IUT- Département d'Informatique
G. Rey
2015

Introduction à l'IHM – M2105
TD n° 1 (Séance 1 à 3)

1 Objectifs

Ce TD/TP illustre la partie du cours sur la programmation des interfaces homme-machine. Les concepts principaux abordés seront:

- Principe de séparation du code et de la description de l'interface
- Principe de la gestion des interfaces : notion d'évènement,
- Notion de modèle et de vue
- Réalisation d'interface en Java

Un compte rendu donnant la réponse à chacune des questions posées devra être rendu à votre responsable de TD/TP par email à la fin de chacune des séances. L'objet (sujet) de l'email devra être le suivant :

[S2T][IHM][Gx][TDx]Nom-Prénom / Nom-Prénom

Avec Gx le numéro de votre groupe (exemple G1 pour le groupe 1) et TDx le numéro du TD/TP inscrit sur le sujet du TD/TP (exemple TD1 pour le premier TD).

Vous ne pouvez pas faire 2 séances avec le même binôme. De même, vous ne pouvez pas faire plus de la moitié des séances seul.

Veuillez, sauf indication contraire de votre responsable de TD, faire l'ensemble des exercices marqués « Obligatoire » dans l'ordre de la feuille de TD/TP. Les exercices « Conseillés » sont à faire si vous avez terminé les précédents ou lors de vos révisions. Les exercices « optionnels » sont là pour les plus passionnés d'entre vous.

2 Prise en main de l'environnement

Pour la majeure partie des TD/TP de ce module, nous utiliserons le langage Java pour illustrer les concepts vus en cours. Pour des raisons de facilité nous utiliserons l'IDE Eclipse déjà installé dans nos salles de TP. Si vous avez besoin de faire le TP chez vous, vous aurez besoin :

- Du JDK Java (Java Standard Edition) : http://www.oracle.com/technetwork/java/javase/downloads/index.html
- IDE Eclipse (Eclipse IDE for Java Developers) : https://www.eclipse.org/downloads/

Il est conseillé de créer un répertoire M215 sur compte. Celui-ci contiendra un répertoire TDx (x le numéro du TD) par TD.

Pensez à sauvegarder régulièrement votre travail et à commenter votre code. N'hésitez pas à sauvegarder les différentes versions, évolutions d'un même exercice de manière à pouvoir facilement réviser ou le réutiliser par la suite.

3 Exercice 1 : Java JFrame (obligatoire)

Dans ce premier TD/TP, nous allons découvrir les principaux composants Java, explorer quelques gestionnaires d'agencement et conteneurs. Enfin, nous illustrerons simplement la notion d'évènements.

Le TD n°1 n'a pas pour vocation à construire une application n'y même sa GUI (Graphical User Interface : interface graphique). Même simplement de comprendre le fonctionnement de la conception d'une GUI. Il a pour but de vous faire manipuler les concepts techniques principaux sur des exemples-jouer. Et cela dans le but de vous préparer au TD2 qui consistera en la réalisation une application complexe.

Introduction à l'IHM – M2105
TD n° 1 (Séance 1 à 3)

3.1 Ma première fenêtre

Nous allons commencer par créer une fenêtre classique (JFrame).

- Créez un projet java TD1-exo1
- Créez une classe Exo1 (du package TD1).
- Ajoutez le code suivant dans votre classe

```
public static void main(String[] args) {
	SwingUtilities.invokeLater(new Runnable(){
		@Override
		public void run(){
			//On crée une nouvelle instance de notre JFrame
			JFrame window = new JFrame();
			window.setSize(300, 200);//On lui donne une taille pour qu'on puisse la voir
			window.setVisible(true);//On la rend visible
		}
	});
}
```

On remarquera qu'on utilise `SwingUtilities.invokeLater` comme vu en cours. Je vous rappelle que cela a pour effet de faire exécuter le code de la méthode run() par l'Event Dispatch Thread qui est le seul habilité à faire des modifications graphiques.

- Pensez à ajouter les bons imports :

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
```

- Compilez et exécutez votre code Java pour tester votre application. Pour cela vous pouvez soit utiliser votre IDE, soit utiliser les commandes suivantes depuis la racine de votre projet (dans src si vous avez créé un tel répertoire).

```
javac TD1\Exo1.java
java TD1.Exo1
```

3.2 Explorons un peu la JFrame

Pour simplifier la suite du TD, on va créer notre propre classe de type.

- Créez la classe suivante

```
package TD1;
import javax.swing.JFrame;
public class MaFenetre extends JFrame {
	/**
	 * Nécessaire pour la sérialisation
	 */
	private static final long serialVersionUID = 1345376003869391813L;
	public MaFenetre(){
		super(); //On appelle le constructeur de la JFrame
		InitializeComponent();
	}
	private void InitializeComponent() {
		this.setTitle("Ma fenêtre Java !!");
		this.setSize(600,400);
		this.setLocationRelativeTo(null);
		this.setResizable(false);
	}
}
```

Introduction à l'IHM – M2105
TD n° 1 (Séance 1 à 3)

- D'après vous (et sans chercher sur le web) que font les 4 méthodes de la méthode « InitializeComponent »
- Modifiez la classe Exo1 pour utiliser votre classe « MaFenetre » en lieu et place de la JFrame. Pensez à supprimer la ligne « `window.setSize(300, 200);` ». Vous pouvez sauvegarder votre précédent exercice avant de faire les modifications.
- Quelle autre ligne/instruction n'est plus nécessaire dans votre classe Exo1 ?

Poursuivons notre étude de la JFrame.

- Créez, depuis votre programme principal, une deuxième fenêtre identique à la première.
- Vérifiez que vous avez bien 2 fenêtres.
- Que se passe-t-il quand vous fermez une fenêtre ?
- Ajoutez l'instruction suivante dans la méthode **InitializeComponent**

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

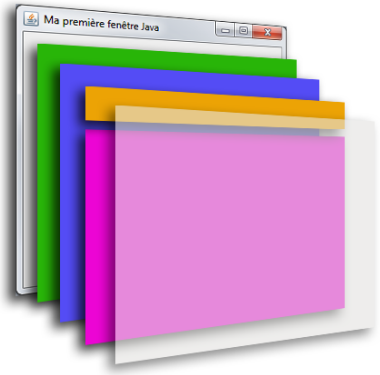
- Toujours sans regarder sur le web, indiquez ce que fait cette nouvelle instruction.
- Supprimez la 2eme fenêtre de votre code avant de passer à la suite.

Il existe de nombreuses autres options que nous n'allons pas détailler par manque de temps. Pour plus de détails sur la JFrame, vous pouvez consulter la JavaDoc officielle.

A savoir :
La fenêtre principale (JFrame) est un composant graphique comme les autres (ou presque). Elle peut être configurée en fonction des besoins.

3.3 Comment ajouter des composants à notre fenêtre

Une JFrame est découpée en plusieurs parties superposées, comme le montre la figure suivante. Nous avons, dans l'ordre (extrait d'openclassroom) :



- la fenêtre ;
- le **RootPane** (en vert), le conteneur principal qui contient les autres composants ;
- le **LayeredPane** (en bleu), qui forme juste un panneau composé du conteneur global et de la barre de menu (MenuBar) ;
- la **MenuBar** (en orange), la barre de menu, quand il y en a une ;
- le **ContentPane** (en rose) : c'est dans celui-ci que nous placerons nos composants ;
- le **GlassPane** (en transparence), couche utilisée pour intercepter les actions de l'utilisateur avant qu'elles ne parviennent aux composants.

Pour commencer, nous allons nous restreindre à l'utilisation du **ContentPane**.

- Créez un bouton et ajoutez-le à votre fenêtre.

```
JButton bouton1 = new JButton("Mon Bouton");
this.getContentPane().add(bouton1);
```

On notera que nous n'ajoutons pas les éléments directement dans la fenêtre mais dans un conteneur (ici le ContentPane) qui a en charge l'organisation des composants.

En réalité, pour une JFrame, l'ajout direct d'un composant dans celle-ci (myJFrame.add(monComposant)) ajoute en réalité le composant dans le contentPane. Cela est donc équivalent à ce que nous avons écrit précédemment (myJFrame.getContentPane().add(monComposant)).

Introduction à l'IHM – M2105
TD n° 1 (Séance 1 à 3)

- Commentez le code précédent et ajoutez un bouton « anonyme » de la manière suivante :

```
this.getContentPane().add(new JButton("CENTER"), BorderLayout.CENTER);
```

- Quelle est la différence entre un bouton et un bouton anonyme ?
- Ajoutez 4 autres boutons « anonymes », pour tester les 4 valeurs de `BorderLayout` qui correspondent aux points cardinaux. On finale on devrait avoir un bouton dans chaque zone.

En réalité, le **ContentPane** contient, par défaut, un gestionnaire d'agencement (layout manager) de type `BorderLayout`.

4 Exercice 2 : Layout manager (obligatoire)

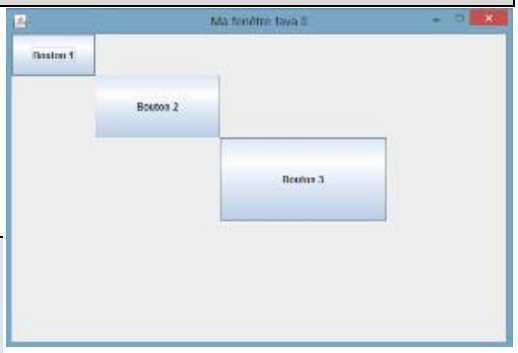
Les gestionnaires d'agencements (ou de placement) permettent de disposer les composants dans un conteneur en fonction (1) de la stratégie du gestionnaire d'agencement, (2) des paramètres sur le gestionnaire d'agencement et (3) des caractéristiques des composants. De nombreux gestionnaires d'agencements sont disponibles en java. Dans la suite de ce TD, nous allons seulement en voir quelques-uns.

On peut changer le gestionnaire d'agencement de notre fenêtre (en réalité de notre contentPane) via l'une des 2 méthodes suivantes :

```
this.getContentPane().setLayout(monLayoutManager); // A utiliser de préférence
this.setLayout(monLayoutManager);
```

4.1 Pas de gestionnaire

Ce cas correspond à celui où le conteneur n'a pas de gestionnaire d'agencements. Dans notre cas il faut supprimer celui par défaut (myJFrame.getContentPane().setLayout(null)). Les composants ont un emplacement et une taille fixe.



- Ajoutez trois boutons à votre fenêtre et supprimez le gestionnaire d'agencements. Vous pouvez utiliser les fonctions suivantes pour donner une taille à votre bouton, le positionner ou changer son texte (cf. l'image ci-dessous).

```
monBouton.setSize(new Dimension(100,50)); //équivalent à monBouton.setSize(100,50);
monBouton.setLocation(new Point(100, 50)); // équivalent à monBouton.setLocation(100, 50);
monBouton.setText("Mon Bouton");
```

- Modifiez la propriété de votre fenêtre pour pouvoir modifier dynamiquement sa taille.
- Est-ce que le fait de retailler ma fenêtre impact mon positionnement ?
- Vous vérifierez cela de manière automatique pour chaque gestionnaire d'agencement étudié dans l'exercice 2.

A savoir :
Sans gestionnaire d'agencement, il faut positionner tous les composants graphique un à un. De plus, cette disposition n'évolue pas en fonction de la taille de la fenêtre (ou du conteneur)

4.2 BorderLayout

Il s'agit du gestionnaire par défaut du **ContentPane**. Le conteneur est divisé en 5 zones : nord, sud, est, ouest, et centre. Il ne peut donc pas contenir plus de 5 composants. Dans les zones nord, sud, est et ouest, les composants sont placés en fonction de leur taille préférée, le composant du centre occupe alors toute la place restante.

- Modifiez votre projet pour que le ContentPane utilise à nouveau son gestionnaire d'agencement par défaut.
- Modifiez l'ajout des composants dans le ContentPane pour ajouter le bouton 1 au centre, le bouton 2 au nord et le bouton 3 au sud.
- Testez. Que constatez-vous ? Est-ce que les boutons ont les tailles souhaitées ?

Introduction à l'IHM – M2105
TD n° 1 (Séance 1 à 3)

- Utilisez la méthode setPreferredSize en lieu et place de la méthode setSize pour vos 3 boutons.
- Testez. Que constatez-vous ? Est-ce que les boutons ont les tailles souhaitées ?

Les composants sont séparés, horizontalement et verticalement, par des espaces qui peuvent être modifiés par les méthodes monBorderLayout.setHgap(int g) et monBorderLayout.setVgap(int g). Ceci est vrai pour une majorité de gestionnaire d'agencement.

- Testez cela via la méthode suivante :

```
((BorderLayout)this.getContentPane().getLayout()).setVgap(10);
```

A savoir :
Le BorderLayout est un des gestionnaires d'agencement proposé en Java. Il permet de découper l'espace en 5 espaces (Nord, Sud, Est, Ouest et Centre. FlowLayout

4.3 FlowLayout

Les composants sont placés les uns à la suite des autres horizontalement, en passant à la ligne suivante quand il n'y a plus de place sur la ligne. Chaque composant a sa taille préférée. Le gestionnaire de placement peut :

- **FlowLayout.CENTER** : centrer les composants (valeur par défaut)
- **FlowLayout.LEFT** : aligner les composants à gauche
- **FlowLayout.RIGHT** : aligner les composants à droite
- **FlowLayout.LEADING** : aligner les composants à partir du bord du conteneur. Ce bord est défini en fonction de l'orientation du conteneur (à gauche si le conteneur est orienté de gauche à droite (**ComponentOrientation.LEFT_TO_RIGHT**) et à droite si le conteneur est orienté de droite à gauche (**ComponentOrientation.RIGHT_TO_LEFT**).
- **FlowLayout.TRAILING** : aligner les composants à partir du bord du conteneur. Ce bord est défini en fonction de l'inverse de l'orientation du conteneur (à droite si le conteneur est orienté de gauche à droite (**ComponentOrientation.LEFT_TO_RIGHT**) et à gauche si le conteneur est orienté de droite à gauche (**ComponentOrientation.RIGHT_TO_LEFT**).

- Ajoutez un **FlowLayout** à votre **ContentPane** (attention le **ContentPane** ne peut avoir qu'un seul gestionnaire d'agencement, pensez à corriger votre code)

```
//le paramètre du FlowLayout peut être omis. FlowLayout.CENTER est la valeur par défaut
this.getContentPane().setLayout(new FlowLayout(FlowLayout.CENTER));
```

- Attention, il faut corriger l'ajout des boutons dans le **ContentPane**
- Testez les autres paramètres du **FlowLayout**

```
FlowLayout.RIGHT, FlowLayout.LEFT, FlowLayout.LEADING, FlowLayout.TRAILING
```

- Vous pouvez changer l'orientation du **ContentPane** de la manière suivante

```
this.getContentPane().setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
```

A savoir :
Le FlowLayout est un des gestionnaires d'agencement proposé en Java. Il place les composants graphiques les uns à la suite des autres horizontalement.

Avant de poursuivre le TD/TP, on va faire un petit aparté pour voir le fonctionnement de la méthode pack().

- Ajoutez des boutons de la taille du bouton2 et de manière à avoir 2 ou 3 lignes de composants dans votre fenêtre.
- Ajoutez this.pack() ; à la fin de votre méthode InitializeComponent().
- Testez. Que se passe-t-il ?
- Commenter la méthode qui donne une taille à votre fenêtre. Que se passe-t-il ?

A savoir :
Pack adapte la fenêtre à la taille préférée et à la mise en page de ses sous-composants.

4.4 GridLayout

Les composants sont placés dans un tableau à deux dimensions. S'il y a plus de composants que le nombre de cases (colonnes*lignes), le système ajoute autant de colonnes qu'il faut pour que le nombre de cases soit supérieur ou égal au nombre de composants du conteneur. On notera que l'agencement des composants dépendra de la propriété componentOrientation du conteneur.

- Modifiez la **LayoutManager** de votre application pour en faire un GridLayout.

```
this.getContentPane().setLayout(new GridLayout()); //constructeur par défaut
this.getContentPane().setLayout(new GridLayout(nbLignes,nbColonnes));
```

- Que se passe-t-il avec le constructeur par défaut ? et avec les paramètres (1,2) ? et avec (2,1) ?
- Imaginez ce qu'il va se passer avec les paramètres (4,5).
- Vérifiez et testez ce qui se passe en changeant la propriété componentOrientation du conteneur.

4.5 BoxLayout

Permet de placer des composants soit horizontalement, soit verticalement. Ce gestionnaire de placement respecte la taille maximum et l'alignement de chaque composant.

Les composants sont insérés dans le conteneur en fonction du second paramètre du BoxLayout :

- **X_AXIS** : les composants sont placés horizontalement de gauche à droite, en respectant leur alignement par rapport à une ligne horizontale passant au milieu du conteneur.
- **Y_AXIS** : les composants sont placés verticalement de haut en bas, en respectant leur alignement par rapport à une ligne verticale passant au milieu du conteneur.
- **LINE_AXIS** : comme X_AXIS, mais en respectant l'orientation du conteneur.
- **PAGE_AXIS** : comme Y_AXIS, mais en respectant l'orientation du conteneur.

- Ajoutez des tailles maximales à vos différents boutons. Chaque bouton devant avoir une taille maximale différente de celle des autres boutons et supérieure à sa taille et à sa taille préférée.

```
bouton1.setMaximumSize(new Dimension(200,100));
```

- Modifiez le gestionnaire d'agencement de votre fenêtre :

```
this.getContentPane().setLayout(new BoxLayout(this.getContentPane(), BoxLayout.X_AXIS));
```

- Testez ce nouveau gestionnaire d'agencement.
- Testez également les autres valeurs du 2eme paramètre.

Il existe, en Java, de nombreux autres gestionnaires d'agencement, plus ou moins complexe, mais nous n'avons malheureusement pas le temps de tous les explorer. Vous pouvez les consultez sur la Java Doc officielle. Vous pouvez également écrire votre propre gestionnaire d'agencement.

5 Exercice 3 : Les composants de base (obligatoire)

Nous allons maintenant rapidement parcourir certain des quelques-uns des principaux composants de l'API SWING. Nous ne reviendrons pas sur le bouton que nous avons déjà un peu utilisé.

5.1 JCheckBox

La JCheckBox est une boîte à cocher. La méthode **maJCheckBox.isSelected()** permet de savoir si la boîte à cocher est sélectionnée ou non. La méthode **maJCheckBox.setSelected(boolean selection)** permet de sélectionner ou désélectionner la boîte à cocher.

Introduction à l'IHM – M2105
TD n° 1 (Séance 1 à 3)

- Avant d'aller plus loin, créé un nouveau projet avec une fenetre (JFrame) et le gestionnaire d'agencement de votre choix. Je vous conseille d'utiliser un FlowLayout qui reste assez simple d'utilisation pour ce qu'on veut faire.
- Ajoutez un JCheckBox dans votre fenêtre.

```
JCheckBox maJCheckBox = new JCheckBox("message", false);
```

- Testez.

5.2 JRadioButton

Un bouton radio est un bouton qui peut être sélectionné ou non. Un ButtonGroup permet de grouper des boutons radio, de façon à ce qu'il n'y ait qu'un seul bouton sélectionné dans le groupe. Les méthodes **isSelected** et **setSelected** sont également disponibles pour les boutons radio.

- Créez 3 boutons radio. Seul le premier sera sélectionné.

```
JRadioButton monRadioB1 = new JRadioButton("un", true);//Constructeur avec texte et état
JRadioButton monRadioB2 = new JRadioButton("deux");//Constructeur avec texte (état=désélectionné)
```

- Testez.
- Ajoutez un groupe et ajoutez les boutons radio dans le groupe.

```
ButtonGroup monGroupeB = new ButtonGroup();
```

- Testez.

On notera que le groupe n'est pas un élément graphique et qu'il ne sera pas ajouté dans la JFrame.

A savoir :
Une case à cocher (JCheckBox) sert à définir une option « unique », activable ou désactivable.
Un bouton radio (JRadioButton) ne s'utilise jamais seul. Il s'utilise en groupe (plusieurs boutons radio dans un ButtonGroup) pour présenter un choix.
Dans les deux éléments précédents, il est très important de bien choisir la valeur par défaut (case cochée ou non, lequel des boutons radio est sélectionné).

5.3 JLabel

Un JLabel sert à afficher du texte.

- Ajoutez un JLabel à votre fenêtre

```
JLabel monLabel = new JLabel("Mon étiquette");
```

- Testez.

5.4 JTextField

Un JTextField est une zone d'édition de texte comportant une seule ligne. Les méthodes **monJTextField.getText()** et **monJTextField.setText(String s)** permettent de récupérer le texte tapé par l'utilisateur, ou d'affecter un texte au JTextField.

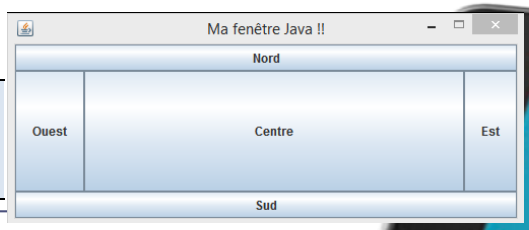
- Ajoutez un JTextField à votre fenêtre

```
JTextField monJTextField = new JTextField("Mon texte");
monJTextField.setColumns(20); //permet de choisir la largeur de la zone d'édition
```

- Testez.

5.5 JPanel

Le JPanel est un conteneur. On va voir comment il fonction et comment on peut s'en servir.



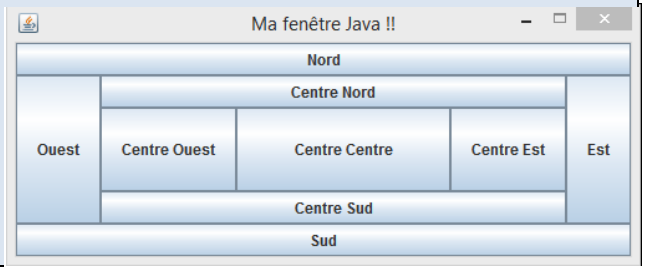
- Créez un nouveau projet et ajoutez une fenêtre semblable à celle ci-contre.
- Ajoutez un 6éme bouton au centre. Que se passe-t-il ?

Voyons maintenant, comment avec le JPanel on arranger cela.

- Retirez les 2 boutons qui sont au centre de la fenêtre et créez un JPanel

```
JPanel jp = new JPanel();
```

- Ajouter les 2 boutons dans le JPanel (celui-ci a par défaut un FlowLayout)
- Ajoutez le JPanel au centre de la fenêtre.
- Observez le résultat
- Ajoutez les boutons nécessaires (9 au total) et modifiez votre code pour obtenir un résultat similaire à celui-ci-contre.



Comme on a pu le voir avec les 2 exemples ci-dessus, on peut utiliser les JPanel pour passer outre les limitations de certains gestionnaires d'agencement.

Il existe une cinquantaine de composants SWING en Java. Nous aurons probablement l'occasion d'en rencontrer et dans tester d'autres durant la suite de ce TD. Au besoin vous pouvez consultez la Java Doc officielle pour plus d'information.

A savoir :
Chaque composant à des propriétés qui lui sont propres. Il est important d'utiliser le composant en adéquation avec l'objectif recherché. Par exemple, un JTextField n'est pas approprié pour la saisie d'un mot de passe.

6 Exercice 4 : Evènements (obligatoire)

Nous allons maintenant ajouter des évènements à nos différents composants. Pour cela nous allons reprendre les différents éléments étudiés en cours au début de l'année.

6.1 Écouteurs et les classes anonymes

Je vous rappelle qu'en Java l'abonnement à un évènement consiste à l'ajout d'un écouteur (**Listener**). Dans le cas d'un bouton (ou d'une case à cocher ou d'un bouton radio), nous pouvons écouter le fait qu'il y a eu une action sur le bouton.

- Revenez sur le projet précédent à celui du JPanel. Celui où vous aviez des cases à cocher, …
- Ajoutez un écouteur d'action sur le 3eme bouton radio, qui changera le texte du label par le texte du bouton radio

```
monRadioB3.addActionListener(new ActionListener() {
	@Override
	public void actionPerformed(ActionEvent e) {
		monLabel.setText("trois");
	}
});
```

- Faites de même pour les 2 autres boutons.

A savoir :
Il est très simple d'associé (abonnement) un écouteur d'évènements (observateur) à un composant graphique (observable). En Java, un écouteur d'évènements est un Listener.

6.2 Factorisation du code

On voit que cette méthode, même si elle est pratique pour écrire rapidement un écouteur pour un composant, ne permet pas de factoriser le code.

- Modifiez votre code de manière à utiliser le même ActionListener pour tous vos boutons radio.

```
ActionListener listener = new ActionListener() {
	public void actionPerformed(ActionEvent e) {
```

Introduction à l'IHM – M2105
TD n° 1 (Séance 1 à 3)


```
            // A vous d'écrire le code qui vient ici
        }
};
monRadioB1.addActionListener(listener);
```

- Maintenant vous allez modifiez votre code pour qu'il fonctionne pour tous les boutons radio qui écoute l'évènement « Action ». Pour rappel, l'ojbjet `e` (de type `ActionEvent`), passé en paramètre, vous permet de récupérer la source de l'évènement via la méthode `e.getSource();`
- Sachant qu'un `JRadioButton`, ou qu'une `JCheckBox` hérite tous de `AbstractButton`, modifiez pour que le label contienne le texte du dernier élément sélectionné parmi tous les `JRadioButton` et les `JCheckBox`.

A savoir :
Les évènements contiennent des références à la source (observable). Cette référence est utilisable par l'écouteur d'évènements.
En Java, l'écouteur d'évènements reçoit 1 seul paramètre, qui correspond à l'évènement lui-même.

6.3 Création de son propre écouteur

Avant de terminer cet exercice, nous allons voir comment créer son propre écouteur. Cela est relativement simple il suffit d'implémenter l'interface correspondante à votre écouteur et donc de définir l'ensemble des fonctions de cette interface.

- Modifiez votre code pour indiquer que votre classe implémente un `ActionListener`

```
public class MaFenetre extends JFrame implements ActionListener{
```

- Cette interface ne définit qu'une seule méthode (). Il ne vous reste plus qu'à ajouter cette méthode et à la compléter.

```
public void actionPerformed(ActionEvent e) {}
```

- Attention, n'oubliez pas d'abonner vos différents éléments à votre nouvel écouteur

```
monRadioB1.addActionListener(this);
```

7 Exercice 5 : Explorons quelques écouteurs évènements (obligatoire)

Nous allons voir différents écouteurs d'évènements et principalement ceux liés au clavier et à la souris.

7.1 ActionListener

Commençons par l'**ActionListener** que nous avons déjà utilisé sur nos boutons.

- Créez une nouvelle classe héritant d'une JFrame.
- En vous inspirant de l'exercice 1, configurez celle-ci pour quelle s'affiche. On choisira un `FlowLayout` comme gestionnaire d'agencement.
- Ajoutez un champ texte qui vérifiera son si son contenu est un entier (font vert) ou non (font rouge) lors d'une action (appuie sur la touche Return) sur le champ texte. Le code ci-dessous vous permet de savoir si le contenu du champ texte est un entier.

```
try {
        Integer.parseInt(monChampText.getText());
        monChampText.setBackground(Color.green);
}catch (Exception ex){
        monChampText.setBackground(Color.red);
}
```

7.2 Le KeyListener

Regardons maintenant comment s'abonner à des évènements clavier. Cela se fait en abonnant notre composant aux évènements clavier via un **KeyListener**. Cet écouteur propose 3 méthodes (**keyTyped**, **keyReleased** et **keyPressed**) qui permettent respectivement de réagir lorsqu'une touche a été pressée, tapée et relâchée (envoyé dans cet ordre).

Introduction à l'IHM – M2105
TD n° 1 (Séance 1 à 3)

La classe **KeyEvent** étend **java.util.EventObject** et dispose donc des méthodes déclarées dans cette classe (notamment **getSource**()), mais fournit également une dizaine d'autres méthodes spécifiques aux évènements relatifs au clavier. Parmi elles :

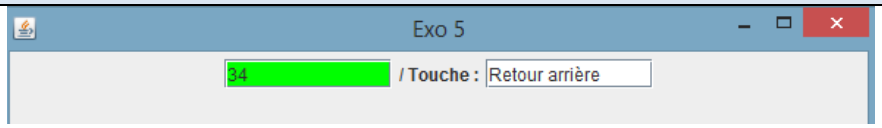
- **getKeyChar()** retourne le caractère associé à la touche appuyée,
- **getKeyCode()** récupère le code de la touche pressée,
- **isActionKey()** retourne true si la touche appuyée est une touche d'action (CAPS LOCK, Verr Num, etc), et
- **getKeyText(int keyCode)** retourne le texte associée à la touche (par ex. F1, A, etc).

Une dernière méthode, **getKeyText**, est très couramment utilisée. Attention celle-ci est statique, et ne s'utilise donc pas avec l'instance de **KeyEvent** fournie mais directement en faisant **KeyEvent.getKeyText(monKeyCode)**. [Extrait de java.developpez.com]

- Ajoutez une trace (message de sortie vers la console) pour vérifier l'ordre d'appel des 3 méthodes lors d'une saisie clavier dans le champ texte. Pour cela vous pourrez utiliser simplement la méthode suivante :

```
System.out.println("message à afficher dans la console");
```

- Modifiez votre code pour que la vérification précédente soit effectuée à chaque action sur le clavier. Vous testerez les 3 méthodes pour voir les différences.
- Modifiez votre fenêtre pour qu'elle soit similaire à l'illustration suivante. Le deuxième champ texte affichant automatiquement la dernière touche appuyée.



Dans la majorité des cas, on ne souhaite pas utiliser l'ensemble des méthodes d'un écouteur. Dans ce cas, on peut substituer un « Type »Listener par un « Type »Adapter. Il s'agit globalement de la même chose. L'avantage du second c'est qu'il ne nécessite pas de redéfinir l'ensemble des méthodes.

Prenons un exemple. On souhaite redéfinir seulement la méthode keyTyped. Avec un Listener on doit écrire :

```
monChampText.addKeyListener(new KeyListener() {
	@Override
	public void keyTyped(KeyEvent e) {
		// mon code
	}
	@Override
	public void keyReleased(KeyEvent e) {}
	@Override
	public void keyPressed(KeyEvent e) {}
});
```

Avec un Adapter, on peut écrire seulement :

```
monChampText.addKeyListener(new KeyAdapter() {
	@Override
	public void keyTyped(KeyEvent e) {
		// mon code
	}
});
```

7.3 MouseListener

Le **MouseListener** est utilisé pour tous les événements relatifs à la souris (clics et déplacements).

5 méthodes sont déclarées dans l'interface **MouseListener** : **mouseClicked(MouseEvent e)** prévient des clics (la souris a été pressée puis relâchée), **mousePressed(MouseEvent e)** pour les pressions sur la souris (donc on enfonce le bouton

Introduction à l'IHM – M2105
TD n° 1 (Séance 1 à 3)

sans le relâcher), **mouseReleased(MouseEvent e)** prévient du relâchement d'un bouton de la souris, **mouseEntered(MouseEvent e)** indique que la souris est entrée dans l'espace d'un composant, **mouseExited(MouseEvent e)** indique que la souris est sortie de l'espace d'un composant.

La classe **MouseEvent** étend **java.util.EventObject** et dispose donc des méthodes déclarées dans cette classe (notamment **getSource()**), mais fournit également 12 autres méthodes spécifiques aux événements relatifs à la souris, notamment **getButton()** retourne le bouton qui a été cliqué, **getClickCount()** retourne le nombre de clics (utile pour gérer le double clic), **getLocationOnScreen()** retourne un objet Point représentant la position de la souris à l'écran, et enfin **isPopupTrigger()** précise si le bouton cliqué est celui habituellement utilisé pour afficher la liste déroulante (bouton droit sur le bureau Windows par exemple). [Extrait de java.developpez.com]

Pour identifier le bouton de la souris qui a été utilisé, il existe plusieurs solutions :

- 3 méthodes statiques de la classe **SwingUtilities**. Il s'agit dee trois méthodes suivantes : **isLeftMouseButton(MouseEvent e)**, **isMiddleMouseButton(MouseEvent e)** et **isRightMouseButton(MouseEvent e)**).

```
if ( SwingUtilities.isLeftMouseButton(e) ) { /* le code de gestion du bouton 1 (gauche) */ }
```

- La méthode getButton() (à appeler sur l'instance d'un **MouseEvent**) dont le retour est à comparer avec les entiers **MouseEvent.BUTTON1**, **MouseEvent.BUTTON2** ou **MouseEvent.BUTTON3** pour savoir s'il s'agit (dans l'ordre) du bouton de gauche, du milieu ou de droite.

```
if (e.getButton() == MouseEvent.BUTTON2){ /* ici le code de gestion du bouton 2 (milieu) */ }
```

- La méthode getModifiers() (à appeler sur l'instance d'un **MouseEvent**) dont le retour est à filtrer avec le masque du bouton rechercher puis à comparer avec le masque de ce bouton.

```
if ( ( e.getModifiers () & InputEvent.BUTTON3_MASK ) == InputEvent.BUTTON3_MASK ) {
      // ici le code de gestion du bouton 3 (droite)
}
```

- Modifiez votre code, pour que l'arrière-plan du 2eme champ texte soit bleu quand la souris survole ce composant.
- Ajoutez une trace (message de sortie vers la console) pour vérifier l'ordre d'appel des 3 méthodes lors d'un clic souris dans la fenêtre. Vous devez utiliser les 3 solutions ci-dessus. L'ensemble des messages affichés devront avoir la forme suivante : « Click : Pressed Bouton DROIT »

8 Exercice 6 : QCM (conseillé)

A vous de mettre en œuvre de façon ergonomique les éléments vu en cours et dans ce TD dans l'exercice suivant.

- Créez un questionnaire à choix multiples dont le sujet des questions devra porter sur ce TD. Le QCM devra avoir un titre, 2 questions avec qu'une seule réponse de juste et 2 questions avec plusieurs réponses de justes, un bouton « évalué le QCM » et espace d'affichage de la note après l'évaluation.
- L'appui sur le bouton devra bloquer la modification du QCM, lancer l'évaluation et afficher le résultat (une note sur 4)
- Ajoutez une question « libre » où la réponse sera saisie par l'utilisateur et modifiez le code pour que cette nouvelle question soient prise en compte.