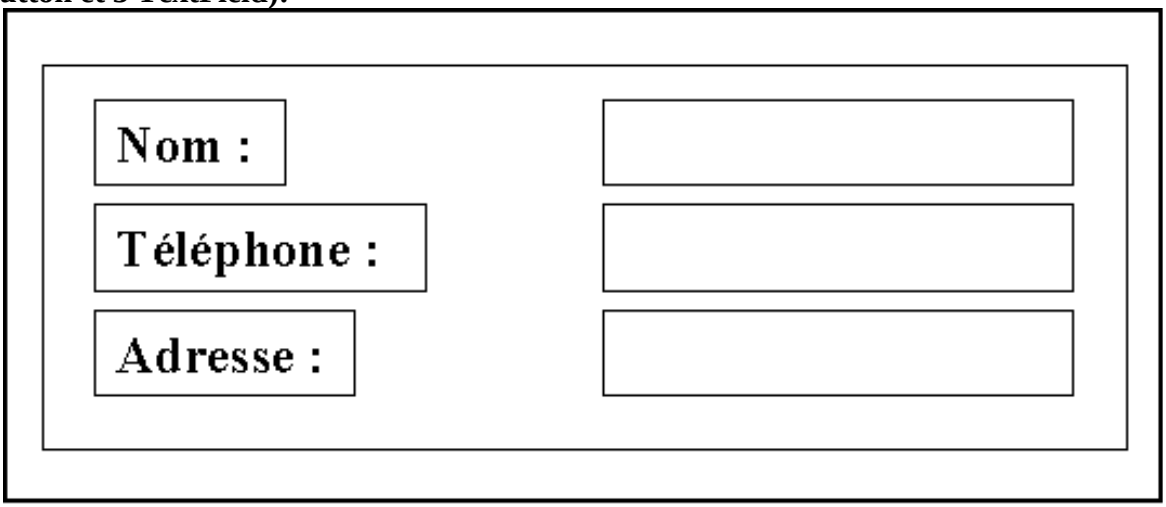
ED N° 5 : Les interfaces graphiques en Java

1°) Construire en langage Java une application indépendante qui affiche le formulaire (3 Button et 3 TextField):



une solution :

Il faut préciser les composants et leurs conteneurs. Pour les conteneurs il faut ensuite préciser la politique de positionnement parmi `GridLayout`, `FlowLayout`, `BorderLayout` (voire d'autres). Pour un exercice aussi simple on doit pouvoir utiliser chacun des trois. Voici une solution complète avec un `GridLayout`.

```
import java.awt.*;

public class FormulaireGrid extends Frame {
	Button[] bt;
	TextField[] champ;

	public static void main(String args[]) {
		new FormulaireGrid().affiche();
	}

	public void affiche() {
		bt = new Button[3];
		champ = new TextField[3];
		for (int i=0; i<3; i++) {
			bt[i] = new Button();
			champ[i] = new TextField(20);
		}
		bt[0].setLabel("Nom");
		bt[1].setLabel("Téléphone");
		bt[2].setLabel("Adresse");

		// le placement
		setLayout(new GridLayout(3, 2));
		for (int j=0; j<3; j++) {
			add(bt[j]);
			add(champ[j]);
		}
		// setSize(300, 125);
		pack();
		setVisible(true);
	}
}
```

On veut désormais prendre en compte les réactions de l'utilisateur. Lorsqu'il clique sur l'un des boutons le texte se trouvant dans le `TextField` en vis à vis s'affiche.

2°) Indiquer la manière que le langage Java propose pour traiter les événements (depuis sa version 1.1). On précisera quelle est la méthode du langage qui est nécessairement lancée et sur quel objet et on indiquera comment le langage s'y prend pour faire cette vérification à la compilation. On indiquera aussi comment l'association est effectué entre le composant graphique et le code à lancer lorsqu'il est utilisé.

Java propose depuis sa version 1.1, la programmation par délégation pour traiter les événements : c'est un objet d'une classe qui est chargé de lancer le code approprié lorsque l'utilisateur actionne un composant graphique (clic sur un bouton, action sur une touche calvier lorsqu'un `TextField` a le focus, utilisation d'une barre de défilement, etc.).

Il faut donc non seulement construire un objet d'une certaine classe mais aussi que le langage Java ait "standardisé" les méthodes de cette classe qui seront lancées lors d'une action déterminée de l'utilisateur sur le composant graphique.
Toutes ces méthodes ont une signature de la forme :

```
public void nomMethode (XXXEvent evt)
```

où XXX est le type de l'événement.
Par exemple lorsqu'on clique sur un bouton poussoir, c'est la méthode

```
public void actionPerformed (ActionEvent evt)
```

de l'objet délégué qui lui est associé qui sera lancée.

En Java on impose qu'une classe donne bien un corps à une méthode donnée en demandant que cette classe implémente une interface possédant la signature de cette méthode et le compilateur fait cette vérification. Ainsi le langage a défini des interface comme

```
public interface ActionListener {
     public void actionPerformed(ActionEvent evt);
}
```

dans le paquetage `java.awt.event` et impose qu'une classe qui doit fournir des objets délégué à l'écoute d'un clic sur un bouton implémente cette interface.
On a donc :

```
public class MaClasseEcouteurBouton implements ActionListener
{
     public void actionPerformed(ActionEvent evt){
          // le code lancé lorsqu'on clique sur le bouton
}
```

Il reste donc a construire un objet de la classe `MaClasseEcouteurBouton` comme

```
MaClasseEcouteurBouton refEcouteur =
               new MaClasseEcouteurBouton();
```

et a l'associer au bouton comme :

```
Button bt = new Button(…);
bt.addActionListener(refEcouteur);
```

Les associations entre objet graphique et leur listener sont aussi de syntaxe habituelle (design patterns) de la forme

```
addXXXListener (XXXListener)
```

où XXX est toujours la classe de l'événement (cf. ci dessus).

2°) Donner le code complet pour prendre en compte l'appui sur les boutons.
Il y a, la aussi plusieurs solutions. On en donne ici une qui approche le plus possible la correction de la question précédente. Le code additionnel est en gras.

```
import java.awt.*;
import java.awt.event.*;

public class FormulaireGrid extends Frame {
	Button[] bt;
	TextField[] champ;

	public static void main(String args[]) {
		new FormulaireGrid().affiche();
	}

	public void affiche() {
		bt = new Button[3];
		champ = new TextField[3];
		for (int i=0; i<3; i++) {
			bt[i] = new Button();
			champ[i] = new TextField(20);
			bt[i].addActionListener(new  MaClasseEcouteurBouton
(champ[i]));
		}
		bt[0].setLabel("Nom");
		bt[1].setLabel("Téléphone");
		bt[2].setLabel("Adresse");

		// le placement
		setLayout(new GridLayout(3, 2));
		for (int j=0; j<3; j++) {
			add(bt[j]);
			add(champ[j]);
		}
		// setSize(300, 125);
		pack();
		setVisible(true);
	}
}

class MaClasseEcouteurBouton implements ActionListener {
	TextField  leTextFieldAssocie;

	public MaClasseEcouteurBouton(TextField tf) {
		leTextFieldAssocie = tf;
	}

	public actionPerformed(ActionEvent evt) {
		System.out.println(leTextFieldAssocie.getText());
	}
}
```

La difficulté de l'exercice est que, lors du lancement de la méthode `actionPerformed()` de l'objet écouteur on a besoin du textfield associé pour récupéré son texte. Le seul argument de cette méthode est `ActionEvent` et ne fournit pas le textField : il faut donc le connaître

avant. Comme la classe est externe, la seule technique et de l'avoir passé au constructeur au moment de la construction de l'objet écouteur. D'où le code ci dessus.

Par contre si la classe ecouteur est interne ou anonyme (qui est d'ailleurs un exemple de classe interne), de telles classes peuvent accéder aux champs de la classe englobante. On a donc un code beaucoup plus simple et concis que voici.
Avec les classes internes :
Il faut modifier un peu le code pour repérer avec précision chaque `TextField` et `Button`.

```
import java.awt.*;
import java.awt.event.*;

public class FormulaireGrid extends Frame {
        Button btNom, btTelephone, btAdresse;
        TextField champNom, champTelephone, champAdresse,;

        public static void main(String args[]) {
                new FormulaireGrid().affiche();
        }

        public void affiche() {
                btNom = new Button('Nom");
                btTelephone = new Button("Telephone ");
                btAdresse = new Button("Adresse ");
                }
                champNom = new TextField(20);
                champTelephone = new TextField(20);
                champAdresse = new TextField(20);

                btNom.addActionListener(new  EcouteurBtNom ());
                btTelephone.addActionListener(new  EcouteurBtTelephone ());
                btAdresse.addActionListener(new  EcouteurBtAdresse ());


                // le placement
                setLayout(new GridLayout(3, 2));
                add(btNom); add(champNom);
                add(btTelephone); add(champTelephone);
                add(btAdresse); add(champAdresse);

                // setSize(300, 125);
                pack();
                setVisible(true);
        }

                class EcouteurBtNom implements ActionListener {
                        public actionPerformed(ActionEvent evt) {
                                System.out.println(champNom.getText());
                        }
                }

                class EcouteurBtTelephone implements ActionListener {
                        public actionPerformed(ActionEvent evt) {
                                System.out.println(champTelephone.getText());
                        }
                }
```

```
                class EcouteurBtAdresse implements ActionListener {
                        public actionPerformed(ActionEvent evt) {
                                System.out.println(champAdresse.getText());
                        }
                }
}
```

avec les classes anonymes
Elle permettent de mettre le code "à la volée" sans définir de nom de classe qui finalement ne sert pas à grand chose. Voici un exemple s'appuyant sur le code précédent (les nouveautés en gras.

```
import java.awt.*;
import java.awt.event.*;

public class FormulaireGrid extends Frame {
        Button btNom, btTelephone, btAdresse;
        TextField champNom, champTelephone, champAdresse,;

        public static void main(String args[]) {
                new FormulaireGrid().affiche();
        }

        public void affiche() {
                btNom = new Button('Nom");
                btTelephone = new Button("Telephone ");
                btAdresse = new Button("Adresse ");
                }
                champNom = new TextField(20);
                champTelephone = new TextField(20);
                champAdresse = new TextField(20);

                // les listener
                btNom.addActionListener(new  ActionListener () {
                        public actionPerformed(ActionEvent evt) {
                                System.out.println(champNom.getText());
                        }
                });

                btTelephone.addActionListener(new ActionListener () {
                        public actionPerformed(ActionEvent evt) {
                                System.out.println(champTelephone.getText());
                        }
                });

                btAdresse.addActionListener(new ActionListener () {
                        public actionPerformed(ActionEvent evt) {
                                System.out.println(champAdresse.getText());
                        }
                });

                // le placement
                setLayout(new GridLayout(3, 2));
                add(btNom); add(champNom);
                add(btTelephone); add(champTelephone);
                add(btAdresse); add(champAdresse);

                // setSize(300, 125);
```

```
                pack();
                setVisible(true);
        }
}
```

3°) Transformer cette application indépendante en applet. On précisera le fichier HTML qui charge cette applet.

On ne peut malheureusement pas faire de la classe FormulaireGrid directement une applet car elle dérive de `Frame` et que pour être une applet il faut dériver de la classe `Applet` (pas d'héritage multiple en Java). On pourrait envisager une applet qui est un bouton poussoir qui, lorsqu'il est cliqué fait apparaître une `Frame` de classe FormulaireGrid. L'immense intérêt de ceci c'est qu'il n'y aurait pas beaucoup de code à écrire et rien a modifier dans la classe FormulaireGrid et donc pas même avoir besoin du code source de cette classe.
Je préfère tout reécrire :

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Formulaire extends Applet {
        Button btNom, btTelephone, btAdresse;
        TextField champNom, champTelephone, champAdresse,;

        public void init() {
                btNom = new Button('Nom");
                btTelephone = new Button("Telephone ");
                btAdresse = new Button("Adresse ");
                }
                champNom = new TextField(20);
                champTelephone = new TextField(20);
                champAdresse = new TextField(20);

                // les listener
                btNom.addActionListener(new  ActionListener () {
                        public actionPerformed(ActionEvent evt) {
                                System.out.println(champNom.getText());
                        }
                });

                btTelephone.addActionListener(new ActionListener () {
                        public actionPerformed(ActionEvent evt) {
                                System.out.println(champTelephone.getText());
                        }
                });

                btAdresse.addActionListener(new ActionListener () {
                        public actionPerformed(ActionEvent evt) {
                                System.out.println(champAdresse.getText());
                        }
                });

                // le placement
                setLayout(new GridLayout(3, 2));
                add(btNom); add(champNom);
                add(btTelephone); add(champTelephone);
                add(btAdresse); add(champAdresse);

                // setSize(300, 125);
```

```
                pack();
                setVisible(true);
        }
}
```

Remarque importatnte :
Si on écrit un tel code, la sortie par `System.out.println()` ne se fait pas dans la fenêtre qui a lancé le navigateur (à vrai dire heureusement) mais dans une fenêtre à part du navigateur qu'il faut souvent activer et qui est la Java Console.

Le code du fichier HTML qui charge cette applet peut simplement être :

```
<APPLET CODE=Formulaire.class WIDTH=400 HEIGHT=400>
</APPLET>
```

mais il est plus raisonnable qu'il soit :

```
HTML><HEAD><TITLE>applet formulaire</TITLE></HEAD>
<BODY>
<APPLET CODE=Formulaire.class WIDTH=400 HEIGHT=400>
</APPLET>
</BODY>
</HTML>
```