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 :	Jeudi le 19 avril 2018, 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 Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires. Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. Veuillez suivre le guide de codage

Informations préalables

Installer QT Creator

https://download.qt.io/official_releases/qtcreator/

Set up le build du projet

Run qMake

Travail à réaliser

Le travail effectué dans ce TP continue celui amorcé par les TP précédents soit une plateforme de type *ebay* 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 vue contrôleur. Par contre, QT fonctionne sur une base simplifiée de ce modèle, le modèle vue. 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 et les classes 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.

Classe Usager

Il s'agit d'une classe abstraite qui possède les attributs et méthodes suivantes :

Attributs :

- Nom
- Prénom
- Identifiant
- Code Postal

Méthodes : Tous les 'obtenir' et 'modifier' de ses attributs.

Classe Client

Cette classe contient un constructeur pas défaut et un constructeur par paramètres.

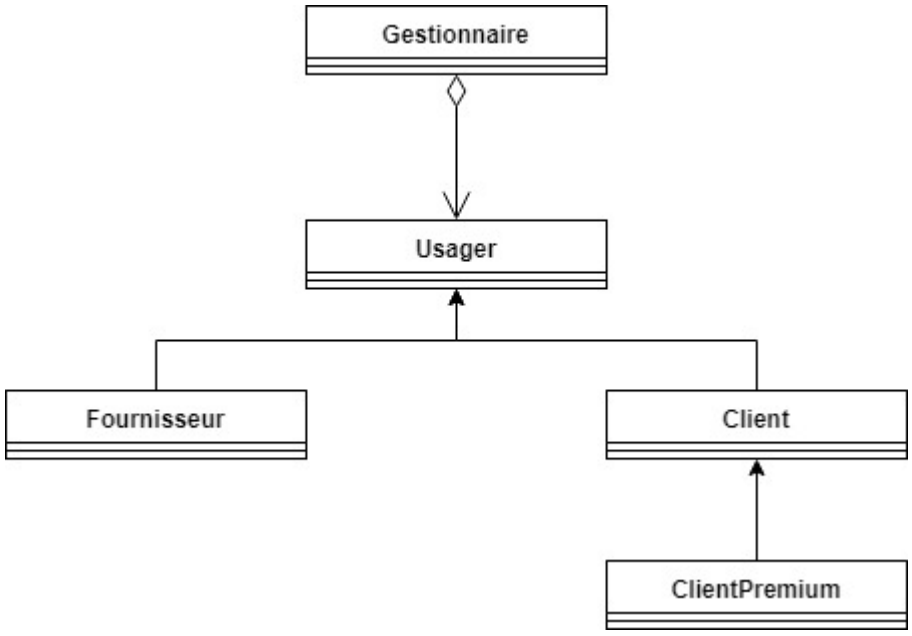
Classe Fournisseur

Cette classe contient un constructeur pas défaut et un constructeur par paramètres.

Classe ClientPremium

Cette classe contient un constructeur pas défaut et un constructeur par paramètres. Elle contient un attribut de plus, jours restants qui se réfère au nombre de jours restants à l'abonnement de l'usager.

Les classes sont liées selon la hiérarchie suivante :



Classe ExceptionArgumentInvalide

Cette classe représente une exception QT. Elle peut se propager à travers différents thread utilisateurs et doit être soulevée si on tente de créer un usager sans avoir complété tous les champs. Elle fonctionne comme n'importe quelle autre exception C++.

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

Méthodes :

- **ExceptionArgumentInvalide :** Constructeur par paramètre prenant un QString et l'assignant à l'attribut de la classe.
- **what :** Overload de la fonction what d'une QException. Cette fonction retourne l'attribut QString par copie et ne prend aucun argument.

Section à faire :

Classe *Gestionnaire*

Cette classe servira de modèle à notre application. Elle définit toutes les fonctions permettant de faire des traitements sur les usagers qu'elle contient. Elle doit avoir un moyen de signifier à la vue qu'un changement sur l'image est survenu, elle possèdera donc des signaux.

Attributs :

- Vecteur<Usager *> usagers_: Contient tous les usagers du gestionnaire.

Signaux :

- signal_usagerAjoute : Émet un signal lorsqu'un usager a été ajouté.

- signal_usagerSupprime : Émet un signal lorsqu'un usager a été supprimé.

Méthodes :

- **obtenirUsagers()** : retourne le vecteur des usagers_.

- **obtenirNombreUsager()** : retourne le nombre d'usager contenu dans le vecteur usagers_.

- **obtenirUsager(int index)** : Retourne l'usager de usagers_ à l'index spécifier.

Méthodes à implémenter:

- **ajouterUsager(Usager* usager) :** Méthode qui ajoute un usager dans le vecteur usagers_ et qui émet un signal qui indique qu'un usager a été ajouté.
- **supprimerUsager(Usager* usager) :** Méthode qui supprime un usager du vecteur usagers_ et qui émet un signal qui indique qu'un usager a été supprimé.

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. Des captures d'écrans sont fournis plus bas pour vous donner une idée de ce à quoi l'application devrait ressembler et interagir.

Attributs :

- gestionnaire_ : Il s'agit d'un objet de type gestionnaire avec des usagers de base. Il est créé dans le Main.
- ajoute_ : Il s'agit de la liste des usagers qui seront créés à partir de la vue.
- indexCourantDuFiltre_ : Il s'agit de l'index qui indique quel est le filtre courant qui s'applique sur la liste d'usager.

Attributs Locaux QObject :

- listUsager : Liste contenant les usagers qui sont affichés sur le panneau de gauche.
- editeurNom : Boîte de texte ou l'on peut insérer le nom d'un usager. Il se situe dans le panneau de droite.
- editeurPrenom : Boîte de texte ou l'on peut insérer le prénom d'un usager. Il se situe dans le panneau de droite.
- editeurIdentifiant : Boîte de texte ou l'on peut insérer l'identifiant d'un usager. Il se situe dans le panneau de droite.
- editeurCodePostal : Boîte de texte ou l'on peut insérer le code postal d'un usager. Il se situe dans le panneau de droite.
- editeurJoursRestants : Boîte de texte ou l'on peut insérer le nombre de jours restant à l'abonnement d'un client premium. Il se situe dans le panneau de droite.
- boutonRadioTypeUsager : Représente les boutons des trois types d'usager (Client, ClientPremium, Fournisseur). Ils se situent sur le panneau de droite.
- boutonAjouter : Bouton permettant d'ajouter un usager après avoir rempli tous les champs requis. Ce bouton se situe sur le panneau de droite, au bas.
- boutonSupprimer : Bouton permettant de supprimer un usager sélectionné dans la liste du panneau de gauche. Ce bouton se situe sur le panneau de droite, au bas.

Méthodes à modifier:

- **setConnections()** : Cette fonction contient deux connect. Le premier connecte le signal signal_usagerAjoute (Usager* u) du gestionnaire au slot usagerAEteAjoute(Usager* u). Le deuxième connecte le signal

signal_usagerSupprime(Usager* u) du gestionnaire au slot usagerAEteSupprime(Usager* u).

- **setMenu()** : Cette fonction est déjà complétée. Elle permet d'ajouter un objet QMenu nommé « Fichier » à la barre de navigation. Dans l'objet QMenu, on ajoute le bouton créé exit de type QAction qui ajoute une option nommée 'Quitter'. Cette option ferme la fenêtre de l'application par un connect entre le signal triggered et le slot close. Cette fonction implémente aussi un raccourci clavier pour quitter l'application (Ctrl + Q).
- **setUi()** : Cette fonction membre génère tous les éléments de la vue. Il s'agit d'une longue fonction dans lequel vous devez ajouter les éléments manquants de la vue aux endroits indiqués. Vous devez donc compléter la vue en ajoutant :
    - showCombobox de la classe QComboBox : le 'dropdown' qui permet d'afficher le type d'usager dans la liste des usagers. Il y a 4 filtres : Tout Afficher, Afficher Clients Reguliers, Afficher Clients Premiums, Afficher Fournisseurs. Pour distinguer chaque filtre, on utilisera la convention que Tout Afficher à l'index 0, Afficher Clients Reguliers l'index 1, Afficher Clients Premiums l'index 2, Afficher Fournisseurs l'index 3. Utiliser la méthode addItem() de la classe QComboBox. Il faut aussi faire un connect entre le signal currentIndexChanged et le slot filtrerListe.
    - boutonAjouterUsager de type QPushButton. Le connect entre le click() du bouton permet de nettoyer la vue en appelant la fonction slot nettoyerVue(). Indice : se calquer des instructions du bouton supprimerTousLesUsagers.
    - Ajouter dans l'objet listLayout, les objets showCombobox et boutonAjouterUsager.
    - Compléter listLayout en y ajoutant tous les éléments de vue (QComboBox et QPushButton) que vous venez de créer.
    - Champ pour l'identifiant de l'usager. Ce champ accepte seulement des nombres entre 0 et 100 000. Il faut imiter les instructions écrites pour les champs nom et prénom en ajoutant l'appel de la méthode setValidator pour la validation du nombre.
    - Champ pour le code postal.
    - Champ pour le nombre de jours restants. Ce champ accepte seulement des nombres entre 0 et 1000.
    - boutonSupprimer dans le panneau de droite. Sert à supprimer l'usager sélectionné dans la liste de gauche. Il faut faire appel à un connect qui lie le click du bouton avec le slot supprimerUsagerSelectionne. Indice : s'aider du bouton boutonAjouter.

    - Dans la colonne de droite, ajouter les 3 champs identifiant, code postal, joursRestants.

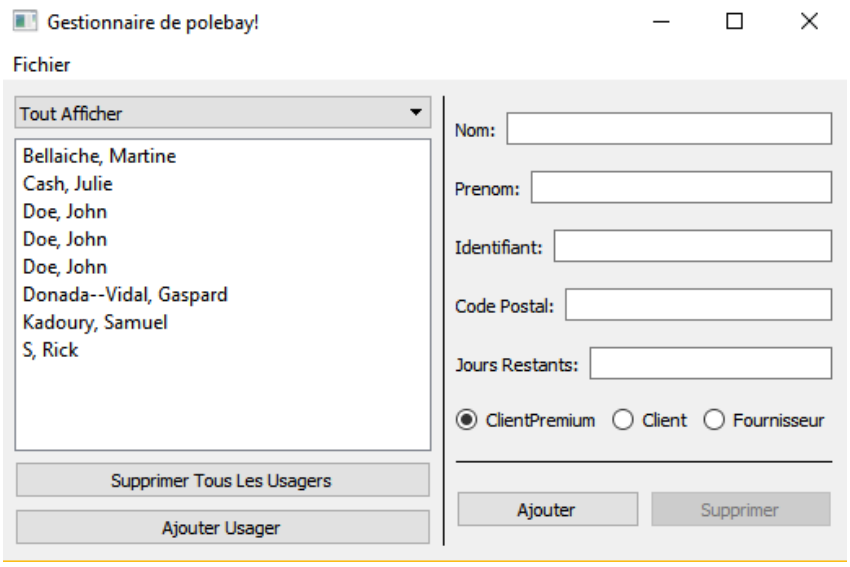
- **afficherMessage(QString msg)** : Cette fonction crée un QMessageBox qui 'pop up' à l'écran et affiche le message 'msg' passé en paramètre.
- **filtrerListe(int index)** : Cette fonction filtre la liste des usagers selon l'index donné en paramètre et affiche seulement les usagers qui correspondent à ce champ (convention : Tout Afficher à l'index 0, Afficher Clients Reguliers l'index 1, Afficher Clients Premiums l'index 2, Afficher Fournisseurs l'index 3). À l'aide d'une boucle, on parcourt le contenu de l'attribut listUsager, on stocke un item de la liste dans un objet de type QListWidgetItem. À chaque item, on appelle la méthode setHidden() qui prend comme paramètre la méthode filtrerMasque().
- **selectionnerUsager(QListWidgetItem* item)** : Il faut compléter cette méthode. En cliquant sur un usager, ces informations sont affichées dans les boîtes de texte de droite. Ces boîtes de texte sont mises à disabled (méthode setDisabled) pour éviter toute modification à l'usager sélectionné. En ce qui concerne le champ 'JoursRestants' il affiche les jours restants s'il s'agit d'un client premium, sinon il affiche : '-'.
- **supprimerTousLesUsagers()** : Cette fonction doit supprimer tous les usagers du gestionnaire. On fait appel à la méthode supprimerUsager() de l'attribut gestionnaire_.
- **supprimerUsagerSelectionne() :** Cette fonction supprime l'usager qui était sélectionné dans la liste. Cet usager sera supprimé du gestionnaire.
- **ajouterUsager()** : Cette fonction ajoute un usager au gestionnaire. Il sera créé et ajouté seulement si tous les champs de la vue ont été complété. On utilise donc un try throw catch dans la fonction avec une exception à message personnalisé pour chaque champ manquant. Par exemple, si le champs prénom n'a pas été rempli, le message d'erreur devrait être 'Erreur : Le prénom n'est pas rempli'.
- **usagerAEteAjoute(Usager* u)** : Cette fonction est appelée lorsqu'un nouvel usager a été créé. Cette fonction met à jour la vue avec l'usager nouvellement créé.

Référez-vous aux commentaires dans le code pour toute autre fonction non documentée ici.

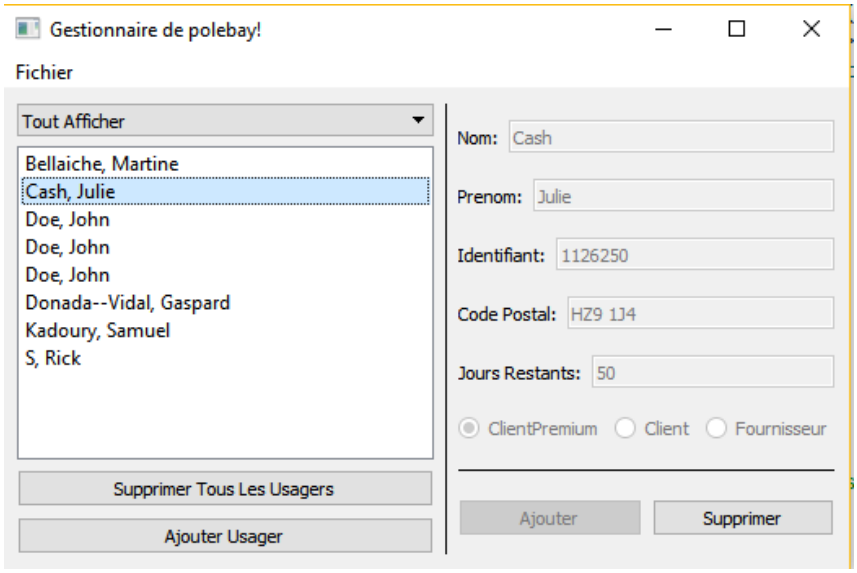
Main.cpp

Un gestionnaire est créé dans le Main. C'est les usagers qui s'affichent à l'exécution. Il n'y a rien à modifier ici.

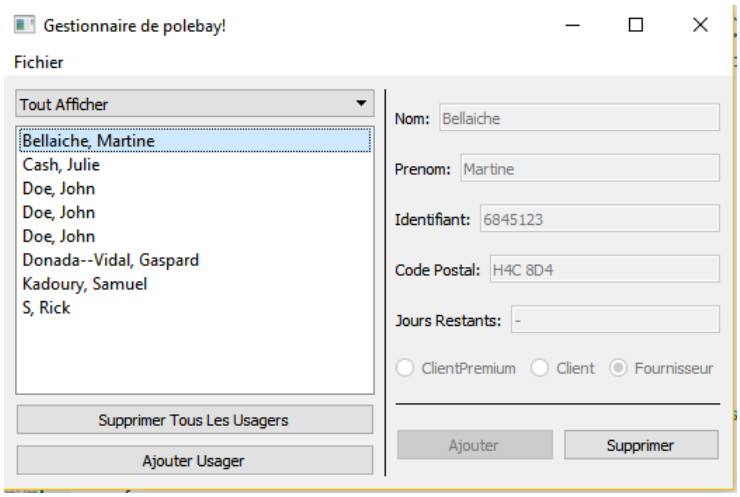
Apparence au démarrage du programme



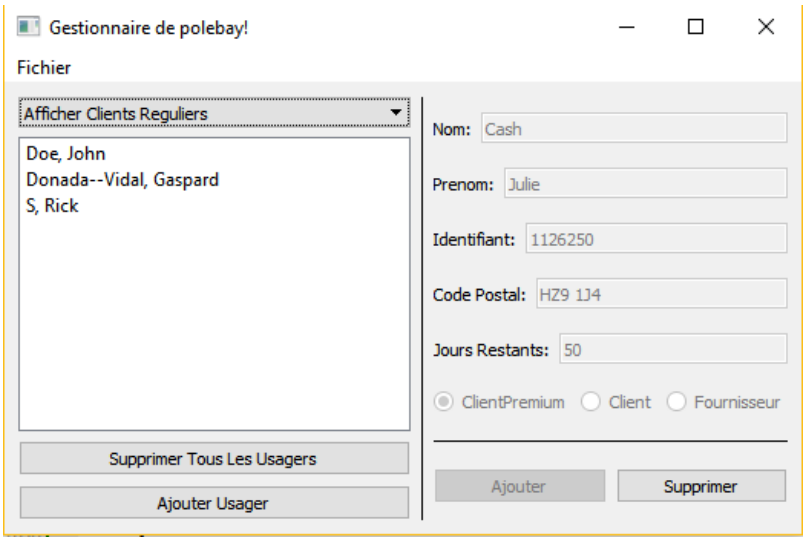
Affichage d’un client premium



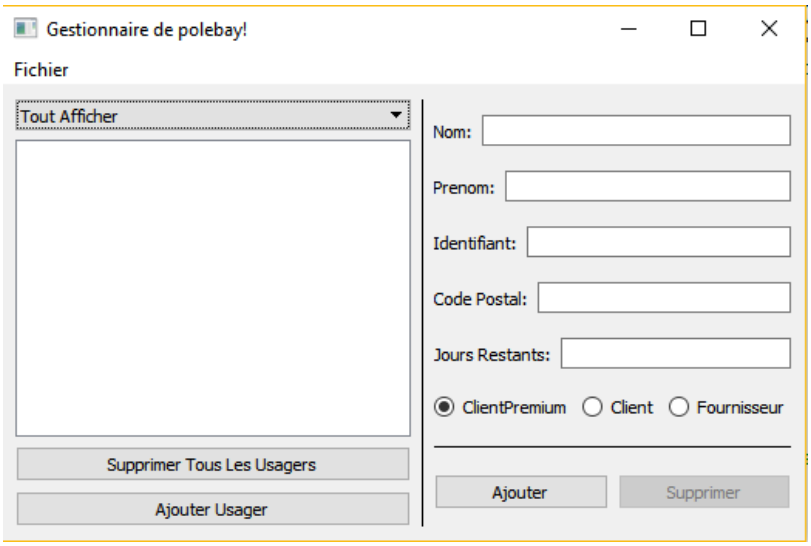
Affichage d'un fournisseur. L'affichage d'un client (régulier) est identique



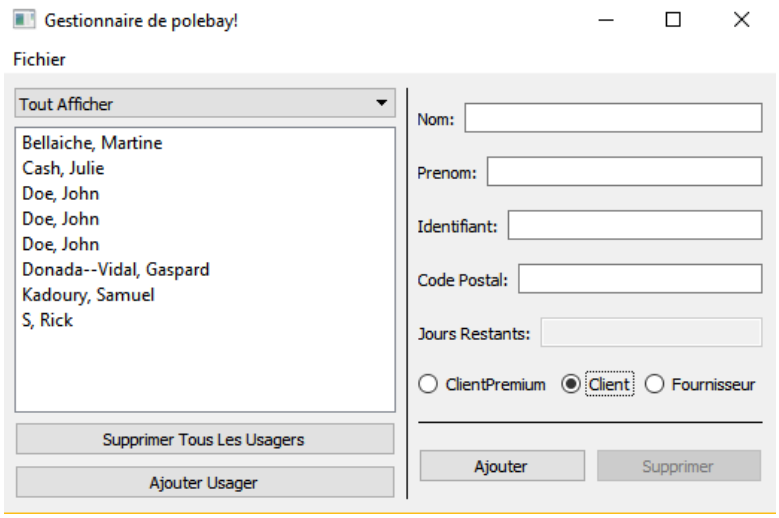
Affichage de tous les clients réguliers uniquement.



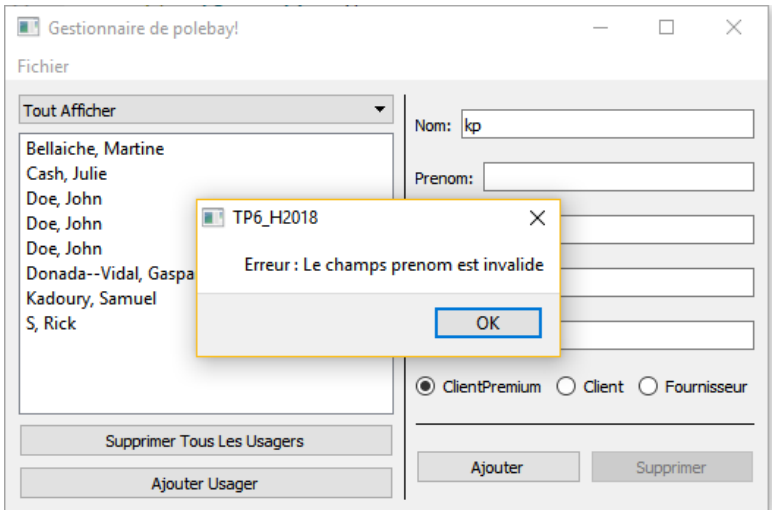
Affichage après avoir cliqué sur le bouton 'Supprimer tous les usagers'



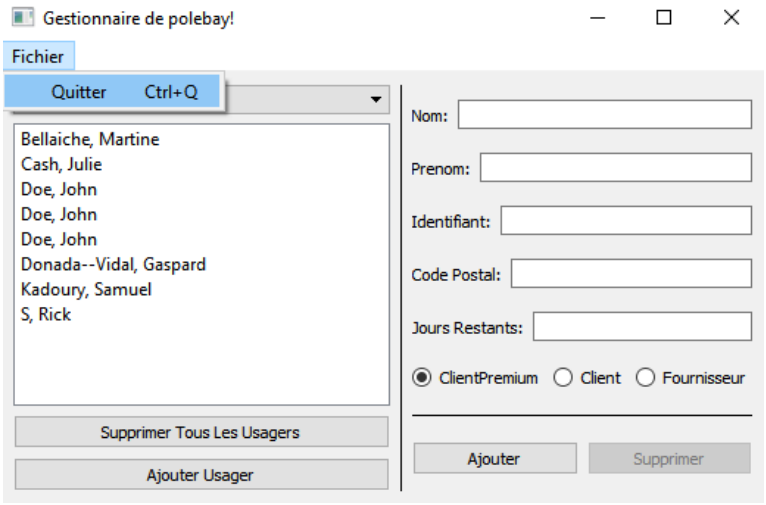
Affichage lors de l'ajout d'un usager qui n'est pas un client premium. Remarquez l'apparence de la boîte 'JoursRestants'.



Erreur produite lorsque l'on clique sur 'Ajouter' mais qu'il reste des champs vides.



Affichage du menu principal avec l'option Fichier contenant la méthode Quitter avec raccourci clavier Ctrl + Q



• 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