IN101 – TP5 2011/2012 v3

Sujet (n'imprimer le pdf qu'une seule fois par poste de travail)

Durée : 3 h

1 OBJECTIFS

- Maîtriser l'utilisation des tableaux
- Maîtriser l'utilisation des boucles

2 TRAVAIL A REALISER

Nota : le travail demandé doit être terminé, en séance ou, à défaut, hors séance.

Booter sous Linux. Visualiser le sujet **dans un navigateur** pour bénéficier des liens, mais la version imprimable est plus agréable à lire (à l'écran ou sur papier) .
Créer un répertoire **`tp5`** dans `In101` sur votre compte.

2.1 Exercice 1 : Projet "weblog-analyzer" (tableaux)

Les serveurs web maintiennent habituellement des fichiers d'historique des accès aux pages web qu'ils supportent. Ces fichiers sont appelés fichiers log.
L'analyse de ces fichiers permet d'obtenir des informations utiles comme : quelles sont les pages les plus consultées, quelles sont les périodes de consultation les plus chargées, ...
Le projet weblog-analyzer sur lequel nous allons travailler est un programme qui réalise une analyse élémentaire d'un fichier log simplifié (fourni). C'est dans ce cadre que nous créerons et gèrerons des tableaux.

2.1.1 Ouvrir le projet

Télécharger le fichier `weblog-analyzer.jar` lié à cet énoncé, et l'enregistrer dans le répertoire `tp5` précédemment créé.

Lancer *BlueJ* et ouvrir, le fichier `.jar` sauvegardé ci-dessus. [menu *Project*, choix *Open non-BlueJ* ...].

2.1.2 Prendre connaissance du projet weblog-analyzer

Le projet weblog-analyzer est composé de 4 classes. Nous nous intéresserons essentiellement à la classe LogAnalyzer. Ce programme réalise une analyse temporelle d'un petit fichier log.

Nommé weblog.txt, il se trouve maintenant dans le répertoire tp5/weblog-analyzer. Il contient une suite de dates au format "année mois jour heure minute", chaque ligne étant censée correspondre à un accès à une page web du serveur. Visualiser le contenu de ce fichier et en prendre connaissance.

Editer et prendre connaissance de la classe LogAnalyzer. La méthode analyzeHourlyData compte combien d'accès ont été réalisés dans chaque tranche horaire pendant toute la durée couverte par le log. Le résultat est mémorisé dans le tableau hourCounts.

Créer un objet de la classe LogAnalyzer et exécuter la méthode analyzeHourlyData puis la méthode printHourlyCounts. Quels sont les moments les plus chargés et les moins chargés d'une journée ?

2.1.3 Compléter la classe LogAnalyzer

On se propose d'ajouter à la classe LogAnalyzer de nouvelles méthodes d'analyse. **Aucune** de ces méthodes ne devra comporter d'**instruction d'affichage** autre que celle déjà présente.

1. **Écrire** au bon endroit (voir les commentaires) une fonction numberOfAccesses ***(combien de paramètres ?)*** qui, en sommant les éléments du tableau aHourCounts, retourne le nombre total d'accès enregistrés dans le fichier log.
   Tester cette méthode à l'aide de la classe `WeblogAnalyzerTest` fournie, et cliquer sur le bouton `[Run Tests]`. *Tout est vert ?* ***(interdiction de modifier WeblogAnalyzerTest)***

2. Si ce premier test s'est bien passé, **écrire** une fonction busiestHour ***(combien de paramètres ?)*** qui, en analysant le tableau aHourCounts, renvoie la première tranche horaire la plus chargée.
   (c'est-à-dire un nombre entre 0 et 23)
   Tester cette méthode en décommentant `[F7]` la méthode de test correspondant, et `[Run Tests]`. *Tout est vert ?* ***(interdiction de modifier WeblogAnalyzerTest)***

3. **Écrire** une fonction quiestHour ***(combien de paramètres ?)*** qui, en analysant le tableau aHourCounts, renvoie la première tranche horaire non vide la moins chargée. *(c'est-à-dire un nombre entre 0 et 23)* Vérifier notamment le bon fonctionnement de cette méthode dans le cas d'un tableau aHourCounts contenant plusieurs zéros, en particulier au début (*ce qui est le cas du fichier weblog.txt fourni*), et retourner -1 s'il n'y a que des zéros.
   Contrainte : Ne parcourir le tableau qu'une seule fois.
   Tester cette méthode en procédant comme pour la précédente. *Tout est vert ?*

4. **Écrire** une fonction entière busiestTwoHours ***(combien de paramètres ?)*** qui, en analysant le tableau aHourCounts, détermine (et retourne) la première période de 2 heures consécutives la plus chargée (désignée par la 1ère heure de cette période, *donc un nombre de 0 à 22*).
   Tester cette méthode en procédant comme pour les précédentes. *Tout est vert ?*

5. **Écrire** une classe HourCount et une fonction rankHours2 :
   - Définir dans le projet weblog-analyzer une nouvelle classe de nom HourCount, comportant deux attributs : un entier aHour et un entier aCount, un constructeur HourCount(int pHour, int pCount), et offrant deux méthodes d'accès : getHour() et getCount().
   - Dans la classe LogAnalyzer, définir une fonction rankHours2 ***(combien de paramètres ?)*** qui, en analysant le tableau aHourCounts, renvoie un tableau de type HourCount[] des tranches horaires classées par valeurs décroissantes de compteur (mais tranches croissantes si compteurs égaux).
   - Exemple : Supposons que le tableau aHourCounts soit le suivant :

Elément	45	12	8	72	45	3	75	80	10	12	33	80
Indice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*

   Le tableau vTabHours2 serait initialisé comme suit :

Elément	0;45	1;12	2;8	3;72	4;45	5;3	6;75	7;80	8;10	9;12	10;33	11;80
Indice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*

   Le résultat de vTabHours2 trié serait le tableau suivant :

Elément	**7**;80	**11**;80	6;75	3;72	**0**;45	**4**;45	10;33	**1**;12	**9**;12	8;10	2;8	5;3
Indice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*

   - **Aide :** Pour trier le tableau, on peut employer le « tri à bulles ». Cette méthode consiste à **répéter un « parcours »** jusqu'à ce que celui-ci n'ait produit aucun échange de places. Un parcours consiste à parcourir tout le tableau en comparant les valeurs situées dans 2 cases consécutives et en les échangeant si elles ne sont pas dans le bon ordre.
   - **Tester** cette méthode en décommentant `[F7]` les 2 dernières méthodes de test. *Tout est vert ?*

2.2 Exercice 2 : Utilisation de la ligne de commande

1. Créer un nouveau projet `moyenne` sous BlueJ et créer la classe `Moyenne` sans attribut ni constructeur, dans laquelle nous construirons **progressivement** une méthode permettant de calculer la moyenne des nombres présents sur la ligne de commande. **A la fin de l'exercice 2**, par exemple, la commande `java Moyenne 9.5 10 15` devra au point 5. afficher `moyenne=11.5`

2. Définir la fonction `moyenne()` vue en TD. L'essayer par exemple avec `{9.5, 10.0, 15.0}`
   Tester `fonctionMoyenne()` dans la classe `MoyenneTest` ; *tout est vert ?*

3. Modifier cette fonction pour qu'elle accepte maintenant un tableau de `String` au lieu du tableau de `double`. Nous supposerons que chaque `String` représente bien un `double` (par ex.: `"3.14"`).
   Aide : découvrir la méthode `parseDouble()` de la classe `Double` ne sera pas une perte de temps !
   Contrainte pour tout l'exercice 2 : Ne pas utiliser de tableau intermédiaire.
   Retester `fonctionMoyenne()` dans la classe `MoyenneTest` ; *tout est vert ?*

4. Transformer cette fonction en **procédure à un seul paramètre** (le nombre d'éléments utiles sera considéré comme égal à la taille du tableau). D'autre part, elle ne devra plus retourner le résultat, mais l'afficher. S'il n'y a aucun nombre, afficher le message `pas de nombre ! .`
   Tester `procedureMoyenne()` dans la classe `MoyenneTest` ; *tout est vert ?*

5. Lire les explications (IV.3 et IV.4) sur la fameuse méthode `main()` et modifier le nom de votre procédure pour que votre programme fonctionne à partir de la ligne de commande. Essayer la commande du point 1. dans un terminal Linux.
   Tester `procedureMain()` dans la classe `MoyenneTest` ; *tout est vert ?*

2.3 Exercice 3 : La classe Ératosthène

Cet exercice va consister à réaliser l'exercice décrit dans ce sujet de TD.
Lire les explications sur la méthode du crible d'Eratosthène au début de l'exercice 6 dans le sujet de TD.

2.3.1 Créer un nouveau projet

Créer *dans le répertoire tp5*, précédemment créé, un nouveau projet BlueJ de nom eratosthene.

2.3.2 Créer une nouvelle classe `Eratosthene`

A) **Créer** les 2 attributs `aMax` et `aTab` (*point 1 du TD*).
Tester `testAttributs()` dans `EratostheneTest`. *Tout est vert ?*

B) **Créer** le constructeur à un paramètre entier (*point 2 du TD*) en commentant [F8] l'appel à `prepare()`.
Tester `testConstructeur()` dans `EratostheneTest`. *Tout est vert ?*

C) **Écrire** la procédure `prepare()`(*point 3 du TD*) en commentant [F8] les appels à `initV()` et `raye()`.
Décommentez maintenant [F7] l'appel à `prepare()` dans le constructeur.

D) **Écrire** la procédure `initV()` (*point 4 du TD*).
Tester `testInitV()` dans `EratostheneTest`. *Tout est vert ?*

E) **Écrire** la procédure `raye()` (*point 5 du TD*). Décommentez [F7] les 2 appels dans `prepare()`.
Tester `testRaye()` dans `EratostheneTest`. *Tout est vert ?*

F) **Écrire** la procédure `estPremier()` (*point 6 du TD*).
Tester `testEstPremier()` dans `EratostheneTest`. *Tout est vert ?*

G) **Écrire** la procédure `affiche()` (*point 7 du TD*).
Bouton `[Run Tests]` dans `BlueJ`. *Tout est vert ?*

2.3.3 Créer une méthode `essai()` (en travail personnel)

Passer un paramètre caractère pC et un paramètre entier pN et tenir compte des consignes suivantes :
- Le paramètre pC sera interprété comme une commande : **d**isplay, **g**reatest, **h**elp
- Le paramètre pN sera interprété différemment selon la commande ci-dessous
- `'d'` appellera `affiche()`
- `'g'` devra trouver le plus grand nombre premier inférieur ou égal à `pN`
- `'h'` affichera un message d'aide listant les commandes possibles (`pN` ne sert à rien dans ce cas)
- tout autre caractère provoquera l'affichage d'un message d'erreur signalant la commande `"h,0"`

Vérifier le bon fonctionnement de ce programme en testant extensivement « à la main ».

2.4 Terminer la séance

Si pas fait antérieurement, **générer** (après l'avoir complétée !) la documentation des 2 exercices, puis sauvegarder les projets ouverts, puis fermer BlueJ [menu Project, choix Quit]. Se déloger.