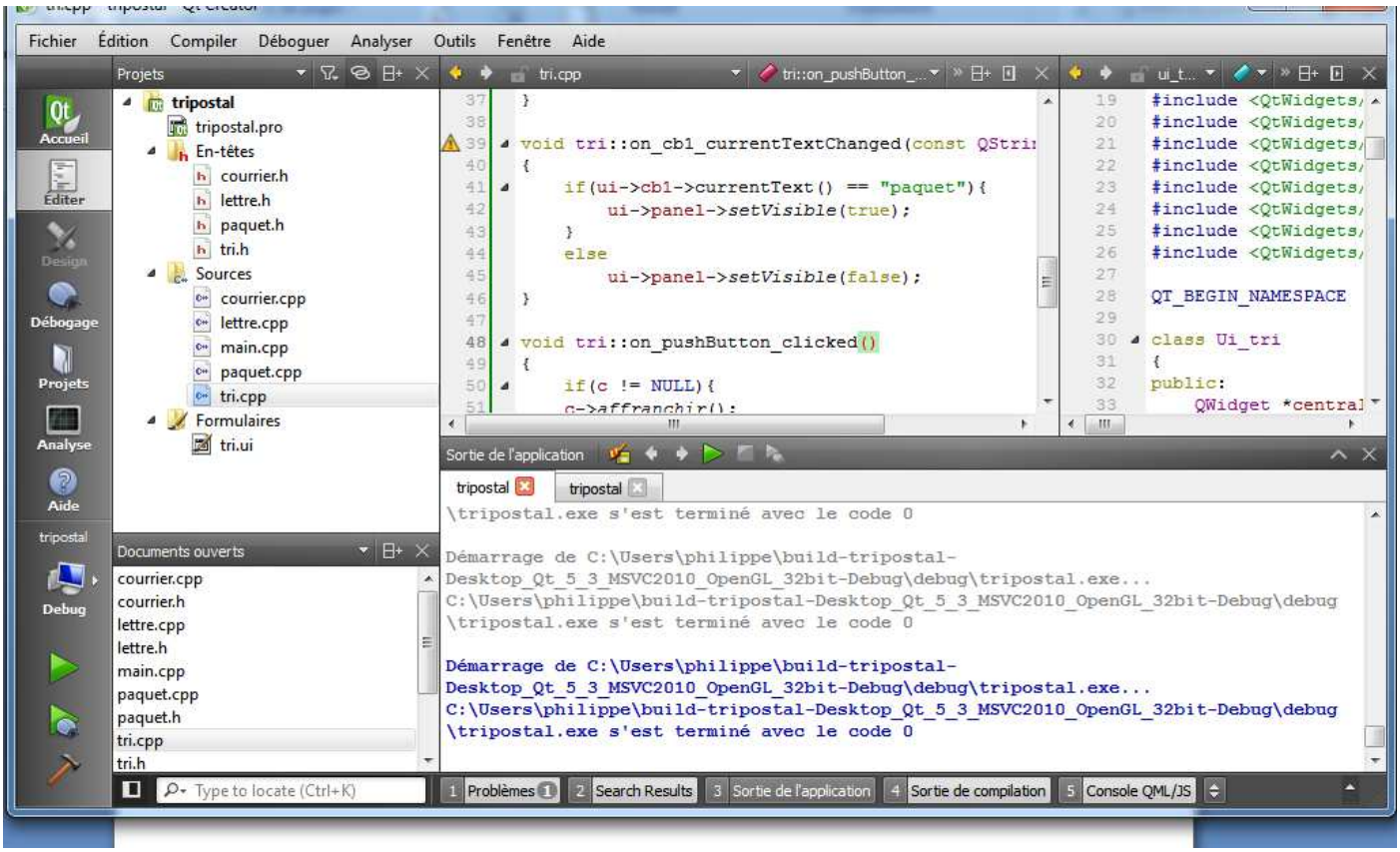
Présentation de QT

1] Présentation

QT est un IDE libre permettant de réaliser des IHM en utilisant des composants visuels nommés WIDGETS.
QT est disponible sous les plateformes window, linux, macos ce qui rend son code portable d'un OS à un autre.
Il supporte le C++, gère des projets et non pas des sources indépendamment les unes des autres.
Il possède aussi un ensemble de classes adaptées aux problématiques courantes tel que : communications réseaux, gestion de bases de données, communication série…
Les IHM se composent de composants graphiques manipulés graphiquement et codés sous forme de classe C++.



L'environnement, une fois ouvert et le projet créé, montre :

- ⇨ Une barre de bouton permettant de :
  - o Editer les fichiers sources afin d'écrire le code C++
  - o Modifier le graphisme de l'application (Design).
  - o Débuguer une application.
  - o Démarrer l'exécution de l'application.
- ⇨ Une visualisation de l'arborescence du projet
- ⇨ Une visualisation des fichiers ouverts
- ⇨ Une zone d'édition permettant :
  - o la modification des sources C++ du fichier sélectionné : mode **Editer**
  - o la modification de l'IHM sélectionnée : mode **Design**
- ⇨ en mode **Design**, une zone de visualisation/édition des propriétés du composant sélectionné.
- ⇨ une visualisation de la classe gérant les composants graphiques de l'IHM
- ⇨ la visualisation des résultats d'exécution ou de compilation.

2] Structure d'un projet

QT gère les applications sous forme de projet. Lorsque celui-ci est créé, QT crée automatiquement les fichiers suivants :

- ⇨ un répertoire au nom du projet créé : **nomProjetUtilisateur**
  - o une classe nommée par l'utilisateur lors de la création du projet représentant la fenêtre principale du projet.
    - ▪ **nomClasseIHMPrincipale**.cpp
    - ▪ **nomClasseIHMPrincipale.h**
  - o une classe codant l'ensemble des composants graphiques de l'application.
    - ▪ **ui_ nomClasseIHMPrincipale.h**

- un fichier **nomClasseIHMPrincipale.ui** définissant les propriétés des composants graphiques (XML)
- un fichier main.cpp permettant de démarrer l'application

```
#include "tri.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);   //création d'un objet QApplication
    tri w;                        //création de la fenêtre principale
    w.show();                     //visualisation de la fenêtre
    return a.exec();
}
```

⇨ un fichier **nomProjetUtilisateur.pro** makefile permettant la compilation et l'édition de liens de l'application.

```
QT       += core gui
QT += serialport

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = tripostal

SOURCES += main.cpp\
        tri.cpp \
    courrier.cpp \
    lettre.cpp \
    paquet.cpp

HEADERS  += tri.h \
    courrier.h \
    lettre.h \
    paquet.h

FORMS   += tri.ui
```

4] Composant widget basiques

Pour une documentation complète sur les classes QT, http://qt-project.org/doc/qt-5/classes.html

QT dispose d'un grand nombre de classe (composants graphiques et autres) rendant impossible une présentation exhaustive. Seules quelques classes vont être sommairement exposées.
Le nom des classes QT sont préfixées par Q. Toutes les classes sont héritées de la classe de base **QObject**.

Ces composants graphiques codés sous forme de classes C++ sont composés de membres :

- ⇨ les données membres, caractéristiques graphiques et comportementales
- ⇨ les méthodes.
- ⇨ Constructeurs et destructeur.
- ⇨ Les slots : gestionnaires d'événements : click, textChange …

Les composants décrits sont :

⇨ les zones d'édition :	QTextEdit
⇨ Les labels :	QLabel
⇨ Les fenêtres principales	QMainWindow
⇨ Les boutons :	QPushButton
⇨ Les combobox	QComboBox

4.1] QTextEdit

Saisie de texte :

- ⇨ Récupérer le texte méthode : `QString toPlainText()`
- ⇨ Ajouter du texte méthode : `void setText(QString & text )`
- ⇨ Convertir le texte en valeur numérique

- méthode : `float toPlainText().toFloat()`

4.2] QLabel

Affichage de texte :
- ⇨ Texte défini lors de la conception graphique
- ⇨ Modifier le texte en cours d'exécution :
  - **`setText(QString & text)`**
- ⇨ récupérer le texte du label.
  - **`QString Text()`**
- ⇨

4.3] QMainWindow

Fenêtre principale de l'application

4.4] QPushButton

Bouton clickable associé à un ou plusieurs gestionnaires d'événements **SLOTS**.
- ⇨ Accéder au texte du boutton
  - `QString Text( )`
- ⇨ Définir le code du gestionnaire d'événement : SLOT
  - **`void nomFenetre::on_nomBoutton_clicked(){…}`**

4.5] QComboBox

- ⇨ définir les items
  - en mode conception graphique
  - en cours d'exécution
    - `setItemText(int index , QString &texte);`
- ⇨ récupérer le texte de l'item sélectionné.
  - `QString currentText()`
- ⇨ Définir le slot lors du changement d'item sélectionné.
  - `void tri::on_cb1_currentTextChanged(const QString &arg1)`
  - arg1 représente le texte sélectionné.

5] La classe QString.

Cette classe permet de coder les textes utilisés par les composants graphiques de QT. Elle est munie des méthodes classiques de gestion de texte. Calcul de la taille, conversion en valeur numérique et inversement, récupération d'une lettre dans un mot, conversion en std ::string…
Les opérateurs de tests ont été redéfinis pour cette classe.

La conversion en char * doit être faite comme suit :

```
QString str1 = "Test";
QByteArray ba = str1.toLocal8Bit();
const char *c = ba.data();
```

6] Manipulation des composants QT

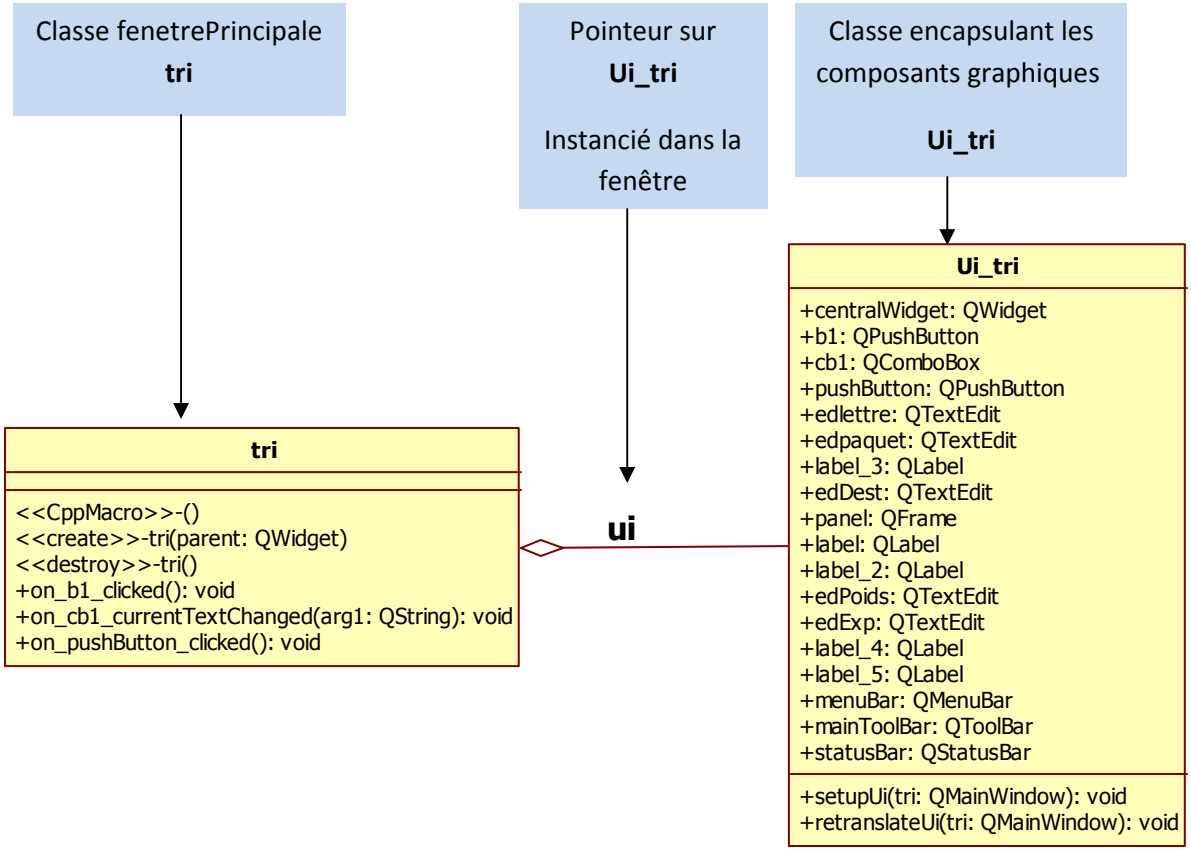
6.1] Utilisation des composants dans le sources C++ :

Pour utiliser ces composants, il faut inclure les fichiers d'entêtes correspondants (**pas de .h**)
- ⇨ **`#include <nomClasseDuComposant>`**

Chaque composant graphique est instancié par pointeur => utilisation de la flèche pour accéder aux membres du composant.

Les composants graphiques composant l'IHM, sont encapsulés par pointeurs dans une classe **ui_nomFenetre**. La classe **fenetrePrincipale** instancie un pointeur sur un objet de type **ui_nomFenetre** permettant l'accès aux composants graphiques.

Illustration :



Ce qui se traduit en C++ :

tri.h

```
class tri : public QMainWindow
{
    …

private:
//lien indirect vers la classe Ui_tri
    Ui::tri *ui;
};
```

ui_tri.h

```
class Ui_tri
{
public:
    QWidget *centralWidget;
    QPushButton *b1;
    QComboBox *cb1;
    QPushButton *pushButton;
    QTextEdit *edlettre;
…
```

Par conséquent, pour accéder à un membre d'un objet graphique à partir de la classe **tri**, il faut respecter la syntaxe du C++ :

Exemple : modifier le texte d'un label ***lab*** dans la classe **tri**.

```
tri::nomMethode(…) {
      ui->lab->setText("coucou");
      …
      }
```

6.2] Mise en œuvre des gestionnaires d'événements

Sous QT, les gestionnaires d'événements sont appelés SLOTS. Ils sont mis en œuvre via l'interface **Design**.
A partir des choix de l'utilisateur, QT les implémentent dans les sources C++ et sont utilisables comme des méthodes C++ classiques. Ces méthodes seront déclenchées en réponse à l'événement pour lequel elles ont été prévues.

Faire pour chaque slot :

- ⇨ Passer en mode design
- ⇨ Sélectionner le composant auquel on désire ajouter un slot.
- ⇨ Bouton droit sur le composant puis '**Aller au slot'**
    - o Choisir l'événement

- Valider

⇨ QT met en place la déclaration et le corps du slot.
⇨ Coder le slot.

6.3] Ecrire dans la console :

Afin de réaliser un débogage sommaire, il est possible d'écrire dans la console de QT en utilisant des méthodes prévues à cet effet. Ces méthodes sont **`qDebug(),qWarning() qCritical()`** et s'utilisent comme suit :

```
qDebug() << "Debug Message";
qWarning() << "Warning Message";
qCritical() << "Critical Error Message";
```

7] Ajout de fichier C++

On peut ajouter au projet des sources existantes ou bien créer de nouvelles classes. Il suffit de faire **fichier** => **nouveau**, choisir ce que l'on désire ajouter (classe, projet …) puis renseigner les champs proposés et enfin valider.
Dans ce cas, il est nécessaire de faire **compiler** => **exécuter make** afin de modifier le makefile pour prendre en compte les nouvelles classes à compiler et lier.

8] TP de prise en main :

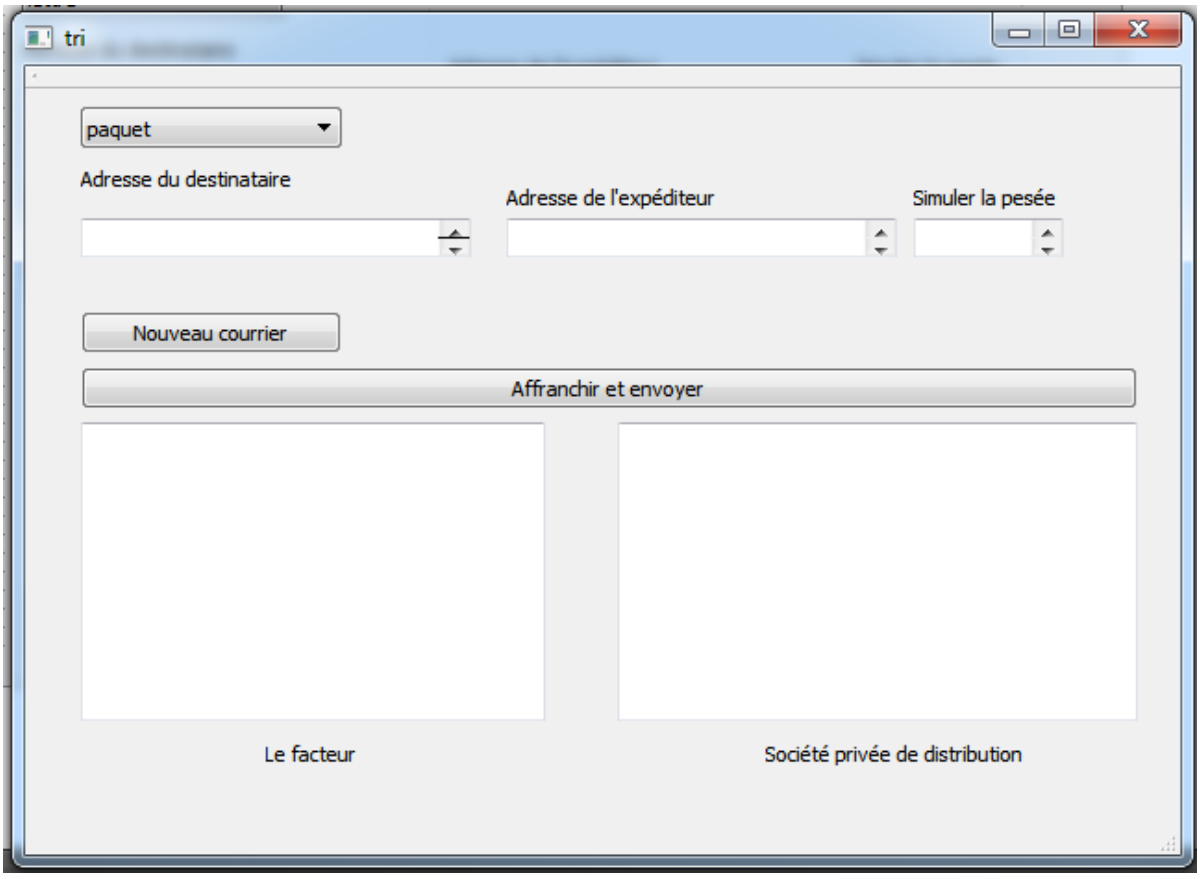
L'objectif est de réaliser une IHM simulant un tri postal qui consiste à envoyer du courrier soit via le facteur en cas de lettre normale soit via une entreprise privée dans le cas de paquet. L'IHM doit permettre de créer des lettres ou des paquets puis d'afficher leur caractéristiques (nom destinataire, expéditeur, tarif…) dans un QTextEdit approprié en fonction du type de courrier (lettre ou paquet). Il s'agit donc de réaliser la partie IHM sans traitement réel des courriers.

8.1] Télécharger et installer QT 5 pour window sur le site officiel de QT **http://qt-project.org/downloads** si ce n'est déjà fait.

8.2] Créer un projet nommé **TP1** dont la fenêtre principale sera nommée **wPrincipale**. L'enregistrer sur le bureau.

8.3] Vérifier l'arborescence du projet (lieu de stockage et présence de tous les fichiers correctement nommés.

8.4] En mode **Design**, réaliser l'IHM semblable à celle-ci :



Cette IHM est composée de :

- ⇨ Un combobox permettant de choisir le type de courrier :
  - o Lettre
  - o Paquet
- ⇨ Des labels configurés en mode conception.
- ⇨ Des zones d'éditions **QTextEdit** pour saisir les adresses destinataires (lettre) ainsi que l'expéditeur et le poids (paquet)
- ⇨ Deux zones d'éditions pour afficher les informations des lettres et paquets.
- ⇨ Deux boutons pour créer un courrier puis pour envoyer le courrier en fonction de son type.

Les noms des objets doivent être représentatifs de leur rôle et non pas les noms par défaut.

Gestionnaire d'événement : Slots

Le traitement réel des courriers est pour l'instant simulé par des affichages et modifications des valeurs visualisées:

A] Lorsque l'utilisateur modifie l'item du **combobox**, en fonction du choix le slot doit permettre de rendre visible ou non les composants permettant la saisie des renseignements du courrier de type paquet (adresse expéditeur et poids pour le calcul du tarif).

- ⇨ Slot associé au combobox gérant l'événement **onItemChange**
  - o Si item == lettre ALORS FAIRE cacher les composants inutiles
  - o Si item == paquet ALORS FAIRE montrer les composants nécessaires.

B] Lorsque l'utilisateur clique sur le bouton nouveau courrier le slot doit:

- ⇨ Modifier le texte du bouton : **courrier à envoyer**
- ⇨ Afficher dans la console du message : **Courrier créé.**

C] Lorsque l'utilisateur clique sur le bouton **Affranchir et envoyer**, le slot doit :

- ⇨ Si item sélectionné du combobox == lettre ALORS afficher dans le **QTextEdit** approprié '**lettre'.**
- ⇨ Si l'item sélectionné du combobox == paquet ALORS
  - o afficher dans le QTextEdit approprié **'paquet'.**
  - o Afficher dans la console le résultat de **l'opération poids*015**
- ⇨ Restituer le texte d'origine du bouton précédent : **nouveau courrier**.