Université de Montréal

Travail #6

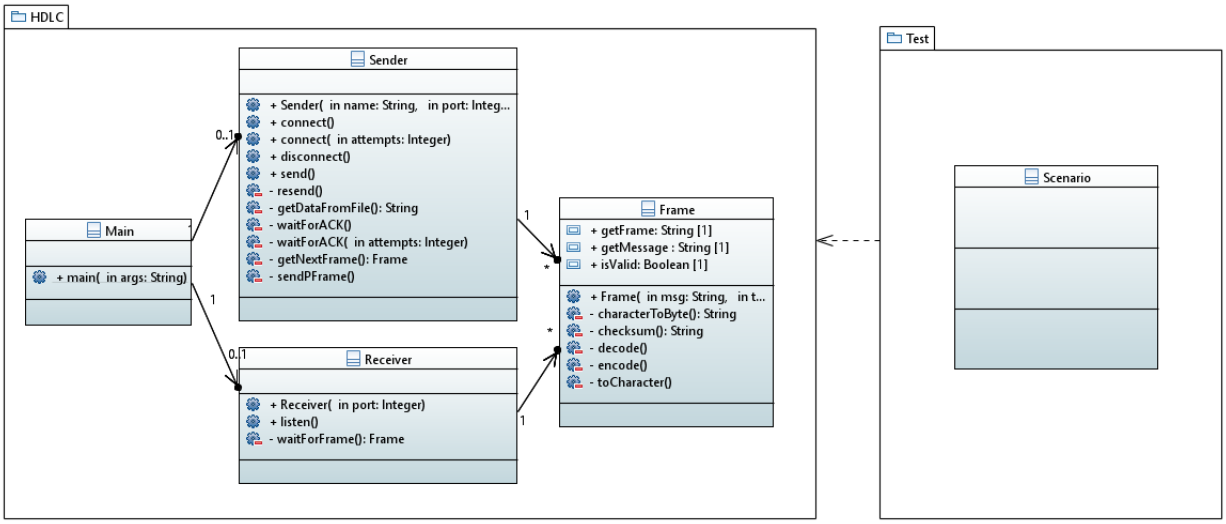
Par
Kevin Belisle
(20018469)
Olivier Bélec
(00000000)

Baccalauréat en informatique
Faculté des arts et des sciences

Travail présenté à Abdelhakim Hafid
Dans le cadre du cours IFT-3325
Téléinformatique

Novembre 2018

Diagramme de classe



Description des classes

Sender

Constantes

```
private final static int DATA_SIZE_LIMIT = 100;
private final static int MAX_BUFFER_WINDOW_SIZE = 8;
private final static int MAX_FRAME_COUNT = 7; /*2^3-1 = 7*/
private final static int MAX_WAIT_TIME = 3000;
private final static int MAX_ATTEMPT = 30;
private final static int ACK_FRAME_SIZE = 48;
```

DATA_SIZE_LIMIT : Le nombre de caractère que nous envoyions par trame. Elle sert à simuler un maximum sur le data pour chaque trame.

MAX_BUFFER_WINDOW_SIZE : La grandeur maximum du buffer pour les trames (2^3, car nous utilisons 3 bits pour numéroter les trames).

MAX_FRAME_COUNT : Le nombre maximum de trame en mémoire à un moment donné ($2^3 - 1$ pour Go-Back-N).

MAX_WAIT_TIME : Le temps maximum d'attente pour le temporisateur (3s pour ce TP).

MAX_ATTEMPT : Le nombre maximum d'essai avant de conclure que la connexion avec le serveur est perdu.

ACK_FRAME_SIZE : la taille des trames de réponse (Elles sont tout le temps de taille 6 octets pour ce TP, donc 6octets*8bits/octets = 48 bits).

Attributs

```
Socket s;
DataInputStream in;
DataOutputStream out;
BufferedReader file;
Frame[] frameBuffer = new Frame[MAX_BUFFER_WINDOW_SIZE];
int nextEmpty = 0;
int lastACK = MAX_BUFFER_WINDOW_SIZE - 1;
int frameCount = 0;
boolean sendingData = false;
```

Nous gardons en mémoire les éléments suivants en ordre de haut en bas : le socket (s), le flux d'entrée du socket (in), le flux de sortie du socket (out), le flux du fichier(file), le buffer pour les trames(frameBuffer), l'index de la prochaine case du tableau qui est vide(nextEmpty), le numéro de la dernière trame qui a été acquiescée(lastACK), le nombre de frame dans le buffer (frameCount) et si nous envoyons ou non des données (sendingData).

Constructeur

```
public Sender(String name, int port, String fileName)
```

Unique Construteur qui initialise le socket vers le *serveur* sur le *port*, ouvre le fichier et garde en mémoire le flux d'entrée, de sortie et du fichier.

Méthodes

```
private void waitForACK() /*waitForACK(0)*/
private void waitForACK(int attempts)
```

Attend la prochaine trame de réponse du serveur et la traite.

```
public void connect() /*connect(0)*/
```

```
private void connect(int attempt)
```

Envoie la trame de demande de connexion au serveur et attends la trame de réponse.

```
public void disconnect()
```

Envoie la trame de connexion et relâche toutes les ressources (socket et flux).

```
private String getDataFromFile()
```

Lit du flux du ficher le nombre de caractère maximal spécifié par la constante DATA_SIZE_LIMIT

```
public void send()
```

Commencer l'envoie des données vers le serveur.

```
private void resend()
```

Réenvoie toutes les trames qui sont en mémoire.

```
private Frame getNextFrame()
```

Bloque et attends (si nécesaire), lit du flux d'entrée la trame de réponse provenant du serveur et retourne la trame pour futur traitement.

```
private void sendPFrame(int attempts)
```

Envoie une trame P-bit au serveur.

Receiver

Attributs

```
Frame lastMsg;
int next;
long timer;
String txt="";
/*socket*/
ServerSocket server;
Socket socket;
DataInputStream input;
DataOutputStream output;
```

Nous gardons en mémoire les attributs suivants en order de haut en bas: la trame du dernier message (lastMsg), le prochain numéro de trame (next), le temps courant (timer), le data reçu du sender (txt), le serveur de socket (server), le socket (socket), le flux d'entrée du socket (input) et le flux de sortie du socket (output).

Constructeur

```
public Receiver(int port)
```

Instancie le Receiver, créer le serveur pour les sockets sur le port spécifié, instancie un socket sur le serveur et garde en mémoire les flux d'entrée et de sortie du socket.

Méthodes

```
public void listen()
```

Écoute sur le socket pour des trames, sauvegarde le data des trames et envoie les trames de réponse appropriés

```
private Frame waitForFrame()
```

Bloque et attends (si nécessaire) pour des trames du Sender et les retourne pour un futur traitement.

Frame

Attributs

```
private boolean valid;
private String message,encoded;
public String getMessage()
public String getFrame()
public Boolean isValid()
```

Nous gardons en mémoire si la trame est valide ou non (valid et isValid()), le message décodé (message, getMessage()) et le message encodé (encoded, getFrame()).

Constructeur

```
public Frame(String msg,boolean toBeEncoded)
```

Instancie une trame. En fonction de (toBeEncoded), nous encoderons la trame et nous la décoderons sinon.

Méthodes

```
private void encode()
```

Encode le message de la trame, fait du bit-stuffing et calcule le CRC.

```
private void decode()
```

Décode le message encodé de la trame, défait le bit-stuffing et vérifie le contenu selon le CRC.

```
private String checksum(String msg)
```

Calcule le CRC à partir du contenu de la trame.

```
private String characterToByte()
```

Transforme une chaîne de caractères en octets.

```
public void toCharacter()
```

Transforme des octets en caractères.

Main

Point d'entrée de l'application, valide les arguments d'entrée et créer le Sender ou Receiver selon les arguments d'entrée