TP 3 : Graphique

1 Toujours plus loin

1.1 Les Packages

Un package est généralement un répertoire où ce situe des fichiers class ou d'autre répertoire. Ces répertoires peuvent être compresser en fichier donnant lieu à ces fameux .jar. Vous pouvez considérer aussi que le répertoire où vous avez compilé vos fichier java par exercice est un package.

Exemple créez un répertoire du nom de "dossier" et écrivez les deux fichiez suivant :

fichier ./dossier/TestPack.java :

```
package dossier;
public class TestPack{
         TestPack(){
                System.out.println("ca marche");
        }
}
```

fichier ./TestP.java :

```
public class TestP{
        public static void main(String arg[]){
                dossier.TestPack p=  new dossier.TestPack();
        }
}
```

1. *Que remarquez vous lors de la compilation ? Expliquez pourquoi.*
2. *Modifiez le fichier "./dossier/TestPack.java" afin de permettre la compilation.*
3. *Vous remarquez que le nom du package apparaît devant la classe pour l'appeler. Il existe un moyen de dire à java qu'il peut chercher dans un package : c'est la commande* `import`
   Dans le fichier TestP.java enlevez tout les "dossier." Que remarquez vous ? et ajoutez `import dossier.TestPack ;` *au début du fichier. Que remarquez vous ?*

Remarque : *Avec ceci plus besoin de mettre des java.io devant les* `FileReader`.
Remarque2 : *On peut utiliser le symbole '*' pour dire de prendre tout le contenu du package.*

1.2 Les classes abstraites

Les classes abstraites sont des classes qui ne sont pas destinées à avoir d'instance (et ne peuvent pas en avoir). En général, en plus d'un certain nombre de méthodes "normales", elles contiennent des méthodes abstraites qui sont simplement des déclarations de méthodes contenant leur signature mais pas leur code.

En Java, pour déclarer une méthode abstraite, il faut préfixer sa signature avec le mot-clé `abstract`*. Pour déclarer une classe abstraite, il faut préfixer sa déclaration avec le même mot-clé* `abstract`*. Toute classe conte- nant une methode abstraite doit être déclarée abstraite mais il n'est pas nécessaire qu'une classe contienne une méthode abstraite pour qu'on puisse la déclarer abstraite.*

```
abstract class A {
  void methode1() {
     ...
  }
  abstract int methode2(int x); // méthode abstraite
...
}
```

Toute classe qui hérite d'une classe abstraite est abstraite et doit être explicitement déclarée abstraite sauf si elle redéfinit toutes les méthodes abstraites de sa superclasse. Les classes abstraites servent à modéliser des types d'objets qui n'ont pas de réalité concrète.

Par exemple, un "moyen de transport" est un concept abstrait tandis qu'une "voiture" peut correspondre à un objet concret. Si dans une application on a besoin de manipuler des objets dont les comportements sont voisins, il peut être utile de créer une superclasse abstraite qui caractérise et factorise ce qu'ils ont en commun.

1.3 Interface

Les interfaces sont une spécificité du langage Java. Il s'agit de classes abstraites particulières : tous ses attributs doivent explicitement être static et final et toutes ses méthodes sont implicitement abstraites. On les déclare comme des classes mais en remplaçant le mot-clé class par interface. Elles sont implicitement abstraites.

Exemple :

```
interface MonInterface
{
final static int uneConstante = 10;
void uneMethode(); // méthode implicitement abstraite
}
```

Contrairement à une classe qui ne peut hériter que d'une seule classe, une interface peut hériter de plu- sieurs interfaces. Dans ce cas, on fait comme habituellement, on utilise le mot-clé extends mais en indiquant la suite d'interfaces desquelles on hérite en les séparant par des virgules. Une classe peut aussi hériter de plusieurs interfaces mais il faut alors utiliser le mot-clé implemclasse peut hériter d'une autre classe et de plusieurs interfaces.

```
interface Interface1 extends InterfaceMere {
  void nouvelleMethode(int x);
}
class UneClasse extends ClasseMere implements Interface1, Interface2 {
  void nouvelleMethode(int x){
     ...
```

```
  }
  ...
}
```

2 Ma première application graphique

2.1 Hello Window !

```
import javax.swing.*;

public class HelloWindow extends JFrame{
        public static void main(String args[]){
                new HelloWindow();
        }
        HelloWindow(){
                setTitle("Ma premiere Fenetre !");
                setSize(300,200);
                getContentPane().add(new JLabel("Salut a tous !!"));
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                setVisible(true);
        }
}
```

1. *implementez ce programme et testez le.*
2. *Comprenez le en essayant différents paramètres.*

2.2 JPanel

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

public class Gui extends JFrame{
        Panneau p=new Panneau();
        public static void main(String args[]){
                new Gui();
        }
        Gui(){
                setTitle("Mon premier dessins !");
                setSize(300,200);
                getContentPane().add(p);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                setVisible(true);
        }
}

class Panneau extends JPanel{
        public void paintComponent(Graphics g){
                super.paintComponent(g);
                g.drawLine(15,10,100,50);
        }
```

```
}
```

1. *implementez ce programme et testez le.*
2. *Comprenez le en essayant différents paramètres.*
3. *Modifiez le pour voir afficher des courbes du type* e^x , $\frac{sin(x)}{x}$, ...
   Attention à la taille du panneau qui peut être variable selon la taille de la fenêtre. vous pouvez récupérer les information de taille avec la fonction `int getWidth()` *et* `int getHeight()`.

3 Graphiques et fichiers

1. *Créer avec votre éditeur de texte préféré, un fichier contenant les dates et les notes d'élève.*
2. *Calculer la moyenne mensuelle, trimestrielle et annuelle.*
3. *Placez le tout dans un graphique permetant de suivre l'évolution de l'élève.*