Travaux Pratiques N°2

Objectifs :

Ce TP couvre les chapitres suivants :

- Classes et objets
- Les tableaux
- Les chaînes de caractères

Il a comme objectifs de:

- S'exercer à l'utilisation des tableaux et les listes avec Java
- S'exercer à la manipulation des chaînes de caractères
- S'exercer à la spécification et à la programmation de classes élémentaires à partir d'un énoncé textuel ou d'un diagramme de classes UML.
- Sensibiliser l'élève ingénieur aux problèmes de fuite de mémoire liés aux références d'objets obsolètes.
- Apprendre coder proprement en respectant les conventions de nommage et en documentant le code avec JavaDoc.

Livrable :

Ce TP devra être réalisé en groupe de 3 étudiants. Le livrable est à rendre avant le 15/10/2018 à 23h59 par email. Le titre de votre email doit contenir l'identifiant du groupe. Votre livrable doit être au format *zip* ou *rar* et doit comprendra les codes sources commentés sous forme d'un projet java Eclipse.

Exercice 01 :

1. Ecrire une méthode *generatePassword* qui permet de générer et renvoyer un mot de passe de N caractères générés aléatoirement en utilisant les deux algorithmes simplifiés données ci-dessous.
2. Analyser les défauts de ces algorithmes et proposer une autre implémentation meilleure.

Algorithme 1 :

Tant que le nombre de caractère généré est inférieur à N faire :
    Générer un entier qui représente le code d'un caractère Unicode aléatoirement.
    Stocker ce caractère dans un tableau
Fin Tant que
Convertir le tableau de caractères en une chaîne de caractères. grâce au constructeur String(char[] value)) de la calasse String.

Algorithme 2 :

Motdepass = chaine vide
Tant que le nombre de caractère généré est inférieur à N faire :
    Choisir un caractère au hasard avec la méthode random de la classe Math parmi un ensemble de caractères autorisés représenté par un tableau
    Ajouter le caractère à Motdepass
Fin Tant que
retourner Motdepass

Exercice 02 :

Ecrire une classe ayant les méthodes statiques suivantes :

1. *readMatrix* qui permet de lire une matrice au clavier
2. *getPointCols* qui retourne dans une liste d'objets les positions et les valeurs de tous les points-cols trouvés dans une matrice passée en paramètre.
   (Les points cols est les éléments qui sont à la fois un maximum sur leur ligne et un minimum sur leur colonne)
3. Ecrire un programme de test

Exercice 03 *(Extrait d'un contrôle continu 2015/2016)*

Un nombre complexe z se présente en général en coordonnées cartésiennes, comme une somme a + bi, où a et b sont des nombres réels quelconques et i (l'unité imaginaire) est un nombre particulier tel que $i2 = -1$.

Deux nombres complexes sont égaux si et seulement s'ils ont la même partie réelle et la même partie imaginaire.

Le conjugué d'un nombre complexe ***a+bi*** est ***a-bi***.

Ecrire une classe ***Complexe*** qui dispose des méthodes suivantes :

- Un constructeur pour initialiser la partie réelle et imaginaire.
- Une méthode *parseComplexe(String pComplexe)* qui prend une chaîne de caractères en paramètres et retourne un nombre complexe si la chaîne de caractère représente bien un nombre complexe (cf. exemples ci-dessous), sinon elle déclenche une exception de type ***Exception***.

  Exemples :

  parseComplexe("2+3i") ➔ *Un objet complexe sera créé correctement*
  parseComplexe("2+i3") ➔ *Un objet complexe sera créé correctement*
  parseComplexe("23i") ➔ *Un objet complexe sera créé correctement*
  parseComplexe("0+23i") ➔ *Un objet complexe sera créé correctement*
  parseComplexe("23i+0") ➔ *Un objet complexe sera créé correctement*
  parseComplexe("23") ➔ *Un objet complexe sera créé correctement*
  parseComplexe("23+i0") ➔ *Un objet complexe sera créé correctement*
  parseComplexe("23+0i") ➔ *Un objet complexe sera créé correctement*
  parseComplexe("23+23") ➔ *Une exception doit être déclenchée*
  parseComplexe("i+23i") ➔ *Une exception doit être déclenchée*

- Une méthode : *Complexe conjugue()* qui retourne le conjugué de l'objet appelant cette méthode.
- Une redéfinition de la méthode *equals* permettant de comparer deux nombres complexes.
- Une méthode *toString* qui retourne la présentation sous forme d'une chaîne de caractère d'un nombre complexe : a+bi.

Exercice 04 *(Extrait d'un contrôle continu 2015/2016)*

La méthode statique *compareTab* de la classe **TabUtils** prend comme paramètres deux tableaux d'entiers *pTab1* et *pTab2*. Si les deux tableaux passés en paramètres sont de même taille, la méthode doit renvoyer un tableau d'entiers de même longueur et qui contient les nombres 1 et 0. Le $i^{ème}$ élément du tableau renvoyé contient 1 *si* $pTab1[i] \neq pTab2[i]$ et 0 si $pTab1[i] = pTab2[i]$.

Si les deux tableaux passés en paramètres ne sont pas de même taille, la méthode *compareTab* lève une exception.

Par exemple, Si *pTab1 = {1, 2, 3}* et *pTab2 = {3, 2, 1};* l'appel TabUtils.compareTab(pTab1, pTab2) renvoie le tableau {1,0,1}.

Ecrire la classe ***TabUtils*** et sa méthode *compareTab*.

Exercice 05 *(Extrait d'un contrôle continu 2015/2016)*

Dans cet exercice on suppose que les pages web traitées sont bien formées et que les balises HTML sont en minuscule et ne contiennent pas d'espaces inutiles. (Exemple : On ne peut pas trouver des balises telles que <h 1> , < h1 > ou <H1>).).

Le corps d'une page HTML est le code donné entre <*body*> et </*body*>.

On suppose que la densité d'un mot dans une page web peut être calculée par la formule:

$$densité\ (\boldsymbol{motX}) = \frac{nombre\ de\ fois\ le\ mot\ \boldsymbol{motX} est\ apparu\ dans\ le\ corps\ de\ la\ page\ \times nombre\ de\ caractères\ du\ mot\ \boldsymbol{motX}}{le\ nombre\ de\ caractères\ dans\ le\ corps\ de\ \ la\ page \times 0.5}$$

On modélise une page Web par une classe ***WebPage*** caractérisée par les propriétés suivantes :

- Son titre (le texte de la balise *title*)
- Son corps (le code HTML donné entre <*body*> et </*body*>)

Questions :

1- Ecrire la classe ***WebPage*** permettant de représenter une page web. En plus de ses attributs, cette classe a les méthodes suivantes :
- Un constructeur permettant d'initialiser ses attributs.
- Les accesseurs (getters) et mutateurs (setters) nécessaires.
- Une méthode ***double getDensity(String mot)*** pour calculer la densité d'un mot dans la page représentée par un objet de cette classe. (**Indication** : *On peut utiliser la méthode indexOf)*

2- Ecrire la classe ***WebPageAnalyser*** ayant les méthodes suivantes :
- ***static List<String> getAllH1TitlesFromWP(WebPage page) :*** Elle prend en paramètre une page web et retourne la liste de tous les titres h1 de la page web. (**Indication** : *On peut utiliser les méthode indexOf et substring*).

Exemple :

Pour la page web ci-dessous la méthode ***getAllH1TitlesFromWP*** doit retourner la liste:

{" Un nouveau tremblement de terre dans la région d'Al Hoceima"," Séisme à Al Hoceima : 15 blessés selon le ministère de la Santé","Séisme entre le Maroc et l'Espagne"}

<html><head><title> Séisme </title></head>

<body>

<h1>Un nouveau tremblement de terre dans la région d'Al Hoceima </h1>

Une nouvelle secousse sismique …

<h1>Séisme à Al Hoceima : 15 blessés selon le ministère de la Santé</h1>

Le séisme qui a concerné la région du Rif …

<h1>Séisme entre le Maroc et l'Espagne</h1>

Un séisme de magnitude 6,1 s'est produit lundi matin,…

</body></html>

- ***static List<WebPage> getEarthquakeWB(WebPage[] pages) :*** Elle prend en paramètres des pages web et retourne toutes les pages qui parlent du séisme d'Al Hoceima. On considère qu'une page parle du séisme d'Al Hoceima s'elle a au moins un titre de type h1 contenant la chaîne de caractères «Séisme d' Al Hoceima».

Exercice 06 : Fuite de mémoire dans une pile.

Lorsque l'on passe d'un langage à gestion de mémoire manuelle comme le C à un langage à base de ramasse-miettes comme JAVA, C# …, le travail de développeur est grandement simplifié du fait de la récupération automatisée des objets dont on n'as plus besoin. Cela peut conduire à croire qu'il n'y a plus besoin de se soucier de la gestion de la mémoire, ceci n'est pas entièrement vrai.

Considérons l'exemple de la classe ci-dessous qui gère une pile écrite par un débutant en programmation, cette classe contient une erreur subtile (fuite de mémoire).

1. Détecter la fuite de mémoire et expliquer les conséquences qui peuvent êtres engendrées dans une application qui utilise une telle pile.
2. Corriger la classe (en une seule instruction).
3. Ecrire un programme testant cette nouvelle implémentation

```
/**
 * Classe qui modélise une pile
 * @author  Beginner Programmer
 *
 */
public class BeginnerStack {
   private Object[] elements;
   private int size = 0;

   public BeginnerStack(int initialCapacity){
          elements = new Object[initialCapacity];
   }

   public void push(Object e){
          guaranteeCapacity();
          elements[size++] = e;
   }

   public Object pop() throws EmptyStackException{
          if(size == 0) throw new EmptyStackException();
          Object result = elements[--size];
          return result;
   }

/**
* Permet de s'assurer qu'il y a pas de palce pour au moins un élément
*  supplémentaire en doublant la capacité à chaque accroissement de la
*taille du tableau
*/
   private void guaranteeCapacity(){
          if(elements.length == size){
                 Object[] elementsAnciens = elements;
              elements = new Object[2* elements.length+1];
              System.arraycopy(elementsAnciens,0, elements, 0,size);
          }
   }
}
```

Exercice 7 : Utilisation des classes ArrayList et LinkedList

En utilisant la classe *ArrayList* écrire une classe *MessageManger* qui permet de gérer les messages d'un forum de discussion. Un message est caractérisé par un id, un titre, une date et un contenue.

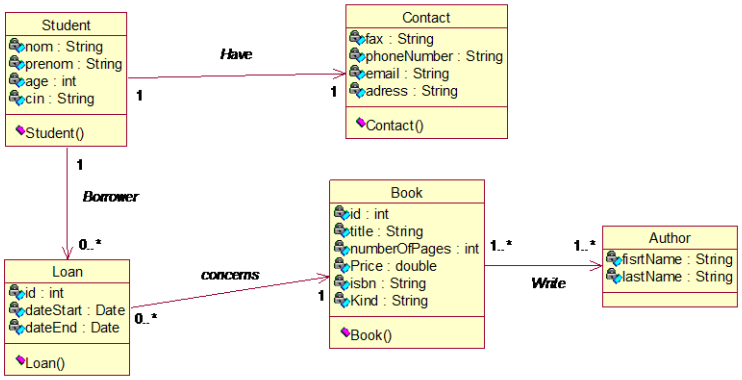
Dans la classe *MessageManger* Implémenter les méthodes suivantes :

- *getAllMessages : Retourne la liste des messages*
- *deleteMessage : Supprime un message de la liste en connaissant son id.*
- *UpdateMessage : Permet de mettre à jour un message existant dans la liste.*
- *findMessageById: permet de retrouver un message par id*
- *findMessageByTitle: permet de retrouver un message par titre*
- *getNumberOfMessages : retourne le nombre de message de la liste.*
- *getFirstMessage: retourne le premier message dans la liste*
- *getLastMessage: retourne le dernier message dans la liste.*
- *dddMessage : Ajoute un message à la liste.*

Refaire l'exercice en utilisant la classe LinkedList

Exercice 8 : Digramme de classes et implémentation en JAVA :

1- Implémenter les classes représentées dans le digramme de classes ci-dessus.
2- Ajouter un constructeur avec arguments dans les deux classes.
3- Ajouter les accesseurs (*getters*/*setters*) dans les deux classes.
4- Ajouter une méthode *showContact* qui permet d'afficher un contact.
5- Ajouter une méthode *showDetails* qui permet d'afficher les informations d'un étudiant (nom, prénom, adresse ….) .
6- Ajouter une méthode *equals* dans la classe *Student* et dans la classe Contact.
7- Ajouter dans la classe Contact la méthode *isEmailValide* utilisant les expressions régulières et qui permet de tester la validité d'une adresse email. (Pour simplifier on considère qu'un email est valide *ssi* il s'écrit sous la forme xx@yy.zz avec xx, yy et zz sont des chaines de caractères non vides).
8- Ajouter dans la classe Contact la méthode *isCinValide* qui permet de tester la validité d'un numéro d'identité national en utilisant une expression régulière.
9- On complète le digramme précédent en ajoutant les classes *Loan* , *Book* et *Author* (cf. digramme ci-dessous).



Implémenter les nouvelles classes et faire les modifications nécessaires sur les classes existantes.

10- Ajouter les méthodes suivantes dans la classe Book :
   - *addAuthor* qui permet l'ajout d'un auteur à la liste des auteurs d'un livre dans la classe Book.
   - *getAuthors*
   - Les accesseurs nécessaires
11- Ajouter les méthodes nécessaires pour la gestion des emprunts d'un étudiant.
12- Ajouter une classe *PrincipalProg* qui permet de tester toutes les méthodes implémentées dans cet exercice.

Exercice 9 : spécification et programmation de classes élémentaires à partir d'un énoncé textuel

1. Cahier des charges

Il s'agit de programmer une application simple qui simule la gestion des comptes bancaires. Cette application permet de créer et utiliser autant de comptes bancaires que nécessaires, chaque compte étant un objet, instance de la classe Compte.

Un compte bancaire est identifié par un numéro de compte. Ce numéro de compte est un entier positif permettant de désigner et distinguer sans ambiguïté possible chaque compte géré par l'établissement bancaire. Chaque compte possède donc un numéro unique. Ce numéro est attribué par la banque à l'ouverture du compte et ne peut être modifié par la suite. Dans un souci de simplicité (qui ne traduit pas la réalité) on adoptera la politique suivante pour l'attribution des numéros de compte :

- Générer un nombre de 8 numéros aléatoirement
- Vérifier que ce numéro n'est encore attribué à une personne.
- Si il est déjà pris en régénère un nouveau numéro.

Un compte est associé à une personne (civile ou morale) titulaire du compte, cette personne étant décrite par son nom. Une fois le compte créé, le titulaire du compte ne peut plus être modifié.
La somme d'argent disponible sur un compte est exprimée en DH. Cette somme est désignée sous le terme de solde du compte. Ce solde est un nombre décimal qui peut être positif, nul ou négatif.
Le solde d'un compte peut être éventuellement (et temporairement) être négatif. Dans ce cas, on dit que le compte est à découvert. Le découvert d'un compte est nul si le solde du compte est positif ou nul, il est égal à la valeur absolue du solde si ce dernier est négatif.

En aucun cas le solde d'un compte ne peut être inférieur à une valeur fixée pour ce compte. Cette valeur est définie comme étant - (moins) le découvert maximal autorisé pour ce compte. Par exemple pour un compte dont le découvert maximal autorisé est 2000 DH, le solde ne pourra pas être inférieur à -2000 DH. Le découvert maximal autorisé peut varier d'un compte à un autre, il est fixé arbitrairement par la banque à la création du compte et peut être ensuite révisé selon les modifications des revenus du titulaire du compte. Créditer un compte consiste à ajouter un montant positif au solde du compte. Débiter un compte consiste à retirer un montant positif au solde du compte. Le solde résultant ne doit en aucun cas être inférieur au découvert maximal autorisé pour ce compte.

Lors d'une opération de retrait, un compte ne peut être débité d'un montant supérieur à une valeur désignée sous le terme de débit maximal autorisé. Comme le découvert maximal autorisé, le débit maximal autorisé peut varier d'un compte à un autre et est fixé arbitrairement par la banque à la création du compte. Il peut être ensuite révisé selon les modifications des revenus du titulaire du compte.

Effectuer un virement consiste à débiter un compte au profit d'un autre compte qui sera crédité du montant du débit.

Lors de la création d'un compte seul le nom du titulaire du compte est indispensable. En l'absence de dépôt initial le solde est fixé à 0. Les valeurs par défaut pour le découvert maximal autorisé et le débit maximal autorisé sont respectivement de 800 DH et 3000 DH. Il est éventuellement possible d'attribuer d'autres valeurs à ces caractéristiques du compte lors de sa création.

Toutes les informations concernant un compte peuvent être consultées : numéro du compte, nom du titulaire, montant du découvert maximal autorisé, montant du débit maximal autorisé, situation du compte (est-il à découvert ?), montant du débit autorisé (fonction du solde courant et du débit maximal autorisé).

2. Travail demandé

1- A partir du cahier des charges élaborer une spécification puis une implémentation sous forme de classes JAVA :
   - Définir les attributs (variables d'instance, variables de classe) de la classe Compte.
   - Identifier et implémenter les méthodes proposées par la classe Compte. Pour chaque méthode on prendra soin, outre la définition de sa signature, de spécifier son comportement sous la forme d'un commentaire JAVA DOC.
   - De proposer un ou plusieurs constructeurs pour la classe Compte. Là aussi on complétera la donnée de la signature de chaque constructeur avec un commentaire JAVA DOC détaillant son utilisation.

2- Tester les méthodes ajoutées.

3- Réaliser une application console permettant d'effectuer les opérations décrites dans le menu ci-dessous:
   1. Créer Compte
   2. Créditer Compte
   3. Débiter Compte
   4. Effectuer un virement
   5. Afficher un Compte
   6. Modifier un Compte

4- Nous souhaiterions pouvoir afficher les opérations (effectuées ou refusées) sous forme de journalisation de l'application dans la console en cas de litige (il serait aussi possible d'utiliser en sortie un fichier texte). Ce service doit être implémenté par une classe distincte nommée *BankAppLogger*. Il faudrait garantir que notre application utilise une seule et même instance de la classe *BankAppLogger*. La classe *BankAppLogger* dispose de deux méthodes propres à l'utilisation de cette classe : *addLog*(string) et *showLog*().

   a- Implémenter la classe *BankAppLogger*
   b- Ajouter les appelles nécessaires à la méthode *addLog* dans la classe Compte.
   c- Mettre à jour le programme principal pour tester les nouvelles modifications.