

TD4 - Socket sous Qt

Banque - Côté serveur



Code less.
Create more.
Deploy everywhere.

-
- - Date : novembre 2022
 - Version : 2
 - Référence : TD4 – Socket sous Qt (Banque Serveur).odt

1. Objectif

- Utilisation des sockets avec la bibliothèque Qt
- Comprendre le fonctionnement des sockets en mode événementiel.
- Codage de l'information
- Communication réseau

2. Conditions de réalisation

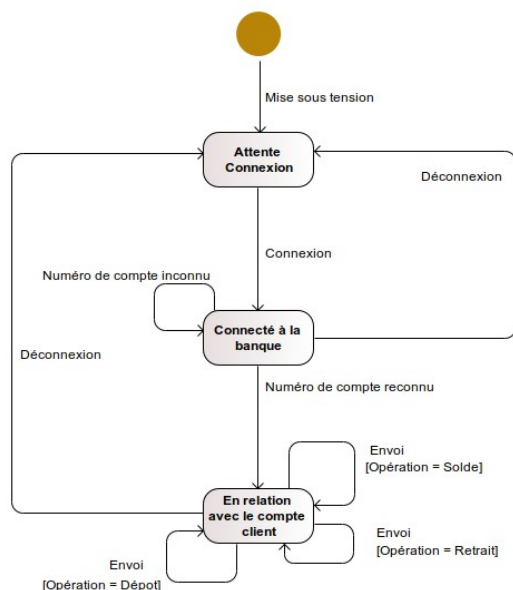
- Ce fichier contient des liens hypertextes.
- Ressources utilisées :
 - Un PC sous Linux
 - Un client est disponible
- Qt-creator

3. Ressources

Les classes sockets dans la technologie QT, consulter le site <https://doc.qt.io/qt-6/qtnetwork-programming.html> et plus particulièrement la classe, **QTcpServer** <https://doc.qt.io/qt-6/qtcpserver.html>.

4. Rappel du besoin

Les distributeurs automatiques de billets (DAB) permettent de se connecter à une banque. Ils offrent la possibilité d'interroger un compte pour en connaître le solde, de déposer de l'argent et d'en retirer. Les fonctionnalités attendues côté client sont décrites par le diagramme d'état du système suivant :



Le diagramme à états ci-contre indique que c'est seulement lorsque le client est connecté à sa banque et que son numéro de compte est reconnu que les opérations vers la banque peuvent être réalisées. La déconnexion reste possible à chaque état.

4.1. Rappel du protocole d'échange entre les clients et le serveur

La trame en provenance du client pour sélectionner un numéro de compte :

Taille des données	Commande	Valeur
De type quint16	'N' - Un caractère sous la forme d'un QChar	Un entier de type int

La trame en provenance du client pour effectuer un retrait :

Taille des données	Commande	Valeur du retrait
De type quint16	Pour le retrait : 'R' - Un caractère sous la forme d'un QChar	Un réel de type float

La trame en provenance du client pour effectuer un dépôt :

Taille des données	Commande	Valeur du dépôt
De type quint16	Pour le dépôt : 'D' - Un caractère sous la forme d'un QChar	Un réel de type float

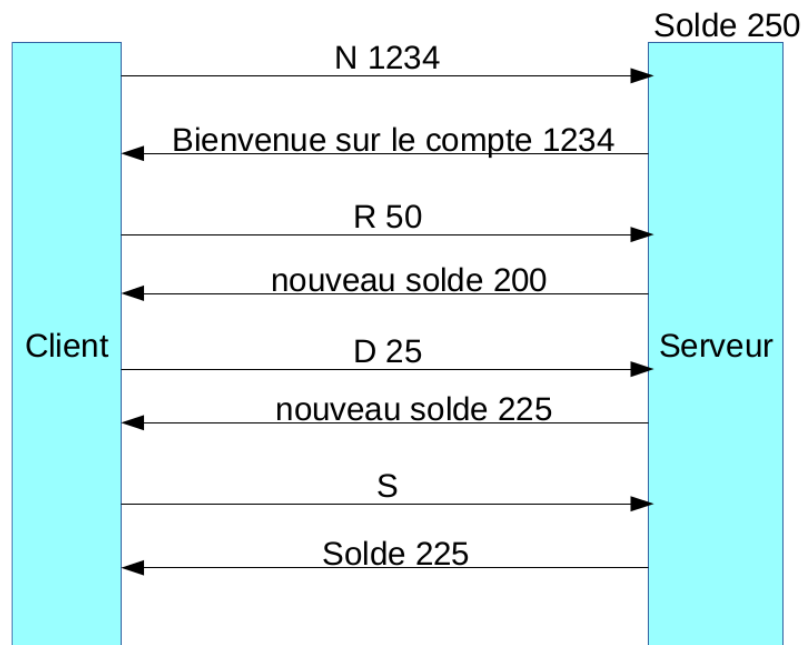
La trame en provenance du client pour obtenir le solde du compte :

Taille des données	Commande	Valeur
De type quint16	Pour le solde : 'S' - Un caractère sous la forme d'un QChar	Sans objet

Dans tous les cas de figure, le serveur répond un message sous la forme

Taille des données	Message
De type quint16	Une chaîne de caractère sous la forme d'une QString

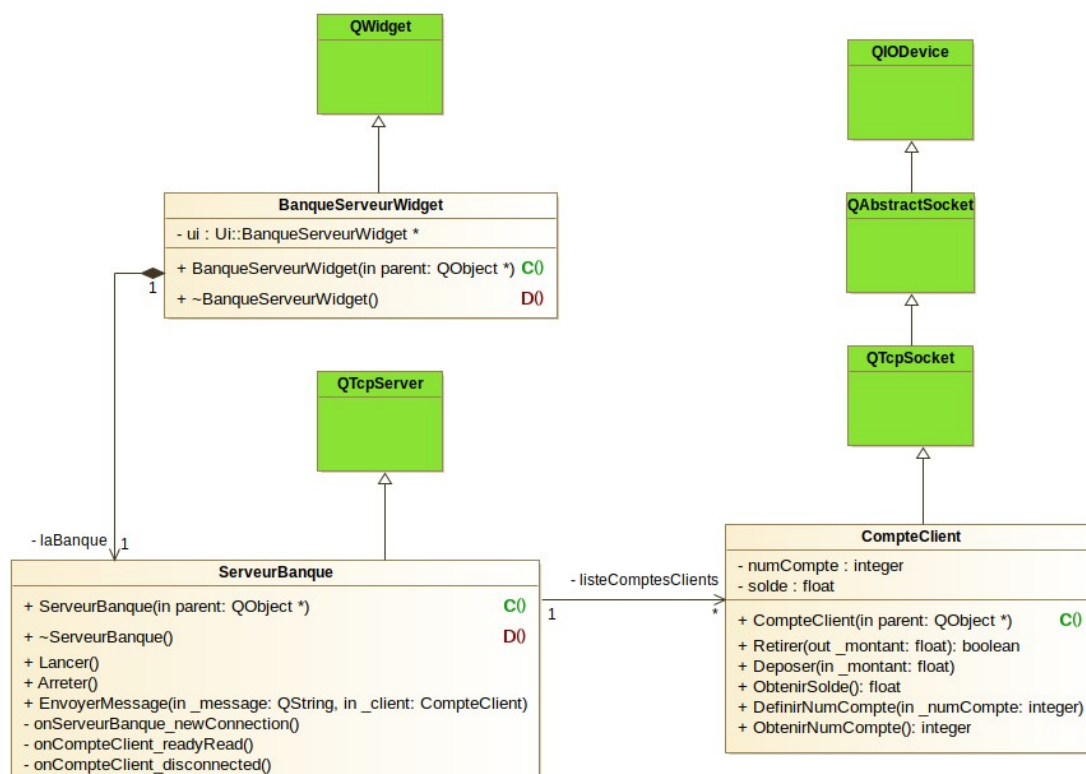
4.2. Exemple de communication client-serveur :



5. Réalisation du serveur

5.1. Création du projet

1. Créez le projet **ServeurBanque** de type Application graphique en C++ sous QT6 avec Qt Creator. La classe principale se nomme **BanqueServeurWidget**. Ce projet utilise une communication réseau entre un serveur et des clients, modifier le .pro de votre projet en conséquence.



Attention : Identifiez bien les différentes relations entre chaque classe. Elles sont différentes des Travaux dirigés précédents.

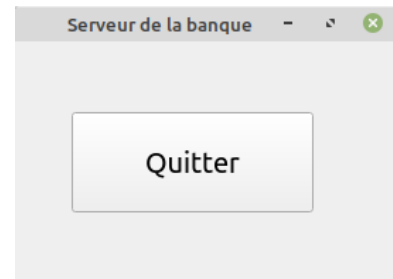
La relation entre la classe **BanqueServeurWidget** et la classe **SeveurBanque** est implémentée de manière statique.

Pour la relation d'association entre la classe **ServeurBanque** et la classe **Compteclient** la relation sera instanciée lors de la connexion du client au serveur.

L'IHM de la banque comporte juste un bouton **Quitter** qui ferme l'application.

Le constructeur de la classe **BanqueServeurWidget** lance le serveur de la banque, son destructeur l'arrête.

Pour cette application, la gestion des **signaux** et des **slots** est réduite au minimum.



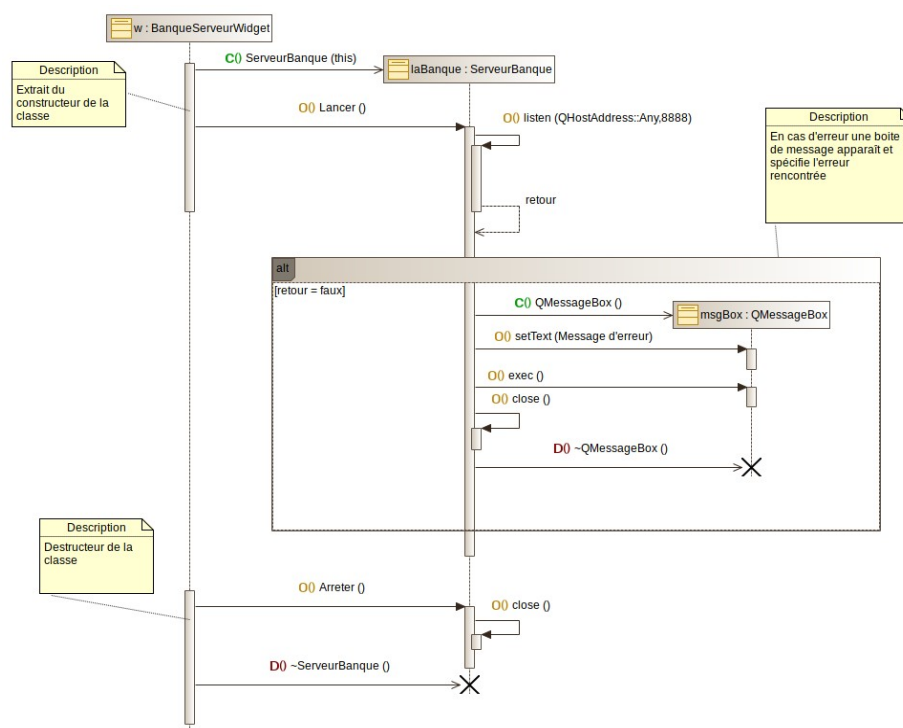
5.2. Réalisation de la classe **ServeurBanque**

Le constructeur de la classe reçoit en paramètre un pointeur sur **QObject** initialisé par défaut avec la valeur **nullptr** et le transmet à sa classe de base.

Il réalise également la liaison entre le signal indiquant la présence de la connexion d'un nouveau client et le **slot** correspondant.

- Ajouter cette classe à votre projet et codez le constructeur

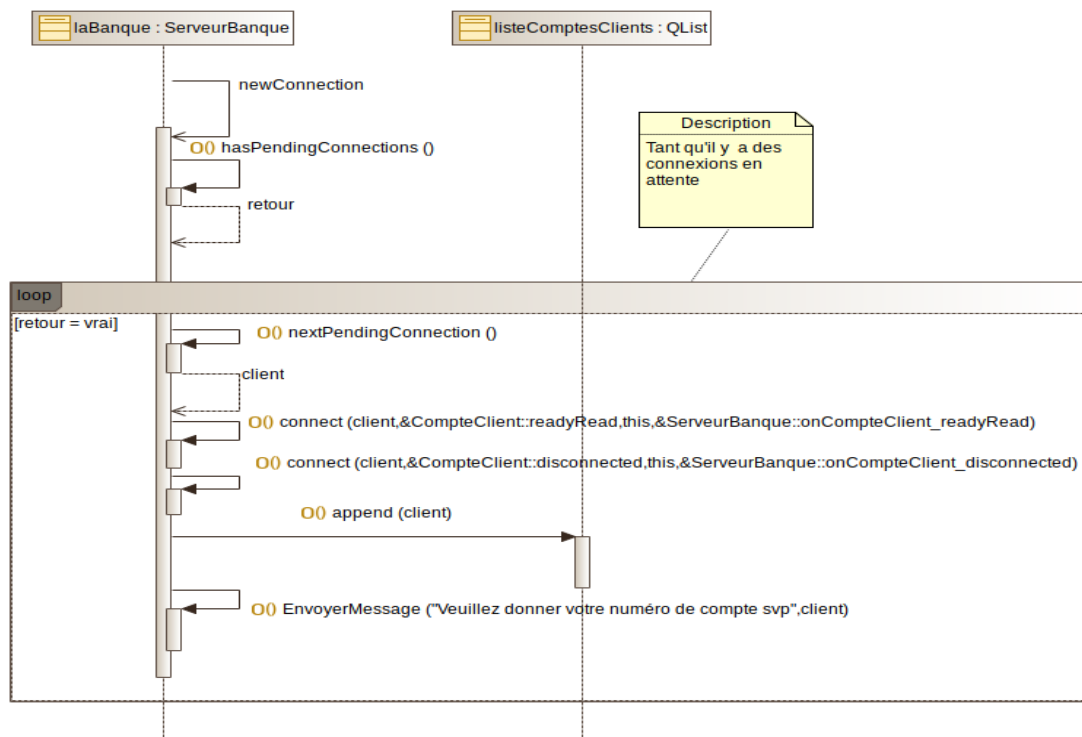
La description du fonctionnement des méthodes **Lancer()** et **Arreter()** est représentée par le diagramme de séquence suivant :



- Codez ces deux méthodes ainsi que le destructeur de la classe **BanqueServeurWidget**.

4. Déclarez la classe **CompteClient** en respectant la relation indiquée dans le diagramme de classe et complétez son constructeur. Il reçoit en paramètre un pointeur sur **QObject** initialisé par défaut avec la valeur **nullptr** et le transmet à sa classe de base.
5. Codez la relation entre la classe **ServeurBanque** et la classe **CompteClient** sous la forme d'une **QList**, comme le montre le diagramme de classe.

Le diagramme de séquence suivant montre l'enchaînement des opérations lorsqu'un client cherche à se connecter à la banque.



6. Réalisez le codage du **slot** correspondant au signal indiquant qu'un client cherche à se connecter à la banque.
7. Codez le **slot** en relation avec le signal indiquant qu'un client s'est déconnecté. S'il n'y a pas d'erreur, celui-ci retire le client de la liste, déconnecte les **signaux** et les **slots** qui lui sont associés et détruit la socket de dialogue vers le client.
8. Lancer votre **serveur banque** et votre **client distributeur automatique de billets** vérifiez la connexion et la déconnexion de votre client.
9. Le codage du dernier slot se fera après le codage de la classe **CompteClient**.

5.3. Réalisation de la classe *CompteClient*

10. Complétez le codage du constructeur en initialisant les attributs solde et numéro de compte avec une valeur nulle
11. Les méthodes ne posent aucune difficulté :
 - **Retirer()** : si cela est possible, le solde est débité du montant passé en paramètre.
 - **Deposer()** : crédite le solde si le montant passé en paramètre est supérieur à zéro.
 - **ObtenirSolde()** : renvoie la valeur du solde
 - **DefinirNumCompte()** : Affecte la valeur du paramètre au numéro de compte.
 - **ObtenirNumCompte()** : renvoie le numéro du compte

5.4. Finalisation de la classe *ServeurBanque*

12. Codez le dernier **slots** correspondant au signal indiquant que des données sont parvenues au serveur. Vous respecterez le protocole rappelé au début du sujet et vous utiliserez un **QDataStream** après avoir vérifié qu'un nombre suffisant de données a bien été reçu.

Extrait de l'utilisation QDataStream

```
if (socketVersClient->bytesAvailable() >= static_cast<qint64>(sizeof(taille)))
{
    QDataStream in(socketVersClient);
    in >> taille;
    // à compléter
}
```

SocketVersClient représente un pointeur sur l'objet qui a généré le signal **readyRead**.