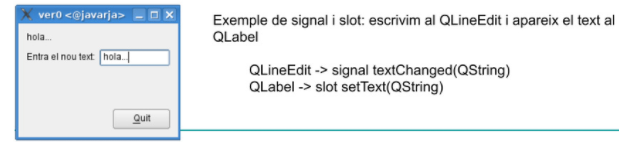
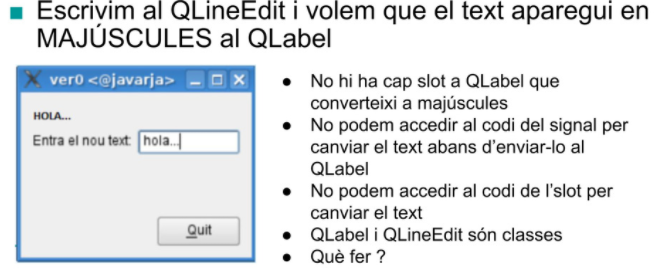
**Custom Widgets a Qt**

**Recordatori**

* Widgets existents i configurables
* Connexions entre components mitjançant signals i slots:
  + Signal: Esdeveniment que succeeix durant l’execució. Ex: Clic sobre un widget...
  + Slot: mètodes especials que es poden connectar amb signals.



**Problema 1:**

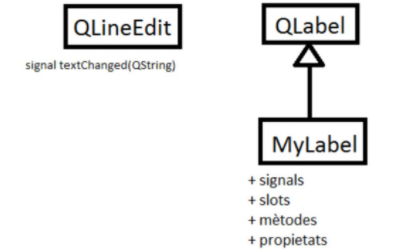
****

****

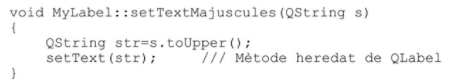
* Cap de les dues classes es pot modificar.
* Cap dels dos mètodes es pot modificar.

Solució:

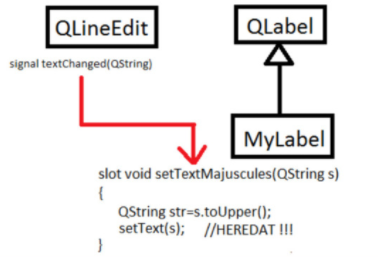
Heretant.



* Heretar una classe nova de QLabel: MyLabel.
* MyLabel hereta l’aspecte, mètodes i propietats, slots i signals de QLabel.
* A MyLabel podem afegir nous slots i signals, si cal.
* Podem escriure el codi dels slots, però no dels signals.
* Podem crear un slot que, abans de posar el text al QLabel, el converteixi a majúscules.



* Després es pot connectar el signal textChanged(QString) a aquest slot.
* Es pot heretar de qualsevol classe de Qt.



Crear els dos fitxers següents:



MyLabel no conté codi C++, cal preprocessar-lo (MOC), es fa automàticament.

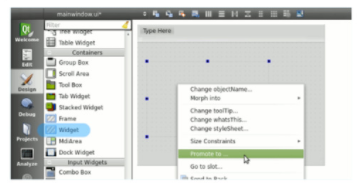
Només cal afegir el .h i el .cpp (els dos fitxers anteriors) al .pro.

HEADERS += MyLabel.h

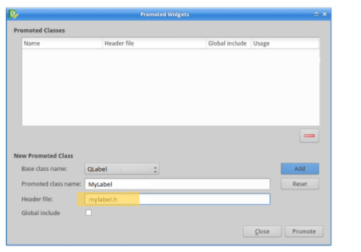
SOURCES += MyLabel.cpp

**Promote:**

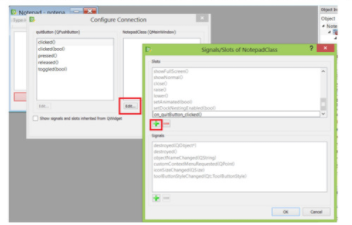
El resultat és un .h i un .cpp. Però, el designer no detecta la nova classe que hem creat. Per tant, al designer cal posar el widget de la classe base: QPushButton, QLCDNumber... i sobre el widget prémer el botó dret del ratolí i fer Promote to...



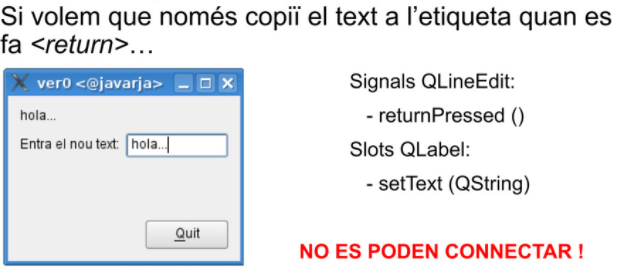
Posar la classe del nostre custom widget. Vigilar amb el nom del fitxer .h.



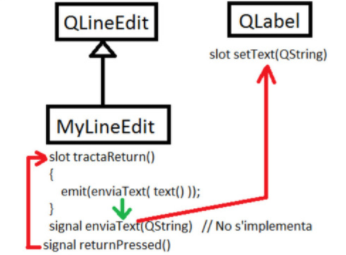
Cal connectar els signals/slots de la nostra classe amb altres signals/slots. El designer no detecta els nous signals i slots del .h, cal afegir-los a mà al diàleg de connexió. VIGILAR ELS NOMS I ELS PARÀMETRES, han de ser iguals als que estan declarats al .h.



**Problema 2:**



Heretant.



* Heretar de la classe QLineEdit: MyLineEdit.
* Crear un slot a MyLineEdit sense paràmetres: tractaReturn().
* Connectar returnPressed() amb tractaReturn().
* tractaReturn emet un signal que envia el text guardat al QLineEdit i que es pot consultar amb el mètode text(). Aquest mètode es heretat per MyLineEdit.
* El signal s’ha de crear a MyLineEdit i ha d’enviar un QString: enviaText(Qstring).
* El signal es connecta al slot setText del Label.

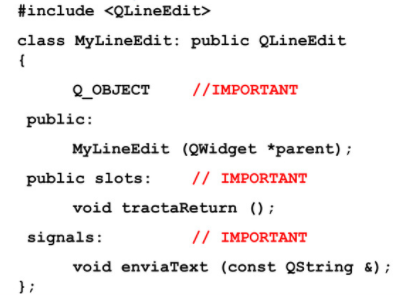
Notes importants:

* Els slots els implementarem a “MyLineEdit.cpp”.
* Els signals no s’implementen, però es poden llençar en qualsevol punt del codi cridant la funció: emit nom\_signal(paràmetres).
* Els signals només es poden connectar a slots, no s’implementen.
* Quan s’emet el signal, s’executen tots els slots que hi estan connectats:

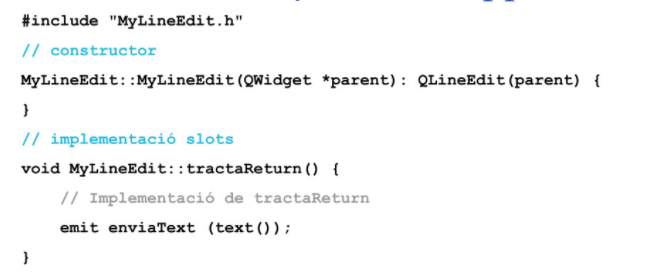
emit enviarEnter(5).

Fitxers:

**MyLineEdit.h**



**MyLineEdit.cpp**

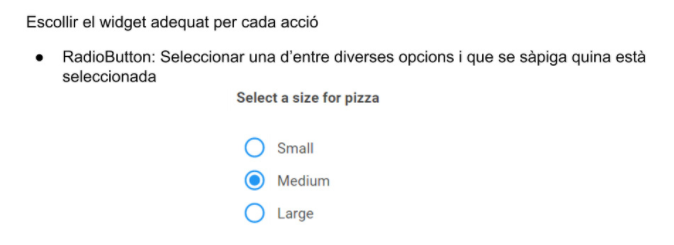


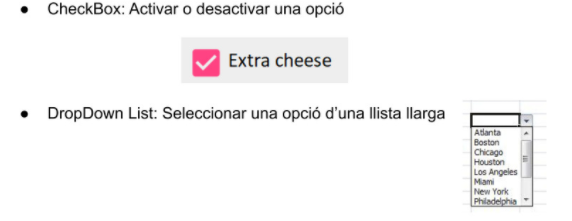
**La classe MyGLWidget:**

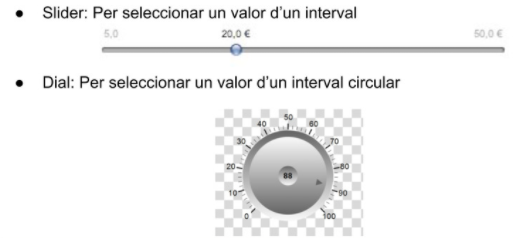
* Com podeu veure, la nostra classe d’OpenGL MyGLWidget, en realitat és una classe pròpia derivada de QOpenGLWidget de Qt…
  + Podeu veure que el .h inclou la macro Q\_OBJECT.
  + I que tenim el fitxer .h en el tag HEADERS del .pro.
* Per tant podem usar-la per afegir comportament si volem que es pugui lligar amb altres components de Qt (és a dir, podem afegir-li signals i slots).
* Recordeu afegir el “makeCurrent()” al principi de qualsevol slot que hagi d’usar codi OpenGL.

Notes importants:

* Els custom widget no funcionen amb CTRL+R, cal compilar i executar des de la consola.
* Els LCDNumber han de tenir una mida una mica gran per a què es vegin en color.
* Feu servir stylesheet pels colors, millor que la classe QPalette.
* Si feu servir QtCreator vigilar amb el Shadow Build. Assegureu-vos que podeu compilar i executar després des de la consola.
* Per a fer una neteja: make distclean.
* Usar l’assistant.







**Selecció d’un element d’interfície**

Principals elements d’interfície:

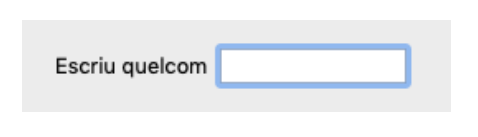
1. Botó (button):

* El botó serveix per executar una única acció quan es prem.
* Acostuma a indicar quina és la seva funció amb un text, una icona o ambdós alhora.
* Aquest text (o icona) acostuma a no variar, per tal de no confondre l’usuari.
* El botó no és una bona opció en cas que sigui necessari mostrar informació diferent (estat).



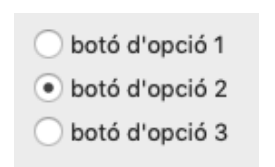
2. Camp de text (text field):

* El camp de text permet a l’usuari introduir un text.
* Depenent de com estigui configurat l’element permet introduir una única línia de text o vàries.
* En Qt tenim diverses classes que implementen els camps de text: QLineEdit, QTextEdit, ...



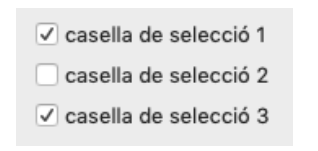
3. Botó d’opció (radio button):

* Els botons d’opció permeten seleccionar una sola opció d’un conjunt. Per tant mai usarem un únic botó d’opció en el disseny d’interfícies (com a mínim van en parelles).
* Inicialment una de les opcions hauria d’estar seleccionada.
* El nombre d’opcions no hauria de ser massa elevat.



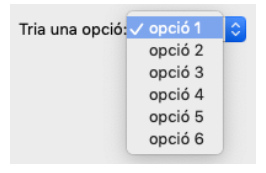
4. Casella de selecció (checkbox):

* Les caselles de selecció permeten seleccionar opcions d’un conjunt (entre 0 i totes les opcions).
* La millor opció acostuma a ser col·locar les caselles de selecció una sota l’altra en una llista vertical.
* Només és acceptable usar més d’una columna quan la llista és massa llarga i sigui necessari desplaçar-se per veure totes les opcions, o si la comparació de termes fos necessària.



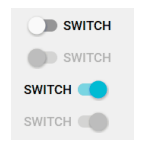
5. Llista desplegable (dropdown list o comboBox):

* Una llista desplegable permet els usuaris seleccionar una sola opció alhora de manera similar als botons d’opció.
* Aquest element d’interfície és més compacte i permet estalviar espai (respecte els botons d’opció).
* Cal considerar afegir un text a aquest element, com per exemple “Tria una opció”, per ajudar a l’usuari a reconèixer quina acció hauria de fer.



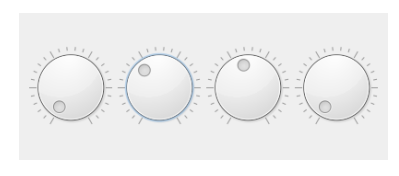
6. Commutador (toggle switch):

* El commutador és un element d’interfície que alterna entre dos estats.
* Bàsicament és un botó on/off que indica quin és el seu estat actual.
* L’avantatge principal sobre els botons d’opció és que és molt més compacte.
* Aquest element d’interfície no es troba de manera nativa en Qt.



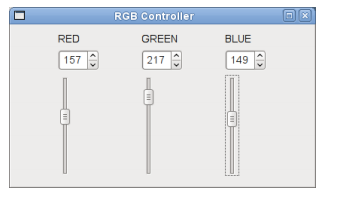
7. Dial:

* El dial és un element d’interfície d’entrada de dades que replica un sistema de control circular (similar a un velocímetre o potenciòmetre).
* Els dials s’usen quan l’usuari necessita controlar un valor entre un rang definit de forma circular.

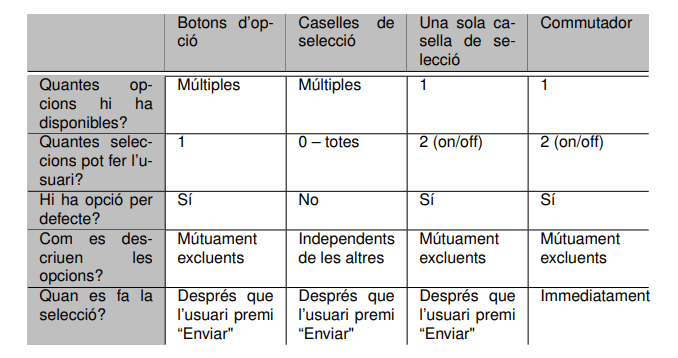


8. Botó lliscant (slider):

* El botó lliscant és el widget clàssic que permet controlar un valor delimitat.
* Permet a l’usuari moure el botó a través d’una ranura de manera que la posició del botó es tradueixi en un valor enter.



Quin element d’interfície triar?

Regla d’or: Cal donar a l’usuari el màxim d’informació amb la mínima interacció possible. Però cal evitar sobrecarregar la interfície d’informació no rellevant.

Botons d’opció VS llista desplegable:

Botons d’opció -> quan hi ha poques opcions. (<4)

Llista desplegable -> quan hi ha moltes opcions. (>3)

Botó lliscant VS dial:

Botó lliscant: el botó lliscant s’ha de triar en la majoria dels casos. Pels usuaris és més fàcil navegar horitzontalment i vertical que fent corbes i cercles. Amb el ratolí és complicat fer rotacions i corbes.

Dial: cal triar dial quan ens doni una major affordance. S’hauria d’usar un dial en llocs on és natural utilitzar un element amb interacció circular. Per exemple, un velocímetre, selecció de graus, un rellotge,. . . El dial és molt adequat en interfícies tàctils.

\*S’anomena Affordance a qualsevol característica d’un objecte que ens comunica com interactuar.