TP_1 : introduction à Java et à la programmation orientée objet

Contraintes et remise des TPs

- Indentez vos fichiers (Eclipse peut le faire pour vous !).
- La correction tiendra compte de la briéveté des méthodes que vous écrivez (évitez les fonctions de plus de 25 lignes) ; n'hésitez pas à découper une méthodes en plusieurs sous-méthodes (privées) plus courtes.
- Vos classes et méthodes doivent être documentées en utilisant JavaDoc, et vos méthodes (hors accesseurs et autres méthodes triviales) doivent avoir des tests unitaires JUnit.
- Pour l'UML, vous pouvez utiliser le logiciel UMLet (`http://www.umlet.com/`) ou l'outil en ligne `http://www.draw.io/`.
- Les noms de classe commencent par une majuscule.
- Les noms de méthodes et d'attributs commencent par une minuscule.
- La convention de nommage des accesseur est getNomAttribut() et setNomAttribut(...).
- Utilisez **un projet Eclipse par TP avec comme nom de projet TPX_nom1_nom2** pour le TP numéro X, et **un package par exercice** (ou série d'exercices dépendant les uns des autres).
- Pour le rendu, placez les schémas UML et autre documents à rendre éventuellement dans le répertoire du projet puis faites une archive tar.gz [1] de **l'ensemble du répertoire du projet** (dans votre workspace).
- Le rendu est à déposer sur Moodle **au plus tard 15 min après la fin du TP** (un rendu par binome).

Eclipse et UMLet

Pour programmer en Java, nous allons utiliser Eclipse [2]. C'est un logiciel libre que vous pourrez aussi utiliser sur votre ordinateur personnel ; tapez `eclipse` dans un terminal pour le lancer.

Pour realiser les diagrammes UML, vous pouvez utiliser l'outil UMLet [3] ou le site `http://www.draw.io/`

Notez que ces deux logiciels sont écrits en Java !

Barème

Tous les exercices : coefficient 1. Propreté, lisibilité et commentaires : coefficient 1.

Exercice 1 – Syntaxe

1. Pour faire une archive tar.gz : `tar cvzf mon_archive.tar.gz mon_repertoire`
2. `http://www.eclipse.org`
3. `http://www.umlet.com/`

1. Écrivez un programme qui calcule puis affiche la somme des 100 premiers entiers (de 0 à 100, 100 inclus). Vous programmerez cet algorithme dans la fonction main() d'une classe appelée Somme.
2. Écrivez un programme qui calcule puis affiche la factorielle de n, où n sera spécifié sur la ligne de commande. La classe devra s'appeler Factorielle.
   Pour convertir une chaîne de caractère, on peut utiliser la méthode de classe parseInt de la classe Integer :

```
int n = Integer.parseInt(s);
```

   Testez votre programme avec Eclipse (il faut ajouter l'argument dans le petit menu « run... ») puis dans un terminal, en dehors d'éclipse. Écrivez la ligne de commande à utiliser dans les commentaires de votre classe.

Exercice 2 – ArrayList

En vous aidant de l'annexe (documentation de ArrayList et de Random), réalisez un programme RandomArray ayant le comportement suivant :

- Créer un ArrayList<String>, initialement vide
- Remplissez le de 10 entiers aléatoires (entre 0 et 10, 0 inclut, 10 exclut) convertis en chaînes de caractères. Pour convertir un entier de type **int** en String, vous pouvez utiliser la fonction String.valueOf(**int** mon_entier) (attention, il ne s'agit pas d'une méthode comme nous les avons vues mais d'une méthode de classe – détails au prochain cours), par exemple :

```
int a = 42;
String s = String.valueOf(a);
```

- Afficher l'ArrayList.

Exemple :

```
$ java MonProgramme
1
2
1
3
5
4
9
2
7
0
```

Exercice 3 – La classe Vecteur

Dans cet exercice, on souhaite modéliser des vecteurs en trois dimensions à l'aide d'une classe que l'on nommera Vecteur. Ce type de classe est présent dans la plupart des logiciels manipulant des objets en 3D (moteurs de jeu, modeleurs, ray-tracers, ...). Les opération suivante doivent pouvoir être effectuées :

— créer un vecteur avec pour coordonnées $[0, 0, 0]$;
— créer un vecteur avec les coordonnées x, y, z fournies en argument au constructeur ;
— additionner le vecteur this avec un vecteur passé en argument puis renvoyer le résultat sous la forme d'un nouveau vecteur — on ne modifie pas this (méthode additionner) ;
— calculer puis renvoyer la norme de this (méthode calculerNorme) ;
— calculer puis renvoyer le produit scalaire de this avec un vecteur passé en argument (méthode calculerProduitScalaire) ;
— calculer puis renvoyer le vecteur correspondant à this après une rotation autour du centre du repère de α radians dans le plan x, y, α étant passé en argument — on ne modifie pas this (méthode tourner) ;
— afficher les coordonnées du vecteur sur la sortie standard (méthode afficher).

1. À l'aide de l'outil UMLet, dessinez un diagramme de classe UML représentant cette classe. Faites attention à bien spécifier tous les types de retour et les types des arguments.
2. Implémentez la classe Vecteur.
3. Testez votre classe à l'aide d'une fonction main() qui *vérifie* que chaque opération implémentée a le résultat attendu et affiche un message d'erreur si ce n'est pas le cas.

Exercice 4 – La classe Triangle

On souhaite maintenant s'appuyer sur la classe Vecteur pour réaliser une classe Triangle. Cette classe doit permettre d'effectuer les opérations suivantes :
— créer un triangle à partir de trois vecteurs (donnant les coordonnées des trois points) ;
— faire tourner le triangle autour du centre du repère en fonction d'un angle α passé en argument de la méthode — on modifie this (méthode tourner) ;
— afficher les coordonnées des vecteurs constituant this sur la sortie standard (méthode afficher).
— faire effectuer une translation du triangle — on modifie this — selon un vecteur passé en argument (méthode deplacer).

1. Ajoutez la classe Triangle au diagramme UML contenant déjà la class Vecteur.
2. Implémentez puis testez comme précédemment la classe Triangle.

Exercice 5 – La classe Objet3d

Un objet 3D est constitué :
— d'une liste de triangle (que l'on représentera par un ArrayList ;
— d'un centre de gravité ;
— d'une couleur, représentée par une classe Couleur (qui correspond à un triplet de réels $[r, v, b]$).

Les opérations que l'on souhaite pouvoir effectuer sur un Objet3D sont les suivantes :
— créer l'objet à partir de son centre de gravité et de sa couleur ;
— afficher les coordonnées de tous les points de l'objet (méthode afficher) ;
— ajouter des triangles à l'objet (méthode ajouter Triangle)

— effectuer une translation de l'objet en fonction d'un vecteur passé en argument (méthode déplacer) ;

1. Ajoutez les classes Couleur et Objet3D au diagramme UML.
2. Implémentez les classes Objet3D et Couleur.

Exercice 6 – 421

On souhaite maintenant réaliser un jeu inspiré du 421. Ce jeu se joue avec 3 dés. Au début de chaque tour, chaque joueur lance 3 dés. Chaque joueur peut alors décider de relancer un ou deux des trois dés (on ne peut pas relancer deux fois le même dé). On observe alors les combinaisons et on compte les points :

— 4, 2, 1 : 10 points ;
— Deux « 1 » et un autre dé : rapporte la valeur du troisième dé (par exemple 5 points pour 5, 1, 1) ;
— Trois chiffres consécutifs (par exemple 4, 5 ,6) : 2 points.

Pour demander à l'utilisateur d'entrer des entiers et des chaînes de caractères, on peut utiliser la classe Scanner. Par exemple :

```
import java.util.*;
...
    Scanner sc = new Scanner(System.in);
    System.out.println("x?");
    int x = sc.nextInt();
    System.out.println("tapez␣0␣pour␣rejouer␣?");
    int y = sc.nextInt();
    if (x == 0)
      ...
...
```

1. Dessinez un diagramme UML de classes pour modéliser ce jeu. Utilisez trois classes : une classe pour représenter *un* dé, une pour représenter un joueur et une autre pour le jeu.
2. Implémentez et testez votre jeu.

A ArrayList

ArrayList est une classe de l'API Java permettant de représenter un tableau de longueur variable. Documentation complète :
http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html

A.1 Import

Au début de chaque fichier utilisant un ArrayList, vous devez taper :

```
import java.util.ArrayList;
```

L'instruction import ressemble à #include en C. Nous y reviendrons en cours.

A.2 Déclaration et instanticiation

Lorsque l'on déclare un ArrayList, on doit spécifier le type des objets contenus dans le tableau en l'encadrant par « < » et « > » (il s'agit d'une *classe générique*, nous les aborderons en cours plus tard). N'importe quel nom de classe valide peut être utilisé mais on ne peut pas utiliser de types primitifs (int, float, etc.). Exemple :

```
import java.util.ArrayList;
public class Test
{
  //--- attributs ---
  // un ArrayList de String
  ArrrayList<String> a3;

  //---- methodes ---
  /**
  * Constructeur par defaut
  */
  Test()
  {
    // lors de la creation, il faut preciser le type complet
    a3 = new ArrayList<String>();
  }
}
```

A.3 Méthodes principales

Pour les exemples suivants, on suppose que l'on dispose d'une classe Element.

A.3.1 Ajout d'un élément à la fin

```
ArrayList<Element> mon_arraylist = new ArrayList<Element>()
Element mon_element = new Element();
mon_arraylist.add(mon_element);
```

A.3.2 Suppression d'un élément

```
int numero_element = 2;
//supprime le 3eme element (les indices commencent a 0)
mon_arraylist.remove(2);
```

A.3.3 Taille

```
int size = mon_arraylist.size();
```

A.3.4 Accès à un élément

```
// accede au 3eme element (les indices commencent a 0)
Element e = mon_arraylist.get(2);
```

B Tirage aléatoire

Pour tirer des nombres aléatoires, on utilise souvent la classe Random. Documentation complète :
http://http://java.sun.com/j2se/1.5.0/docs/api/java/util/Random.html

B.1 Import

```
import java.util.Random;
```

B.2 Déclaration et instanciation

```
// dans une methode :
Random r = new Random();
```

B.3 Utilisation

Tirage d'un entier entre 0 et n (exclu) :

```
Random r = new Random();
int x = r.nextInt(n);// n : valeur max (exclue)
```