# INF1010

## *Programmation Orientée-Objet*

### Travail pratique #6

### Interfaces Utilisateurs et Exceptions

---

| | |
|---|---|
| **Objectifs :** | Permettre à l'étudiant de se familiariser avec le concept de modèle MVC par l'entremise de la librairie QT. |
| **Remise du travail :** | Lundi le 4 décembre 2017, 8h |
| **Références :** | Notes de cours sur Moodle et documentation QT http://doc/qt/io/ |
| **Documents à remettre :** | Les fichiers .cpp et .h complétés sous la forme d'une archive au format .zip. |
| **Directives :** | Directives de remise des Travaux pratiques sur Moodle<br>Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires.<br>Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe.<br>Veuillez suivre le guide de codage |

# Informations préalables

***La directive de précompilation « #ifndef »***

La directive de précompilation « #ifndef » signifie « if not defined » (si non défini). Comme le type de directive le laisse deviner, cette directive est évaluée avant la phase de compilation du code source. Dans les travaux pratiques, vous l'utiliserez dans les fichiers d'en-têtes (.h), pour éviter la double inclusion. Un fichier peut inclure deux fichiers d'en-tête, par exemple prenons deux fichiers a.h et b.h. Il se peut que a.h soit inclus dans le fichier b.h. On se retrouve alors à inclure deux fois le fichier a.h, ce qui entraînerait une erreur de compilation, car on ne peut définir deux fois la même classe. La directive « #ifndef » nous évite cette double inclusion. Pour utiliser la directive « #ifndef », il faut respecter la syntaxe suivante :

```
#ifndef NOMCLASSE_H

#define NOMCLASSE_H

// Définir la classe NomClasse ici

#endif
```

***La directive de précompilation « #include»***

La directive de précompilation pour l'inclusion de fichiers « #include »

1. #include <nom_fichier>
2. #include "nom_fichier"

Ce qui différencie ces deux expressions est l'emplacement où le fichier spécifié est recherché. Pour la seconde forme, le précompilateur commence tout d'abord par rechercher dans le même répertoire que le fichier compilé. Par la suite, il procède de la même manière que la première forme, c'est-à-dire dans des répertoires prédéfinis par l'environnement de développement intégré.

En résumé, lorsqu'on inclut un fichier source qui se trouve dans le projet, on utilise la seconde forme. Et lorsque l'on inclut un fichier qui provient d'une bibliothèque externe au projet, on utilise la première forme.

# Travail à réaliser

Le travail effectué dans ce TP continue celui amorcé par les TP précédents soit une application de gestion d'images en y intégrant les notions d'interfaces graphiques et de gestion des exceptions.

Les interfaces graphiques sont souvent conçues selon un modèle MVC, soit modèle vu contrôleur. Par contre, QT fonctionne sur une base simplifiée de ce modèle, le modèle vu. Le contrôleur est intégré aux différentes vues et il faut utiliser les fonctions propres à l'api QT pour les modifier.

Pour réussir ce TP il faudra vous familiariser avec les concepts de programmation événementielle. QT fonctionne avec des signaux et des « slots » permettant d'accepter ces signaux et de les traiter. La documentation complète de l'API QT est disponible en ligne.

Pour vous aider, des fichiers .cpp et .h vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes décrites plus bas. Les attributs ou méthodes qui ne sont plus nécessaires ont été supprimés. Et les méthodes à modifier vous ont été indiquées.

**ATTENTION : Tout au long du TP, assurez-vous d'utiliser les opérateurs sur les objets et non sur leurs pointeurs ! Vous devez donc déréférencer les pointeurs si nécessaires.**

**ATTENTION : Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.**

**ATTENTION : Vous devez utiliser les fichiers fournis pour ce TP.**

**Remarque : Pour plus de précision sur le travail à faire et les changements à effectuer, veuillez-vous référer aux fichiers .h. TOUTE fonction dans les fichiers .h doivent être implémentées.**

# Information supplémentaire:

## Classe InterfaceCommande (déjà complète)

Cette classe fournit une abstraction sur ce que devrait contenir une commande, soit une méthode Executer et une méthode AnnulerCommande.

## Fichier Commande.h (déjà complété)

Contiens des commandes pouvant être effectuées sur une image. Elles dérivent toutes de la classe InterfaceCommande et possède une fonction d'exécution, qui applique les modification, et une fonction AnnulerCommande, qui ramène l'image à son état précédent l'application de cette commande.

## Classe UndoRedo (déjà complétée)

Cette classe contient deux stacks pouvant contenir des commandes dérivant de la classe InterfaceCommande. Elle utilise des pointeurs intelligents shared_ptr pour ne pas à avoir à gérer la mémoire associée à l'allocation dynamique des commandes. Elle permet de faire des retours en arrière au cas où l'utilisateur se serait trompé et utilise le patron de conception commande.

# À Faire :

## Classe ExceptionMauvaisFormatImage

Cette classe représente une exception QT. Elle peut se propager à travers différents thread utilisateurs et doit être soulevée si la lecture d'une image d'un format inconnu est tentée. Elle fonctionne comme n'importe quelle autre exception C++. Vous devrez rechercher comment les exceptions QT sont faites et créer les fonctions demandées. **Implémentez cette classe directement dans le .h.**

### Attributs :

**QString s_:** Attribut contenant le message de l'erreur.

### Méthodes à faire :

- **ExceptionMauvaisFormatImage :** Constructeur par paramètre prenant un QString et l'assignant à l'attribut de la classe.
- **raise :** Fonction void et ne prenant aucun paramètre. Cette fonction doit être implémentée par toute classe héritant de QException. **(voir la documentation de QException pour plus d'informations sur son implémentation).**
- **clone :** Fonction qui retourne un pointeur ExceptionMauvaisFormatImage* **(voir la documentation de QException pour plus d'informations sur son implémentation)**.

- **what :** Overload de la fonction what d'une QException. Cette fonction retourne l'attribut QString par copie et ne prend aucun argument.

# Classe *Image*

Cette classe servira de modèle à notre application. Elle définit toutes les fonctions permettant de faire des traitements sur l'image. Elle doit avoir un moyen de signifier à la vue qu'un changement sur l'image est survenu et donc qu'il faut qu'elle se mette à jour (le signal QT signal_changementImage).

## Attributs :

- QImage image_ : Une QImage permettant de lire, écrire et modifier une image.
- QFileInfo nomDuFichier_ : QFileInfo est un objet qui permet d'aller chercher des informations sur un fichier.
- UndoRedo undoRedo_ : undoRedo est un objet qui gère les Commandes (voir commande.h) du programme.

## Signales :

- **signal_changementImage :** Signal indiquant qu'un changement s'est effectué dans l'image. Ce signal ne retourne rien et prend en paramètre l'image qui a été modifiée.

## Méthodes à modifier:

- **ouverturDeImage :** Cette fonction permet d'ouvrir une image sur le disque et de la stocker dans un objet QImage, soit l'attribut image_. Si l'image ne peut pas être ouverte, vous devez lancer une exception ExceptionMauvaisFormatImage(QString) avec le message d'erreur déjà implémenté dans la fonction ouverturDeImage (à tester avec l'image Lasagne.jpg). Si l'image a réussi à être ouverte, vous devez émettre le signal de mise à jour de l'interface graphique.

## Slots à créer ou à modifier:

- **convertirEnNuanceDeGris :** Cette fonction est déjà implémentée et vous donne une idée de comment fonctionne la classe UndoRedo ainsi que les classes dans le fichier commandes.h. L'image doit se modifier dans l'affichage du programme une fois cette fonction terminée.
- **convertirEnNoirEtBlanc :** Vous devez compléter cette fonction en suivant l'implémentation de convertirEnNuanceDeGris. L'image doit se modifier dans l'affichage du programme une fois cette fonction terminée.
- **mettreEnNegatif :** Vous devez compléter cette fonction en suivant l'implémentation de convertirEnNuanceDeGris. L'image doit se modifier dans l'affichage du programme une fois cette fonction terminée.
- **undo :** Cette fonction doit appeler undo de l'attribut undoRedo_ et modifier l'affichage de l'image dans l'interface.

- **redo :** Cette fonction doit appeler redo de l'attribut undoRedo_ et modifier l'affichage de l'image dans l'interface.

# Classe *MainWindow*

Cette classe est la classe principale du TP. C'est dans cette classe que se situent toutes les informations sur l'interface graphique. Il s'agit de la vue de notre application. Le widget principal que vous devrez créer est un pointeur intelligent sur un label QT (QPointer<QLabel>) qui définit un label sur l'endroit où doit s'afficher l'image.

## Attributs :

- Ui::MainWindow *ui : interface par laquelle les menus, boutons et actions crées avec le QTDesigner sont accessibles.
- Image image_ : Modèle d'une image et de ces fonctionnalités.
- QAction *actionOuvrir_ : Action Ouvrir du menu Fichier + Ouvrir.
- QAction *actionAPropos_ : Action À Propos du menu Aide + À Propos.
- QAction *actionEnregistrerSous_ : Action Enregistrer sous du menu Fichier + Enregistrer sous
- QAction *undo_ : Action undo du menu Edition + undo.
- QAction *redo_ : Action redo du menu Edition + redo.
- QGroupBox *groupBox_ : Groupe contenant les boutons.
- QPushButton *boutonNB_ : Bouton Noir et Blanc.
- QPushButton *boutonGris_ : Bouton Nuances de Gris.
- QPushButton *boutonNegatif_ : Bouton Négatif.
- QScrollArea *imageScrollArea_ : Portion de l'affichage avec barre de déroulement.
- QPointer<QLabel> imageLabel_ : Label contenant l'image.

## Méthodes à modifier:

- **MainWindow :** Constructeur de la fenêtre principale. Vous devez créer la variable imageLabel_, tout le reste est fait pour vous.
- **creeActions :** Cette fonction doit créer les actions de la barre de menu. Elle doit créer les actions **actionOuvrir_ (Open), actionEnregistrerSous_ (SaveAs), actionAPropos_ (About), undo_ (Undo) et redo_ (Redo)**. **Vous devez respectivement les nommer Ouvrir, Enregistrer sous, Undo, Redo et À Propos.** Cette fonction, en plus de créer ces actions, leur assigne les raccourcis clavier habituels **(voir la documentation QT pour ajouter les raccourcis usuels aux actions d'une barre de menue)**.
- **initUI :** Cette fonction initialise l'interface graphique et donne un accès plus facile aux boutons, actions et menus déclarés dans l'attribut ui. Vous devez **ajouter les menus Fichier, Edition et Aide**. Ces menus doivent contenir les actions appropriées (Fichier doit contenir Ouvrir et Enregistrer sous, Edition doit contenir Undo et Redo et Aide doit contenir À Propos). Vous devez aussi créer les boutons et les ajouter aux groupBoxLayout. Finalement, vous devez ajouter le code

permettant d'intégrer imageLabel_ (créer dans le constructeur) imageLabel_ au widget imageScrollArea_. C'est ce qui permettra d'afficher une image lorsqu'elle sera ouverte ou modifiée. **Si l'image est plus petite que l'interface, elle doit être centrée dans son l'espace scrollArea. (Avant de poser toutes questions par rapport à ce qui est demandé pour cette fonction, recherchez dans la documentation QT)**.

- **ouvrirImage :** Vous devez ajouter la portion qui gère l'exception pouvant être lancée par la fonction ouvertureDeImage de la classe Image. **Vous ne devez gérer que cette exception et aucune autre, vous ne devez pas être en mesure d'attraper des exceptions d'autre type que celle lancée par la fonction de la classe Image.** Si l'exception est attrapée, vous devez envoyer le message d'erreur à la fonction afficherMessage.
- **initConnections :** Vous devez ajouter toutes les connections de tous les boutons, actions ou menu disponible dans l'application.
- **afficherMessage :** La fonction doit créer une QMessageBox et afficher le message contenu dans la variable qu'elle reçoit.
- **activateButtons :** La fonction doit activer les boutons lorsque la première image est reçue et pas avant (on ne peut modifier une image qui n'existe pas).

### Slot :

- **updateImage :** La fonction reçoit une QImage par référence, vous devez modifier l'imageLabel_ de manière à ce qu'il modifie ses pixels par ceux de la nouvelle image reçue par copie (de cette manière, si le modèle fait des changements, l'interface ne sera modifiée que lorsqu'il aura terminé et les deux threads ne compétitionnent pas pour les accès mémoires). La première fois, les boutons d'actions doivent être activés, car cela signifie qu'une image a été ouverte et donc qu'il est possible de mener des actions sur celle-ci.
- **updateTableViewWidget :** Fonction qui update le QTableWidget. Dois afficher le nom de l'image (**seulement le nom, pas le chemin complet, voir la documentation sur le QFileInfo de la classe Image**), le nombre de pixels en largeur et le nombre de pixels en hauteur **(voir la documentation de QImage)**.

### Autres méthodes et Slot présente À NE PAS TOUCHER :

- **fileExplorer :** Fonciton permettant de créer une boîte de dialogue (QFileDialog) de type explorateur de fichier permettant d'aller chercher la position d'une image sur le disque. C'est pourquoi vous devez tester votre implémentation de l'exception par l'ouverture de l'image Lasagne.jpg qui n'est en fait pas une image, bien que son extension soit .jpg. Cela ne devrait pas permettre à la fonction ouvertureDeImage de la classe Image de générer une QImage et donc, l'exception sera lancée et une QMessageBox doit s'afficher.
- **enregistrerImage :** Fonction qui permet d'enregistrer une image sur le disque avec le nom qui lui a été fourni.

# Main.cpp

Les directives principales sont dans MainWindow et non dans le main.

# Spécifications générales

• Ajouter un destructeur pour chaque classe chaque fois que cela vous semble pertinent
• Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs
• Ajouter le mot-clé const chaque fois que cela est pertinent
• Documenter votre code source
• Note : Lorsqu'il est fait référence aux valeurs par défaut, pour un string cela équivaut à la chaîne vide, et pour un entier au nombre 0

Correction

La correction du TP se fera sur 20 points.

Voici les détails de la correction :

- (3 points) Compilation du programme ;
- (3 points) Exécution du programme ;
- (4 points) Comportement exact
- (4 points) Gestion adéquate des signaux QT
- (1 points) Gestion des exceptions;
- (1.5 points) Utilisation adéquate des widgets QT (Boîte de message, TableWidget)
- (1.5 points) Documentation du code ;
- (2 points) Allocation / désallocation appropriée mémoire