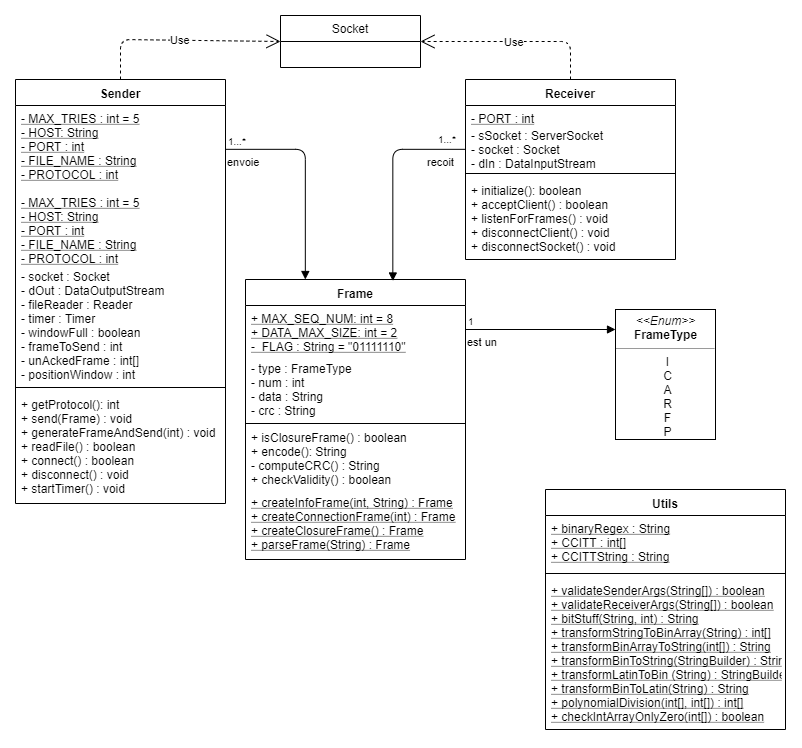
**Alexandre Dufour (p1054564) - Jonathan Caspar (p1129743 - 20059041)**

# **Téléinformatique - Devoir #2 - Rapport**

Nous avons organisé notre programme en 3 classes principales : **Sender**, **Receiver** et **Frame** :



L’idée générale est que deux entités (**Sender** et **Receiver**) se transmettent des suites de bit (dans notre programme, il s’agit de String composées uniquement de 0 et de 1) via un socket.

Cette suite de bits est une représentation bas-niveau d’un **Frame**.   
Une fois un frame construit, on l’encode (**encode()**) dans sa représentation bas-niveau (String de 0/1), on l’envoie au destinataire qui reconstruit un objet Frame à partir de ce String.

Ainsi, quand l’objet Frame est reconstruit : on peut vérifier grâce à un **checksum**() si les données n’ont pas été corrompues durant la transmission.

# **Classes principales**

# 1) Frame

Un objet Frame est une représentation plus haut niveau d’un paquet de bits. Il respecte un certain pattern et certains de ses champs ont des tailles fixes (ce qui nous permet de « parser » facilement une suite de bits).

Il peut être instancié uniquement en appelant les méthodes (createInfo/Closure/ConnectionFrame, etc.) puisque le constructeur de Frame est privé. Cela est voulu car on souhaite qu’une création de Frame mène au calcul automatique du code CRC (qui se fait dans le premier constructeur avec **this.crc = this.computeCRC()**)

Il peut aussi être **encodé** (Frame -> String) ou « **parsé** » (String -> Frame).  
  
Ses méthodes sont les suivantes :

* **isClosureFrame()** : retourne s’il s’agit d’une **trame de fermeture de connexion** (Type F) ou non
* **encode()**: retourne une représentation binaire (en String) d’un objet Frame en introduisant des bits supplémentaires si les données n’occupent pas tout l’espace binaire alloué (avec **Utils.bitStuff(String bits, int maxSize)**)
* **createInfoFrame(int num, String data)** : retourne un objet Frame représentant une trame d’information
* **createConnectionFrame(int protocol)** : retourne un objet Frame paramétré pour une demande de connexion avec un protocole spécifique
* **createClosureFrame()** : retourne un objet Frame paramétré pour une indication de fermeture de connexion
* **parseFrame(String frame)**: retourne un objet Frame reconstruit à partir d’un String binaire passé en paramètre
* **computeCRC**() : méthode privée (car utilisé uniquement par les méthodes « createInfo/Closure/ConnectionFrame, etc.. ») retournant le code CRC (en String) calculé à partir des attributs « type », « num » et « data » de l’objet Frame ciblé.
* **checkValidity**() : retourne si le frame est corrompu en effectuant un « checksum » grâce au CRC déjà calculé

# 2) Sender

Un objet Sender représente un objet cherchant à transmettre des frames à un client, le Receiver. Le Sender reçoit un fichier .txt à envoyer à un client. Celui-ci lance la connection à un socket via un port, auquel le *Receiver* pourra se connecter. Le *Sender* est activé avec les arguments [127.0.0.1](http://127.0.0.1/?fbclid=IwAR3v1EgGerNrsFwtO7SbZ28kr_s6nSB06YL3-nz-dgMRPowBz0vLn-juPuQ) 8080 test.txt 0 où on retrouve, dans l’ordre, la machine, le port, le fichier contenant les informations à transmettre ainsi que le protocole Go-Back-N.

Les méthodes principales sont les suivantes :

# **getProtocol()** : Retourne le protocole entré en démarrant le *Sender*.

# **send(Frame frame)** : Cette méthode envoi par le socket le frame à envoyer après l’avoir encoder en un string de données binaire par la méthode frame.encode()

# **generateFrameAndSend(int frameToSend)** :

# **readFile()** : Cette méthode *parse* le fichier texte et le divise directement en frame à envoyer. Chaque frame à un nombre de donnée maximale (Frame.MAX\_DATA\_SIZE), et li ait assigné un numéro de frame allant de 0 à 7, soit la taille de la fenêtre du Go-Back-N.

# **connect()** : Tente la connexion à un serveur via un socket. La méthode tente 5 fois de se connecter, par intervalle de 500 millisecondes. Entre temps, le serveur du *Receiver* doit être démarré pour accepter la connexion. Retourne un booléen si la connexion a fonctionné ou non.

# **disconnect()** : Arrête la connexion à un socket suite à la demande par le *Receiver*.

# **startTimer()** : En cas d’un parcours de la fenêtre complet, c’est-à-dire que le tableau représentant la fenêtre est plein, le *Sender* est en attente d’un frame de confirmation (ACK) envoyé par le *Receiver*. Si ce dernier ne l’a toutefois jamais reçu, il n’enverra pas de confirmation. Ainsi, cette méthode permet de démarrer un *Timer*. Si le *Sender* ne reçoit pas de confirmation avant la fin du *Timer* de 1 seconde, il enverra de nouveau toutes les frames qu’il avait envoyé depuis cette frame non confirmé. Il prend donc en note le numéro de frame non confirmé, et envoi toutes celles qui suivent également.

* **main()**: Méthode permettant de gérer l’algorithme de Go-Back-N. Cette méthode crée un *Sender*, se connecte à un socket et appel **readFile()*.*** Si toute se passe bien, il démarre une boucle permettant d’envoyer tous les frames. L’idée est qu’il commence par envoyer les 8 premières, puis attend la réception d’un ACK du *Receiver*. S’il l’a reçu, il envoi le prochain frame. Sinon, il envoi de nouveau les frames. Il cesse la connexion lorsqu’il a reçu une confirmation pour tous les frames.

# 3) Receiver

Le *Receiver* agit comme serveur par le biais duquel sont envoyé les frames. Il est démarré avec comme argument le port utilisé. Après avoir accepté la connexion avec le *Sender*, il démarre une boucle pour recevoir tous les frames. Il analyse leur type, et les traitent comme il se doit. Après avoir reçu tous les frames, il peut reformer le .txt envoyé et traité les données.

Les méthodes principales sont les suivantes :

* **initialize**(): Cette méthode permet la connexion à un socket via le port mentionné en argument. Elle retourne un booléen si la connexion est activée.
* **acceptClient**() : Cette méthode accepte la connexion à un client, soit le *Sender*, via un socket. Elle vouvre un *DataInpuStream* pour pouvoir recevoir de données. Elle retourne un booléen si un client a bien été accepté.
* **listenForFrames**() : Cette méthode permet de capter les frames envoyés par le *Sender*. Elle commence par vérifier la validité de la frame, c’est-à-dire si les données ont été corrompues, via la fonction **frame.checkValidity()**, et traite ensuite la frame selon le type de frame (de FrameType). La réception de frames cesse lorsque le *Sender* a envoyé sa dernière frame.
* **disconnectClient**() : Ferme le *DataStreamInput*, donc cesse sa relation avec le *Sender*.
* **disconnectSocket**() : Se déconnecte du socket auquel il était connecté.

# **Classes secondaires**

# 4) FrameType (enum)

Cette classe n’est qu’un enum qui contient les différents types de frame qu’on peut retrouver, soit en l’occurrence I, C, A, R, F, P. Ceci permet de standardiser le type de frame envoyé.

# 5) Utils (Utilitaires)

Cette classe contient toute les méthode *static* que l’on pourrait dire générique, donc non spécifique au *Sender* ou au *Receiver*. Elle ne contient aucun constructeur ou *main()*, que des méthodes permettant d’obtenir des résultats directement.

Les méthodes sont les suivantes :

* **validateSenderArgs**(String[] args) : Cette méthode vérifie si les arguments passée en paramètre de *Sender* lors de son appel son valide. Elle utilise le système de regex, et retourne un booléen.
* **validateReceiverArgs**(String[] args) : Cette méthode vérifie si les arguments passée en paramètre de *Receiver* lors de son appel son valide. Elle utilise le système de regex, et retourne un booléen.
* **bitStuff**(String bits, int maxSize) : Cette méthode prend en paramètre un string binaire ainsi qu’une taille. Elle ajoute des zéros au début du string jusqu’à attendre la taille **maxSize**. Elle retourne le string de taille **maxSize**.
* **transformStringToBinArray**(String) : Cette méthode prend en paramèetre un string binaire, et retourne un int[] binaire représentant le string en entré.
* **transformBinArrayToString**(int[]) : Cette méthode prend en paramètre un int[], et retourne un string binaire représentant les données du tableau en entrée.
* **transformBinToString**(StringBuilder) : Cette méthode prend en paramètre un stringbuilder binaire et le transforme en string pour pouvoir le manipuler et l’envoyer dans une frame.
* **transformLatinToBin** (String) : Cette méthode prend un string écrit en latin et retourne son code binaire sous la forme d’un stringbuilder.
* **transformBinToLatin**(String) : Cette méthode prend en paramètre un string binaire et retourne sa conversion en un string de texte Latin.
* **polynomialDivision(int[] array, int[] checksum)**: permet d'obtenir le résultat d'une division polynomiale entre deux nombres binaires .
* **checkIntArrayOnlyZero(int[] array)**: vérifie si un tableau de nombres contient uniquement des 0. On l’utilise pour vérifier si une division polynomiale donne un reste égal à 0.

# 6) Test

Cette classe permet de mettre en place des méthodes pour tester le protocole HDLC et Go-Back-N. Essentiellement, elle est appelée avant l’envoi de certaines frames pour modifier soit les données du champs *data* du frame, ce qui engendrerait une corruption des données, ou bien en modifiant le champ *num* du frame pour simuler la perte d’une frame dans le transfert de celle-ci.

Les méthodes sont les suivantes :

* **corruptData (String frame)**: elle reçoit en paramètre une frame en string binaire prête a être envoyé, et modifie 1 bit aléatoire du champ *data* du frame. Si le caractère choisi aléatoirement est un 1, la méthode la change pour un 0, et vice versa. Elle renvoi un nouveau frame binaire en string, prêt à être envoyé par le socket par le *Sender*.
* **Destroyframe(String frame)**: cette méthode reçoit en paramètre un frame en string binaire prête à être envoyé par le *Sender*, et modifie le champs *num* de la frame. Pour ce faire, la méthode commence par extraire les données du champs *num*, détermine sa valeur en décimal, l’augmente de 1, retransforme en binaire, puis l’incorpore au frame. Elle retourne un nouveau frame binaire contenant le nouveau numéro, prêt à être envoyé par le *Sender*.