Alain Eid
Eduard Voiculescu

IFT 3325 - Téléinformatique
Devoir 2

Implémentation d'un protocole HDLC simplifié

Travail présenté à
Zakaria Abou El Houda

Date de remise :
29 novembre 2018

Université de Montréal

Dans le cadre de ce devoir, nous avons implanté une version simplifiée du protocole HDLC dans le langage de programmation Java. Pour ce faire, nous avons notamment utilisé les sockets. Nous vous présenterons prochainement le diagramme de classe et une brève description des composantes de notre programme.

D'abord, nous distinguons les deux classes principales du programme pour la communication soit *Sender* et *Receiver.* La classe *Sender,* implémentant l'interface *Serializable*, contient une méthode *main* qui instancie un objet de type *Sender* avec les paramètres passés au programme. Le constructeur de l'objet va ensuite appelé la méthode *createSocket* qui retourne un entier, soit 0 quand tout est correct ou 1 quand une erreur s'est produite. Cette méthode s'occupe de l'envoi des trames. Elle fait appel à la méthode *createTrames* qui retourne une liste de trames après avoir lu les données d'un fichier avec la méthode *readFile*. Cette dernière retourne une liste des données lu du fichier. Elle contient aussi la méthode *delimiter* qui imprime dans le terminal une série de caractères '#' pour des besoins d'affichage.

La classe *Receiver* contient une méthode *main* qui instancie un objet *Receiver* avec le port passé en paramètre. La classe *Receiver* est une extension de la classe *Thread.* Le constructeur fait appel à la méthode *runReceiver* qui fait le traitement des trames envoyés par l'émetteur. La classe contient également la méthode *delimiter* mentionnée précédemment.

La classe *CheckSum* permet de faire le calcul du checksum. Elle contient la méthode *checkSumData* qui prend en paramètre les données et le polynôme générateur sous forme de chaîne de caractères et retourne un checksum sous forme de chaîne de caractères. Cette méthode fait appel à la méthode *XORDivision* qui prend les même paramètres et fait la division XOR des données avec le polynôme générateur puis retourne un *StringBuilder*. Elle appelle aussi la méthode *addRDegreeZeros* qui retourne un objet *StringBuilder* et qui sert à ajouter le nombre de zéros selon le degré du polynôme générateur. La classe contient également la méthode *validateCRC* qui retourne un booléen indiquant si le CRC contient une erreur.

La classe *Trame* est un modèle contenant les cinq attributs du format présenté dans l'énoncé et une série de méthodes pour accéder et modifier ces attributs, soit des *setters* et *getters*. Elle contient aussi la méthode *makeTrameFormat* qui retourne une chaîne de caractères qui représente la trame. La méthode *prettyPrint* retourne une chaîne de caractère représentant la trame pour des besoins d'affichage. La méthode *bitStuffSendertrame,* quant à elle, sert à "bitstuffer" les attributs de la trame et retourne une chaîne de caractères. Cette classe implémente l'interface *Serializable* pour qu'elle puisse être envoyé dans un *ObjectOutputStream.* Nous avons choisi d'envoyer des objets *Trame* plutôt qu'une chaîne de caractères pour simplifier le traitement du côté serveur. De plus, cela respecte davantage la conception orientée-objet.

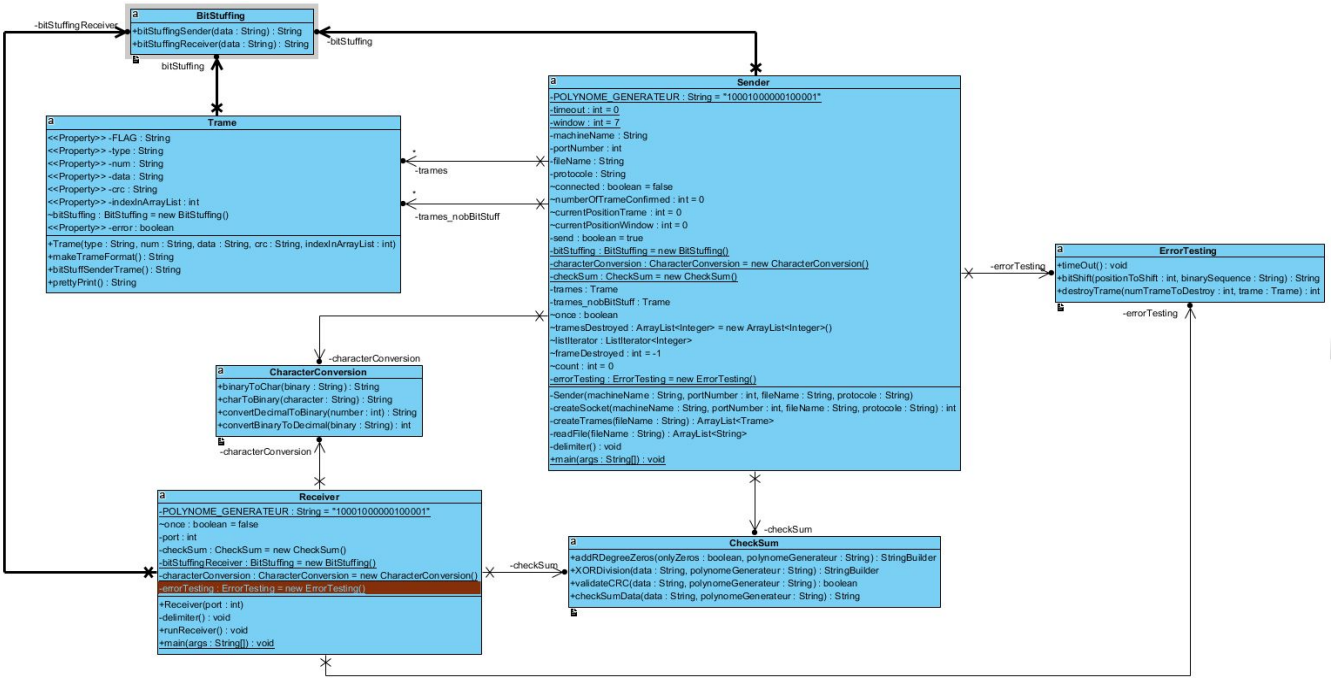
La classe *CharacterConversion* fournit des méthodes pour faire des conversions de décimaux au format binaire et vice-versa. La méthode *binaryToChar* prend en paramètre une chaîne de caractères, soit une suite de binaires, puis retourne sa valeur convertie en chaîne de caractères. La méthode *charToBinary* fait le contraire. Elle prend en paramètre une chaîne de caractères puis la convertit en une suite de binaire sous forme de chaîne de caractères.

La classe *BitStuffing* contient deux méthodes et implémente l'interface *Serializable*. La méthode *bitStuffingSender* prend en paramètre une suite de binaires de type chaîne de caractères et ajoute des bits. Elle est utilisée dans l'envoi de données. La méthode *bitStuffingReceiver* prend en paramètre une suite de binaires de type chaîne de caractères et enlève les bits ajoutés dans l'envoi des données. Les deux méthodes retournent une chaîne de caractères qui est une suite de binaires. La méthode *convertDecimalToBinary* convertit un décimal dans un format de 8 binaires puis le retourne sous forme de chaîne de caractères. Puis, la méthode *convertBinaryToDecimal* fait le contraire. Elle prend en paramètre une chaîne de caractères qui est une suite de 8 binaires puis retourne la valeur décimale qui est un entier.

Finalement, la classe *ErrorTesting* sert à introduire des erreurs dans l'envoi des trames afin de tester le programme. La méthode *timeOut* sert à interrompre le *Thread* pendant 10 secondes. La méthode *bitShift* prend en paramètre une position de la séquence de binaires à inverser et la séquence de binaire elle-même puis retourne la séquence après avoir effectué l'inversion. Puis, la méthode

destroyTrame va détruire le numéro de trame passé en paramètre puis retourne le numéro de la trame détruite ou la valeur -1 s'il n'y a pas eu de destruction.

Diagramme de classe



Vous pouvez consulter l'image dans l'archive pour une meilleur résolution.