Classe de base: QObject. Emission : emit signal(); . Dans le .h, signals : void signal();

bool QObject::connect(const QObject * sender, const char * signal, const QObject * receiver, const char * method, Qt::ConnectionType type = Qt::AutoConnection) const ; signal de sender sera réceptionné par receiver et exécutera le slot method.
Exemple : QObject::connect(&w, SIGNAL(verifQteFarine(unsigned)), pFarine, SLOT(checkMinQty(unsigned)));

MOC: Meta Object Compiler. Interface de pré-compilation du code source (pseudo c++). Modèle intermédiaire. Outil qui fait partie de l'environnement de développement Qt. Utilisé pour générer du code c++ à partir de fichiers de définition de classe QObject (pseudo c++).

#ifndef NOMCLASSE_H #define NOMCLASSE _H #include <QObject>
class NomClasse : public QObject { Q_Object private : public : NomClasse(); pulic slot : signals : }; #endif // NOMCLASSE _H

#ifndef NOMWIDGET_H, #define NOMWIDGET_H #include < QWidget > #include <QLabel> #include <QPushButton>
class WidgetPrincipal : public QWidget { Q_OBJECT private: QPushButton * pbVerifie; QLabel * lEau; public:
WidgetPrincipal(QWidget *parent = nullptr); ~WidgetPrincipal();
signals: void verifQteEau(unsigned int); public slots: void verification(); }; #endif // WIDGETPRINCIPAL_H

#include "widgetprincipal.h" WidgetPrincipal::WidgetPrincipal(QWidget *parent) : QWidget(parent){ setWindowTitle("nom"); QVBoxLayout * vbVertical; QHBoxLayout * hbEau; pbVerifie = new QPushButton("Vérifie Ingrédients"); hbEau = new QHBoxLayout; hbEau->addWidget(lEau); vbVertical = new QVBoxLayout(this); vbVertical->addLayout(hbEau); vbVertical->addWidget(tLog); setLayout(vbVertical); } !! void NomClasse::methode(){}, const pour get et param, exemple : const unsigned int &

Main : #include "widgetprincipal.h" #include "producteureau.h" #include <QApplication> int main(int argc, char *argv[]) { QApplication a(argc, argv); WidgetPrincipal w; ProducteurEau * pEau = new ProducteurEau(); ProducteurFarine * pFarine = new ProducteurFarine(); QObject::connect(&w,SIGNAL(verifQteEau(unsigned)) ,pEau, SLOT(checkMinQty(unsigned))); w.show(); return a.exec();}

Pour contrôler ses propres utilitaires (moc, uic, ...), Qt fournit un moteur de production spécifique : le programme qmake. qmake prend en entrée un fichier de projet .pro et génère en sortie un fichier de fabrication spécifique à la plateforme. Ainsi, avec les systèmes UNIX/Linux, qmake produira un Makefile.
Projet QT défini par un fichier d'extension .pro décrivant la listedesfichierssources, dépendance, parampassés aucompilateur, etc.
QT += gui sql xml.
QDialog : public
À chaque fois que l'on modifie le fichier .pro, il faudra exécuter à nouveau la commande qmake pour que celle-ci mette à jour le fichier Makefile. D'autre part, le fichier Makefile est toujours spécifique à la plateforme. Si vous changez de plateforme (Linux, Windows, MacOs), il vous suffira d'exécuter à nouveau la commande qmake pour générer un fichier Makefile adapté à votre système.

class MyDialog : public QDialog { Q_OBJECT private : public : MyDialog(QWidget ∗parent=0) ; public slots : signals : } ;
#include ”mydialog.h” MyDialog : :MyDialog(QWidgetparent) : QDialog(parent) { //TODO }
QMessageBox msgBox ; msgBox.setText(”The document has been modified.”) ; msgBox.setInformativeText(”Do you want to save your changes?”) ; msgBox.setStandardButtons(QMessageBox : :Save | QMessageBox : :Discard | QMessageBox : :Cancel) ; int ret = msgBox.exec() ;

QString fileName = QFileDialog::getOpenFileName(0, " Ouvrir un fichier", "/home/tv", "Fichiers textes (*.txt);;Fichiers XML (*.xml)");

.h : class MyMainWindow : public QMainWindow {Q_OBJECT private : public : MyMainWindow(QWidget ∗parent = 0) ; public slots : signals : } ;
.cpp : MyMainWindow : :MyMainWindow(QWidget ∗parent) : QMainWindow(parent) 4 { 5 //TODO 6 }

Avantages héritage :
Réutilisation du code : l'héritage permet de réutiliser le code déjà existant dans la classe de base, ce qui peut être très utile pour éviter de devoir réécrire du code qui a déjà été testé et validé.
Meilleure organisation du code : l'héritage permet de regrouper des fonctionnalités similaires dans une classe de base, ce qui peut aider à mieux organiser le code et à le rendre plus facile à comprendre.

Améliore la lisibilité et la maintenance du code : en utilisant l'héritage, il est possible de regrouper des fonctionnalités similaires dans une classe de base, ce qui peut rendre le code plus lisible et plus facile à maintenir.
Permet de créer des hiérarchies de classes : l'héritage permet de créer des hiérarchies de classes qui reflètent les relations existantes entre les objets dans le monde réel. Cela peut aider à mieux organiser le code et à le rendre plus facile à comprendre.

Inconvénients :
Dépendance entre les classes : l'héritage crée une dépendance entre les classes, de sorte que si la classe de base est modifiée, cela peut avoir des effets sur les sous-classes qui en dérivent. Cela peut rendre le code moins flexible et plus difficile à maintenir.
Complexité accrue du code : l'utilisation de l'héritage peut rendre le code plus complexe, en particulier lorsque de nombreuses classes sont impliquées et qu'il y a de nombreuses relations d'héritage entre elles. Cela peut rendre le code plus difficile à comprendre et à maintenir.

Classe ProducteurBoisson abstraite si aucune classe n'a initialisé un objet de type ProducteurBoisson.
Interface c++, toutes les fonctions sont virtuelles pures (virtual void method affiche() = 0). Pour surcharger une méthode, fonction mère : virtual void affiche(). Fille : void affiche(). On peut créer une classe abstraite en mettant au moins une fonction membre virtuelle pure (on écrit retour fonction() =0;)

Menu : Dans .h : #include <QMenuBar> puis private : QMenuBar * barreMenu; Constructeur : barreMenu = new QMenuBar(nullptr); setMenuBar(barreMenu);QMenu*menuFich=new QMenu(QString::fromUtf8("&Fichier"), this);barreMenu->addMenu (menuFich);
Sous menu : QMenu *smenuInfo = new QMenu(QString::fromUtf8("&Information"), this); menuFich->addMenu(smenuInfo);
Action : QAction *actionInformationTaille = new QAction(QString::fromUtf8("&Taille des données") , this); connect(actionInformationTaille, SIGNAL(triggered()), this, SLOT(afficheTaille())); smenuInfo->addAction(actionInformationTaille);

BDD : #include <QtSql>, QT += sql, QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");db.setHostName("acidalia"); db.setDatabaseName("customdb"); db.setUserName("mojito"); db.setPassword("J0a1m8"); bool ok = db.open(); db = QSqlDatabase::addDatabase("QSQLITE"); db.setDatabaseName("/tmp/base.sqlite"); if(!db.open()){} else{}
Private : QSqlDatabase db; QSqlQuery query ("SELECT ∗from Personne") ; QSqlQuery query; query.prepare("INSERT INTO person (id, nom, prenom)VALUES (:id, :nom, :prenom)"); query.bindValue(":id", 1001); query.bindValue(":nom", "Pierre"); query.bindValue (":prenom", "Durand"); if(query.exec()) { while(query.next()) { qDebug() << "id : " << query.value(0).toString();};
Doxygen : doxygen -g <fichierConf>, nom par défaut = Doxyfile. doxygen <fichierConf>

QApplication fournit une boucle principale d'évènements graphiques pour les app QT. QtCore = non graphique, QtGui = composant graphiques, QtNetwork = prog réseau. QtSQL = bddSQL, QtXML = fichier XML.

std::map<std::string, std::string > ms; ms["jd"] = "Dupont"; std::cout << ms["jd"]<<std::endl ; void afficheListeFormes (std::vector < Forme * > lf) {for (auto &f: lf) { f->affiche() ; } } for (const auto & kv : mf) { std :: cout << kv. first << " ␣ / ␣ " << std :: endl; afficheListeFormes (kv. second) ;}

class MainWindow : public QMainWindow 2 {Q_OBJECT QStateMachine * stateMachine ; QPushButton * bAvance ; unsigned int compteur ; public : MainWindow (QWidget * parent = nullptr) ; ~MainWindow () ; private slots : void incrementeCompteur () ; void razCompteur () ; void afficheCompteur () ; bAvance = new QPushButton ("Avance",this);stateMachine=new QStateMachine; QState *st0 = new QState ; QState *st1 = new QState ; QState *st2 = new QState; stateMachine->addState(st0) ;stateMachine->addState (st1);stateMachine->addState (st2);stateMachine->addState(st3) ; st1->addTransition (bAvance ,SIGNAL (clicked()) , st2); st2->addTransition (bAvance , SIGNAL (clicked()) , st3) ;st3->addTransition(bAvance , SIGNAL (clicked()) , st4) ; connect (st1 , SIGNAL (entered()), this, SLOT(incrementeCompteur ())); connect (st2, SIGNAL(entered()), this ,SLOT(incrementeCompteur ())) ; connect (st1 , SIGNAL (exited ()) , this , SLOT (afficheCompteur()));connect (st2 , SIGNAL (exited()), this ,SLOT(afficheCompteur ())); stateMachine->setInitialState(st0);stateMachine->start(); std::function <void(const int &)>pF=maFonction ; std::map<std::string, std::function<float(const std :: vector &)>> mf;