TP 2
Calculatrice

Durée : 2 séances
Objectifs : Utiliser les gestionnaires de disposition (*layout*), une barre de menu et les raccourcis.

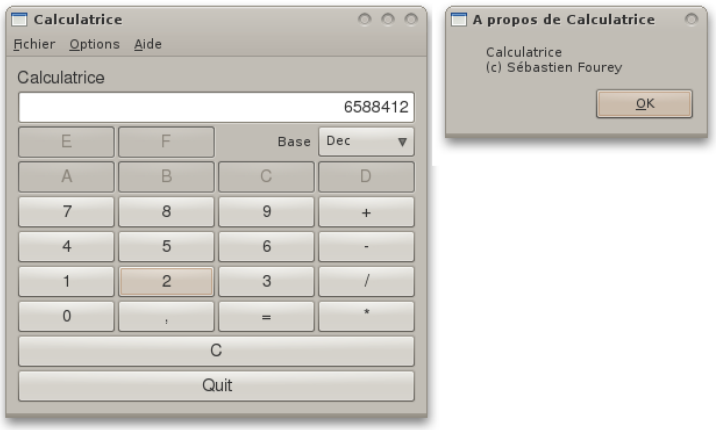
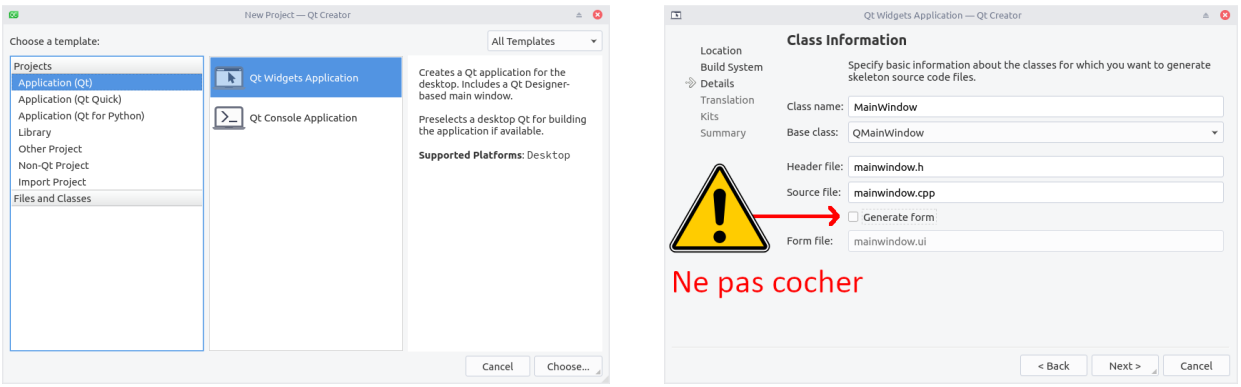


FIGURE 1 – Capture d'écran de l'application à réaliser.

Dans ce TP vous allez utiliser plusieurs classes très utiles dans tout projet Qt : label, menu, action, bouton, liste déroulante, gestionnaire de disposition. Vous ne vous soucierez que de l'interface d'une calculatrice. Le fonctionnement des opérations sera assuré par une classe qui vous est fournie dont l'utilisation vous est présentée en section 1.3.

Vous pouvez créer un nouveau projet depuis le menu « *File* / *New file or project* ». Choisissez ensuite « *Application (Qt)* / *Qt Widget Application* » (figure 2(a)) Pour ce TP, vous n'allez pas utiliser l'outil *Qt-Designer* qui permet de concevoir visuellement une interface [1]. Aussi, dans l'assistant de création du projet, à l'étape 3, pensez bien à *décocher* l'option « *Generate form* » (figure 2(b)).



(a) (b)

FIGURE 2 – Création d'un projet vide basé sur la classe QMainWindow.

1. Ceci fera l'objet du prochain TP. Pour le moment, nous cherchons encore à maîtriser les bases qui nous permettront d'utiliser efficacement cet outil.

1 Description de l'application

Le widget principal de cette application (figure 1) héritera de la classe QMainWindow. Cette classe possède (par construction) une barre de menu (QMenuBar) que l'on équipera d'un menu « Fichier » avec l'option « Quitter », ainsi que d'un menu « Aide » avec une option « A propos » déclenchant l'ouverture d'une boite de dialogue. Le menu « Options » sera évoqué dans la section 2.

Méthodes utiles :

`QMainWindow::menuBar()`	Retourne (un pointeur sur) la barre de menu.
`QMenuBar::addMenu()`	Ajoute et retourne un QMenu (menu déroulant) à partir d'un intitulé.
`QMenu::addAction()`	Ajoute et retourne une entrée dans un menu, sous forme de QAction.
`QAction::triggered()`	Signal émis par une action quand elle est activée.
`QMessageBox::about()`	Méthode statique qui permet d'afficher un dialogue.

Les autres widgets seront insérés au sein d'un QWidget simple. Vous ferez de ce widget le widget central de la fenêtre principale (voir ce schéma ainsi que la méthode `QMainWindow::setCentralWidget()`). La disposition des widgets contenus dans ce widget central sera gérée à l'aide d'objets *gestionnaires de disposition* de types QVBoxLayout et QGridLayout (voir figure 3(a)).

Conseil : vous veillerez pour la création des boutons à éviter les longues séquences d'instructions qui se ressemblent en privilégiant les boucles. Notamment, les adresses des boutons pourront être stockés dans un tableau selon l'ordre donné par le type énuméré CalculatorModel::ButtonID et les intitulés des boutons pourront avantageusement être placés dans le même ordre dans une chaîne de caractère.

1.1 Fonctionnement des gestionnaires de disposition

Un objet *gestionnaire de disposition* (dérivé de QLayout) est créé et associé à un widget père destiné à contenir d'autres widgets (figure 2(b)). Les widgets sont alors créés classiquement en précisant leur widget père lors de la construction. Afin que le gestionnaire de disposition prenne en charge la géométrie de ces widgets, il faut les enregistrer auprès du gestionnaire. Enfin, les gestionnaires peuvent être imbriqués.

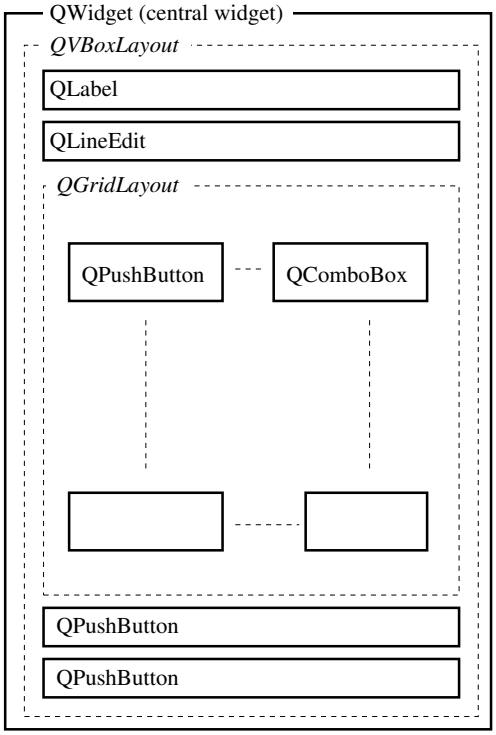
Méthodes utiles :

`QWidget::setLayout()`	Précise quel est le gestionnaire de disposition du contenu d'un widget. Inutile si le widget a été précisé comme *parent* à la construction d'un gestionnaire de disposition (cf. cours).
`Q(Grid|VBox)Layout::addWidget()`	Enregistre un widget pour qu'il soit pris en charge par le gestionnaire. Voir les multiples versions de cette méthode (ajout séquentiel pour un VBox ou avec ligne/colonne pour un Grid).
`Q(Grid|VBox)Layout::addLayout()`	Imbrique un gestionnaire de disposition dans un autre.

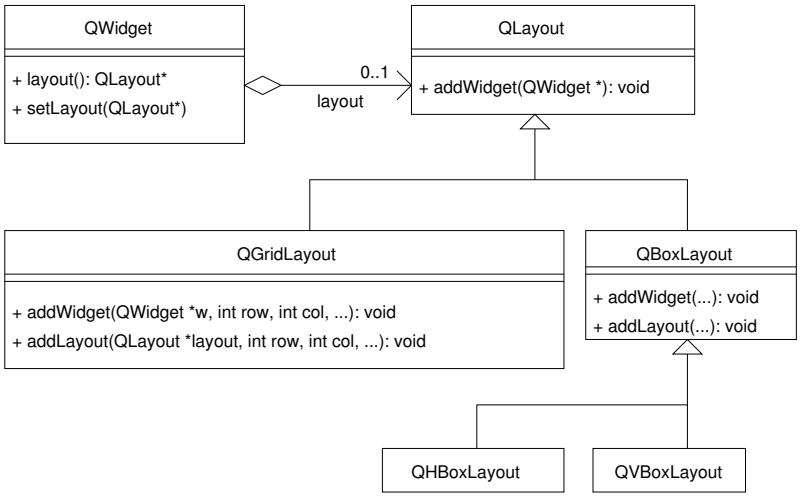
Il vous est conseillé de bien étudier la figure 3(a) et la figure 1 pour comprendre comment organiser les widgets (représentés en traits pleins) et les gestionnaires de disposition (représentés en traits pointillés).

1.2 Gestion des événements liés aux boutons

Un QPushButton émet un signal clicked() lorsqu'il est appuyé puis relâché. Afin de répondre aux événements de tous les boutons, une solution consisterait à utiliser autant de slots que de boutons, ou bien à connecter tous les signaux à un même slot en récupérant l'objet émetteur via la méthode QObject::sender().



(a) Disposition dans le widget central de l'application. Widgets (en traits pleins) et gestionnaires de dispositions (en pointillés).



(b) Les gestionnaires de disposition.

FIGURE 3 – Disposition

Ici, nous choisirons une troisième solution qui consiste à enregistrer les boutons auprès d'un objet de la classe QButtonGroup. Cet objet, qui n'est pas un widget, se connecte de lui-même aux signaux clicked() des boutons qu'il connaît pour n'émettre qu'un signal idClicked(int id) paramétré par le numéro du bouton à l'origine du signal (son *id*). Ce numéro est choisi lors de l'insertion du bouton dans le groupe. D'un point de vue pratique, il sera souhaitable d'utiliser comme identifiant d'un bouton l'entier correspondant à l'énuméré ButtonID défini dans la classe CalculatorModel (section 1.3, voir le fichier `CalculatorModel.h`). De cette façon, il sera immédiat de faire le lien entre les boutons de l'interface et le paramètre de la méthode `CalculatorModel::command()`.

On autorisera aussi l'utilisation de la calculatrice à l'aide du clavier en paramétrant correctement les boutons.

Méthodes utiles :

`QButtonGroup::addButton()`	Associer un bouton à un identifiant entier (*id*).
`QButtonGroup::idClicked()`	Signal émis par le groupe au nom d'un bouton (Qt ≥ 5.15).
`QButtonGroup::buttonClicked()`	Signal émis par le groupe au nom d'un bouton (Qt < 5.15).
`QPushButton::setShortcut()`	Affecter un raccourci clavier à un bouton.

1.3 Modèle à utiliser

Si ce n'est déjà fait, téléchargez sur Moodle l'archive contenant les deux fichiers CalculatorModel.h et CalculatorModel.cpp qui implémentent les opérations d'une calculatrice 4 opérations. Vous pourrez utiliser cette classe comme modèle [2] chargé des opérations effectuées par la calculatrice.

2. Au sens des différents patrons d'architecture qui en comportent un, comme MVP, MVC, etc.

Le fichier CalculatorModel.h définit le type énuméré `ButtonID` pour représenter toutes les touches d'une calculatrice 4 opérations. La classe CalculatorModel définit la méthode command(ButtonID) qui implémente les opérations, ainsi que la méthode getText() pour consulter le texte à afficher à l'écran. La méthode command() ne retourne true que si cette commande a pour effet de modifier le texte à afficher à l'écran.

Il suffit donc finalement de réagir dans un slot au signal du groupe de bouton, d'appeler la méthode CalculatorModel : : command avec l'identifiant du bouton, de consulter le texte à afficher et enfin de modifier l'afficheur en conséquence (QLineEdit). Votre calculatrice est alors fonctionnelle !

2 Travail à effectuer

2.1 Programmez la calculatrice, dans un premier temps sans vous soucier de la gestion des bases de numération (le modèle utilise par défaut la base 10).

2.2 Ajoutez une « ComboBox » permettant de choisir la base de numération parmi binaire, décimale ou hexadécimale. Consultez pour cela le fichier `CalculatorModel.h` avec la méthode `setBase()` et le type énuméré correspondant. La sélection d'une base doit entre autres désactiver les boutons qui deviennent inutiles. Outre certains chiffres et lettres, c'est aussi le cas de la virgule en mode binaire ou hexadécimal (seuls les nombres entiers sont gérés par le modèle dans ces deux modes).

2.3 Ajoutez un choix de type « case à cocher »[3] dans un menu « Options » qui permet d'afficher à l'écran un suffixe correspondant à la base courante, sauf pour la base 10. Quand l'option est activée, le nombre 126 s'affichera `"1111110b"` en base 2 et `"7Eh"` en base 16 (*h* comme hexadécimal).
Attention : Cette option devra être gérée sans modifier la classe CalculatorModel !

2.4 Un bouton n'accepte qu'un raccourci. Pour pouvoir utiliser les touches « Return »[4] ou « = » pour afficher le résultat d'une opération, utilisez des instances de la classe QShortcut et leur signal activated().

3. Autrement dit, une action dont la propriété *checkable* est *true*.

4. Selon la documentation de Qt, la touche « Return » est équivalente à la touche « Enter » mais elle se situe sur le pavé numérique.