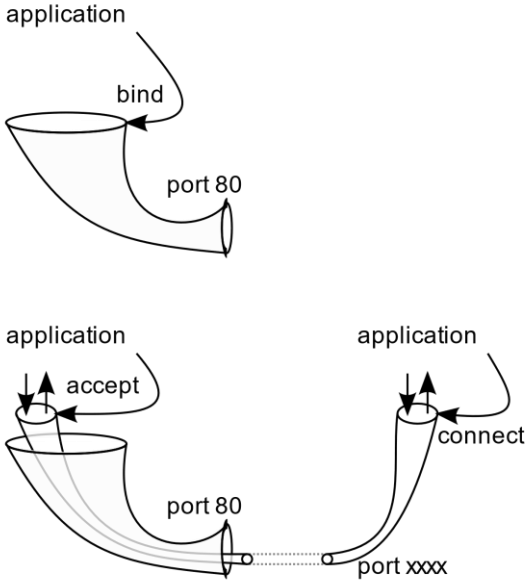
Ce TD porte sur la réalisation d'un programme faisant communiquer plusieurs clients et un serveur sur un réseau local (ou distant \smiley). Pour ce faire, nous utiliserons des sockets au-dessus de TCP/IP. Le programme consistera en une messagerie instantanée simplifiée clavier/écran.

Les sockets sont des dispositifs permettant de communiquer entre 2 machines. Dans tous les cas, il y a un participant qui initie le dialogue (souvent désigné comme client) et l'autre qui se tient prêt à communiquer (souvent désigné comme serveur). L'identifiant d'une connexion se fait à l'aide de 2 valeurs ; l'adresse IP et le port.

1 socket en mode connecté

Il y a un ordre chronologique pour ce genre de connexion. Il faut que le serveur soit lancé avant le client. Un client qui ne trouverait pas de serveur interrompt la connexion.



Pour expérimenter ce genre de connexion nous utilisons les sources disponibles sur le support en ligne.

- RcvCmd.java qui lance un serveur
- SendCmd.java qui lance un client

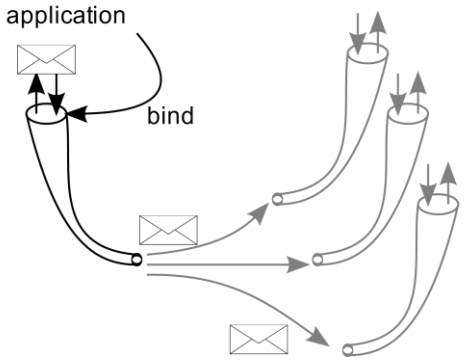
✓ Question : compiler les programme Java et lancez les dans 2 terminaux différents.

Remarquez que le fonctionnement consiste à envoyer une chaîne de caractère du client au serveur et laisse le serveur répondre en une ligne saisie au clavier.

Imaginez des améliorations (nombre de client, longueur des textes échangé).Vous pouvez essayer de modifier les programmes suivant vos idées.

2 Sockets en mode non-connecté

Dans certains cas, nous n'avons pas besoin d'une garantie de transmission. Par exemple un capteur émet sans interruption à intervalles réguliers. La perte de quelques valeurs n'a pas d'importance. Dans ce cas, il faut plutôt voir les échanges comme des messages qui contienne le destinataire et peuvent se perdre ou être détruit à l'arriver si on ne trouve personne à l'adresse indiquée.



Pour expérimenter ce genre de connexion nous utilisons les sources disponibles sur le support en ligne.

- RcvData.java qui lance un serveur et attend des messages
- SendData.java qui lance le même type de serveur mais n'attend aucun message et en envoie un ailleurs.

✓ Question : compiler les programme Java et lancez les dans 2 terminaux différents.

Remarquez que le fonctionnement consiste à envoyer une chaîne de caractère d'un terminal à l'autre.

Imaginez des améliorations (nombre de client, longueur des textes échangé).Vous pouvez essayer de modifier les programmes suivant vos idées.

3 Des protocoles en texte clair

Les protocoles de l'Internet sont à l'origine sous forme d'échanges de commandes texte rudimentaire (en anglais).

Voici un exemple d'échange entre un navigateur et un serveur web :

Client

```
GET /index.html HTTP/1.1 ¶
Host: m4102.parlenet.org ¶
Accept-Language: argoTerrien ¶
User-Agent: Zorglub ¶
¶
```

Serveur

```
HTTP/1.1 200 OK ¶
Date: Sun, 5 Oct 2003 11:00:07 ¶
Server: Apache/2.0.40 ¶
Accept-Ranges: bytes ¶
Content-Length: 2898 ¶
Connection: close ¶
Content-Type: text/html ¶
¶
<!DOCTYPE HTML PUBLIC...
```

Notons que la structure est la même pour le client et le serveur :

- une ligne de protocole composée de 3 mots
- un en-tête MIME (Multipurpose Internet Mail Extensions). Il s'agit d'une suite de lignes associant variable et valeur. L'en-tête se termine par une ligne vide.
- les données échangées :
  - valeurs de formulaire du client vers le serveur ou
  - page demandée du serveur vers le client.

La ligne de protocole est différente entre le client et le serveur.

Pour le client, c'est un nom de méthode (GET, PUT, POST…) suivi d'un URI (chemin local d'informations) et terminé par le protocole utilisé.

Pour le serveur, c'est le protocole suivi d'un code d'erreur et terminé d'une traduction abrégée en anglais.

URL ou URI ?

- L'URI est le chemin local de l'information sur le serveur (qui peut être différent de l'arborescence de fichier)
- l'URL est le chemin unique sur l'Internet (protocole, adresse IP, port, URI, QUERY_STRING…)

Cas de l'hébergement multiple

Notez bien que le protocole HTTP/1.0 permet de préciser dans l'en-tête le nom du serveur final qui fournit les pages (dans le cas de serveurs mutualisés). Par exemple :

```
Host: r305.parlenet.org
```

C'est ce qui permet à un même serveur physique d'héberger plusieurs serveurs web.

Cas d'un mandataire

Notez également que dans le cas de mandataire (en anglais *proxy*), le chemin n'est pas un chemin local (URI) mais le lien complet avec le nom du serveur (URL). Par exemple :

```
GET http://m4102.parlenet.org/ HTTP/1.0
```

C'est ce qui permet au mandataire de savoir à quel serveur physique il doit s'adresser.

Il existe un outil efficace qui connecte un socket à un écran et un clavier *telnet* (aujourd'hui remplacé par *nc* avec les mêmes paramètres). Par exemple :

```
telnet localost 4141
```

4 Compréhension du sujet

Nous allons développer une messagerie instantanée. Le logiciel résultant de ce TD possède les propriétés suivantes :

- Le programme est écrit en Java.
- Le programme ouvre un port en mode serveur.
- Le serveur accepte un nombre indéfini d'utilisateurs.
- Lorsqu'un utilisateur se connecte le serveur lui demande son nom, puis attend un nombre indéfini de messages (lignes).
- Chaque message envoyé par un utilisateur est retransmis à tous les autres, précédé par le nom de l'utilisateur.
- Les utilisateurs se connectent en utilisant simplement la commande « telnet » (ou « nc »).

5 Démonstration

Vous pouvez vous rendre mieux compte des spécifications en suivant une démonstration.

- Ouvrir un terminal
- Saisir la commande :

```
nc parlenet.org 8081   # pour le groupe A
nc parlenet.org 8082   # pour le groupe B
nc parlenet.org 8083   # pour le groupe C
```

- Recevoir une invitation « What's your name? »
- Saisir le **groupe TD** et votre **nom** :

```
GrpA X.Durand
```

- Ces identifiants seront utilisés dans les traces du serveur pour vérifier que vous avez réussi l'exercice)
- Vous aurez en retour un chaleureux message de bienvenu
- Vous pouvez ensuite saisir du texte qui sera répercuté vers tous les autres utilisateurs connectés.

Il existe deux commandes particulières :

- pour connaître la liste des autres participant (mot unique sur la ligne)

```
LIST
```

- pour quitter la session (mot unique sur la ligne)

```
QUIT
```

6 Analyse du logiciel

✓ Combien de tâches devront être mises en œuvre et quels seront leurs rôles ?

✓ Donnez en pseudo-code le comportement des différentes tâches.

constitueront la base de données manipulée par le serveur.

✓ Quelles sont les informations gérées par le serveur ?

✓ Écrire la classe « User » qui représente un utilisateur chez le serveur (attribut et constructeur). Les exemplaires de cette classe constituent la base de données du serveur.

✓ Décrire les risques de conflit qui peuvent apparaître.

✓ Comment le serveur peut-il gérer la diffusion d'un texte envoyé par un utilisateur (attribut et méthode) ?

7 Conception du logiciel

Nous abordons la phase de réalisation.

✓ Où placer l'activité d'attente de texte envoyé par un utilisateur (écrire la méthode) ?

✓ Combien faut-il écrire de programmes (ainsi que ce qui peut être réutilisé d'Unix) ?

✓ Modifier le constructeur de la classe Utilisateur pour ajouter automatiquement un utilisateur dans la base de données du serveur et lancer son activité.

✓ Y a-t-il des activités en concurrence pendant l'utilisation de la messagerie instantanée par 3 utilisateurs ?

✓ Écrire le constructeur du serveur et l'attente de nouveaux utilisateurs.

✓ Écrire la gestion de l'ensemble des utilisateurs dans le serveur. Comment a-t-on réglé le problème d'accès concurrence à la base des utilisateurs ?

✓ Comment le serveur accepte-t-il les connexions des utilisateurs ?

✓ Faire un lanceur qui prenne en argument le port d'écoute.

✓ Lorsqu'un utilisateur envoie un texte, que doit faire le serveur ?

✓ Quelles sont les informations (ou données) que l'on doit mémoriser par utilisateur et qui sont nécessaires à sa création (âge, nom, profession, canal de lecture…) ? Ils