Qt - Module IHM
TP n°1

par Thierry Vaira © v.1.10

Sommaire

Une horloge digitale **2**

Étape 1 : Prototypage de l'IHM 3

Étape 2 : Programmation évènementielle 5

Étape 3 : Squelette de l'application 5

Étape 4 : Implémentation 10

Étape 5 : Test 15

Étape 6 : Évolution 16

Les TP d'acquisition des fondamentaux visent à construire un socle de connaissances de base, à appréhender un concept, des notions et des modèles qui sont fondamentaux. Ce sont des étapes indispensables pour aborder d'autres apprentissages. Les TP sont conduits de manière fortement guidée pour vous placer le plus souvent dans une situation de découverte et d'apprentissage.

Les objectifs de ce tp sont de découvrir la programmation d'IHM sous Qt.

Une horloge digitale

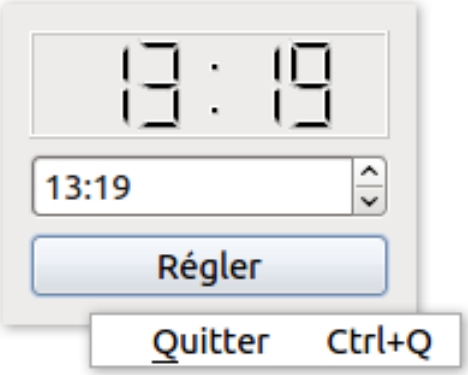
L'objectif de ce premier TP est de réaliser pas à pas une application GUI (*Graphical User Interface*) avec **Qt**.

On travaillera avec l'EDI Qt Creator par contre, on n'utilisera pas l'assistant Qt Designer pour la réalisation de la GUI. Ceci pour permettre de bien appréhender les mécanismes de mise en œuvre des *widgets* et *layout*.

Tous les concepts utilisés dans ce TP sont décrits dans le support de cours sur **Qt**, notamment :

- l'utilisation des *widgets*
- la mise en œuvre du mécanisme *signal/slot*

On désire réaliser une horloge digitale qui s'affichera en permanence sur le bureau. L'application GUI affichera donc digitalement l'heure avec le format `HH:MM` et permettra son réglage. On pourra quitter l'application à partir d'un menu contextuel accessible par le bouton droit de la souris ou le raccourci avec le raccourci Ctrl + Q .



L'horloge digitale

L'application sera affichée en avant-plan pour toujours bénéficier de son affichage sur le bureau. Cela entraîne qu'il ne sera pas nécessaire d'avoir une barre de titre pour cette application (gain de place, esthétisme). On pourra toujours déplacer notre fenêtre en utilisant la combinaison de la touche Alt et du bouton gauche de la souris.

On développera l'application en itérations.

Un développement itératif s'organise en une série de développement très courts de durée fixe nommée itérations. Le résultat de chaque itération est un système partiel exécutable, testé et intégré (mais incomplet).

On va détailler maintenant les différentes étapes pour réaliser cette horloge digitale.

Lorsque l'on réalise une application graphique, il est habituel de commencer par l'IHM (Interface Homme-Machine).

Étape 1 : Prototypage de l'IHM

Cette étape a pour but de concevoir l'IHM.

On commence donc par définir l'architecture de la GUI. L'application est basée sur une seule fenêtre qui contiendra les ***widgets*** **(cf. cours)** suivants :

- un affichage de l'heure sous forme digitale : Qt ne fournit pas directement ce *widget* par contre on pourra utiliser un `QLCDNumber` pour afficher une valeur digitale
- une zone d'édition de l'heure : on utilisera le *widget* `QTimeEdit`
- un bouton pour valider le réglage désiré : on aura besoin d'un *widget* `QPushButton`

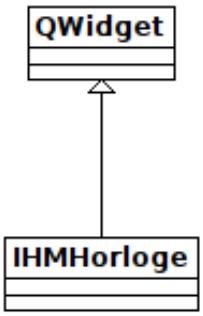
Le menu contextuel (bouton droit de la souris) sera simplement réalisé par l'ajout d'un `QAction` et d'un `QKeySequence` pour le raccourci clavier.

Le **positionnement** **(cf. cours)** de ces *widgets* se fera verticalement avec un `QVBoxLayout`.

Ici, il faut pouvoir créer une **fenêtre personnalisée** **(cf. cours)**. Notre application ne nécessite pas l'utilisation :

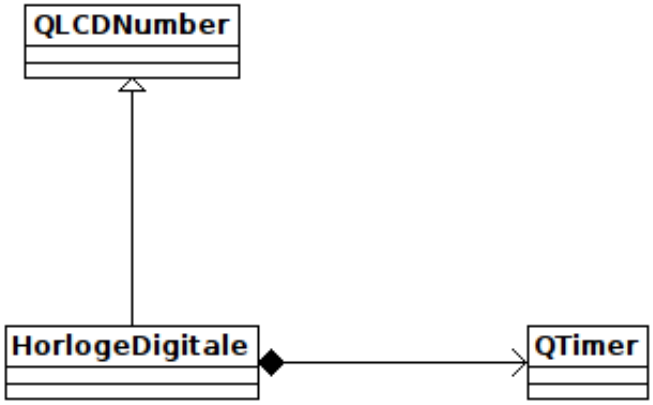
- d'une fenêtre principale (`QMainWindow`) car on n'a ni de menu, ni de barre d'outils, ni de barre d'état.
- d'une boîte de dialogue (`QDialog`)

On basera donc la fenêtre personnalisée de l'application sur un `QWidget` :

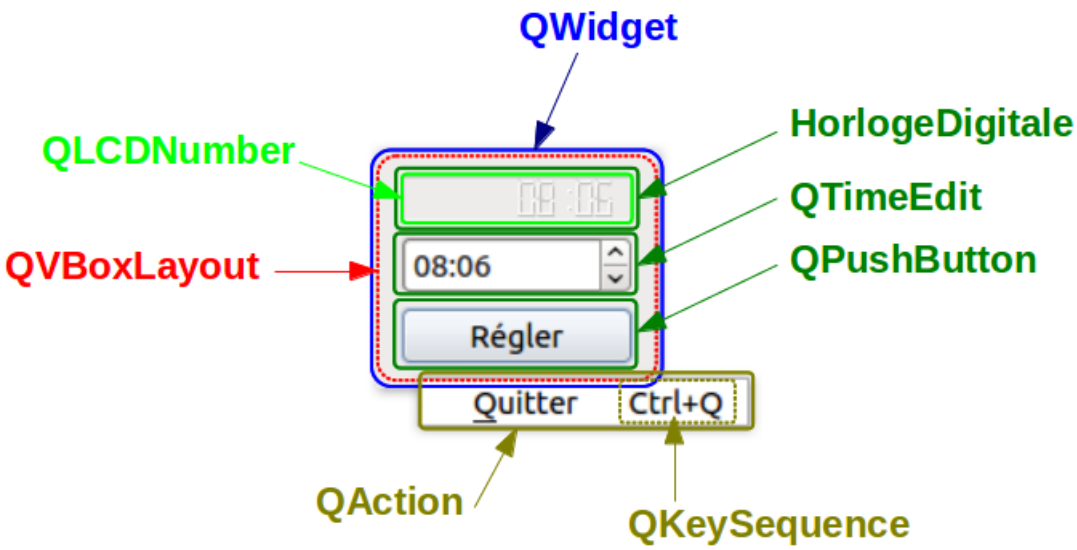


D'autre part, on a besoin d'un *widget* personnalisé pour réaliser le composant "horloge" :

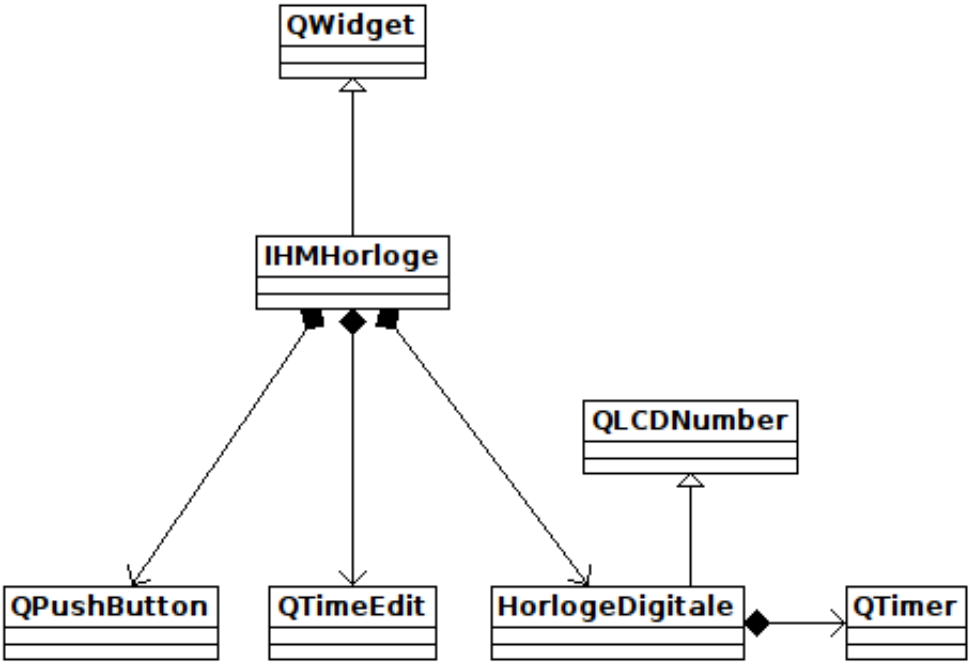
- extension d'un `QLCDNumber` pour l'affichage de la valeur digitale de l'heure
- intégration d'une base de temps avec un `QTimer` pour la gestion de l'heure



Finalement, l'application aura la structure suivante :



On peut établir le premier diagramme de classes pour l'application :



Le `QTimer` n'est pas un *widget* car il n'a pas d'"aspect visible" mais il semblait judicieux de le faire apparaître dès maintenant dans le diagramme.

Étape 2 : Programmation évènementielle

Cette étape a pour but de concevoir la communication évènementielle entre les objets.

La **programmation évènementielle** (cf. cours) d'une application Qt est basée sur le mécanisme ***signal*/*slot*** (cf. cours).

Pour connaître les signaux (et les slots) prédéfinis dans les classes Qt, il faut bien évidemment consulter sa documentation : doc.qt.io.

On détermine l'ensemble des signaux/slots à gérer :

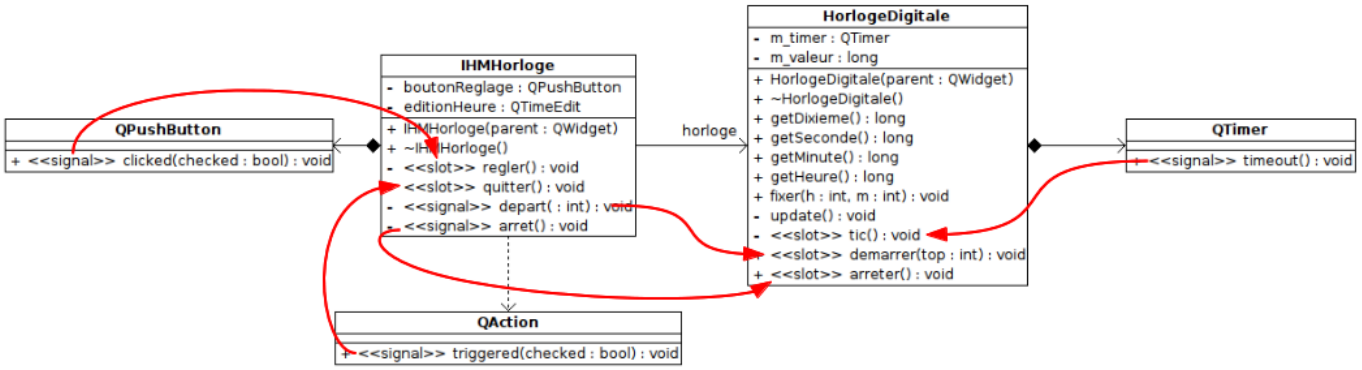
- *signal* `clicked()` émis par le bouton de réglage (`QPushButton`) → *slot* `regler()` (`IHMHorloge`)
- *signal* `triggered()` émis par l'action de quitter (`QAction`) → *slot* `quitter()` (`IHMHorloge`)
- *signal* `timeout()` émis périodiquement par la base de temps (`QTimer`) → *slot* `tic()` (`HorlogeDigitale`)

D'autre part, on devra pouvoir démarrer et arrêter l'horloge. On peut donc définir ces signaux/slots personnalisés :

- *signal* `depart(int)` émis par la fenêtre (`IHMHorloge`) → *slot* `demarrer(int)` (`HorlogeDigitale`)
- *signal* `triggered()` émis par la fenêtre (`IHMHorloge`) → *slot* `arreter()` (`HorlogeDigitale`)

Le *signal* `depart(int)` indiquera en paramètre la **période** (`int`) à fixer pour la base de temps du `QTimer`.

Le plan de connexion des signaux/slots sera le suivant :



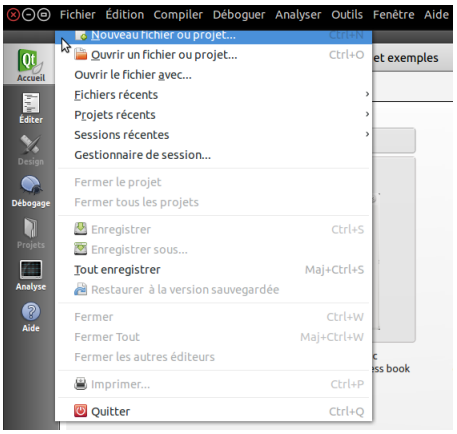
Étape 3 : Squelette de l'application

Cette étape a pour but de créer le squelette de l'application sous Qt.

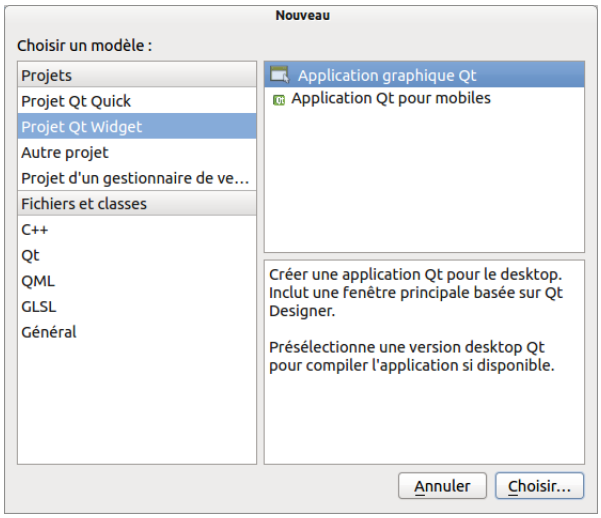
On va créer un **projet Qt** (cf. cours) en l'utilisant l'EDI `Qt Creator` (cf. cours) :

➠ Démarrer Qt Creator.

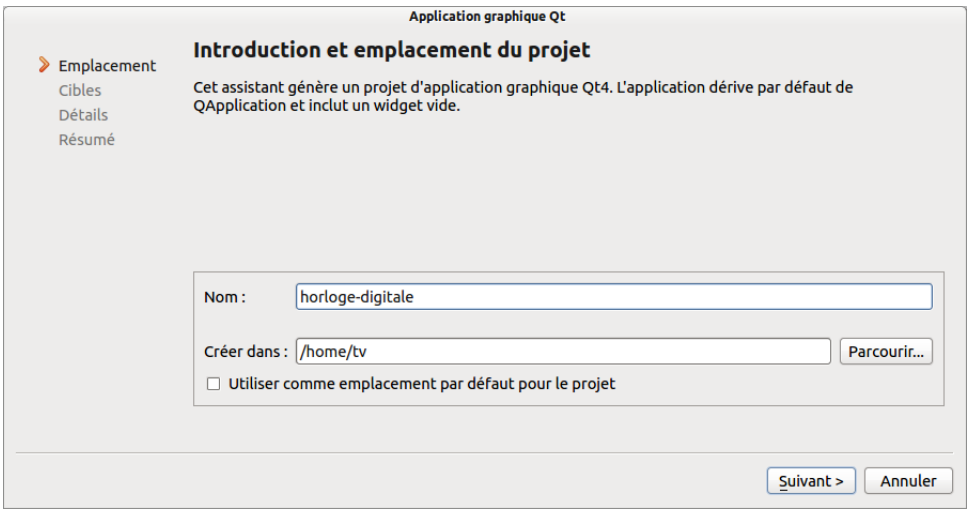
➠ Sélectionner Fichier → Nouveau Projet.



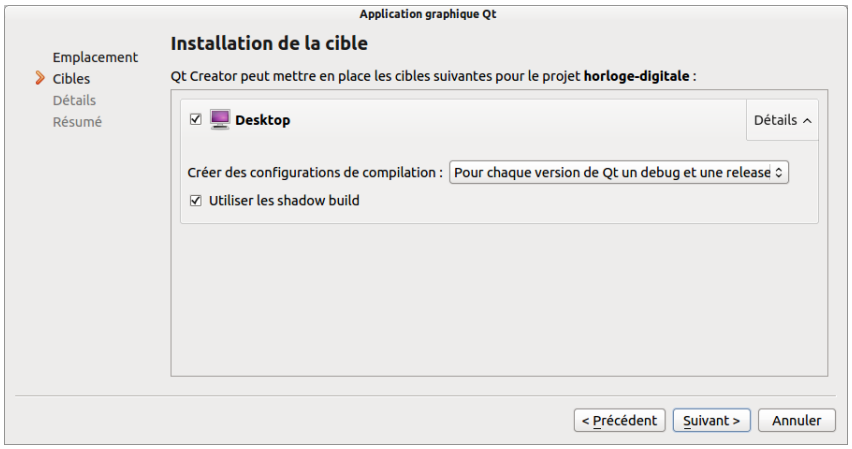
➠ Choisir Projet Qt Widget → Application graphique Qt.



➠ Donner un nom à votre projet (ce sera aussi celui de l'exécutable) et choisir son emplacement.

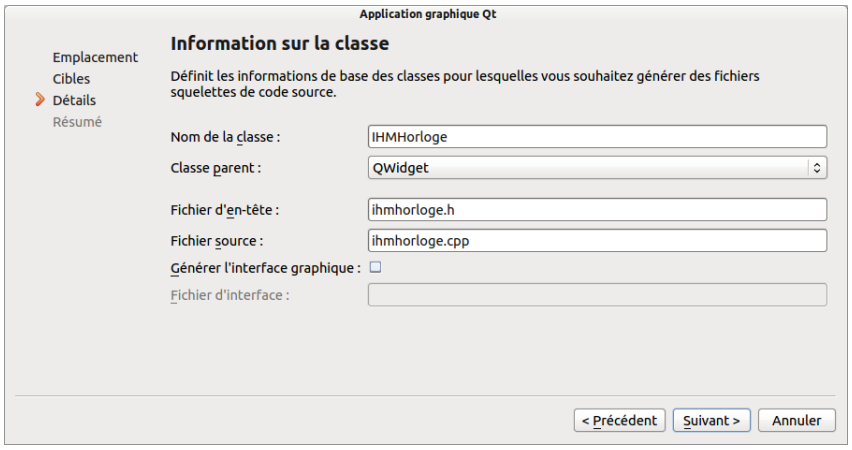


➠ Sélectionner une cible de compilation, choisir les configurations *debug* et *release* et cocher Utiliser les *shadow build*.

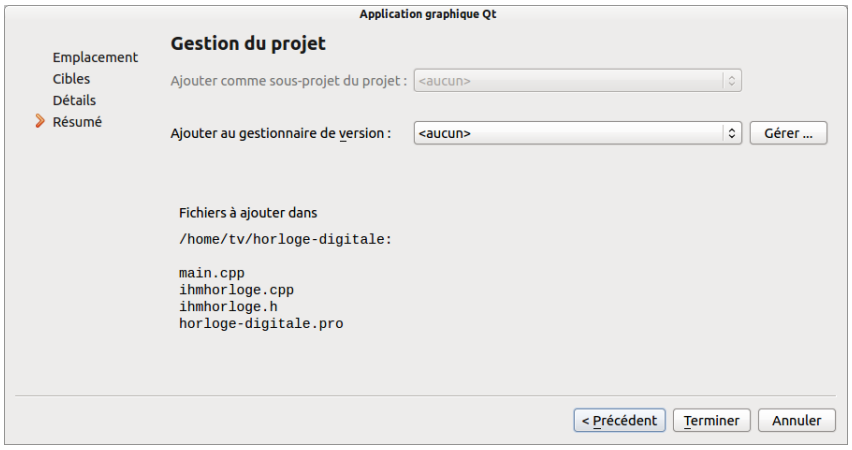


Le *shadow build* permet de séparer les dossiers contenant les codes sources des fichiers générés (exécutable, fichiers objets `.o`, ...) par la fabrication (*build*). Il est important de ne les pas mélanger pour assurer convenablement les tâches courantes (sauvegarde, gestion de versions, portablité multiplateforme, ...). Les fichiers issus de la compilation seront donc stockés dans un répertoire séparé et créé par Qt.

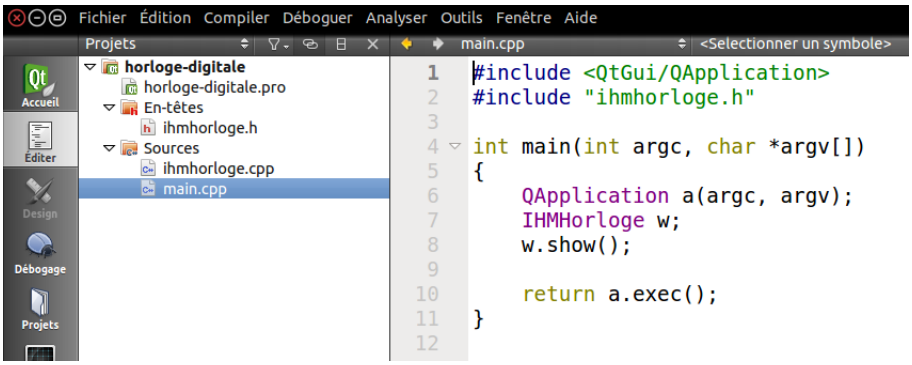
➠ Donner le nom à la classe fenêtre (`IHMHorloge`), sélectionner la classe parent (`QWidget`) et **décocher** la génération de l'interface graphique (puisque l'on a décidé de ne pas utiliser Qt Designer).



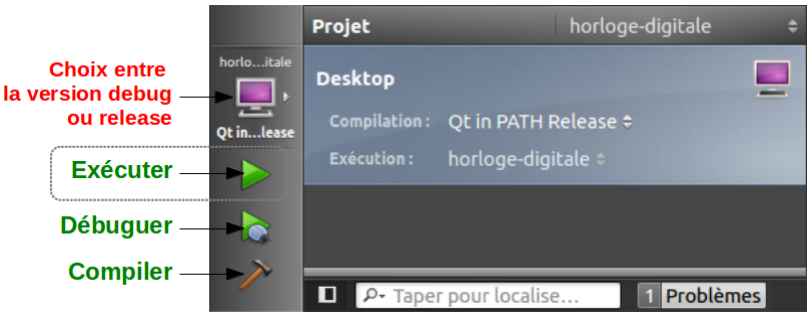
➠ Terminer.



Qt Creator va ensuite créer le projet et l'ouvrir :



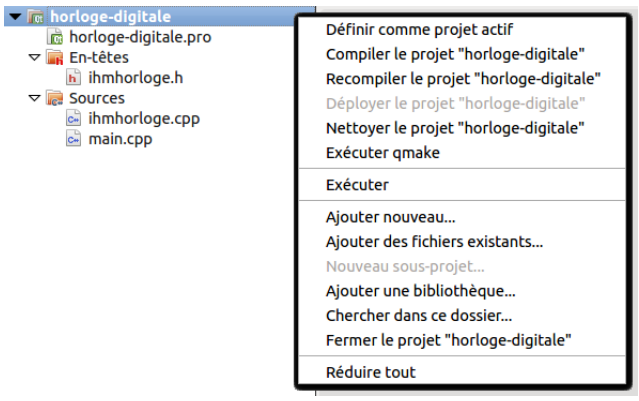
On a accès à quelques raccourcis utiles :



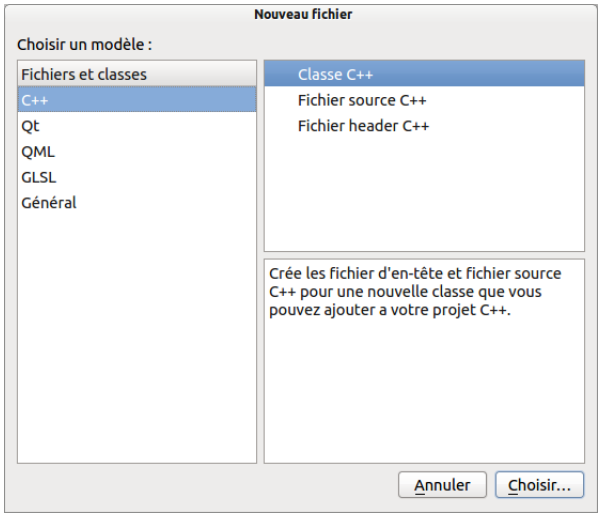
➠ Tester l'application créée en l'exécutant.

On va maintenant ajouter la classe `HorlogeDigitale` au projet.

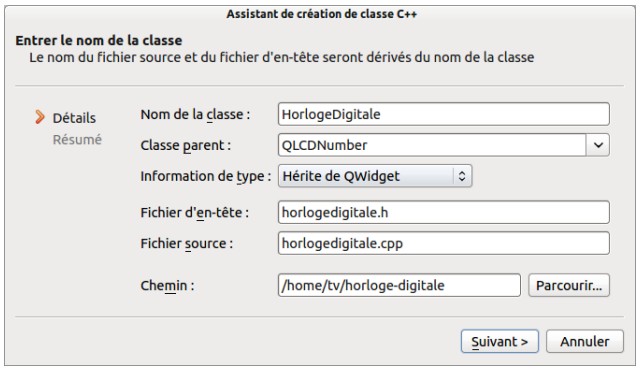
➠ Cliquer avec le bouton droit sur votre projet dans l'explorateur puis sélectionner Ajouter nouveau.



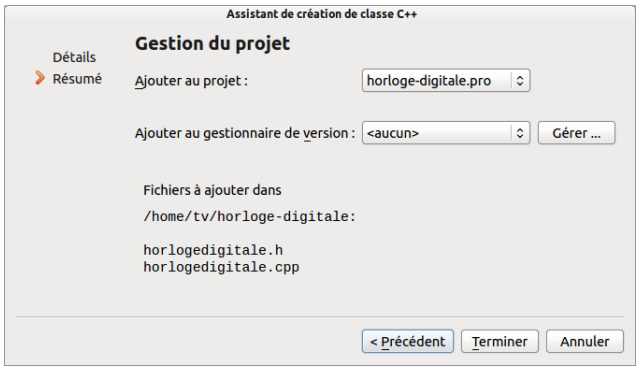
➠ Choisir `C++` puis Classe `C++`.



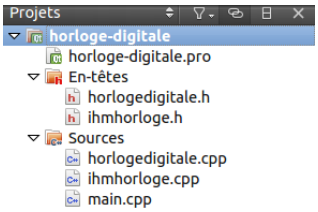
➠ Donner le nom à la nouvelle classe (`HorlogeDigitale`), sélectionner la classe parent (`QLCDNumber`) qui hérite de `QWidget`.



➠ Terminer.



La création du projet est finalisée et on va maintenant passer au codage.



Étape 4 : Implémentation

Cette étape a pour but de coder l'application.

Question 1. Compléter la déclaration de la classe `IHMHorloge` dans le fichier `ihmhorloge.h` en y ajoutant les *widgets*, les *signaux* et *slots* vus précédemment.

```
#ifndef IHMHORLOGE_H
#define IHMHORLOGE_H

#include <qglobal.h>
#if QT_VERSION >= 0x050100
#include <QtWidgets> /* tous les widgets de Qt5 */
#else
#include <QtGui> /* tous les widgets de Qt4 */
#endif

class HorlogeDigitale;

class IHMHorloge : public QWidget
{
    Q_OBJECT

    private:
        // les widgets
        QPushButton    *boutonReglage;
        HorlogeDigitale *horloge;
        QTimeEdit      *editionHeure;

    public:
        IHMHorloge(QWidget *parent=0);
        ~IHMHorloge();

    private slots:
        void regler();
        void quitter();

   signals:
        void depart(int);
        void arret();
};

#endif // IHMHORLOGE_H
```

ihmhorloge.h

Le constructeur de cette classe suit un canevas classique pour les fenêtres :

- créer les *widgets*
- créer le(s) *layout(s)*
- positionner les *widgets* dans *layout(s)*
- paramétrer l'affichage de la fenêtre (titre, taille, type, ...)
- initialiser

Question 2. Compléter la définition de la classe `IHMHorloge` dans le fichier `ihmhorloge.cpp`. Bien lire les commentaires.

```
#include "ihmhorloge.h"
#include "horlogedigitale.h"

/*
 La classe QDebug fournit un flux de sortie pour déboguer les informations. Elles peuvent
     être éliminées de l'exécutable en définissant QT_NO_DEBUG_OUTPUT. Qt fournit quatres
     fonctions globales pour l'écriture de texte d'avertissement et de débogage : qDebug()
     pour les sorties de débogage personnalisées, qWarning() pour des avertissements et des
     erreurs récupérables dans l'application, qCritical() des erreurs critiques et les
     erreurs systèmes et qFatal() pour tracer des erreurs fatales juste avant l'arrêt de l'
     application.
 */
#include <QDebug>

IHMHorloge::IHMHorloge(QWidget *parent) : QWidget(parent)
{
   // Les widgets
   horloge = new HorlogeDigitale(this);

   editionHeure = new QTimeEdit(QTime::currentTime(), this);
   editionHeure->setDisplayFormat("HH:mm");

   boutonReglage = new QPushButton("Régler");
   boutonReglage->setDefault(true);

   QAction *actionQuitter = new QAction("&Quitter", this);
   actionQuitter->setShortcut(QKeySequence(QKeySequence::Quit)); // Ctrl+Q
   addAction(actionQuitter);

   // Le positionnement
   QVBoxLayout *layoutPrincipal = new QVBoxLayout;
   layoutPrincipal->addWidget(horloge);
   layoutPrincipal->addWidget(editionHeure);
   layoutPrincipal->addWidget(boutonReglage);
   //layoutPrincipal->addStretch();
   setLayout(layoutPrincipal);

   // Les signaux/slots
   connect(this, SIGNAL(depart(int)), horloge, SLOT(demarrer(int)));
   connect(this, SIGNAL(arret()), horloge, SLOT(arreter()));
   connect(boutonReglage, SIGNAL(clicked()), this, SLOT(regler()));
   connect(actionQuitter, SIGNAL(triggered()), this, SLOT(quitter()));

   // La fenêtre
   setWindowTitle("Horloge digitale");
   //setFixedHeight(sizeHint().height());
   //setFixedWidth(sizeHint().width());
   // reste en avant plan sans la barre de titre (déplacer avec Alt+clic gauche)
   setWindowFlags(Qt::WindowStaysOnTopHint | Qt::FramelessWindowHint);
   // le menu contextuel
   setContextMenuPolicy(Qt::ActionsContextMenu);
```

```
   // Initialisation
   horloge->fixer(editionHeure->time().hour(), editionHeure->time().minute());
   emit depart(PERIODE); // démarrage de l'horloge
}

IHMHorloge::~IHMHorloge()
{
    // arrêt de l'horloge
    emit arret();
    qDebug() << "~IHMHorloge()";
}

void IHMHorloge::regler()
{
    // arrêt de l'horloge
    emit arret();

    // paramétrage de l'horloge
    int minutes = editionHeure->time().minute();
    int heures = editionHeure->time().hour();
    horloge->fixer(heures, minutes);

     // redémarrage de l'horloge
    emit depart(PERIODE);
}

void IHMHorloge::quitter()
{
    // on ferme la fenêtre
    close();
}
```

ihmhorloge.cpp

Question 3. Compléter la déclaration de la classe `HorlogeDigitale` dans le fichier `horlogedigitale.h` en y ajoutant le `QTimer` et les *slots* vus précédemment.

```
#ifndef HORLOGEDIGITALE_H
#define HORLOGEDIGITALE_H

#include <qglobal.h>
#if QT_VERSION >= 0x050100
#include <QtWidgets/QLCDNumber> /* Qt5 */
#else
#include <QtGui/QLCDNumber> /* Qt4 */
#endif
#include <QTimer>

// réglage au dixième de secondes
#define PERIODE 100

class HorlogeDigitale : public QLCDNumber
{
```

```
   Q_OBJECT

   public:
      HorlogeDigitale(QWidget *parent=0);
      ~HorlogeDigitale();

      long getMinute();
      long getHeure();
      void fixer(int h, int m);

   private:
      QTimer   *m_timer;
      long     m_valeur;

      void update();

   private slots:
      void tic();

   public slots:
      void demarrer(int top=PERIODE);
      void arreter();
};

#endif // HORLOGEDIGITALE_H
```

horlogedigitale.h

Question 4. Compléter la définition de la classe `HorlogeDigitale` dans le fichier `horlogedigitale.cpp`. Bien lire les commentaires.

```
#include "horlogedigitale.h"

#include <QString>
#include <QDebug>

HorlogeDigitale::HorlogeDigitale(QWidget *parent) : QLCDNumber(parent)
{
    // initialise la valeur du compteur de tic d'horloge
    m_valeur = 0;

    // instancie le timer (base de temps) de l'horloge
    m_timer = new QTimer(this);

    // connecte le signal d'expiration (timeout) d'une période (top d'horloge) au slot tic()
    connect(m_timer, SIGNAL(timeout()), this, SLOT(tic()));

    // personnalise l'afficheur
    //setStyleSheet("background-color: black; color: green;");
    #if QT_VERSION >= 0x050000
    setDigitCount(5); /* Qt5 */
    #else
    setNumDigits(5); /* Qt4 */
    #endif
```

```
    setSegmentStyle(QLCDNumber::Filled);
    setFixedSize(this->width()*1.5, this->height()*1.5); // agrandit sa taille
}

HorlogeDigitale::~HorlogeDigitale()
{
    qDebug() << "~HorlogeDigitale()";
}

long HorlogeDigitale::getMinute()
{
    return (m_valeur%36000)/600;
}

long HorlogeDigitale::getHeure()
{
    return m_valeur/36000;
}

void HorlogeDigitale::fixer(int h, int m)
{
    m_valeur = h*36000 + m*600;
}

void HorlogeDigitale::update()
{
   QString heure, minute;

   // met à jour l'affichage de l'horloge
   if (getHeure() < 10)
      heure = "0" + QString::number(getHeure());
   else heure = QString::number(getHeure());
   if (getMinute() < 10)
      minute = "0" + QString::number(getMinute());
   else minute = QString::number(getMinute());

   QString text = heure + ":" + minute;
   display(text);
}

void HorlogeDigitale::tic()
{
    // incrémente le compteur de top d'horloge
    m_valeur++;
    // demande la mise à jour l'affichage de l'horloge
    update();
}

void HorlogeDigitale::demarrer(int top/*=PERIODE*/)
{
    qDebug() << "HorlogeDigitale::demarrer()";
    m_timer->start(top);
}
```

```
void HorlogeDigitale::arreter()
{
    m_timer->stop();
    qDebug() << "HorlogeDigitale::arreter()";
}
```

horlogedigitale.cpp

On termine en modifiant la fonction `main()` (**cf. cours**).

Question 5. Compléter le fichier `main.cpp` afin d'y intégréer la gestion de l'encodage des caractères adaptés aux besoins (ici "UTF-8").

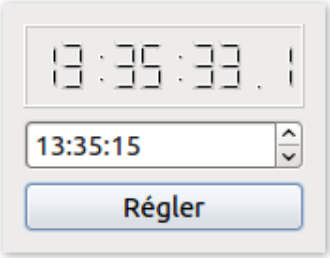
```
#include <QApplication>
#include <QTextCodec>
#include "ihmhorloge.h"

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

    // Choix de l'encodage des caractères
    #if QT_VERSION >= 0x050000
    QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8")); /* Qt5 */
    #else
    QTextCodec::setCodecForCStrings(QTextCodec::codecForName("UTF-8"));
    QTextCodec::setCodecForTr(QTextCodec::codecForName("UTF-8"));
    QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8")); /* Qt4 */
    #endif

    IHMHorloge w;

    w.show();

    return a.exec();
}
```

main.cpp

Étape 5 : Test

Question 6. Exécuter et tester l'application.

Question 7. Observer les appels des destructeurs et conclure sur la gestion de la mémoire par Qt.

Étape 6 : Évolution

Question 8. Modifier l'application existante pour y intégrer la gestion et l'affichage des secondes (et éventuellement des dizièmes). Tester.