Mini-projets Java

H. Mounier

1. Serveur de gestion de notes

a) Écrire un serveur TCP calculant à partir du détail des notes de chaque élève, (plusieurs contrôles, chacun avec plusieurs épreuves, et chacune avec plusieurs questions), la note globale et effectuant des traitements divers :

- histogramme,
- tri par mérite,
- tableau corrélatif entre les épreuves,
- tri par clefs (groupes ou origine des élèves, etc ...)
- sauvegarde et lecture disque

On utilisera des structures de données du paquetage java.util

b) Ecrire un client TCP demandant à l'utilisateur quelle requête il souhaite voir effectuée ; le client envoie ensuite la requête (chaque requête correspondant à un traitement ci-dessus) au serveur (on concevra un mini-protocole avec des paquets adaptés) ; le serveur traite la requête qu'il reçoit et renvoie la réponse correspondante au client. Le client affiche la réponse du serveur à l'écran.

Traiter d'abord le cas d'un serveur itératif, puis celui d'un serveur concurrent.

Conseil : utilisez des fichiers pour stocker vos données.

2. Serveur de calcul de polynômes

a) Écrire un serveur TCP permettant d'effectuer diverses opérations sur des polynômes (+, -, *, /, d/dt, intégrale) donnés par leurs coefficients.

b) Ecrire un client TCP demandant à l'utilisateur quelle requête il souhaite voir effectuée ; le client envoie ensuite la requête (chaque requête correspondant à un traitement ci-dessus) au serveur (on concevra un mini-protocole avec des paquets adaptés) ; le serveur traite la requête qu'il reçoit et renvoie la réponse correspondante au client. Le client affiche la réponse du serveur à l'écran.

On s'attachera à rendre performant le dialogue avec l'utilisateur (voir en particulier la saisie du polynôme, qui ne se fera pas forcément par demande successive de TOUS les coefficients), et à soigner l'affichage des résultats.
Prévoir également la possibilité de rappel et d'édition de l'historique des derniers polynômes saisis.

3. Serveur de calcul par intervalles

a) Écrire un serveur TCP pour le calcul par intervalles, c'est-à-dire une classe permettant de représenter des intervalles numériques, les additionner, les soustraire, les multiplier, les élever à une puissance entière, les comparer, ... sans oublier de quoi les saisir et les afficher.

On prendra le plus grand soin à produire des résultats mathématiquement corrects. Ainsi :
`[1..2] + [-1..1,5]` devra retourner `[0..3,5]`
`[1..2] * [-1..1,5]` devra retourner `[-2..3]`
`[1..2] / [-1..1,5]` devra retourner `[-INFINITY..+INFINITY]`
Il faudra donc être en mesure de gérer la notion de nombre infini (ce que Java fait déjà pour les nombres flottants).

b) Ecrire un client TCP demandant à l'utilisateur quelle requête il souhaite voir effectuée ; le client envoie ensuite la requête (chaque requête correspondant à un traitement ci-dessus) au serveur (on concevra un mini-protocole avec des paquets adaptés) ; le serveur traite la requête qu'il reçoit et renvoie la réponse correspondante au client. Le client affiche la réponse du serveur à l'écran.

Traiter d'abord le cas d'un serveur itératif, puis celui d'un serveur concurrent.

Application (une fois ce qui précède réalisé) : tracer des courbes sans risquer de produire des dessins mathématiquement faux comme c'est parfois le cas lorsqu'on procède par échantillonnage de la fonction à tracer. Ainsi, plutôt que de calculer y=f(x) pour N valeurs flottantes de x comprises entre xmin et xmax et de relier les N points (x,y) obtenus, on peut préférer procéder par intervalles. L'image de chaque intervalle est alors un autre intervalle et le tracé s'obtient en dessinant à l'écran les N rectangles obtenus. La véritable courbe est alors nécessairement à l'intérieur des rectangles. Il suffit ensuite de réduire la taille des intervalles pour affiner la qualité du dessin.

4. Serveur de gestion d'arborescence

a) Écrire un serveur TCP permettant de visualiser et manipuler textuellement l'arborescence des fichiers, à la manière de l'explorateur Windows, mais en format texte. Il faut donc pouvoir :
lister les fichiers et répertoires, avec leurs dates de dernière modification, et leurs tailles (pour les répertoires, on affichera de préférence la somme des tailles de tout ce qu'il y a en dessous dans l'arborescence). Note : on doit pouvoir les afficher triés par ordre alphabétique du nom, par ordre de taille, ou par date de modification,
supprimer, renommer ou déplacer des fichiers ou répertoires,
se déplacer dans l'arborescence pour visualiser le contenu d'un répertoire fils ou du répertoire parent,
rechercher un (des) fichier(s) dans l'arborescence en donnant une *portion* de nom, et éventuellement un critère sur sa date de modification.

b) Ecrire un client TCP demandant à l'utilisateur quelle requête il souhaite voir effectuée ; le client envoie ensuite la requête (chaque requête correspondant à un traitement ci-dessus) au serveur (on concevra un mini-protocole avec des paquets adaptés) ; le serveur traite la requête qu'il reçoit et renvoie la réponse correspondante au client. Le client affiche la réponse du serveur à l'écran.

Traiter d'abord le cas d'un serveur itératif, puis celui d'un serveur concurrent.

5. Logiciel de gestion de QCM

Le but de ce projet est de produire un logiciel de QCM. Le logiciel devra être capable de lire un QCM défini dans un fichier texte, de le mélanger (changer l'ordre des questions, et l'ordre des

réponses pour une question donnée), de le proposer à un utilisateur pour que celui-ci saisisse ses réponses de manière interactive, de stocker les réponses saisies, et enfin de corriger un questionnaire à partir d'un questionnaire de référence contenant les bonnes réponses.

Questionnaire

Un questionnaire est un fichier texte qui fonctionne ligne-à-ligne. Chaque ligne est précédée d'un caractère `T Q M R V F` suivi d'un caractère `:`. Toute ligne ne correspondant pas à cette description sera ignorée.

`T` introduit le titre, unique, du questionnaire. `Q` et `M` introduisent respectivement une question simple (appelant une unique réponse), et multiple (appelant une réponse ou plus). `R V F` introduisent des réponses possibles à la dernière question posée. `V` précède une réponse vraie, `F` une réponse fausse, et `R` précède un choix dont on ne précise pas si il est valide ou non. En cas de besoin éventuel, on numérotera les questions à partir de 1, et les réponses à une question donnée à partir de 1 également.

Exemple :

```
T:Unix pour les utilisateurs
M:quelle(s) commande(s) permet(tent) de changer de répertoire ?
R:ls
R:cd
R:rm
R:pushd
Q:quelle commande permet de compiler un programme java ?
F:java
V:javac
F:mkdir
F:cd
```

Mélange

Cette première partie devra proposer une ou des classes stockant un ensemble de questions et leurs réponses associées. Le stockage d'une série d'éléments (questions, réponses...) sera fondé sur un ou des objets de la classe `java.util.Vector`. Une classe pourra donc ressembler à :

```
import java.util.Vector;
public class QCM {
...
private Vector questions;
...
public QCM() {
...
questions = new Vector();
```

Une méthode `public void lit(Reader r)` permettra de lire un questionnaire à partir d'un flux. Une méthode `public void ecrit(Writer w)` permettra d'écrire un questionnaire dans un flux. Une méthode `public void melange(int seed)` permettra de mélanger le questionnaire, c'est à dire de changer l'ordre des questions et l'ordre des réponses associées. Cette méthode s'appuiera sur un objet `Random` dont la suite pseudoaléatoire sera initialisé avec la graine `seed` précisée en paramètre. Attention, le mélange doit être total, *i.e.* chaque permutation possible doit être obtenue de manière équiprobable aux autres.

Pour tester cette partie, on développera un programme qui :

1. lit un questionnaire à partir d'un fichier ;
2. le mélange à partir d'une graine ;
3. écrit le résultat dans un fichier.

La syntaxe d'utilisation du programme sera du type :

```
shell> java QCM 12345 fichier-qcm-lu fichier-qcm-ecrit
```

Interrogation

Cette seconde partie permettra d'interroger de manière interactive un utilisateur sur un questionnaire. L'interface pourra être en mode texte (à base de menus), ou bien une interface graphique. Les questions seront proposées après un mélange aléatoire. Les réponses choisies par l'utilisateur seront stockées dans un fichier résultat au format libre. L'interface devra tenir compte du type simple ou multiple des questions à poser. La syntaxe d'utilisation du programme sera du type :

```
shell> java Interro fichier-qcm-lu fichier-resultat-ecrit
```

Correction

Cette dernière partie permettra, à partir d'un questionnaire dont les réponses sont explicites (réponses types `V F`) de corriger les réponses données par un utilisateur en précisant les nombres de :

• bonnes réponses (tous les `V` sélectionnés) ;
• mauvaises réponses (un `F` sélectionné) ;
• réponses partielles (certains `V` non sélectionnés) ;
• questions sans réponses (pas de sélection).

La syntaxe d'utilisation du programme sera du type :

```
shell> java Corrige fichier-qcm-lu fichier-utilisateur-lu
```

La correction apparaitra à l'affichage du programme, sur la sortie standard (`System.out`).