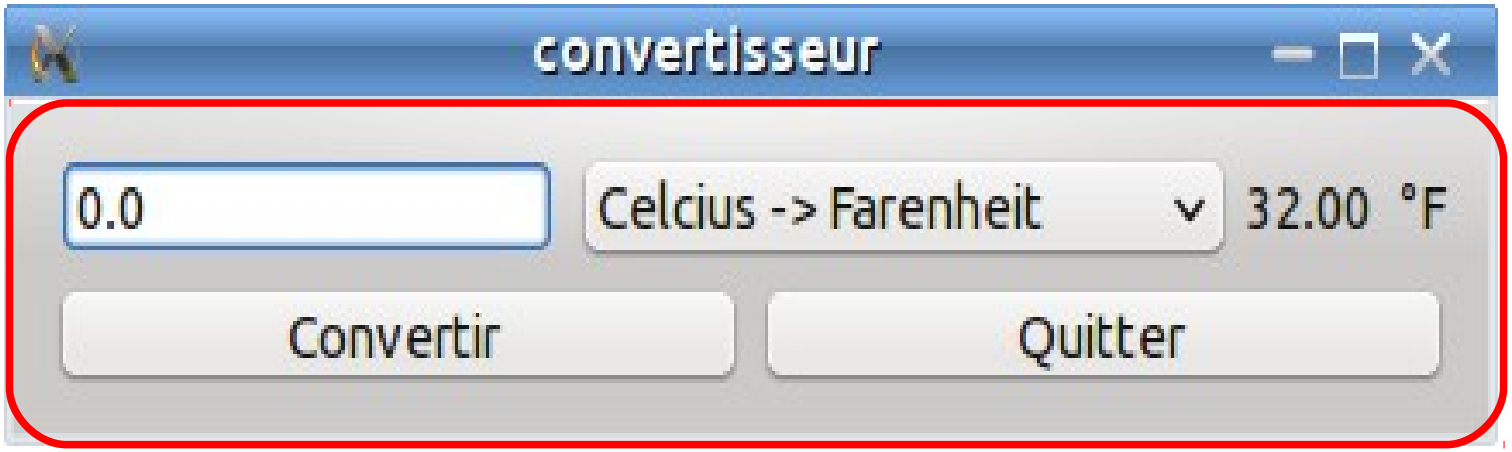
Première application - Principe n°1

- Créer son propre *widget* en créant une nouvelle classe **`MyWidget`** qui héritera de la classe **`QWidget`**.
- Une instance de cette classe représentera la fenêtre de l'application.
- *Rappel :* Un *widget* qui n'est pas incorporé dans un *widget* parent est appelé une fenêtre.



Un widget « fenêtre »

Première application - Principe n°2

- L'application « **convertisseur** » comprendra 3 fichiers :
  - **`main.cpp`** : l'instanciation des objets **`QApplication`** et **`MyWidget`**
  - **`mywidget.h`** : la déclaration de la classe **`MyWidget`**
  - **`mywidget.cpp`** : la définition de la classe **`MyWidget`**
- Le fichier de projet de l'application « **convertisseur** » sera donc :

convertisseur.pro

```
QT += widgets # qt5

TEMPLATE = app
TARGET = convertisseur

# Input
HEADERS += mywidget.h
SOURCES += main.cpp mywidget.cpp
```

Première application - Principe n°3

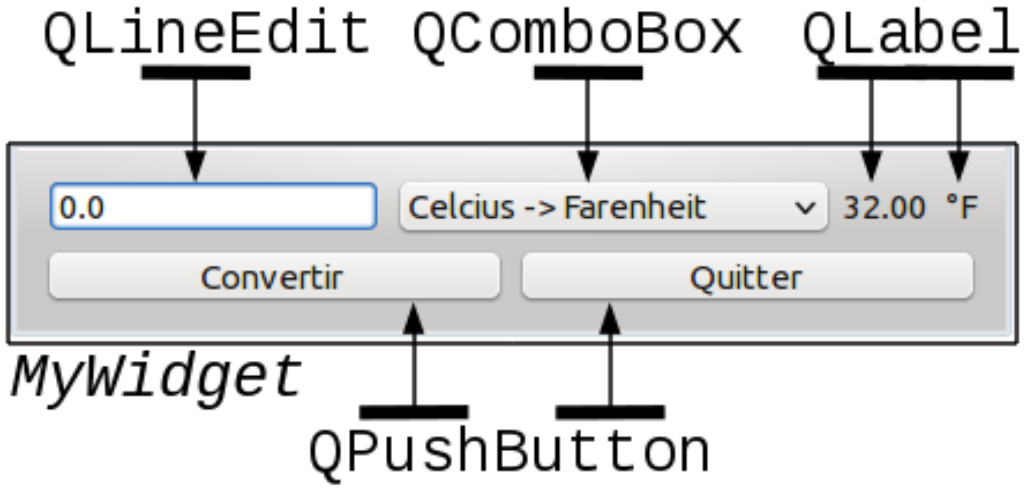
- Un *widget* est toujours créé caché, il est donc nécessaire d'appeler la méthode **show()** pour l'afficher :

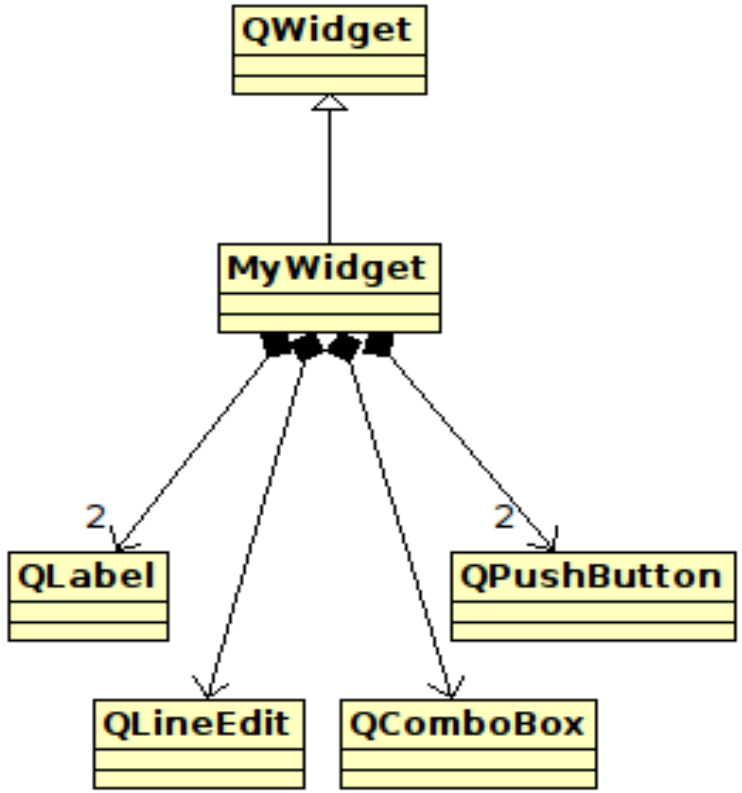
```
#include <QApplication>

#include "mywidget.h"

int main( int argc, char **argv )
{
   QApplication a (argc, argv);

   MyWidget w;

   w.show();

   return a.exec();
}
```

Qt fournit un objet global **qApp** pour accéder à votre objet Application.

Première application - Principe n°4

- L'instanciation des *widgets* (et leur positionnement) se fera dans le constructeur de la classe **`MyWidget`**





Première application - Principe n°5

```
#ifndef MYWIDGET_H
#define MYWIDGET_H
#include <QtWidgets> // ou <QtGui>
class MyWidget : public QWidget {
 Q_OBJECT
 public:
   MyWidget( QWidget *parent = 0 );
 private:
   QLineEdit   *valeur;
   QLabel      *resultat;
   QLabel      *unite;
   QComboBox   *choix;
   QPushButton *bConvertir;
   QPushButton *bQuitter;
 signals:
   void actualiser();
 private slots:
   void convertir();
   void permuter(int index);
};
#endif
```

- Pour créer son propre *widget*, il faut créer sa **propre classe** qui **hérite de QWidget**.

Pour accéder aux déclarations de tous les composants graphiques.

Avec cette **macro**, on précise que l'on doit utiliser le **moc** pour implanter le mécanisme ***signal/slot*** de Qt pour cette classe.

On peut **agréger d'autres *widgets*** (enfants) à son propre *widget*.

On peut déclarer ses propres **signaux**.

On peut déclarer ses propres **slots**.

Première application - Principe n°6

```
#include "mywidget.h"

// Constructeur
MyWidget::MyWidget(QWidget *parent)
: QWidget(parent)
{
  /* instancier les widgets */
  valeur = new QLineEdit(this);
  choix  = new QComboBox(this);
// …
  /* initialiser les widgets */
  valeur->clear();
  choix->addItem("Celcius -> Farenheit");
  choix->addItem("Farenheit -> Celcius");
// …
  /* positionner les widgets */
// …
  /* connecter signaux aux slots */
  connect(choix,
  SIGNAL(currentIndexChanged(int)),
  this, SLOT(permuter(int)));
// …
}
```

- Dans le **constructeur** :

Pour accéder à la déclaration de sa classe

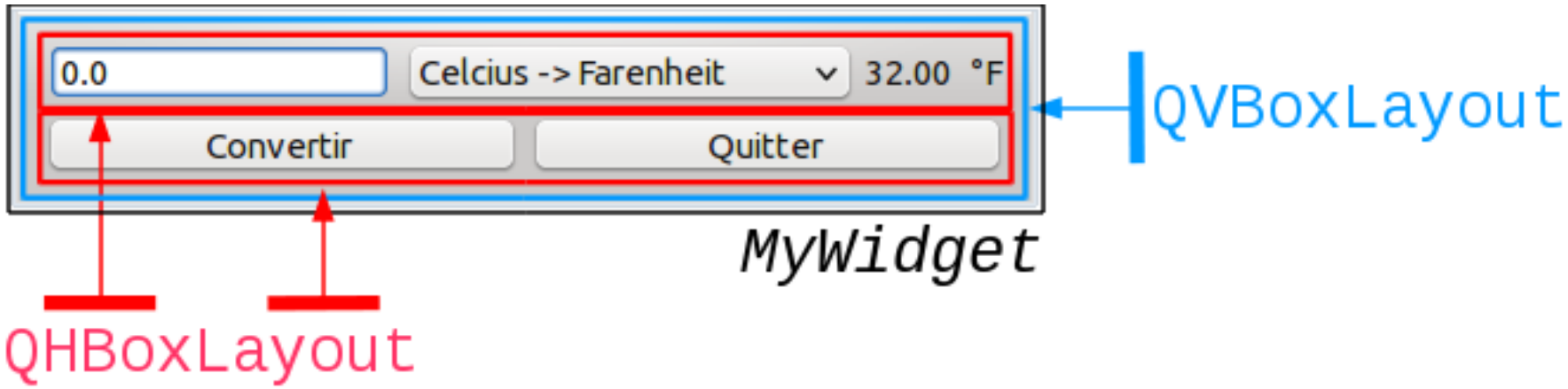
L'objet **valeur** sera « mon enfant »

Fournir leur affichage initial dans la fenêtre

cf. *Layouts*

Un changement de choix dans la liste déroulante provoquera l'exécution de la méthode **`permuter()`**

Les *layouts*




```
QHBoxLayout *hLayout1 = new QHBoxLayout;
QHBoxLayout *hLayout2 = new QHBoxLayout;
QVBoxLayout *mainLayout = new QVBoxLayout;

hLayout1->addWidget(valeur);
hLayout1->addWidget(choix);
hLayout1->addWidget(resultat);
hLayout1->addWidget(unite);
hLayout2->addWidget(bConvertir);
hLayout2->addWidget(bQuitter);

mainLayout->addLayout(hLayout1);
mainLayout->addLayout(hLayout2);

setLayout(mainLayout);
```

On crée les *layouts*

On place les *widgets* dans les *layouts*

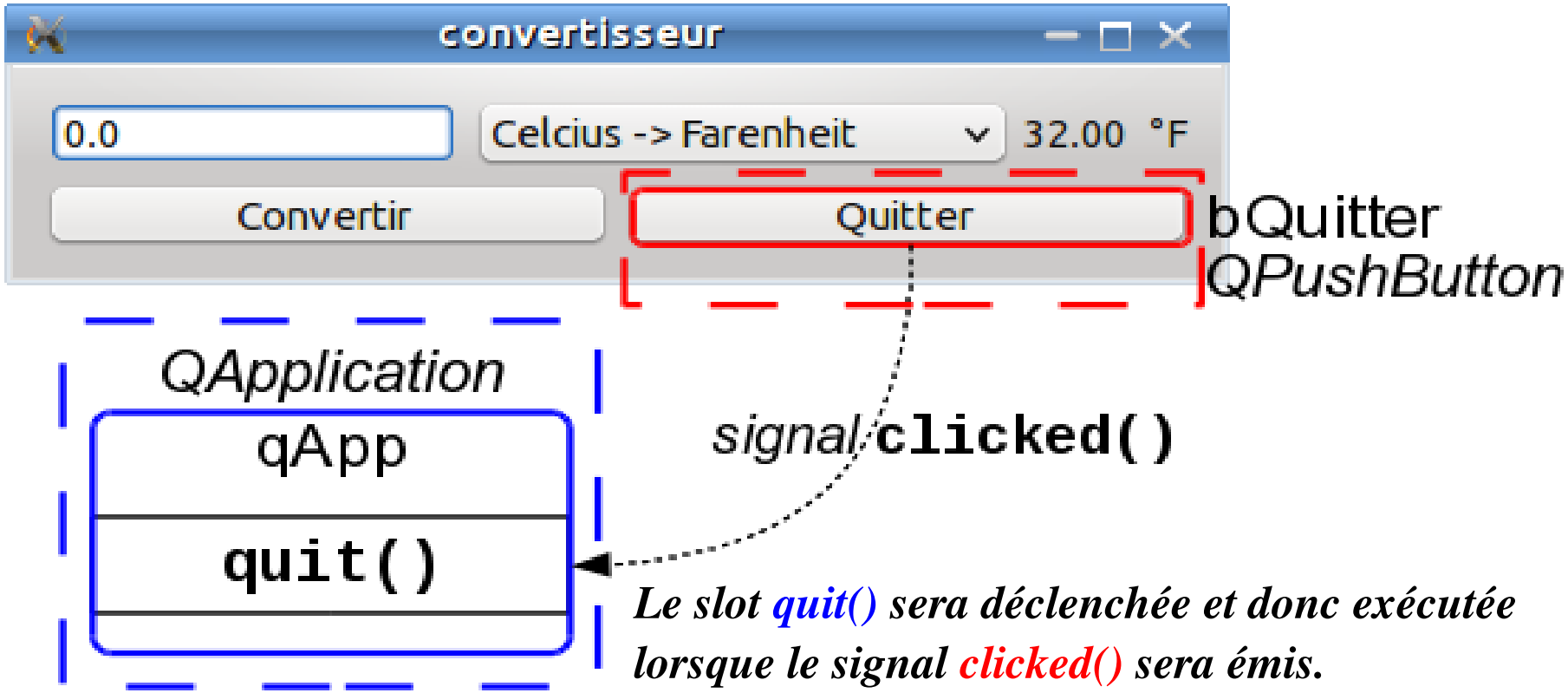
On incoprore les *layouts* dans le *layout* principal

On applique le *layout* principal au *widget* parent

Le mécanisme *signal/slot*

- Il faut connecter :
  - un OBJET émetteur (**bQuitter**) d'un SIGNAL (**clicked()**) à
  - un OBJET récepteur (**qApp**) sur une fonction (ou méthode) SLOT (**quit()**)

```
connect(bQuitter, SIGNAL(clicked()), qApp, SLOT(quit()));
```

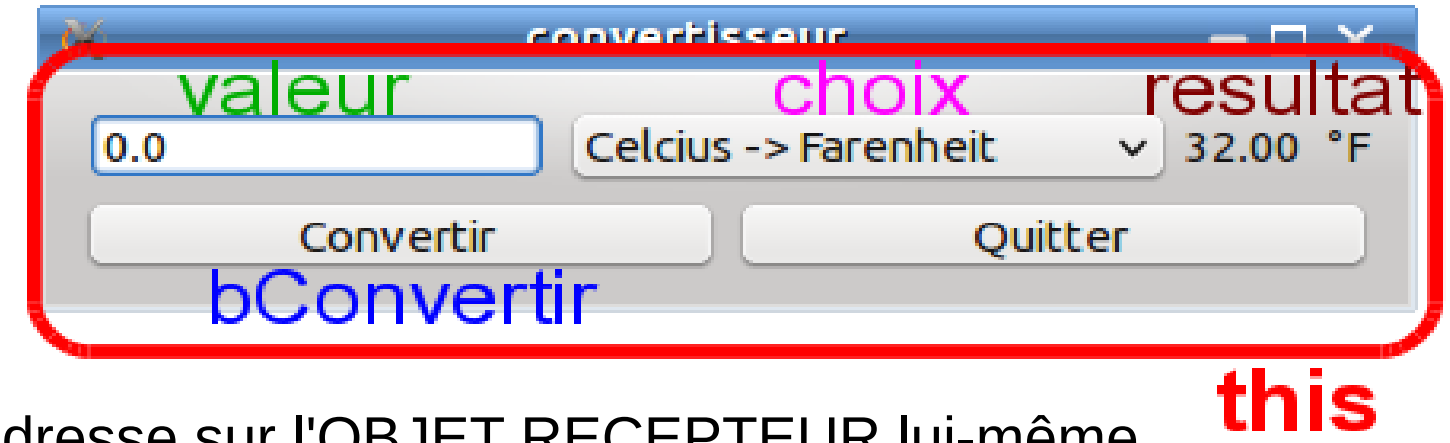


Le slot quit() sera déclenchée et donc exécutée lorsque le signal clicked() sera émis.

Première application - Principe n°7

- On peut assurer la connexion des signaux aux slots dans le constructeur de sa classe :

```
connect(bConvertir, SIGNAL(clicked()), this, SLOT(convertir()));
connect(this, SIGNAL(actualiser()), this, SLOT(convertir()));
connect(choix,SIGNAL(currentIndexChanged(int)),this,SLOT(permuter(int)));
```



this est l'adresse sur l'OBJET RECEPTEUR lui-même. C'est le C++ qui le fournit automatiquement. A chaque fois que l'on utilise "this" dans une classe, il faut comprendre "moi l'objet de cette classe" (auto-pointeur).

Émettre un *signal*

- Pour émettre un signal, on utilise tout simplement le mot clé **emit** :

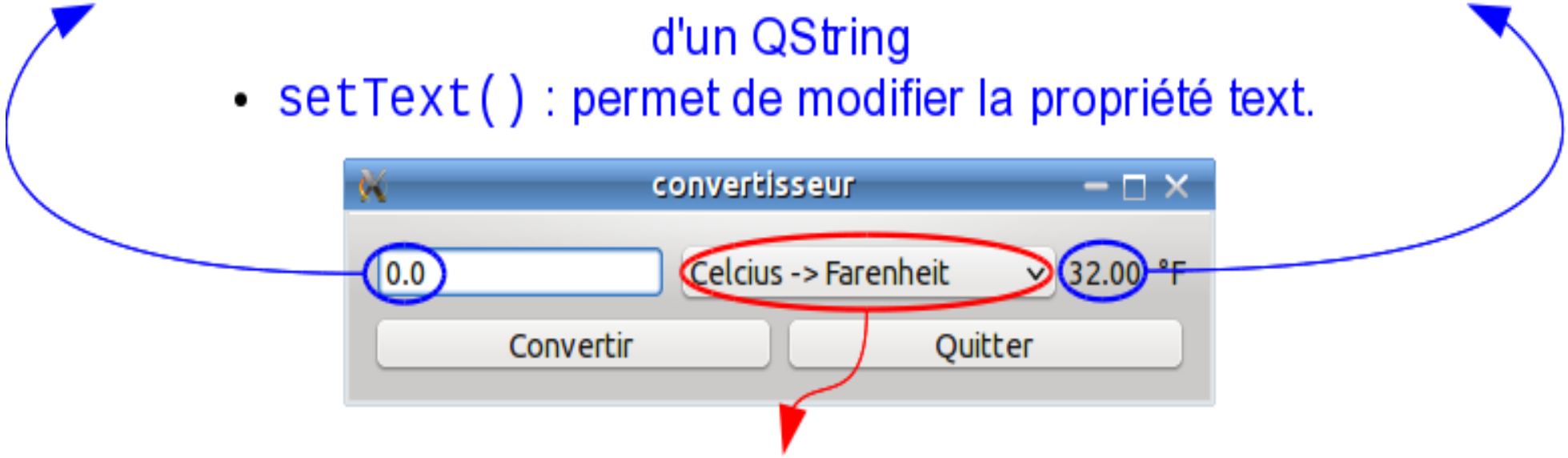
```
void MyWidget::permuter(int index)
{
      valeur->setText(resultat->text());
      emit actualiser();
}
```

Attention : un signal ne se définit pas. On le crée simplement en le déclarant dans le fichier header (.h) de sa classe.

Les propriétes des *widgets*

Les classes QLineEdit et QLabel possèdent une propriété text :

- `text()` : permet de récupérer la propriété text sous forme d'un QString
- `setText()` : permet de modifier la propriété text.



La classe QComboBox possède une propriété currentIndex.

- `currentIndex()` : permet de récupérer la propriété currentIndex (numéro d'élément actuellement sélectionné)
- `setCurrentIndex()` : ...