Téléinformatique

IFT 3325

Devoir n°2

29 Novembre 2023

Auteurs :

- Léo Jetzer (20070432)
- Luchino Allix-Lastrego (20222844)

Université de Montréal
Département d'informatique et de recherche opérationnelle

1 Introduction

Dans le dossier soumis se trouvent : le dossier src qui contient tous les codes pour faire fonctionner l'emetteur et le recepteur, ce rapport, la documentation java et des fichiers tests à transmettre (`lorem\_4096.txt` et Test.txt).

Pour les tests la classe `Logger` permet de noter sur l'invite de commande ou dans un fichier (ou les deux) tout ce qu'il se passe, que ce soit du côté `sender` ou `receveur`.

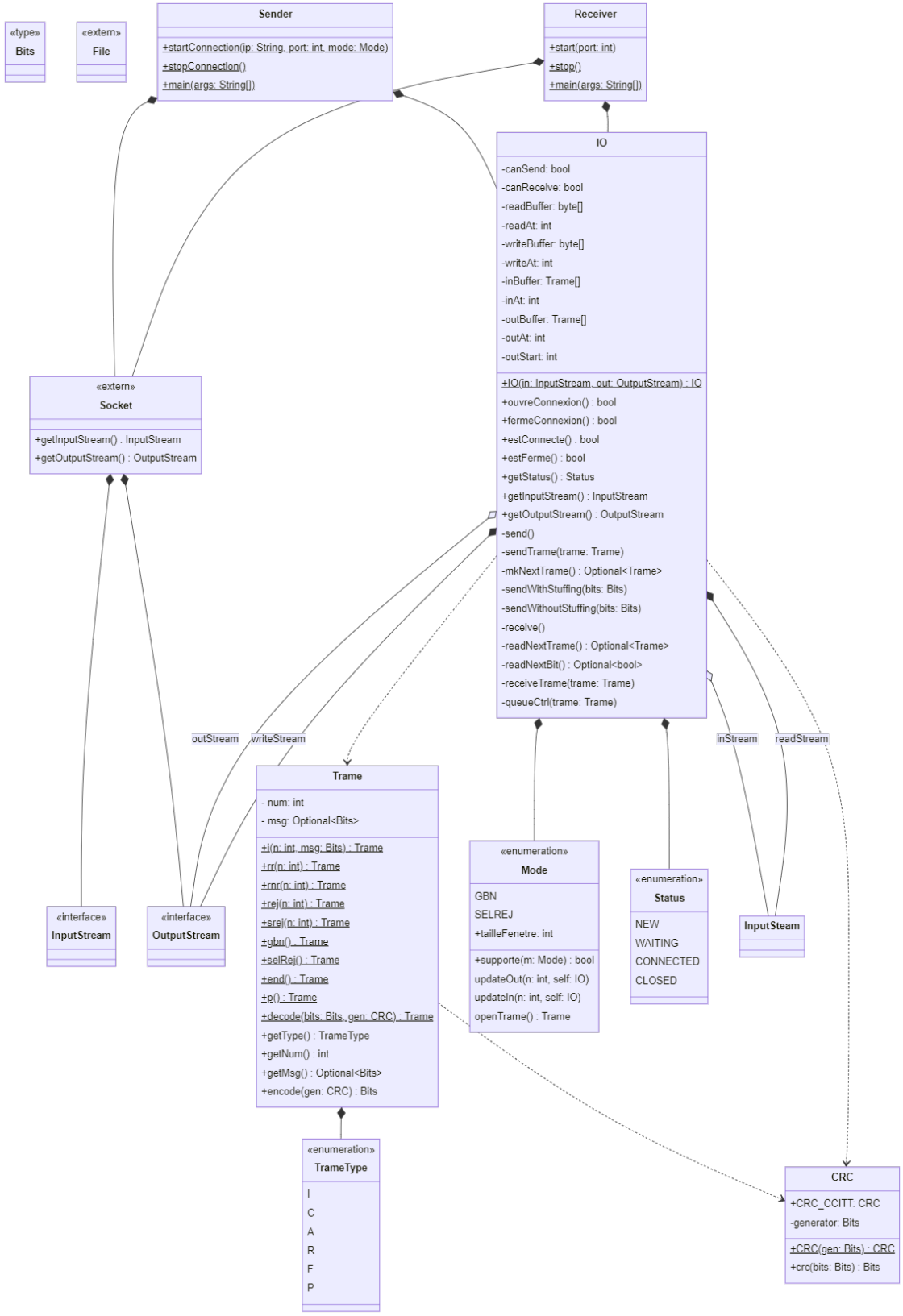
Pour exliquer sans trop de détail, Sender et Receiver établissent une connexion entre eux et s'envoient des données à travers des flux qui sont géré par `IO`. Cette classe s'occupe des trames (de la classe `Trames`), et du protocol *Go Back And*. Tous les bytes sont considéré comme des `Word`, pour faciliter leurs opération dessus comme par exemple le calcul de `CRC`.

Exemple pour lancer les programmes :

```
receiver 6674

Sender "127.0.0.1" 6674 "test.txt" 0
```

2 Diagrame de classe



3 Description des classes

Voici un résumé des classes importantes. Le javadoc sera également inclus avec le devoir pour plus de détails.

3.1 Bits

La classe `Bits` est un wrapper autour d'un tableau de `byte`. Il permet de pouvoir manipuler plus facilement des bits individuels et des chaînes de bits de longueur arbitraire.

3.2 Trame

Représente les différents types de trame et contient la logique pour encoder/décoder en/à partir d'une chaîne de bits.

3.3 CRC

Contient la logique pour calculer le code crc d'une chaîne de bits à partir d'un générateur donné.

3.4 IO

Contient la logique pour l'envoi/réception et traitement des trames. Lors de la création, il prend un `InputStream` — sur lequel il va lire les bits entrant — et un `OutputStream` pour écrire les bits sortant. L'utilisation prévu pour le devoir est de lui passer les streams d'un socket, mais n'importe quel stream peut lui être passé.

À l'interne, il contient deux threads : un est chargé de constament lire sur le stream d'entrée pour y trouver des trames et de les traiters, l'autre s'occupe d'écrire toutes les trames sortantes sur le stream de sortie. Les opérations sont synchronisées pour éviter que les threads se pile sur les pieds. L'envoi et la réception se fait bit par bit.

La classe offre une interface permettant de facilement lire et écrire sans se soucier des trames.

3.4.1 Envoi de trame

À chaque itération, le thread responsable d'écrire les trames a trois étapes :

1. Si on avait précédemment envoyer un RNR, mais que l'on peut maintenant recevoir plus de données, rajouter un RR à la queue de contrôle.
2. Envoyer toute les trames de contrôle dans la queue
3. Envoyer le plus de trame I possible

Pour créer la prochaine trame I, la méthode `mkNextTrame()` est appelé. Celle-ci prend jusqu'à 1024 bytes du buffer d'écriture comme message de trame. S'il n'y a plus de byte à envoyer, elle retourne rien et on passe à la prochaine itération.

L'envoi d'une trame se déroule comme suit :

1. La trame est encodé avec `CRC_CCITT` et transformé en chaîne de bits
2. on envoi (sans bit stuffing) le flag de début de trame
3. on envoi (avec bit stuffing) la chaîne
4. on envoi (sans bit stuffing) le flag de fin de trame
5. on envoi 11111111 pour bien délimiter — Ça permet de ne pas inventer une trame s'il y avait une erreur dans la précédentes et qu'on avait abandonner la lecture

3.4.2 Réception de trame

La recherche de la prochaine trame reçu se déroule de la manière suivante :

1. On lit les bits reçu jusqu'à ce que l'on trouve le flag de début de trame
2. On récolte les bits de la trame un par un jusqu'à ce qu'on arrive au flag de fin de trame. On enlève les 0 de bit stuffing, et si l'on trouve 7 1 d'affiler, on lance une erreur et on rejette la trame (ça va simplement nous faire passer à la prochaine itération)
3. on décode les bits reçu. la vérification CRC se fait en même temps et s'il y a une erreur on rejette la trame
4. la trame est traité selon son type, le mode de connexion et le status

3.4.3 Traitement d'une trame

Toute les trames sont ignorées si la connexion est fermé. Si le status est à NEW, toute les trames sont ignorées sauf les trames C. En status WAITING, tout est ignoré sauf les trames F et R

3.4.3.1 A

Lorsque l'on reçoit une trame A, les choses suivantes se passe :

1. Si le status est WAITING, passer à CONNECTED
2. avancer la fenêtre d'envoi selon le numéro
3. indiquer que l'on peut envoyer d'autre trame ou non selon si c'est un RR ou RNR

3.4.3.2 C

On ignore ces trames sauf si on est en status NEW. Dans ce cas, on envoi un RR et un P initial en réponse.

3.4.3.3 F

lors de la réception d'un F, on ferme les streams et on arrête de lire/envoyer. Le status passe à CLOSED

3.4.3.4 P

À la réception d'un P, on ne fait qu'en envoyer un à notre tour

3.4.3.5 R

Lors de la réception d'un R, on passe la logique à notre mode. Dans le cas de GBN, on ne fait que déplacer la fenêtre d'envoi pour l'aligner avec la trame désirée afin de signaler que c'est la prochaine à envoyer.

3.4.3.6 I

Encore une fois, on passe la logique à notre mode. Dans le cas de GBN, on vérifie que le numéro de la trame est

1. Dans la fenêtre de réception (Si elle ne l'est pas, on l'ignore)
2. La trame désirée

S'il s'agit de la bonne trame, on ajoute les données contenues à notre buffer de lecture, on avance la fenêtre de réception et on envoi un RR (ou un RNR si notre buffer est plein) Sinon, on envoi un REJ

3.4.4 Interface

Après la création de l'objet, il faut que l'un utilise la méthode `ouvreConnexion()` pour établir la connexion avec l'autre.

Une fois établi, l'objet donne accès à un `OutputStream` pour envoyer des données. Tout les bytes que ce stream reçoit sont ajouter à un buffer d'écriture et ils seront utilisés pour créer les trames I. De l'autre côté, il suffit de prendre l'`InputStream` que fourni l'objet afin d'y lire les bytes dans son buffer de lecture.

À la fin, il est important d'appeler la méthode `fermeConnexion` afin d'arrêter les threads et de libérer les ressources. IO ne ferme pas les streams qui lui sont donné au début. Il faudra donc les fermer vous même.

3.4.5 Temporisateur

IO possède également un troisème thread sous la forme d'un `Timer` qui agit comme temporisateur. Trois différents renvoi sont gérer par le temporisateur présentement :

— Le renvoi des P si tu es le serveur et que tu n'a pas reçu le P depuis un bon moment. Reset à chaque P reçu

— Le renvoi de RR si tu attend des trames mais que tu n'en a pas reçu depuis un moment. Reset à chaque RR, REJ ou SREJ envoyé. Arrêté lors de l'envoi d'un RNR

— Le renvoi des trames I à partir du début de la fenêtre si aucune réponse n'a été reçu. Reset à l'envoi d'un I, arrêter à la réception d'un RR, RNR, REJ et SREJ