Programmation Orientée Objet
TP 1 : bases de Java

https ://dpt-info.u-strasbg.fr/~kvanhoey

Introduction

Le code écrit pour les exercices de ce TP devra être remis sur la page Moodle de ce module d'ici le début de la prochaine séance. Le code devra être clair et commenté. Vous devrez également rédiger un rapport contenant les réponses aux questions posées, ainsi que tout ce qui vous semblera pertinent : remarques sur votre code, stratégie d'implémentation, difficultés rencontrées, ...

Documentation de l'API Java

La documentation de l'API officielle de Java se trouve sur ce site. Vous y trouverez la liste des classes disponibles nativement dans le JDK de Java et utilisables par tout vos programmes. Connectez-vous à cette page et familiarisez-vous avec cette documentation. Elle est organisée comme suit :

- en haut à gauche : liste de tous les paquetages ;
- en bas à gauche : liste de toutes les classes ;
- à droite : documentation de la classe sélectionnée.

N'hésitez donc pas à utiliser l'option de recherche de votre navigateur (CTRL+*f*) pour trouver une classe en particulier.
Exercez-vous à trouver quelques classes et méthodes vues en cours ou en TD, et regardez leur documentation. Vous pouvez par exemple chercher :

- classe *Double* ; méthode *parseDouble*
- classe *Integer* ; méthode *parseInt*
- classe *String* ; méthode *charAt*
- classe *Math* ; méthode *pow*

1 Programmation des exercices vus en TD

1.1 Classe *Calculatrice*

Programmer la classe *Calculatrice* avec les modifications proposées (affichage plus explicite du résultat, opération modulo et puissance, lecture interactive des données [1]).

1. Il est possible que l'opérateur de multiplication ne fonctionne pas. Si tel est le cas, remplacez l'argument « * » par « * ». Quel est selon vous le problème ?

1.2 Classes *Point* et *CreationPoint*

- Exécutez la 1ère version du programme tel qu'il est donné dans la fiche de TD ;
- testez vos différents affichages des points : affichage dans *main*, méthode *affiche* dans la classe *Point* ;
- modifiez les variables x et y afin qu'elles deviennent privées et modifiez le code source en conséquence ;
- répartissez les classes dans 2 fichiers ;
- compilez et exécutez le programme.

1.3 Types

- Testez votre programme ;
- complétez le programme pour calculer les nombres minimum et maximum de l'ensemble transmis comme argument lors de l'appel. Pour cela, dans la version avec des *int*, commencez par définir les méthodes suivantes :
  ```
  static int min(int a, int b)
  static int max(int a, int b)
  ```
  qui calculent respectivement le minimum et le maximum de 2 entiers. Pensez à utiliser l'opérateur conditionnel
  ```
  <condition> ? <expression_si_vrai> : <expression_si_faux> ;
  ```
  plutôt qu'une instruction *if*.
- effectuez le calcul dans la méthode *main* et affichez le résultat en mettant en œuvre l'algorithme suivant.
  Initialisation : définir un attribut minimum (respectivement maximum) initialisé avec la valeur du premier argument.
  Boucle : prendre l'argument suivant et calculer le minimum (maximum) entre le minimum (maximum) déjà calculé et la valeur qui vient d'être lue.
  Arrêt : tous les entiers transmis ont été lus.
  Le minimum (maximum) se trouve dans la variable minimum (maximum).
- vérifiez et expliquez pourquoi les méthodes **min** et **max** sont déclarées *static*. Faire les mêmes calculs de minimum et de maximum avec des *Integer*. La méthode **min** aura pour profil :
  ```
  static Integer min (Integer a, Integer b)
  ```
  Elle appellera la méthode d'instance définie dans la classe *Integer* :
  ```
  int compareTo(Integer b)
  ```
  dont le résultat entier est :
  - $= 0$ si les deux *Integer* sont égaux ;
  - < 0 si l'instance courante est strictement inférieure à b
  - > 0 si l'instance courante est strictement supérieure à b.

2 Introduction aux outils du JDK

Parmi les outils les plus utilisés de la JDK, on peut citer **java** et **javac** que vous connaissez déjà, ainsi que **Javap**, *jar* ou **javadoc**. Informez-vous sur celles-ci ainsi que les autres facilitées fournies par le JDK Java.

2.1 Javadoc

Javadoc est un outil permettant de générer un fichier de documentation de votre classe, dans le même style que l'API Java. Pour cela, cet outil lit le code source et les commentaires au style **javadoc** que vous y aurez ajouté.

Exercice :

- retournez dans le répertoire où se trouve la classe *Point* ;
- créez un sous-répertoire nommé *doc/* ;
- générez la documentation de la classe *Point* grâce à la commande

```
javadoc -d doc/ *.java
```

- regardez quels fichiers ont été générés dans le répertoire *doc/* et ouvrez le fichier *index.html* avec votre navigateur ;
- commenter votre code en vous aidant de la documentation **javadoc** (et en particulier de la section « Commenting the Source Code ») de façon à générer une documentation riche de votre classe *Point* ;
- regénérez la documentation et visualisez le résultat.

2.2 Compilation et exécution

Questions :

1. Après compilation, un fichier *.class* est généré. Que contient-il ?
2. À quoi sert la commande **javap** ? Que fait l'option **javap -c** ?
3. Qu'est-ce qu'un fichier *.jar* et quel est son intérêt ? Outre la doc Java, le manuel en ligne de commandes pourra également vous aider à répondre à cette question.
4. Outre le répertoire *doc/*, créez deux répertoires *src/* et *bytecode/*. Déplacez les fichiers sources dans le répertoire *src/* et, depuis le répertoire parent, compilez les fichiers de façon à ce que le bytecode se trouve dans le répertoire dédié (aidez-vous de **man javac**). Quelle commande avez-vous utilisé ?
5. Toujours depuis le répertoire parent, exécutez votre programme. Aidez-vous du manuel de la commande **java** si cela ne fonctionne pas. Quelle commande avez-vous utilisé ?

2.3 Applets

Voici un exemple minimal d'applet :

```
<html>
  <head>
    <title>Mon applet</title>
  </head>
  <body>
    Mon applet :
    <applet code="MonApplet.class" width="90" height="90"></applet>
  </body>
</html>
```

Une applet est une fenêtre HTML qui charge le bytecode d'une classe qui étend la classe *JApplet* du package *swing* et qui définis la méthode *paint*. Dans l'applet définie ci-dessus, le bytecode *MonApplet.class* est chargé. Un exemple quasi minimal de classe pouvant fournir le code d'une applet est donné ci-dessous :

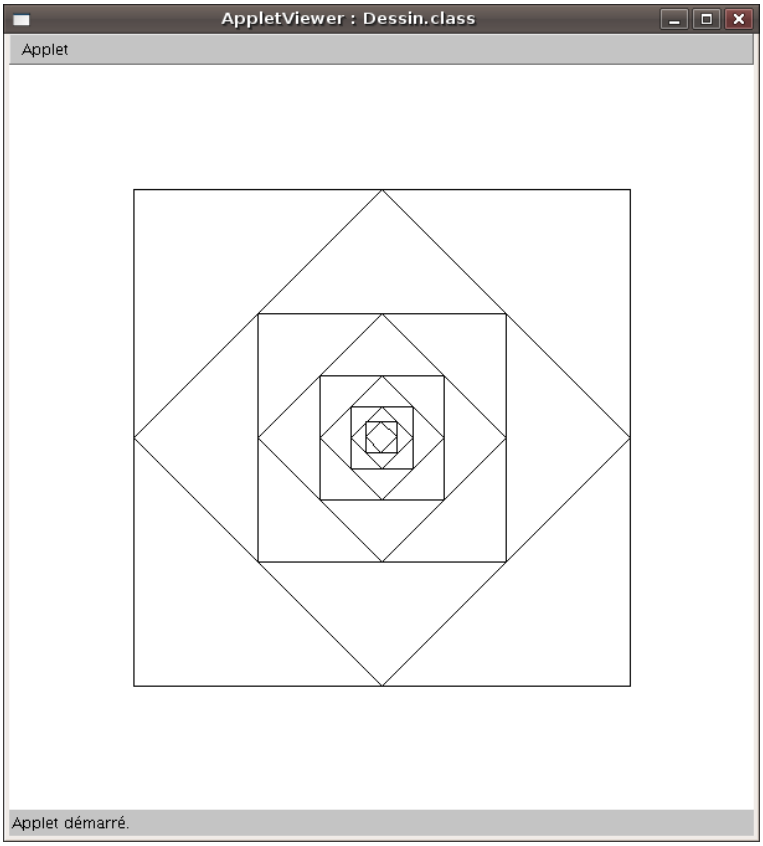
```
import java.awt.* ;
import javax.swing.* ;

public class MonApplet extends JApplet
{
  public void paint(Graphics g)
  {
    g.drawString(``Hello world!'',20,20) ;
  }
}
```

Compilez cet exemple et lancer l'applet (avec la commande **appletviewer**).

Exercice : applet de dessin

La classe `Graphics` du `awt` définit un certain nombre de méthodes de dessin. Reprenez la structure du code source et du fichier HTML de l'exercice précédent et modifiez les pour construire une applet qui dessine les formes simples suivantes :

- un segment est défini par ses extrémités A et B : `drawLine(int,int,int,int)` prend en paramètres les coordonnées xA, yA, xB, yB.
- un rectangle est défini par son coin supérieur gauche A, sa largeur l et sa hauteur h : `drawRect(int,int,int,int)` et `fillRect(int,int,int,int)` prennent en paramètres xA, yA, l, h. `fillRect` remplit le rectangle avec la couleur courante.
- un polygône est défini par les coordonnées de ses sommets et son nombre de sommets : `drawPolygon(int[],int[],int)` et `fillPolygon(int[],int[],int)` prennent en paramètres la liste des coordonnées x, la liste des coordonnées y, et le nombre de sommets. `fillPolygon` remplit le polygône avec la couleur courante.
- une ellipse est dessinée en fonction du rectangle dans lequel elle est inscrite : `drawOval(int,int,int,int)` et `fillOval(int,int,int,int)` prennent les mêmes paramètres que `drawRect`.
- pour sélectionner une nouvelle couleur : `setColor(Color c)` prend en argument un objet `Color` dont les champs sont par exemple `black, green, blue,` etc...
  Faites un programme qui dessine automatiquement la figure 2.3 ci-dessous ou toute autre figure qui vous plaît.


AppletViewer : Dessin.class
Applet
Applet démarré.