Laboratoire 4 – Programmation évènementielle

ITI 1521. Introduction à l'informatique II

Semaine 22-26 Février 2021

Dû en ligne après une semaine de votre Lab

/10

Objectifs

- Réaliser une application graphique simple
- Utiliser l'héritage
- Implémenter :
  - Une interface
  - Une fonction de rappel

I. Application graphique simple

Ce laboratoire consiste à réaliser une application ayant une zone où s'affiche des points aléatoires qu'on va joindre pour réaliser une forme géométrique quelconque à l'aide d'un bouton. Un deuxième bouton servira à fermer la fenêtre.

Question 1: (1 POINT)

Interface graphique : GUI

Créez une sous-classe de la classe **JFrame** que vous nommerez **GUI**. Ce sera la fenêtre principale de l'application. Nous avons vu en classe que son gestionnaire est un objet de type **BorderLayout.**
Plus tard dans ce laboratoire, nous ajouterons tout ce qu'il faut afin de tracer des segments colorés en joignant des points sur cette surface.
Pour commencer la plus réduite possible : rien qu'un cadre de dimension (400, 400) :

- https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html

Vous savez qu'il faut ajoutez une directive au haut de l'unité de compilation afin de rendre le nom **JFrame** disponible. Sinon, il faudra utiliser le nom complet j**avax.swing.JFrame**.
```
import javax.swing.JFrame ;
```

•Déclarez une constante (variable de classe «public», «static», et «final») nommée **DRAW_SIZE**, dont la valeur est 400.
•Ajoutez un constructeur à la classe **GUI**. Dans une application graphique, le constructeur est souvent l'endroit où l'on construit la représentation graphique de l'application. Le constructeur étant appelé au moment de la création de l'objet.
Voici ce que vous devez ajouter au constructeur.

• Donner le titre "GUI 1" à votre fenêtre. Il y a deux façons de changer le titre de la fenêtre, soit l'on appelle le constructeur de la superclasse et on lui passe le titre comme argument (une chaîne de caractères), ou encore, on utilise la méthode **setTitle**.
• L'appel de méthode suivant fera en sorte que l'on quitte l'application (appel à la méthode **System.exit**) lorsque l'usager ferme la fenêtre :
setDefaultCloseOperation (EXIT_ON_CLOSE) ;
https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html#setDefaultCloseOperation-int-

• Ajoutez une méthode principale (**main**) à la classe **GUI**. Cette méthode va tout simplement créer un objet de la classe **GUI**. Compilez votre application et assurez-vous que votre application ressemble à l'application ci-bas.

Question 2: (1 POINT)

Ajout de quelques composants au cadre : un panneau (**JPanel**) supérieur, ici bleu par exemple, portant deux boutons (**JButton**) et un panneau central :

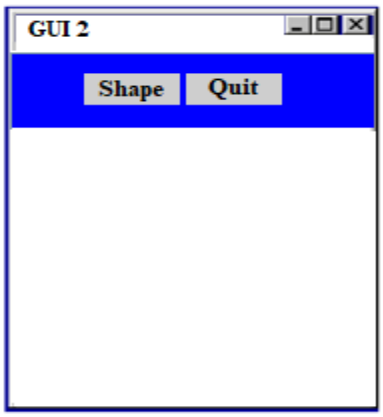
Nous ajouterons les boutons de contrôle en haut de la fenêtre. Pour ce faire, nous devons les ajouter à un panneau d'affichage (objet **JPanel**). Vous devez donc créer un objet **JPanel**. Assurez-vous que l'arrière-plan de cette composante graphique est blanc.
• Ajouter les deux directives **import** suivantes :
import javax.swing.;*
import java.awt.;*

• Créez 2 objets de la classe **JButton**. Utilisez les étiquettes «Shape», et «Quit».
Ajoutez ces boutons à l'objet de type **JPanel**.

• https://docs.oracle.com/javase/8/docs/api/javax/swing/JPanel.html

Voici ce que vous devez ajouter au constructeur.
• Faites appel à la méthode **setBackground**, afin de changer la couleur de l'arrière-plan de cette composante. Utilisez la couleur **java.awt.Color.WHITE**, ou simplement

Color.WHITE, si vous ajoutez une directive **import java.awt.Color** au début du fichier, avec l'autre directive **import**.
Color.BLUE pour le un panneau supérieur.
Ajoutez une méthode principale (**main**) à la classe **GUI**. Cette méthode va tout simplement créer un objet de la classe **GUI**. Compilez votre application et assurez-vous que votre application ressemble à l'application ci-bas.



Question 3: (1POINT)

Gestionnaire d'évènements :

Vous devez maintenant modifier l'application afin de gérer les évènements. Pour ce faire, vous devrez modifier le constructeur de la classe **GUI** afin d'ajouter (enregistrer) un gestionnaire d'évènement pour chaque bouton.
Modifiez la classe **GUI** afin qu'elle serve de gestionnaire d'évènements.
• https://docs.oracle.com/javase/8/docs/api/java/awt/event/ActionListener.html

• Ajoutez une directive **import** afin de rendre le nom **java.awt.event.ActionListener** disponible à la classe **GUI :**
import java.awt.event.;*
• Modifiez la déclaration de la classe **GUI** afin de réaliser l'interface **ActionListener**.
• Il faut donc implémenter la méthode **actionPerformed(ActionEvent e)**.
• Pour l'instant, notre implémentation de la méthode **actionPerformed** n'affichera qu'un message, par exemple «actionPerformed was called».
• Modifiez le constructeur de la classe **GUI**, chaque bouton possède une méthode **addActionListener.**

• L'objet **GUI** est l'auditeur d'événements *action* (**ActionListener**) des deux boutons ce qui oblige à identifier la source d'un tel événement lorsqu'il se produit :

Vous pouvez maintenant compiler et tester l'application. Chaque fois que l'un des deux boutons est cliqué, le message suivant sera affiché sur la console :
actionPerformed was called

Question 4: (1 POINT)

Il y a deux boutons. Comment la méthode **actionPerformed** pourra-t-elle déterminer le bouton qui a été cliqué par l'utilisateur ?
Le paramètre de la méthode **actionPerformed** est une référence vers un objet de la classe **ActionEvent** ayant été généré au moment où l'utilisateur a cliqué le bouton.
• Modifier la méthode **actionPerformed**. Spécifiquement, concaténez le résultat d'un appel à la méthode **e.getActionCommand()** au message de la méthode **println**. Ici, **e** est le nom du paramètre de la méthode **actionPerformed**. Voici la méthode a**ctionPerformed**:

```
public void actionPerformed ( ActionEvent e ) {
System.out.println ( " actionPerformed was called : " + e.getActionCommand ( ) ) ;
}
```

Vous pouvez maintenant compiler et tester l'application. Chaque fois que l'un des deux boutons est cliqué, l'un des messages suivants sera affiché selon le cas sur la console :
actionPerformed was called : Shape
actionPerformed was called : Quit

Nous utiliserons donc **getActionCommand** afin de distinguer les différentes situations.

```
public void actionPerformed (ActionEvent e ) {
String command ;
command = e.getActionCommand ( ) ;
if (command.equals ( " Shape " ) ) {
.....;
}
else if (command.equals ( " Quit " ) ) {
.....;
}
}
```

Question 5: (2.5 POINTS)

Il ne reste plus qu'à dessiner des points sur l'écran et les joindre.
Nous allons créer des points aléatoirement sur le panneau central avec le bouton *Shape* et les joindre.
La bonne manière de dessiner consiste à mémoriser tout ce qui est à tracer et à effectuer *tout* le dessin dans la méthode **`paint`** du composant sur lequel on dessine (cela oblige à écrire une classe spécifique, généralement sous-classe de **`JPanel`**) :

```
private class DrawPanel extends JPanel;
```

Chaque fois qu'**AWT** doit afficher notre objet **GUI**, il fait un appel à la méthode **paint(Graphics g)**. Modifiez la classe **GUI comme suit :**
• Ajouter une variable *tabPoints* qui représentera un tableau de 10 points (`Point`) de type **java.awt.Point**. Vous devez ajouter la directive **import** suivante :

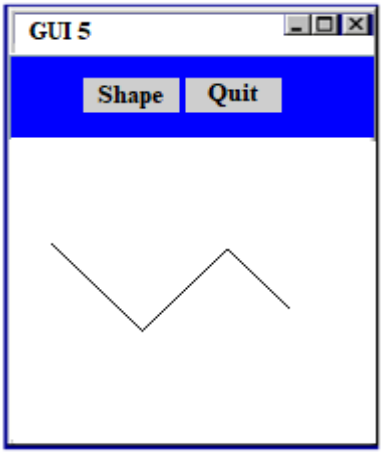
import java.awt.Point;
Noter que les paramètres du constructeur à deux arguments de `Point` sont de type int.

• Ajouter une méthode *addPoint*:
private void addPoint(Point p) ;

qui doit ajouter un point **aléatoire** à votre dessin chaque fois que vous cliquez sur le bouton *Shape* (il suffit d'ajouter votre point p dans votre tableau *tabPoints)*.
Les coordonnées d'un point sont créées aléatoirement comme :
*((int)(Math.random() * DRAW_SIZE),(int)(Math.random() * DRAW_SIZE)));*

• Y ajouter la classe *DrawPanel,* qui contient une méthode **paint(Graphics g)** que vous devez redéfinir.
• L'implémentation de la méthode **paint** fera deux choses :
– D'abord, elle fera appel à la méthode **paint** de la superclasse. Pour ce faire, ajoutez l'appel **super.paint(g)** comme premier énoncé de la méthode **paint**. Ainsi, tout le travail de la méthode **paint** héritée de la superclasse **JPanel** sera fait d'abord, puis il y aura le travail spécifique à notre sous-classe. Elle doit joindre deux points consécutifs pour former un segment. Vous faites appel à la méthode *drawLine*() :
g.drawLine(x1, y1,x2,y2) ;
où (x1, y1) sont les coordonnées du premier point et (x2, y2) sont celles du point suivant.
Noter que les paramètres de *drawLine()* sont de type *int*.
• Modifiez la méthode **actionPerformed**. À la suite de l'appel à la méthode ***addPoint()***, ajouter un appel à la méthode **repaint()** sans arguments.
Consultez la documentation de l'objet **Graphics** pour obtenir la liste des méthodes disponibles.
– https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html

Compilez et testez l'application.
Voici le rendu visuel recherché :



II. Introduction d'un menu pour choisir la couleur :

On souhaite maintenant ajouter un menu déroulant afin de changer la couleur du dessin.

Question 6: (2.5 POINTS)

Modifiez la classe **GUI comme suit :**

• Ajouter une variable `changeColorDraw`, dans laquelle vous allez stocker la nouvelle couleur de votre dessin:

```
private Color changeColorDraw = Color.BLACK;
```

• Ajouter à la classe **GUI,** la méthode createMenu() sans argument suivante qui va créer un menu de couleur et qui doit retourner un type ***JMenuBar*** **:**

```
JMenuBar createMenu() {
        JMenuBar bar = new JMenuBar();

        JMenu menu = new JMenu("ColorMenu");
        bar.add(menu);

        JMenuItem item = new JMenuItem("Black");
        item.addActionListener(this);
        menu.add(item);

        item = new JMenuItem("Red");
        item.addActionListener(this);
        menu.add(item);

        item = new JMenuItem("Green");
        item.addActionListener(this);
         menu.add(item);

        item = new JMenuItem("Blue");
        item.addActionListener(this);
        menu.add(item);
        return bar;

    }
```

Consultez la documentation de l'objet ***JMenu*** (et ***JMenuBar*** *…)* pour obtenir la liste des méthodes disponibles.
https://www.javatpoint.com/java-jmenuitem-and-jmenu

• Faites un appel à la méthode suivante :

```
setJMenuBar(createMenu());
```

• dans votre classe *DrawPanel*, vous redéfinissez la méthode **paint(Graphics g)**.

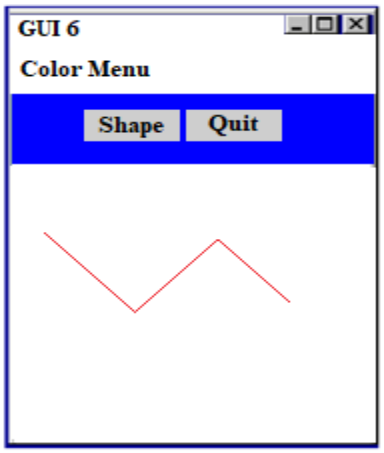
Elle doit en plus cette fois-ci changer la couleur de votre dessin chaque fois que vous cliquez sur une couleur de votre menu:

```
g.setColor(changeColorDraw);
```

• Modifiez la méthode ***actionPerformed***. Vous y ajoutez tous les changements nécessaires :

- Changement de couleur du dessin pour chaque appui sur une touche de menu.
- Ajouter un appel à la méthode ***repaint*()** sans arguments à chaque fois.

Voici le rendu visuel recherché :



Question 7: (1 POINT)

Dans cette dernière question, on va activer le 2ème bouton «Quit». Chaque appui sur ce bouton doit fermer votre fenêtre. Il suffit d'ajouter un appel à la méthode exit() dans votre méthode **actionPerformed**:

```
System.exit(0);
```

Créer et soumettre un fichier zip comme d'habitude (Q1, …Q7)