UFR ST - Besançon- L2 Info - Année 2015/16
Programmation par Objets
TP 4 - API Java, composants graphiques

L'objectif de ce TP est de savoir utiliser des objets déjà programmés en Java, et disponibles dans l'API (*Application Programming Interface*) du langage. Ce TP constitue aussi une introduction à la réalisation d'interfaces graphiques en Java. Vous afficherez une fenêtre d'une interface graphique, et y positionnerez des composants graphiques tels que des boutons, des cases à cocher, etc. Malheureusement, cette interface ne sera qu'une maquette, inerte. En effet, faire réagir une interface à des événements tels que des clics de souris ou des saisies clavier demande des connaissances techniques supplémentaires qui ne pourront être abordées qu'en L3.

La documentation de l'API Java est disponible à l'URL suivante :
`http://java.sun.com/javase/7/docs/api/`
Le lien est sur Moodle. Vous aurez besoin de consulter cette documentation pour réaliser ce TP.

La bibliothèque `swing` de Java fournit des *composants graphiques* prêts à être utilisés. Ce sont des éléments habituels d'une interface graphique tels que des fenêtres, des boutons, des zones de saisie de texte, etc.

Pour utiliser cette bibliothèque, vous devrez placer les deux lignes suivantes au tout début des fichiers Java que vous définirez :

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

N.B. N'oubliez pas le 'x' dans `javax.swing`!

Exercice 1. Construire et afficher une première fenêtre

Les fenêtres sont décrites par le type Java `JFrame`. On peut donc disposer d'un objet nommé `fen` en instanciant JFrame :

```
JFrame fen = new JFrame();
```

La fenêtre existe en mémoire, mais elle doit être manipulée au moyen des méthodes de `JFrame` avant d'être visible sous une forme acceptable à l'écran. Voici quelques méthodes de `JFrame` :

- `setSize(int largeur, int hauteur)` permet de fixer en pixels la largeur et la hauteur de la fenêtre. Par exemple, l'instruction `fen.setSize(50,80);` assigne à la fenêtre `fen` une largeur de 50 et une hauteur de 80 ;
- `setLocation(int posX, int posY)` permet de déterminer la position en X et en Y du coin supérieur gauche de la fenêtre sur l'écran (en pixels) ;
- `setTitle(String titre)` permet d'attribuer un titre à la fenêtre ;
- `setVisible(boolean visible)` permet d'afficher (avec un paramètre true) ou de masquer (avec un paramètre false) la fenêtre à l'écran. Chaque fois que vous apporterez une modification à la fenêtre, il sera nécessaire de la ré-afficher par `setVisible(true)` pour que les modifications deviennent visibles à l'écran ; il n'est pas nécessaire de rendre la fenêtre invisible entre deux modifications.

Notez aussi que pour que le fait de fermer la fenêtre `fen` au moyen de son bouton de fermeture mette fin au programme, il est nécessaire de placer l'instruction

```
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

après l'instanciation de `fen`.

Question 1. Créez une classe nommée `VoirFenetre`, sans attributs et avec juste une méthode `main()`. Dans la méthode `main()` :

- déclarez et créez une fenêtre `fen` ;

– faites en sorte que la fermeture de `fen` provoque l'arrêt de votre programme ;
– affectez à `fen` une largeur de 300 et une hauteur de 500 ;
– placez `fen` au point $(50, 50)$ de l'écran ;
– donnez le titre *Premiere Fenetre* à `fen` ;

Question 2. Affichez `fen` à l'écran.

Exercice 2. Placer des composants graphiques à l'intérieur d'une fenêtre

En Java, on n'ajoute pas des composants graphiques à une fenêtre directement, mais on les ajoute à son *contenu*. Le contenu d'une fenêtre est du type Java `Container`, et il peut être récupéré au moyen de la méthode `getContentPane()` de `JFrame`. Ainsi, nommons `contFen` le contenu de la fenêtre `fen`. Pour disposer d'une instance du contenu de `fen` auquel on puisse ajouter des composants graphiques, il est nécessaire d'effectuer l'instruction

```
Container contFen = fen.getContentPane();
```

La méthode `add()` d'un `Container` permet ensuite d'ajouter le plus simplement du monde une instance d'un composant graphique au contenu de la fenêtre. Par exemple, si b est une instance d'un bouton (type `JButton` en Java), l'ajout de b au contenu `contFen` de `fen` se fait par l'instruction

```
contFen.add(b);
```

Attention. Il est techniquement nécessaire de définir une politique de placement pour indiquer à Java comment doivent être répartis les composants qu'on ajoute à l'intérieur de la fenêtre. Une façon simple de procéder est d'effectuer l'instruction

```
contFen.setLayout(new FlowLayout());
```

sur `contFen` (les composants seront alors répartis équitablement les uns à côté des autres au fur et à mesure de leur ajout).

Les boutons à cliquer en Java. Les boutons sont du type `JButton`. On peut fixer le texte contenu par un bouton directement lors de son instanciation. Par exemple, la création d'un bouton nommé `bQuitter` contenant le texte `Quitter` est obtenue par

```
JButton bQuitter = new JButton("Quitter");
```

Le bouton `bQuitter` est alors prêt à être ajouté à un conteneur, comme par exemple dans

```
contFen.add(bQuitter);
```

La méthode `setEnabled(boolean actif)` permet de rendre actif ou pas un bouton. Un bouton inactif apparaît grisé. Par défaut, un bouton est actif. Pour le désactiver, on doit invoquer sur lui la méthode `setEnabled(false)`.

Question 1. Ajoutez deux boutons `Quitter` et `Sauvegarder` à votre fenêtre. N'oubliez pas de ré-afficher la fenêtre avec `setVisible(true)` pour constater l'ajout.

Question 2. Modifiez votre programme pour que le bouton `Sauvegarder` apparaisse désactivé.

Les cases à cocher en Java. Elles sont du type `JCheckBox`. On peut fixer leur intitulé dès la création comme dans l'exemple suivant :

```
JCheckBox ccChoix1 = new JCheckBox("Choix 1");
```

Il est possible de simuler un clic sur la case par la méthode `doClick()` de `JCheckBox`.

Question 3. Ajoutez des cases à cocher à votre fenêtre avec les choix suivants :
– Java est génial
– J'aime la Programmation Objet
– J'aime aussi les jeux vidéo
– Moi j'aime pas les vacances
Libre à vous de cocher les cases vous concernant :-)

Les boutons radio en Java. Contrairement aux cases à cocher, ils n'offrent en principe que des choix exclusifs (par exemple *refuser* ou *accepter*). Ils sont définis en Java par le type `JRadioButton`, et on peut fixer leur intitulé à la création, comme les cases à cocher.

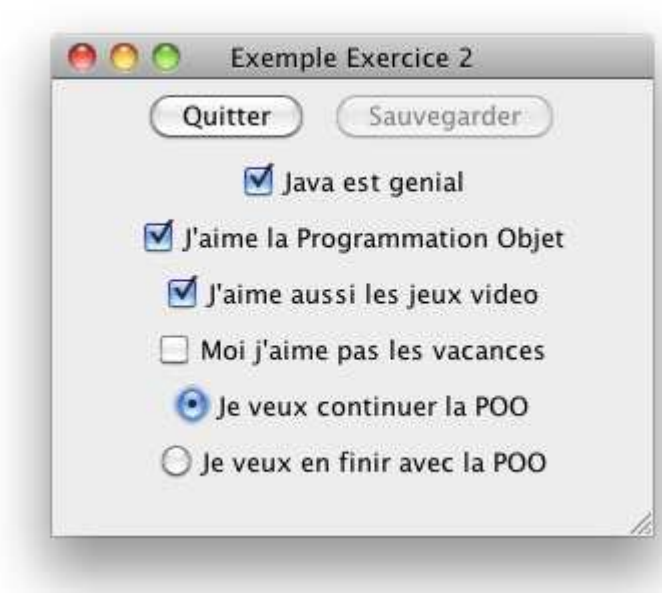
Mais pour qu'ils offrent des choix mutuellement exclusifs, ils doivent être réunis au sein d'un même *groupe* (type Java `ButtonGroup`). En pratique :

- créer une instance de `ButtonGroup` ;
- créer autant d'instances de `JRadioButton` qu'il y'a de choix possibles ;
- ajouter chaque bouton radio au groupe par la méthode `add()` de `ButtonGroup` ;
- ajouter ensuite chaque bouton radio au conteneur de la fenêtre par la méthode `add()` du conteneur.

Question 4. Ajoutez deux boutons radio à choix exclusif à votre fenêtre avec les choix suivants :

- Je veux continuer la POO
- Je veux en finir avec la POO

Voici un exemple de ce à quoi pourrait ressembler cette fenêtre :



Exercice 3. Coder une classe `Horloge` qui s'initialise à l'heure du système

Question 1. Codez en Java une classe `Horloge` dotée

- de 3 attributs `h, m, s` (pour heure, minute et seconde),
- d'un constructeur récupérant l'horaire courant du système,
- d'une méthode `update()` qui ajuste l'horaire enregistré dans les attributs à celui du système.

Comment récupérer l'horaire courant ?

N.B. La méthode décrite dans cette section repose sur une version de java antérieure à la nouvelle version java 1.8. Elle fonctionne toujours, mais java 1.8 a introduit une nouvelle API sous la forme d'un package nommé Time, qui introduit en particulier deux nouvelles classes JavaTime et JavaDate qui remplacent avantageusement les anciennes API. Si vous préférez, vous pouvez utiliser cette nouvelle API pour cet exercice : `https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html`

Sinon, avec l'ancienne API, on utilisera une instance de la classe Java `GregorianCalendar`, dont vous consulterez la documentation dans l'API Java. À la création de chaque nouvelle instance de `GregorianCalendar`, la date et l'horaire actuel sont enregistrés sous forme de constantes dans l'instance. On peut les récupérer par la méthode `get()`. De plus, la classe GregorianCalendar propose une méthode statique nommée getInstance() qui retourne une instance de GregorianCalendar. Cela permet de ne pas enregistrer dans une variable une référence à cette instance, en ne la créant que « à la demande ».

Ainsi, on peut par exemple obtenir une heure actualisée au moyen de :
`GregorianCalendar.getInstance().get(GregorianCalendar.HOUR_OF_DAY);`

N.B. Vous devez inclure la bibliothèque `java.util.*` pour avoir accès à `GregorianCalendar` :
`import java.util.*;`

Question 2. Rajoutez à la classe `Horloge` une méthode `toString()` qui calcule une chaîne présentant l'horaire au format `hh:mm:ss` (2 chiffres pour l'heure, 2 pour les minutes et 2 pour les secondes, et le symbole `':'` comme séparateur). Par exemple, pour `h=9`, `m=12` et `s=5`, la chaîne à calculer est `09:12:05`.

Exercice 4. Affichage graphique de votre horloge

Codez en Java un programme principal sous la forme d'une classe `FenHorloge`, destinée à afficher l'horloge sous la forme d'une fenêtre d'une interface graphique. Cette classe ne possédera que la méthode `main()`, dans laquelle vous réaliserez tous les traitements.

Un exemple d'une telle horloge est fourni dans le fichier `HorlogeGraphique.class` que vous pouvez télécharger depuis le site des TPs. Pour visualiser l'exemple, récupérez ce fichier ainsi que le fichier `Horloge.class`, puis exécutez le :
`java HorlogeGraphique`

Vous pouvez utiliser le composant graphique `JLabel` pour afficher l'horaire de l'horloge : `JLabel` est un composant permettant d'afficher du texte. Le texte à afficher est fourni au composant `JLabel` sous la forme d'un paramètre de la méthode `setText()` de `JLabel`.

L'idée pour réaliser ceci est de :
- créer une instance de `Horloge` ;
- ajouter à la fenêtre le composant `JLabel` contenant le texte calculé par la méthode `toString()` de votre instance de `Horloge` ;
- dans une boucle
  - mettre à jour à chaque itération l'horaire de votre instance de `Horloge` (grâce à la méthode `update()`) ;
  - mettre à jour le composant `JLabel` avec `setText` ;
  - actualiser l'affichage de la fenêtre avec la méthode `setVisible(true)` de la fenêtre.

N.B. Cette manière de faire est facile à programmer, mais elle est susceptible de poser des problèmes, selon le système sur lequel vous exécutez votre programme ! Le problème vient du fait que chaque invocation de update() crée une nouvelle instance de GregorianCalendar. Or comme cette invocation se fait en boucle, on a des allocations mémoire permanentes qui peuvent facilement saturer la mémoire. Tout dépend de la capacité du *Garbage Collector* à récupérer la mémoire efficacement. **Si vous rencontrez ce problème :** rajoutez la méthode suivante dans votre classe `FenHorloge` :

```
public static pause(int nbMillisecondes) {
      try {
            Thread.sleep(nbMillisecondes);
      }
      catch (Exception e) {
      }
}
```

Elle provoque une pause du nombre de millisecondes précisé en paramètre. Il suffit de l'invoquer dans la boucle avec une valeur d'environ 800 ou 900 ms, pour qu'il n'y ait plus d'instanciation qu'une fois toutes les 800 ou 900 ms.