pero vamos a mantener esto porque enseña más cómo recorrer un QStringList.

Ahora falta volver a la clase DPrincipal y crear el diálogo de transferencia usando el nuevo constructor:

dprincipal.cpp

```
void DPrincipal::slotLanzarDialogoTransferencia(){
    if ( dTransferencia == NULL  ) {

      dTransferencia = new DTransferencia(listaCuentas);

      connect(dTransferencia,SIGNAL(accepted()),
                  this, SLOT(slotDialogoAceptado()));
      }
    dTransferencia->show();
}
```

Observa que le pasamos el atributo listaCuentas donde están las cadenas numéricas que son las cuentas admisibles.

# Borrado de Diálogos

Aviso: esta sección trata un experimento breve que va a salir mal, pero con ello aprendemos más

Situacioń de partida: Deseamos eliminar (borrar, destruir) el diálogo Transferencia, una vez se ha cerrado al ser aceptado (o cancelado, pero este caso no lo vemos)

La destrucción o borrado del diálogo es la acción opuesta a la línea:

```
dTransferencia = new DTransferencia(listaCuentas);
```

Si arriba estamos creando un objeto (con **new**) en la siguiente línea estamos destruyendo y borrando de la memoria dicho objeto:

```
delete dTransferencia;
```

Esto, sólo podemos hacerlo desde el diálogo principal, el diálogo de transferencia no puede autodestruirse cuando se cierra.

Tal como hemos dicho, esto debe ocurrir al ser aceptado el diálogo transferencia, por tanto, en el slot siguiente:

dprincipal.cpp

```
void DPrincipal::slotAceptadaTransferencia(){
	lEstadoTransferencia->setText("Transferencia Aceptada");

}
```

vamos a añadir las líneas necesarias (

dprincipal.cpp

```
void DPrincipal::slotAceptadaTransferencia(){
	lEstadoTransferencia->setText("Transferencia Aceptada");
	delete dTransferencia;
	dTransferencia = NULL;
}
```

Date cuenta de que el puntero debemos dejaro después de borrar con **delete** apuntando a **NULL** (que es el valor "seguro" cuando no existe diálogo)

Si pruebas esto, verás que el programa explota al ser cerrado el diálogo y aunque estás programando _bien_, ocurre que el Motor de QT y tú compartís el objeto. Y tú no puedes, a espaldas de Qt, borrarle un objeto que está manipulando con la confianza de que va a seguir existiendo.

La solución a esto no se trata ahora

# Transferir datos usando métodos

Volvamos al diálogo en el que transferíamos una lista de cadenas (números de cuenta ) al díalogo. Recuerda que en el constructor del diálogo Transfernecia, recibíamos la lista de cuentas , y por tanto al crear el diálogo se debía pasar esa lista:

Recordatorio: La declaración del constructor

DTransferencia.h

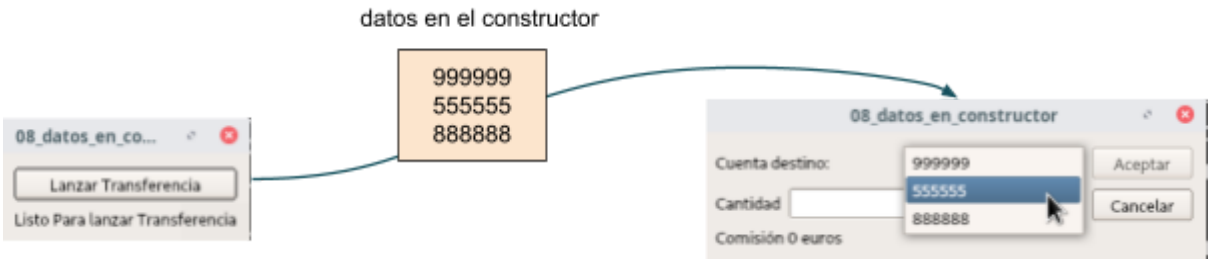
```
...
class DTransferencia : public QDialog {
...
    DTransferencia(QStringList, QWidget * parent = 0) ;
...
};
```

Recordatorio: La creación del diálogo

dprincipal.cpp

```
void DPrincipal::slotLanzarDialogoTransferencia(){
    ...

      dTransferencia = new DTransferencia(listaCuentas);

     ...
```

Si el constructor requiere datos, esos datos deben estar disponibles en el momento de la creación (al hacer new DTransferencia(...) )

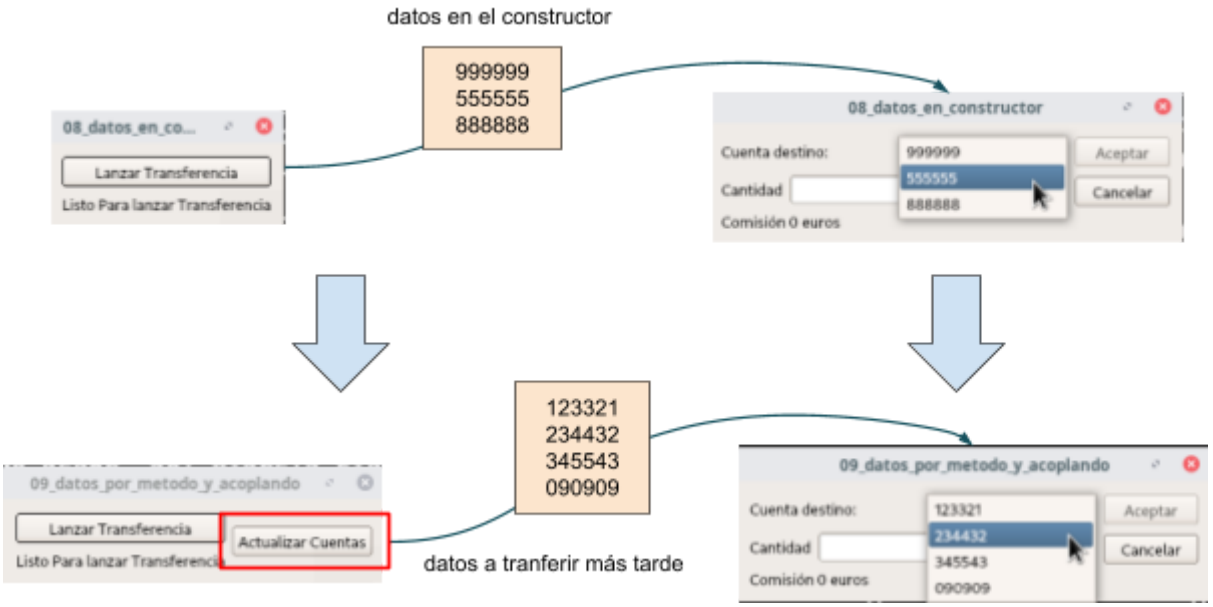



Pero ¿Qué pasaría si después de que el diálogo se cree y sea visible deben cambiarse los números de cuenta que muestra el diálogo de transferencia?

Supón que el diálogo va a estar mucho tiempo visible y se mantiene creado permitiendo varias operaciones (suponlo simplemente). Y los datos del banco cambian (suponlo, otra vez) de forma que algunas cuentas se borran y otras nuevas se crean.

Esa información debe actualizarse en el diálogo... después de que éste haya sido creado. (podrías pensar en solucionarlo cerrando y abriendo otro nuevo diálogo para recibir la información actualizada... pero no! así no).

Vamos a crear un nuevo botón en el diálogo principal que simulará la actualización de la información sobre cuentas disponibles en el diálogo de la transferencia




Lo que necesitamos es una nueva forma de transferir información desde el diálogo principal al de la transferencia. Soluciones para este problema hay varias, la primera que exploraremos es la más sensata y consiste en :

> Añadir un método a la clase DTransfernecia que permita actualizar la lista de cuentas mostrada en el comboBox

Este método estará disponible desde la clase principal, que otra vez, es la que almacena la lista de cuentas (la original y la nueva  a usar en la actualización)

Suponemos hecho el botón y el correspondiente slot en la clase principal, veámos el código del slot

| dprincipal.cpp | |
|---|---|

```
void DPrincipal::slotActualizarCuentas() {
     QStringList lista = {"123321", "234432","345543","090909"};

     dTransferencia->actualizarCuentas(lista);
```

```
}
```

Ese método actualizarCuentas, no está hecho todavía pero ya puedes ver cómo se va a usar desde fuera de la clase DTransferencia. (Existe una alternativa muy muy guarra que comentamos después)

Suponiendo que ya has escrito lo necesario en el fichero de cabecera, el método tendrá el siguiente código

dtransferencia.cpp

```
void DTransferencia::actualizarCuentas(QStringList lista){
   cmbDestino->clear();
     for (int i=0; i < lista.size();i++)
                cmbDestino->addItem(lista.at(i));

}
```

volvamos al diálogo principal para ver cómo podríamos lograr el mismo objetivo de la forma más cerda posible. La idea es acceder directamente al ComboBox... ¡ desde la clase principal!! sin métodos en la clase DTransferencia

dprincipal.cpp

```
void DPrincipal::slotActualizarCuentas() {
    QStringList lista = {"123321", "234432","345543","090909"};

    dTransferencia->cmbDestino->clear();
    for (int i=0; i < lista.size();i++)
          dTransferencia->cmbDestino->addItem(lista.at(i));
     }
}
```

Digamos que lo que se hacía desde el método DTransferencia::actualizarCuentas ahora lo hacemos directamente desde aquí. Esto es posible ya que:

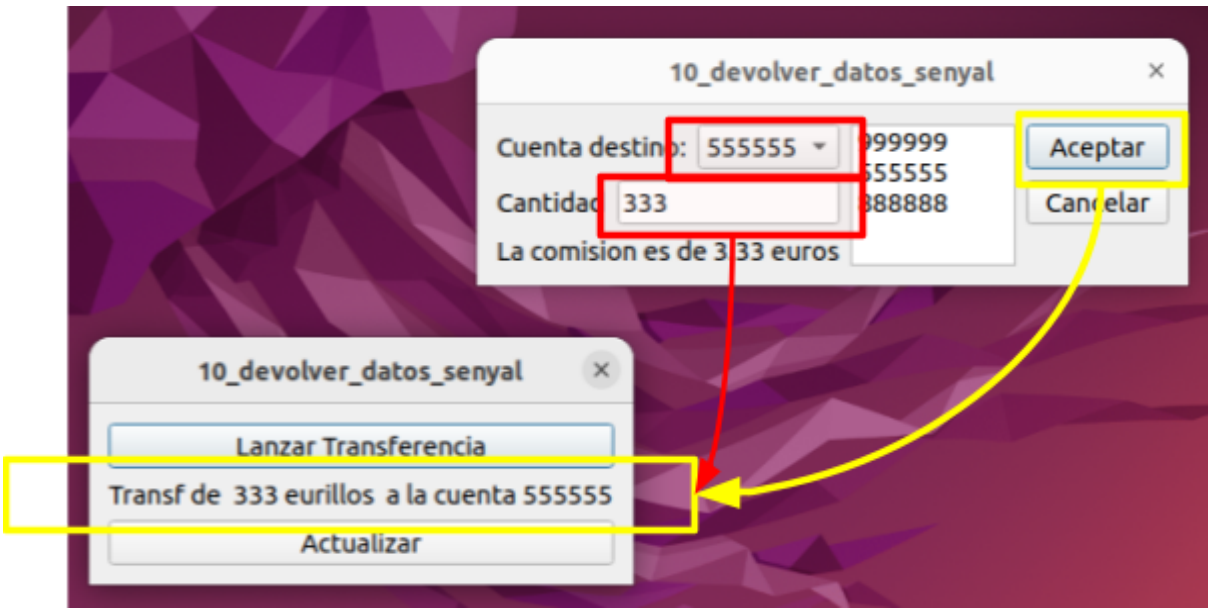
1. Tenemos un puntero que apunta al diálogo (dTransferencia)
2. Todos los atributos allí declarados son públicos
3. Somos muy guarros

¡Ojito! Si pruebas el programa y se te ocurre pinchar primero al botón de actualizar cuentas, vas a provocar un error gravísimo. Descúbrelo tú, diagnostícalo, y soluciónalo.

| | |
|---|---|
| | |

# Devolver datos con señales

Ahora vamos a ampliar nuestros objetivos. Deseamos que al Aceptar el diálogo de la transferencia y éste cerrarse, aparezca la información de la transferencia ordenada en el diálogo principal.




Esto es bastante serio, ya que desde el diálogo principal, podemos fácilmente acceder y gestionar el diálogo Transferencia, pero al revés no es fácil. El diálogo de transferencia no tienen ni idea de la existencia del diálogo principal (no hay un puntero llamado, por ejemplo, "dPrincipal" en el diálogo de la transferencia con el que cambiar cosas en el diálogo principal)

Hay varias formas de resolver esto, algunas muy guarras. Vamos a explorar dos que son limpias y dignas de un buen informático. La primera es usando señales y slots

Empecemos recordando lo que existe. Recuerda que en el diálogo principal, tenemos un slot propio, hecho por nosotros que se activa al pulsar el botón de aceptar. Así mismo, tenemos en el slot, la llamada a accept()  (método del diálogo). Al llamara a accept(), el diálogo emite una señal.

dtransferencia.cpp

```
connect(bAceptar,SIGNAL(clicked()),
                  this, SLOT(slotAceptar()));

...
void DTransferencia::slotAceptar() {

   accept();
}
```

La señal emitida por el diálogo transferencia es aprovechada desde el diálogo principal para detectar que la transferencia está ordenada. Hemos hecho un connect al crear el diálogo

dprincipal.cpp

```
void DPrincipal::slotLanzarDialogoTransferencia(){
    if ( dTransferencia == NULL  ) {
       dTransferencia = new DTransferencia(listaCuentas);

      connect(dTransferencia,SIGNAL(accepted()),
                    this, SLOT(slotDialogoAceptado()));

       }
}
```

Es decir, vinculamos el cierre del diálogo de la Transferencia (gracias a la señal accepted()) al slot siguiente

dprincipal.cpp

```
void DPrincipal::slotDialogoAceptado(){
    lEstado->setText("TRansferencia Realizada");
}
```

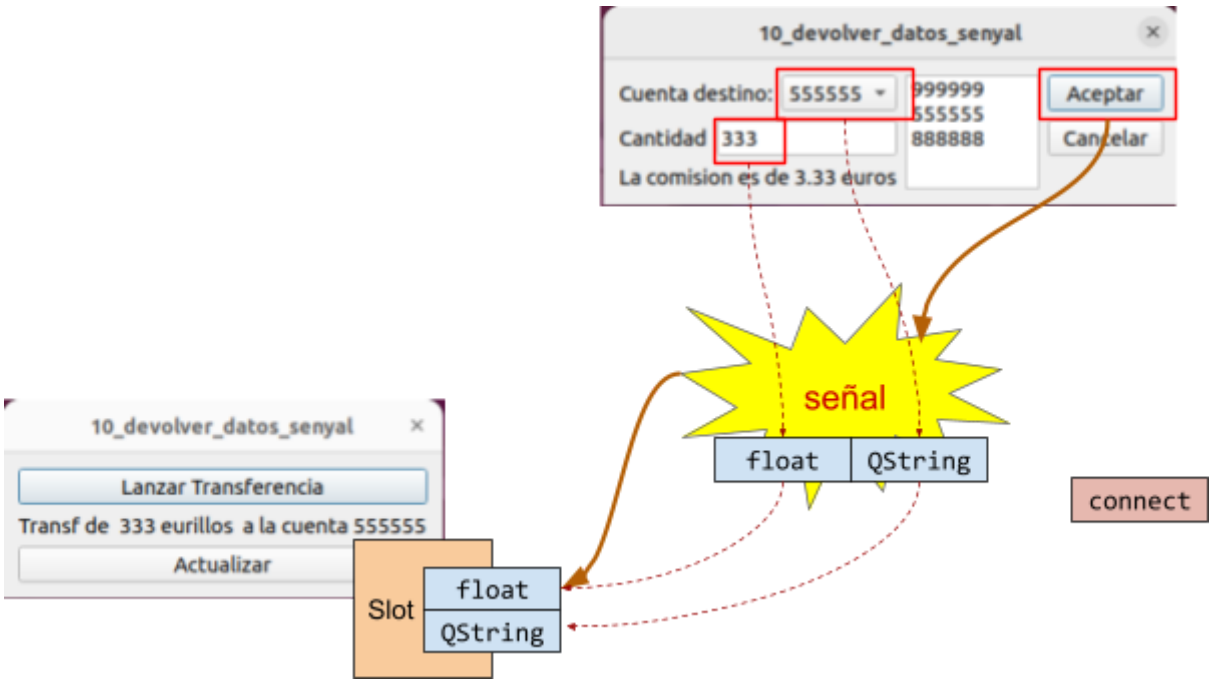
Pero ahora no queremos contentarnos con emitir un mensaje sin información detallada, ahora queremos mostrar el número de cuenta y cantidad. Lo que buscamos podríamos escribirlo algo más aproximadamente como sigue:

dprincipal.cpp

```
void DPrincipal::slotDialogoAceptado(){
    QString mensje = ???
    lEstado->setText(mensaje);
}
```

El problema es que la señal accepted() sólo avisa de que el diálogo se ha cerrado, pero ahora necesitaríamos una señal que además, nos pasase la cuenta y la cantidad introducidas. Esta señal sería emitida por el diálogo DTRansferencia y conllevaría dos argumentos (float y QString).

El slot adecuado en DPrincipal reemplazaría al anterior slotDialogoAceptado() en el sentido en que tendría esos dos argumentos . El siguiente diagrama muestra nuestras intenciones:




El slot del diálog principal podemos concretarlo ya

| dprincipal.cpp | |
|---|---|

```
void DPrincipal::slotDialogoAceptado(Qstring Cuenta, float
cantidad ){
    QString mensaje(" Transf de ");
    mensaje += QString::number(cantidad);
    mensaje += " a lacuenta " + cuenta ;
    lEstado->setText(mensaje);
}
```

Lo que vamos a hacer es **¡¡ Añadir una nueva señal al diálogo DTransferencia !!**

La señal se añade simplemente declarándola en el fichero de cabecera en la sección signals: (y no "public signals:")

dtransferencia.h

```
#ifndef DTRANSFERENCIA_H
#define DTRANSFERENCIA_H
...
class DTransferencia : public QDialog {

Q_OBJECT

public:
...
public slots:
...
signals:
        void senyalCompletaTransf(float,QString);
};
#endif
```

La señal NO hay que implementarla, tan sólo debemos decidir cuándo se emite. En nuestro caso, en el diálogo DPrincipal, en el slot donde se reaccionaba al botón de aceptar. Allí llamábamos al método accept() que a su vez emitía la señal accepted(). Ahora queremos que se emita esta nueva señal. Observe cómo se hace

dprincipal.cpp

```
void DTransferencia::slotAceptar() {
    float cantidad = leCantidad->text().toFloat();
    QString cuenta = cmbDestino->currentText();
    emit senyalCompletaTransf(cantidad,cuenta );
    accept();

}
```

Habiéndole dado al diálogo DTransferencia la capacidad de emitir nuevas señales, es hora de aprovecharlas e el diálogo principal para conectarlas con el nuevo slot (pensado para

esta señal. Lo hacemos concretamente aquí reemplazando el connect previo . Todo en DPrincipal

dprincipal.cpp

```
void DPrincipal::slotLanzarDialogoTransferencia(){
    if ( dTransferencia == NULL  ) {
       dTransferencia = new DTransferencia(listaCuentas);
       /*connect(dTransferencia,SIGNAL(accepted()),
                    this, SLOT(slotDialogoAceptado()));*/

    connect(dTransferencia,
            SIGNAL(senyalCompletaTransf(float,QString)),
            this,
            SLOT(slotTransfCompleta(float,QString)));
       }
    dTransferencia->show();
}
```

El slot es muy simple, toma los valores pasados de número de cuenta y cantidad, y los muestra en la etiqueta de estado:

```
void DPrincipal::slotTransfCompleta(float cant,QString cuen){

    slotTransfCantAccept(cant);
    lEstado->setText(lEstado->text() + " a la cuenta " + cuen );

}
```

# Retorno de valores por método.

Ahora, el objetivo va a ser el mismo del punto anterior, seguimos queriendo ver la información detallada de la transferencia en el diálogo principal. El cambio es que ahora no vamos a usar (tantas) señales y realizaremos esto de forma más **convencional**.

La idea es volver a tener el slot que teníamos y que reaccionaba a la aceptación y cierre del diálogo Transferencia, pero sin argumentos:

dprincipal.cpp

```
void DPrincipal::slotAceptadaTransferencia(){
      lEstadoTransferencia->setText("Transferencia Aceptada");

}
```

Con este slot, no se nos pasa información del número de cuenta ni de la cantidad (como antes). Pero ahora, vamos preguntarle al diálogo DTransferencia esa información llamando a un método que devuelva la información. Estos métodos conocidos como getters.

La idea es

dprincipal.cpp

```
void DPrincipal::slotAceptadaTransferencia(){

      float cant = dTransferencia->getCantidad();
      QString cuenta = dTransferencia->getCuenta();

    // ahora formar un mensaje a base de concatenar y convertir
    QString mensaje(" Transf de ");
    mensaje += QString::number(cantidad);
    mensaje += " a lacuenta " + cuenta ;

    //establecer el valor de la etiqueta
    lEstado->setText(mensaje);

}
```

Hay que implementar los métodos getters. Esto es fácil, impleméntalo tú.

Ahora si tienes lo que hay que tener, implementa esto:

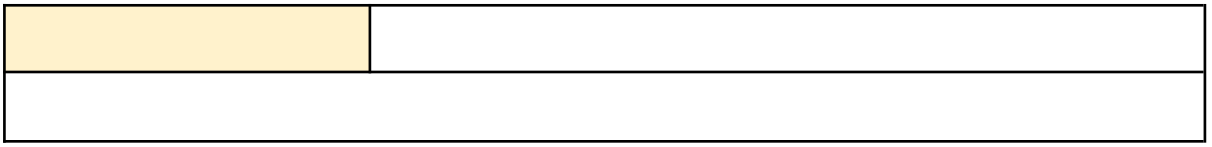
dprincipal.cpp

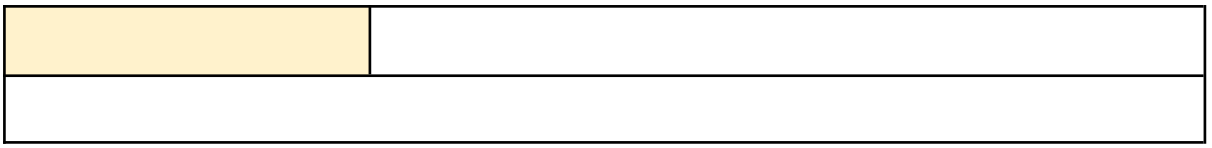
```
void DPrincipal::slotAceptadaTransferencia(){

      float cant ;
      QString cuenta ;

    if (! getInfo(cant,cuenta) ) {
       qDebug()<<"no se puede preguntar todavia ";
       return;
```

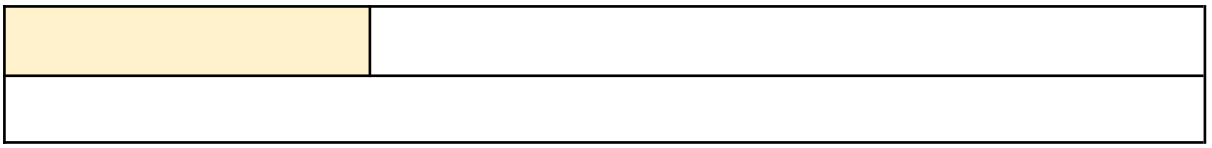
```
     }
     // ahora formar un mensaje a base de concatenar y convertir
    QString mensaje(" Transf de ");
    mensaje += QString::number(cantidad);
    mensaje += " a lacuenta " + cuenta ;

    //establecer el valor de la etiqueta
    lEstado->setText(mensaje);

}
```

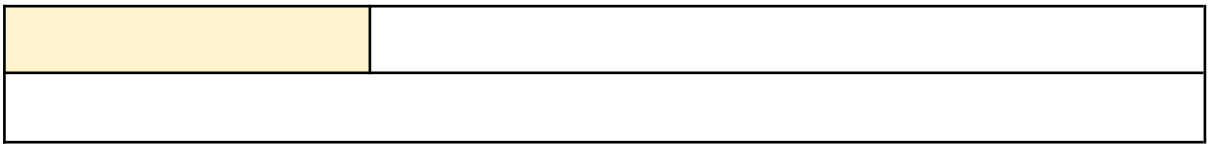
# Creación de componentes dinámicos

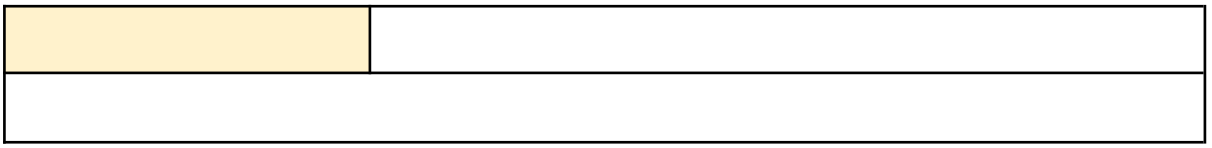
Supongamos el











| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |

| dtransferencia.h | |
|---|---|
| | |