Premier contact

Quelques conventions avant de commencer :

- La première lettre du nom d'une classe est en majuscule et toutes les autres en minuscules sauf si le nom de la classe est la concaténation de plusieurs mots. Dans ce cas, la première lettre de chaque mot est en majuscule et toutes les autres en minuscules, comme dans NombreComplexe ou EtudiantDeuxiemeAnnee.
- Les noms de fonctions ou de variables s'écrivent en minuscules sauf s'ils sont la concaténation de plusieurs mots. Dans ce cas, la première lettre de chaque mot **hormis le premier** est en majuscule et toutes les autres en minuscules comme dans temperatureMoyenne et notesEtudiantsDeuxiemeAnnee.
- Le source d'une classe est sauvé dans un fichier dont le nom est celui de la classe suffixé par .java. Le source de la classe Addition sera donc sauvé dans le fichier Addition.java. C'est ainsi que le compilateur retrouve les descriptions de classes dont il a besoin.

1) Taper le programme suivant sous emacs.

```
/**
 *  Petit exemple document&eacute;
 *
 * @author stage IPI IUP2
 **/

public class Addition
{
  /**
    Classe sans reel inter&ecirc;t
    **/

  public static void main ( String [] arg )
  {
    // ceci est une simple addition pour voir si tout marche bien
    System.out.print ( " 123 + 321 font ") ;
    System.out.println ( 123 + 321) ;
  }
  /* Commentaire qui n'ira pas dans la documentation */
}
```

Comment se comporte emacs au cours de l'édition ?

Compiler le programme (`javac Addition.java`) puis exécuter le (`java Addition`). Comment fonctionne t-il ? Définir le rôle des symboles '+' contenus dans ce programme.

Générer une documentation de la classe Addition en tapant `javadoc Addition.java`. Regarder la documentation ainsi générée avec netscape en lui donnant le chemin absolu du fichier généré.

2) Compiler et exécuter le programme Variables.java suivant.

```
/**
 *  Petit exemple de carre
 *
 * @author stage IPI IUP2
 **/

public class Variables
{
  public static void main ( String [] arg )
  {
    /** affiche le carre de l'entier suivant**/
    int number = 25;
    int carre_number = number * number;

    System.out.println ( number + " * " + number + " = " + carre_number ) ;
  }
}
```

Définir le rôle des symboles '+' ?

Générer une documentation commune aux classes Addition et Variables avec le nom des auteurs (`javadoc -author Addition.java Variables.java`).

Modifier le programme Variables.java de façon à effectuer l'élévation au carré dans une fonction membre carre() qui n'accepte aucun argument. Cette fonction sera appelée dans la fonction main() pour afficher le résultat.

3) Taper et compiler le programme BogusPocus.java.

[

Qu'observez-vous ? Corriger toutes les erreurs.

4) Ecrire un programme qui calcule $((2 + 6) * 5)/25$ et affiche le résultat à l'écran. Le calcul doit être réalisé sur les réels et non pas les entiers.

L'instruction if

1) Ecrire un programme utilisant une instruction if pour afficher le message 7 divise 179 467 si l'entier 179467 est divisible par 7 et le message 7 ne divise pas 179 467 sinon.

2) Ecrire un programme qui permute les valeurs de deux variables entières a et b si et seulement si a est positif et b inférieur à a.

3) Qu'affiche le programme Test.java suivant ? Pourquoi ?

```
public class Test {
   /**  Attention à la signification des tests !! */
   public static void main ( String arg [] ) {
      String s1 = "Test";
      String s2 = s1;
      String s3 = new String ( s1 );
      System.out.println ( "s1 = \"" + s1 + '\"');
      System.out.println ( "s2 = \"" + s2 + '\"');
      System.out.println ( "s3 = \"" + s3 + '\"');
      if ( s1 == s2 )
          System.out.println ( "s1 == s2" );
      if ( s1 == s3 )
          System.out.println ( "s1 == s3" );
      if ( s1.equals ( s2 ) )
          System.out.println ( "s1.equals ( s2 )" );
      if ( s1.equals ( s3 ) )
          System.out.println ( "s1.equals ( s3 )" );
   }
}
```

4) Ecrire un programme qui place la valeur vrai dans un variable boléenne a si et seulement si l'un des cas suivants est vérifié :

- le logarithme d'une variable réelle b est supérieur au carré d'une variable réelle c et le logarithme de la variable réelle b est inférieur à la variable réelle c,
- le carré de la variable réelle b est inférieur au logarithme de la variable réelle c et la variable réelle b est supérieure au logarithme de la variable réelle c.

Simplifier ce programme en écrivant une fonction pour vérifier les conditions ci-dessus.

5) Ecrire le programme RouleSaBille d'aide à la reconnaissance des truands notoires suivants en se basant sur un ensemble de questions-réponses astucieuses.

- Piano est petit, trapu et blond,
- RinceDoigts est grand, trapu et blond,
- MacIntosh est petit, fluet et brun,
- Babine est grand, fluet et roux,
- LaVente est petit, trapu et brun,
- Encephalor est grand, trapu et brun.

6) Ecrire un programme qui affiche le message Positif si l'inverse de la variable réelle x est positif et le message Indéterminé sinon. Faire attention au cas $x = 0$.

Paramètres et Boucles

1) Taper et compiler le programme suivant :

```
/**  Affiche les arguments recus  */

public class Argument {
   public static void main ( String []  arg ) {
      for ( int i = 0 ; i < arg.length ; i++ ) {
         System.out.println ( arg [ i ] );
      }
   }
}
```

Exécuter le programme en lui passant des paramètres :

```
java Argument premier?  a-b 2*3 4.5
```

Modifier le programme pour qu'il n'affiche plus qu'un argument sur deux.

2) Ecrire un programme Salutation qui affiche le message Bonjour, Hello ou Gutentag suivant que le premier paramètre de la ligne de commande est Fr, UK ou De. Compléter ce programme pour que le mot affiché soit encadré avec des étoiles de la façon suivante :

```
***********
* Bonjour *
***********
```

3) Ecrire un programme Max qui retourne le plus grand entier parmi trois entiers passés en paramètres sur la ligne de commande. Modifier ce programme pour qu'il accepte jusqu'à dix entiers sur la ligne de commande. On stockera ces entiers dans un tableau.

4) Ecrire un programme Statistiques qui réponde aux spécifications suivantes :

- java Statistiques max entier1 ... : retourne le plus grand entier parmi les entiers passés en paramètres.
- java Statistiques min entier1 ... : retourne le plus petit entier parmi les entiers passés en paramètres.
- java Statistiques moyenne entier1 ... : retourne la moyenne arithmétique des entiers passés en paramètres.
- java Statistiques somme entier1 ... : retourne la somme des entiers passés en paramètres.
- java Statistiques positif entier1 ... : retourne le nombre d'entiers positifs parmi les entiers passés en paramètres.
- java Statistiques affichage entier1 ... : affiche les entiers passés en paramètres.

5) Ecrire un programme Pascal qui prend un entier n en paramètre, et affiche les n premières lignes du triangle de Pascal. On rappelle que la ligne numéro n du triangle de Pascal se déduit de la ligne $n-1$ grâce aux deux relations suivantes :

$$\forall k \in [1, n[\,, \quad C_n^k = C_{n-1}^{k-1} + C_{n-1}^k \quad et \quad \forall n \in \mathbb{N}, \quad C_n^0 = C_n^n = 1$$

Tests et boucles

1) Réécrire l'algorithme du tri par sélection.
Le programmer en java au sein de la classe suivante :

```
/** Classe servant à tester une méthode de tri par sélection */

class Tri {

   static int [] aTrier = { 0, 8, 9, 1, 7, 6, 4, 5, 2, 3 };
   // tableau a trier pour experimenter la methode

   static void triSelection ( int [] donnees ) {
   /**  Fonction realisant le tri par Selection
        du tableau qui lui est donne en parametre  */
   /*
      À COMPLETER
   */
   }

   public static void main ( String [] arg ) {
   //  fait afficher le tableau avant et apres le tri
      afficher ( aTrier);
      triSelection ( aTrier );
      afficher ( aTrier);
   }
}
```

Il faut définir une fonction d'affichage du tableau.
Expliquer le typage de la fonction triSelection.

2) Que donnerait le bout de code suivant dans la méthode Main précédente :

```
int [] tab = { 1, 2, 3 };
int [] bleau = tab;
afficher ( tab );
bleau [ 1 ] = -1;
afficher ( tab );
```

3) Expliquer le typage et le rôle des différentes variables dans le programme suivant.
Expliquer le typage et la syntaxe d'appel des différentes méthodes proposées.
Complétez ce programme.

```
import java.util.Date;

class InfoContact
{
  public void afficheToi()
    {
      System.out.println(nom + "->" + email + " (cree le " + creation +")");
    }

  public String tonNom()
    {
      // completez
    }

  public String tonEmail()
    {
      // completez
    }

  public InfoContact(String unNom, String unEmail)
    {
      nom = unNom;
      email = unEmail;
      creation = new Date();

      derniereCreation = new Date();
    }

  public static Date taDerniereCreation()
    {
      // completez
    }

  private static Date derniereCreation;

  private String nom;
  private String email;
  private Date creation;
}

class TestInfoContact
{
  public static void main(String[] arg)
    {
      InfoContact ic = new InfoContact("toto", "toto@titi.fr");

      ic.afficheToi();

      System.out.println(InfoContact.taDerniereCreation());
    }
}
```

Classes et Objets

1) Taper et compléter le programme InfoContact de la page suivante. Regarder L'API pour la classe Date.

2) Ajouter dans la classe InfoContact une variable tableau. Cette variable sera une variable de classe, c'est-à-dire une variable visible et donc utilisable par toutes les méthodes de la classe. Cette variable désignera un tableau d'au plus MAX objets du type InfoContact. MAX sera pour l'instant fixé a 10.

- Ecrire dans la classe InfoContact une méthode
  public static void ajouter(InfoContac ic), qui ajoute l'objet ic dans le tableau.

- Ecrire dans la classe InfoContact une méthode
  public static void afficherTout(), qui affiche tous les éléments du tableau.

- Ecrire dans la classe InfoContact une méthode
  public static InfoContact chercher(String unNom), qui cherche si un objet de nom unNom est présent dans le tableau.

La méthode main créera plusieurs objets de type InfoContact, les ajoutera dans le tableau, fera afficher le contenu du tableau, cherchera la présence éventuelle d'un élément particulier dans le tableau.

```
import java.util.Date;

class InfoContact
{
  public void afficheToi()
    {
      System.out.println(nom + "->" + email + " (cree le " + creation +")");
    }

  public String tonNom()
    {
      // completez
    }

  public String tonEmail()
    {
      // completez
    }

  public InfoContact(String unNom, String unEmail)
    {
      nom = unNom;
      email = unEmail;
      creation = new Date();

      derniereCreation = new Date();
    }

  public static Date taDerniereCreation()
    {
      // completez
    }

  private static Date derniereCreation;

  private String nom;
  private String email;
  private Date creation;
}

class TestInfoContact
{
  public static void main(String[] arg)
    {
      InfoContact ic = new InfoContact("toto", "toto@titi.fr");

      ic.afficheToi();

      System.out.println(InfoContact.taDerniereCreation());
    }
}
```

Héritage

1) Créer une classe Ville. Cette classe aura deux variables d'instance : nom et nbHabitants. Ecrire le constructeur, ainsi que les méthodes que vous jugerez utiles.

2) Rajouter à l'exercice précédent la classe Capitale qui hérite de la classe Ville. Cette nouvelle classe aura une variable d'instance supplémentaire par rapport à la classe Ville : la variable pays.
Certaines méthodes seront simplement celles de la classe mère. D'autres devront être spécialisées.

Lectures d'Objets dans un Fichier

1) Documenter la classe InfoContact.
Rappel : la commande qui permet de générer une documentation pour la classe InfoContact, avec le nom de son auteur, est :
`javadoc -author InfoContact.java`.
Regarder la documentation ainsi générée, avec netscape, en lui donnant le chemin absolu du fichier généré.

2) On voudrait maintenant que le tableau, contenant les données de type InfoContact, ne figure plus dans la classe InfoContact, mais qu'il devienne un Objet de type Agenda.
Créer donc une nouvelle classe Agenda, avec un (ou plusieurs) constructeur(s), dans laquelle on placera désormais les méthodes ajouter, chercher, et AfficheTout. Attention : dans la classe Agenda, ces méthodes deviennent des méthodes d'instances.
Tout les agendas crées auront une taille fixe. Pour indiquer que la taille est une constante, on pourra qualifier la variable taille de final, en plus de static.

3) Copier le fichier carrere/MIAGE/IPI/agenda.obj dans votre répertoire.
Consulter dans l'API sur jaguar (à l'adresse : http ://opt/doc/lang/java/jdk1.1.6/api/packages.html), les classes java.io.ObjectInputStream et java.io.FileInputStream. Le fichier agenda.obj contient des objets de la classe InfoContact. En utilisant les méthodes de la classe java.io.ObjectInputStream, il est possible d'extraire un à un, les objets de ce fichier pour les écrire dans le tableau d'un Agenda.
Les objets contenus dans le fichier agenda.obj sont des objets "sérialisables", c'est-à-dire appartenant à une classe, qui imlémente l'interface Serializable. Vous devez donc indiquer dans la classe InfoContact, qu'elle imlémente l'interface Serializable. D'autre part il faudra préciser, pour chaque objet récupéré, qu'il est de type InfoContact, par un "cast" (cela consiste à faire précédé le nom de cet objet par le type InfoContact entre parenthéses).

Ajouter dans la classe Agenda un constructeur Agenda(String NomFichier), qui constuira un agenda contenant toutes les données présentes dans le fichier de nom NomFichier. On utilisera pour cela les méthodes de la classe java.io.ObjectInputStream. Attention : la plupart de ces méthodes peuvent "lever" une exception, c'est-à-dire signaler une erreur, par exemple une erreur du type IOException. Vous devez alors rajouter les mots : throws IOException (ou tout autre type d'exception), juste avant l'accolade ouvrante du code de chacune de vos fonctions, qui utilisent des méthodes levant des exceptions. On testera ensuite ce constructeur avec en argument le nom de fichier agenda.obj.

4) Modifier la méthode ajouter de la classe Agenda pour que les données (de type InfoContact) d'un agenda soient classées en ordre lexicographique par rapport à leur nom (consulter les méthodes de comparaison de la classe String).
Modifier la méthode chercher de la classe Agenda afin de faire une recherche dichotomique.

Interfaces, héritage

1) On veut créer une classe Ville et une classe Trajet implémentant une interface Représentable. On rappelle qu'une interface est semblable à une classe sans variables d'instance, dont les méthodes sont abstraites (c'est-a-dire non implémentées). Elle sert à unifier le comportement de toutes les classes qui implémentent cette interface.

La classe Ville aura deux variables d'instance : nom et nbHabitants déclarées private.
La classe Trajet aura trois variables d'instance : depart, arrivee et distance déclarées private.

Ces deux classes implémentent l'interface Représentable, contenant les méthodes abstraites : afficheNom() et affichePlus(). Pour la classe Trajet, la méthode afficheNom() affichera le nom de la ville de départ et de la ville d'arrivée, séparées par un tiret. La méthode affichePlus() affichera toutes les variables d'instances.

Ecrire le code de ces classes et de l'interface.

2) Rajouter à l'exercice précédent la classe Capitale qui hérite de la classe Ville. Cette nouvelle classe aura une variable d'instance supplémentaire par rapport à la classe Ville : pays. La méthode afficheNom() sera celle de la classe mère. La méthode affichePlus() devra être spécialisée.

3) Comment créer un package contenant les classes précédentes ?

Ecrire une classe TestRepresentable (à l'extérieur du package) contenant un tableau d' objets Representable (par exemple deux villes, deux capitales et deux trajets) de votre choix. Ecrire une mèthode afficheTout() affichant toutes les caractéristiques de tous les éléments d'un tel tableau (cette méthode doit pouvoir s'appliquer à n'importe quel tableau de Representable).

4) Consulter dans l'API à l'adresse :
http ://jccf.labri.u-bordeaux.fr/java/sundoc/jdk1.1/docs/api/packages.html,
les classes java.io.FileWriter et java.io.BufferedWriter (sous-classes de java.io.Writer).

Rajouter à la classe Restaurant du dernier TP une méthode enregistre, qui copie les renseignements concernant les restaurants dans un fichier, que vous nommerez annuaire. Dans ce fichier, vous devrez faire figurer sur chaque ligne un nom de restaurant, son adresse et son numéro de télépnone, séparés par des /.

Entrées/sorties, exceptions

1) Terminer le TP précédent :
Rajouter à la classe Restaurant du TP6 une méthode enregistre, qui copie les renseignements concernant tous les restaurants de la liste dans un fichier.

Attention, les méthodes d'écriture (et lecture) dans un fichier peuvent lever une exception du type IOException.
Dans un premier temps, on indiquera avant l'accolade ouvrante de la fonction enregistre : throws java.io.IOException.
Dans un deuxième temps, on essaiera de traiter l'exception à l'aide de la clause try - catch.

2) Consulter dans l'API les classes java.io.FileReader et java.io.BufferedReader (sous-classes de java.io.Reader). Puis rajouter à la classe Restaurant une méthode lireListe qui construit la liste de Restaurants à partir des renseignements contenus dans le fichier (on utilisera StringTokenizer).

3) Ecrire une classe PileString òu la pile est implémentée à l'aide d'une liste chainée de cellules. Chaque cellule aura les variables d'instances chaine (de type String) et suivante (de type Cellule).

Listes, Arbres

1) Reprendre l'exercice sur la Pile du dernier TP.

2) Pour faire un arbre, créer une classe Noeud qui poss'ede trois variables d'instance :
- filsDroit et filsGauche, deux références à des instances de Noeud ;
- valeur, un entier.

Cette classe possède un constructeur qui prend comme paramètre un entier et le range dans valeur, les autres variables ne référencent rien.

Elle possède une méthode ajoute prenant un entier en paramètre. Si cet entier est inférieur à valeur : si filsGauche ne référence rien, alors créer un nouveau Noeud (avec cet entier) et filsGauche référencera ce nouveau Noeud, sinon ajouter l'entier à filsGauche. Si l'entier est supérieur à valeur : faire de même avec filsDroit.

Cette classe possède aussi une méthode toString qui concatène le résultat de toString de filsGauche avec valeur et toString de filsDroit.

NB : si possible, aucune variable ne doit être déclarée public.

3) Créer une classe arbre qui contienne une variable racine de type Noeud, une m'ethode ajouter qui ajoute un entier dans l'arbre en bonne place et une méthode toString.

4) Créer une interface Comparable ayant une unique méthode inferieur prenant une autre instance de Comparable en paramètre.

Reprendre la question précédente en prenant la variable valeur de la classe Noeud du type Comparable.

Le tout formera un package Arbre.

5) Ecrire une sous-classe de String, StringComparable qui implémente l'interface Comparable.

Ecrire un programme qui range des chaines de caractères dans un arbre, puis les imprime dans l'ordre.