Projet Optim'EISTI

TP IHM

1) Première Fenêtre

Nous allons commencer par créer une fenêtre vide en mode awt :

```
public class Exercice

{
	public static void main(String[] args)
	{
		Frame f = new Frame("Ma super IHM !");
		f.setSize(300, 300);
		f.add(new Button("Cliquez"));
		f.setVisible(true);
	}
}
```

Que remarquez-vous? Que pensez vous du design?

Nous allons ensuite faire la même en mode Swing :

```
JFrame jf = new JFrame("Ma super IHM !");
jf.setSize(300, 300);
jf.add(new JButton("Cliquez"));
jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jf.setVisible(true);
```

Que remarquez-vous? Quelles sont les différences majeures entre les 2? Quelle est l'utilité du setVisible?

2) Composants et Layout

La plupart des composants que vous connaissez dans vos applications préférées, ont leur équivalent en Java. Nous ne vous les enseignerons pas tous, mais vous pouvez les retrouver ici :http://download.oracle.com/javase/6/docs/api/, au package **javax.swing.**

Voir l'exemple : *Exercice2_1.java*

Vous remarquerez que dans cet exemple, je positionne tous mes composants à la main car j'ai un **Layout null.**

Essayez d'agrandir la fenêtre pour voir ce qu'il se passe.

Je peux utiliser des Layout qui vont me placer automatiquement mes composants suivant un pattern donné :

BorderLayout, GridLayout, ...

Voir les exemples : *Exercice2_2.java* et *Exercice2_3.java*

Que se passe-t-il cette fois ci quand j'agrandi la fenêtre?

3) Ecouteurs d'évènements

Pour l'instant, nous n'avons travaillé que sur l'aspect visuel de notre IHM. Cependant, lorsqu'un utilisateur veut utiliser votre application, celle-ci doit pouvoir réagir au actions de cet utilisateur (clique sur un bouton, déplace sa souris, ...).

Toutes les actions possibles d'un utilisateur sont gérées en Java sous le nom d'**évènements**. Ils ne sont pas gérés par défaut comme nous avons pu le voir avec le setDefaultCloseOperation. C'est donc à vous de créer des **écouteurs d'évènements**, un pour chaque composant dont vous voulez qu'il réagisse a un comportement particulier de l'utilisateur.

Nous allons ajouter un écouteur d'évènement sur nos boutons :

```
JButton bouton = new JButton("X");

bouton.addActionListener(new MonEcouteur());
```

Nous devons donc créer une classe qui implémente l'interface ActionListener :

```
class MonEcouteur implements ActionListener

{
	public void actionPerformed(ActionEvent arg0)
	{
		System.out.println("TOTO");
	}

}
```

La méthode actionPerformed sera alors appelée à chaque fois qu'un utilisateur clique sur le bouton.

Testez ce code et observez le résultat.

On peut avoir besoin de modifier un autre composant de l'IHM lors d'un évènement sur un composant. Dans ce cas on passe les composants modifiables en paramètre de l'écouteur, afin qu'il puisse éventuellement les modifier :

```
JTextField tx = new JTextField("aijaisjaisj");

JButton bouton = new JButton("X"+i+","+j);

bouton.addActionListener(new MonEcouteur(tx));
```

Et la classe écouteur récupère la référence vers se composant pour éventuellement le modifier par la suite :

```
class MonEcouteur implements ActionListener
{
	JTextField tx;

	MonEcouteur(JTextField tx)
	{
		this.tx = tx;
	}

	public void actionPerformed(ActionEvent e)
	{
		tx.setText(((JButton)e.getSource()).getText());
		tx.validate();
	}
}
```