

LES APPLICATIONS GRAPHIQUES

Table des matières

1	Les applications graphiques	2
1.1	Introduction	2
1.2	Programmation événementielle.....	2
1.3	Paquetages graphiques en Java (avec swing)	3
1.4	Librairie plus récente : JavaFX	4
1.5	Composition d'une fenêtre graphique.....	4
1.6	Disposition à l'intérieur d'une fenêtre ou d'un panneau	5
1.7	Structure d'une application graphique à une fenêtre.....	6
1.8	Exemple choisi	7
2	Écrire le programme principal	7
3	Classe qui dérive de JFrame	8
3.1	Attributs	8
3.2	Constructeur.....	9
3.3	Méthodes d'initialisation appelées par le constructeur.....	11
4	Classe internes et les écouteurs	14
4.1	Classe interne.....	14
4.2	Principe de base des écouteurs	14
4.3	Interface ActionListener.....	15
5	Quelques composants et méthodes utiles.....	16
5.1	Résumé des composants utilisés	16
5.2	Méthodes utiles pour tous les composants	17
5.3	Autres classes utiles.....	17
5.4	Les organisateurs de disposition (LayoutManager)	17
6	Architecture MVC	18

1 Les applications graphiques

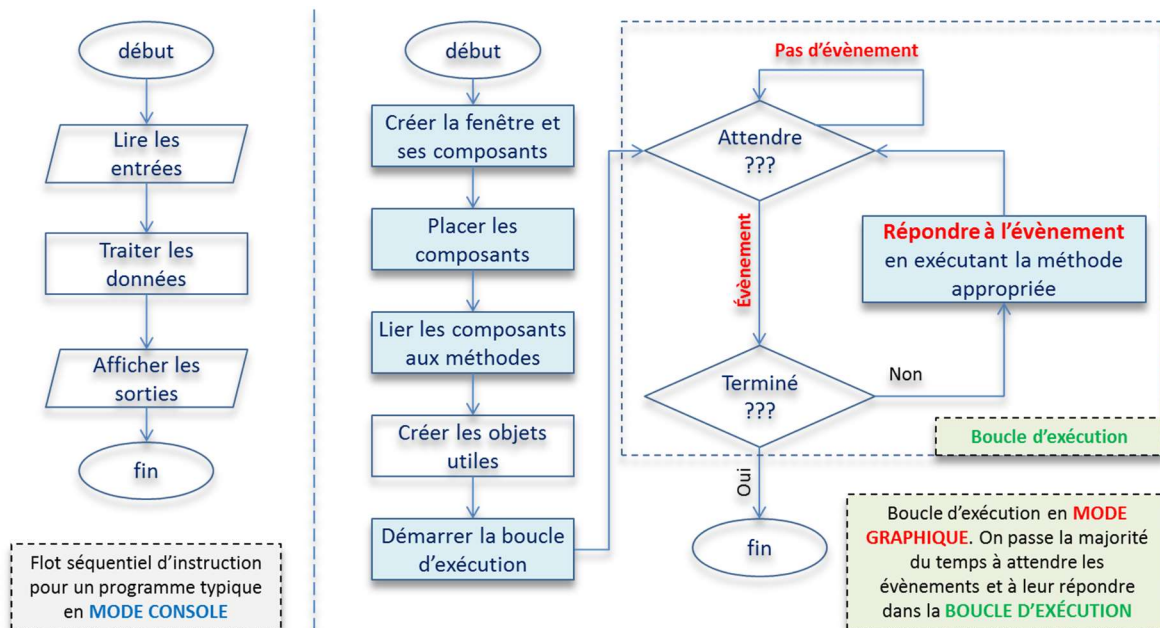
1.1 Introduction

De nos jours, une grande majorité de programmes informatiques fonctionnent avec des interfaces graphiques, que ce soit l'éditeur de texte dans lequel ce chapitre a été écrit, le lecteur de fichier .pdf avec lequel vous le lisez, ou l'application web dans lequel vous exécutez vos transactions bancaires.

Nous allons voir dans ce chapitre les principes de base qui nous permettront de créer des applications graphiques simples, mais tout de même plus dynamiques et d'apparence plus professionnelle que les programmes consoles que nous avons utilisés jusqu'à maintenant.

1.2 Programmation évènementielle

Jusqu'à maintenant nous avons créé des applications de type séquentiel. Dans ces applications, le programme principal contient **une séquence d'instructions exécuté dans un ordre prédéterminé**. Cette séquence peut tout de même s'adapter aux interventions de l'utilisateur avec les branchements conditionnels et les structures répétitives.



Dans les applications avec interfaces graphiques, on peut créer et ouvrir une fenêtre qui **contiendra plusieurs composants graphiques** (boutons, champs de texte, listes déroulantes, ...). **Lorsque l'utilisateur interagit avec un des**

composants graphiques (en cliquant dessus, en y entrant du texte, ...), ce composant graphique **génère un évènement** et **l'application réagit en exécutant le code** approprié. Ici il n'y a pas de code séquentiel à exécuter mais plutôt une application qui contient plusieurs méthodes prêtes à être exécutées lorsqu'elles seront sollicitées par un évènement généré par un clic de bouton, un mouvement de souris, une sélection de touche de clavier,

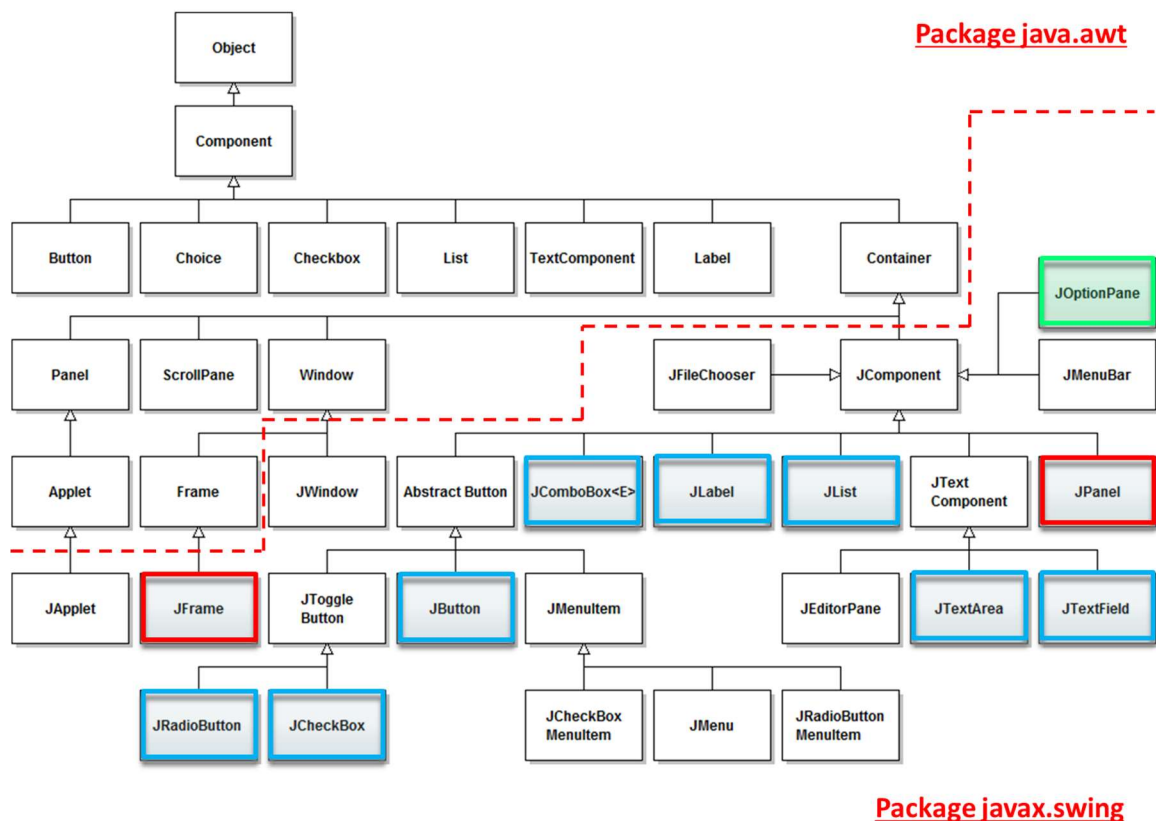
1.3 Paquetages graphiques en Java (avec swing)

Il y a deux paquetages (package) Java qui sont utilisés pour les applications graphiques :

java.awt.* et **javax.swing.***

Le paquetage AWT (Abstract Window Toolkit) fut introduit dans les premières versions de Java. Puis, dans l'optique de créer un système graphique indépendant du système d'exploitation, on a introduit le paquetage Swing, dont la majorité des classes dérivent de celles incluses dans le paquetage AWT.

Voici la hiérarchie des composants graphiques les plus communs de ces deux paquetages :

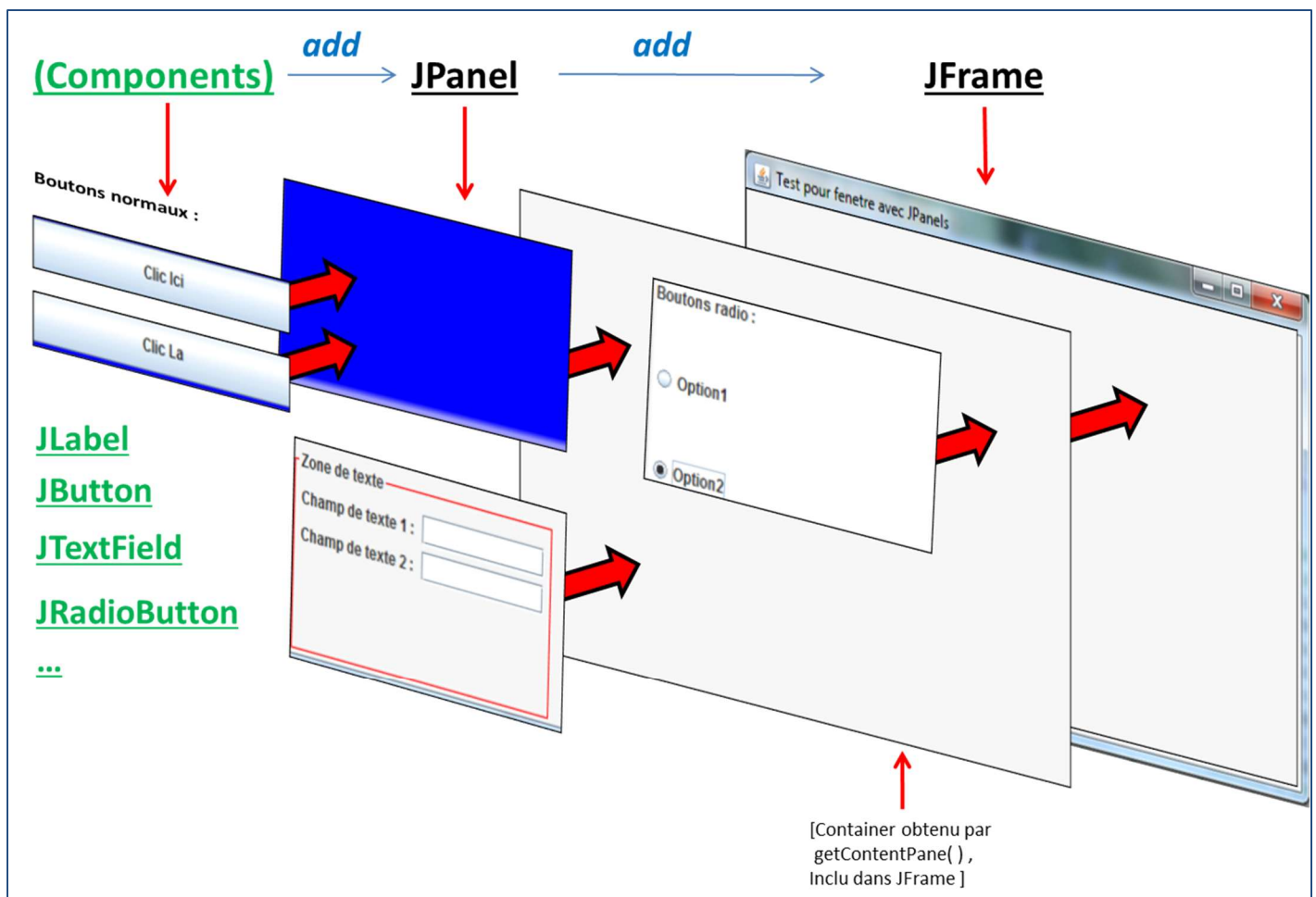


1.4 Librairie plus récente : JavaFX

Dans votre cours d'Applications Natives 1, vous utiliserez une approche plus moderne avec JavaFX ([lien1](#), [lien2](#) et [lien3](#)). Mais plusieurs principes utilisés avec swing, qui sont présentés dans les sections qui suivent, seront encore utilisées avec JavaFX.

1.5 Composition d'une fenêtre graphique

Les fenêtres graphiques sont construites en couches. On crée d'abord une fenêtre vide (**JFrame**), à laquelle on ajoute des panneaux (**JPanel**), auxquels on ajoute des composants graphiques (**tous dérivés de JComponent**).



Il est à noter que :

- On peut ajouter des composants graphiques directement dans la fenêtre sans passer par les panneaux

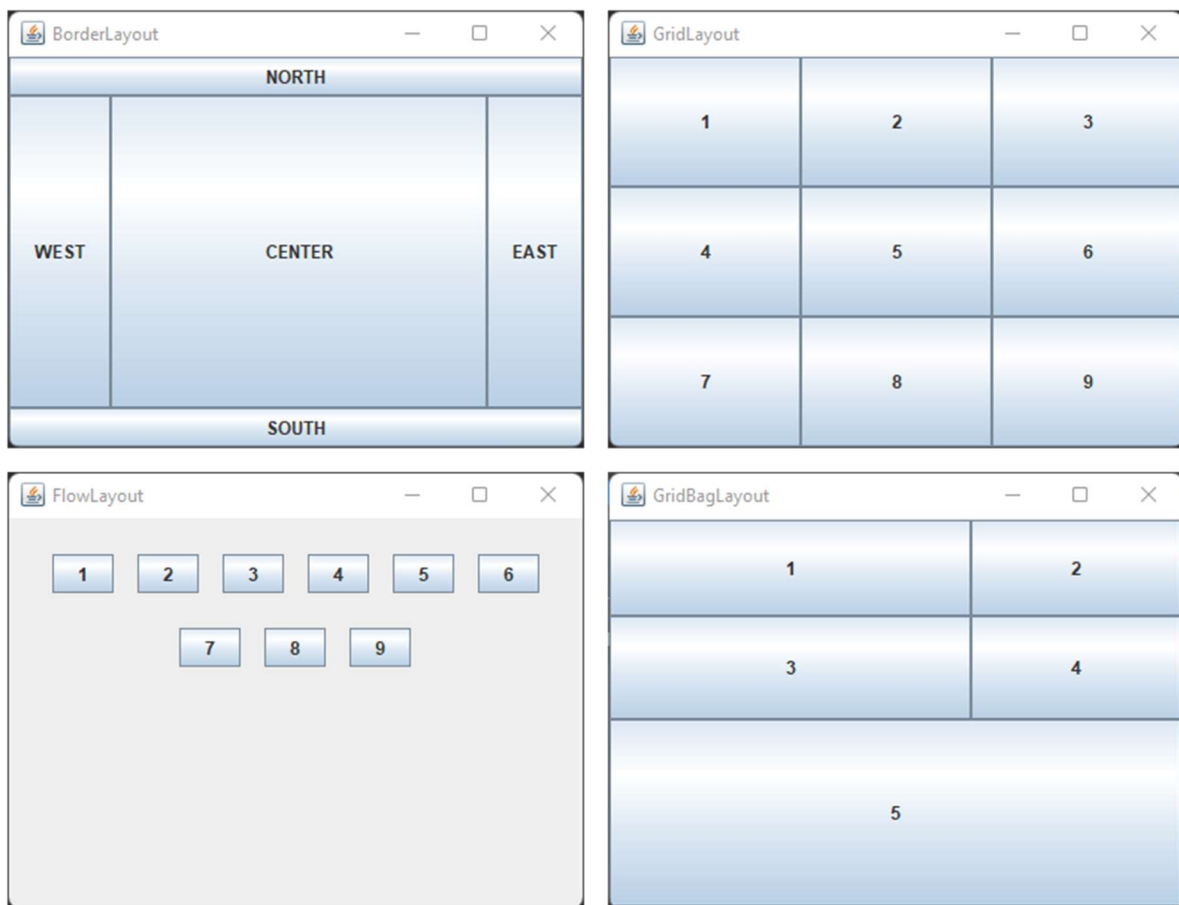
- On peut ajouter plusieurs niveaux de panneaux empilés les uns par-dessus les autres (un panneau dans un panneau dans un ... dans la fenêtre).

1.6 Disposition à l'intérieur d'une fenêtre ou d'un panneau

Lorsqu'on ajoute des composants dans un panneau ou une fenêtre, l'endroit où ils seront placés dépend de deux facteurs :

- L'ordre dans lequel les composants sont ajoutés.
- Le schéma de disposition du panneau ou de la fenêtre.

Il existe plusieurs classes qui permettent d'implémenter un schéma, en voici quelques-unes parmi les plus courantes et les plus simples à utiliser :



Par défaut, la fenêtre JFrame utilise le schéma BorderLayout et le panneau JPanel utilise le schéma FlowLayout. Comme nous allons le voir plus loin, il est possible de changer ces choix en utilisant la méthode **.setLayout()** des classes JPanel et JFrame.

1.7 Structure d'une application graphique à une fenêtre

Voici la structure d'une classe qui permet d'implémenter une application graphique à une seule fenêtre. Nous verrons en détails, dans les prochaines sections, comment implémenter chacun des éléments de cette structure, ainsi que le programme principal qui permettra de démarrer cette application.

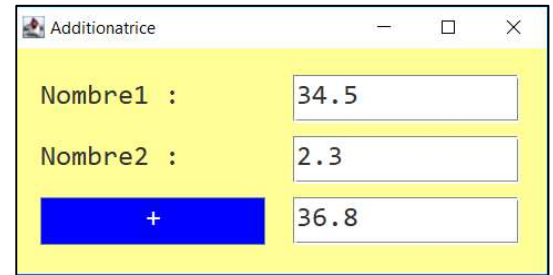
```
public class MonApp extends JFrame {
    // ===== Définition des attributs =====
    // ..... pour les composants graphiques
    // ..... pour la gestion d'évènements
    // ..... pour les données utiles

    // ===== Constructeur =====
    public MonApp(... paramètres ...) {
        // ... JFrame constructeur
        super();
        // ... Personnalisation de la fenêtre
        ...
        // ... Personnalisation des polices
        ...
        // ... Initialisation des données utiles
        this.initDonnees();
        // ... Ajout des composants
        this.ajouterComposants();
        // ... Ajout des écouteurs
        this.ajouterEcouteurs();
        // ... Ajustement de visibilité de la fenêtre
        this.setVisible(true);
    }
    // ===== Méthode pour ajouter les composants =====
    public void ajouterComposants(){ ... }
    // ===== Méthode pour ajouter les écouteurs =====
    public void ajouterEcouteurs(){ ... }
    // ===== Méthode pour initialiser les données utiles =====
    public void initDonnees() { ... }
    // ===== Classes internes pour les écouteurs =====
    public class EcouteurX implements ActionListener {
        public void actionPerformed(ActionEvent e) { ... }
    }
}
```

Remarquez que cette classe dérive de JFrame, ce qui lui donne toute la fonctionnalité de base pour fonctionner comme une fenêtre typique.

1.8 Exemple choisi

Dans **les sections qui suivent**, nous allons utiliser l'exemple d'une application (classe **AppCalculatrice**) qui additionne deux nombres pour illustrer ce qu'il faut faire dans chacune des sections du code présenté au point précédent. Le résultat final est illustré dans la figure ci-contre.



2 Écrire le programme principal

Le programme principal, ne contiendra que la méthode "main" avec une seule instruction, soit la création d'une instance de notre classe dérivant de JFrame.

```
public class ProgrammePrincipalCalculatrice {
    public static void main(String[] arg) {
        // Créer et démarrer l'application graphique avec la fenêtre
        AppCalculatrice fen = new AppCalculatrice("Additionatrice",400,200);
    }
}
```

En créant cet objet qui dérive de **JFrame**, on active un fil d'exécution (thread) qui ouvre une fenêtre qui, par défaut, possède les propriétés d'une fenêtre normale (possibilité de la redimensionner, de la minimiser, de la fermer, ...). Cette fenêtre restera présente et répondra aux commandes de l'utilisateur tant que l'application roulera ou qu'elle ne sera pas fermée par une action de l'utilisateur.

Notes :

- Les paramètres à utiliser pour construire la fenêtre dépendront du constructeur, comme nous le verrons à la prochaine section.
- Normalement, on conserve toutes les classes dans des fichiers séparés et on crée une classe séparée qui contient le programme principal (méthode main). Puisque la méthode main n'a qu'une seule instruction on peut aussi placer la méthode main à l'intérieur de la classe qui contient notre application.

```
public class AppCalculatrice extends JFrame {
    // ===== Méthode main =====
    public static void main(String[] args) {
        AppCalculatrice fen = new AppCalculatrice ("Additionatrice",400,200);
    }
    //... le reste de la classe : attributs, constructeurs, méthodes
}
```

```
}
```

3 Classe qui dérive de JFrame

3.1 Attributs

3.1.1 Attributs pour notre exemple

Voici la liste des attributs que l'on doit définir pour notre application.

```
// ===== Définition des attributs =====  
// ... Attributs pour les composants graphiques  
private JButton btnPlus;  
private JTextField txtNombre1,txtNombre2,txtSomme;  
// ... Attributs pour les écouteurs d'événement  
private EcouteurPlus ecouteur1;  
// ... Attributs pour les données utiles (ex: maBanque, leCompte, ...)  
// il n'y en a aucun dans cet exemple
```

3.1.2 Les attributs des composants

On doit définir seulement des attributs pour les composants avec lesquels nous allons interagir pendant l'exécution du programme (ex: les boutons, les boîtes de textes, les listes déroulantes, ...), mais pas ceux qui sont seulement décoratifs (ex: panneaux, étiquettes, images, ...). Il est une bonne pratique de nommer les composants d'un même type toujours avec le même début de variable (ex: txt..., btn..., lst..., ...).

Dans notre exemple, on a réservé, un attribut pour un bouton, et trois autres pour des zones de textes éditables.

3.1.3 Les attributs pour la gestion des événements

Normalement, pour chaque action à exécuter, nous allons ajouter un objet d'une classe qui implémente l'interface **ActionListener** avec sa méthode **actionPerformed**. Comme nous allons le voir plus tard, ces classes seront implémentées comme des classes internes, et c'est là que tout le code qui permet d'interagir avec les actions de l'utilisateur seront exécutées. Dans notre cas, la classe interne s'appellera **EcouteurPlus** et sera associé à notre attribut **ecouteur1**.

3.1.4 Les attributs pour la gestion des données utiles

En plus des composants graphiques, un programme gère aussi des données utiles, souvent représentés par des objets. Notre exemple est trop simple pour en avoir besoin, mais supposer que vous voudriez faire un mini guichet automatique, il faudrait alors ajouter des attributs tel que :

```
private Banque maBanque;  
private CompteSimple leCompte;  
...
```

3.2 Constructeur

3.2.0 Idée générale

Presque tout le travail à faire se situe dans le constructeur de la fenêtre qui dérive de la classe JFrame. Entre autres, le constructeur doit/peut accomplir les actions suivantes :

1. Appeler le constructeur de son parent avec `super()` (celui de JFrame)
2. Personnaliser la fenêtre (titre, grandeur, couleur, disposition ...)
3. Personnaliser des polices à utiliser
4. Créer, ajuster, disposer, et empiler les différents composants graphiques
5. Créer les écouteurs et les ajouter aux différents boutons
6. Initialiser les données utiles
7. Rendre l'application visible

Puisque ce constructeur peut devenir très volumineux, il est plus pratique de définir quelques **fonctions** qui effectue **les tâches 4 à 6**.

Voyons maintenant tâches par tâche les actions à accomplir.

3.2.1 Appel du constructeur du parent

```
// ===== Constructeur de la fenêtre =====  
public FenetreCalculatrice(String unTitre,int largeur, int hauteur) {  
    // ... Initialisation du parent : JFrame  
    super();  
    // ...  
}
```

Cette tâche n'est pas obligatoire, puisque par défaut, en Java, le `super()` vide est appelé automatiquement s'il est omis dans le code.

3.2.2 Personnaliser la fenêtre

Il est possible de personnaliser la fenêtre qui dérive de la fenêtre de base de la classe `JFrame`. On peut ainsi changer son titre, sa grandeur de départ, sa grandeur minimale, sa couleur d'arrière-plan, ...

Dans notre exemple, nous allons simplement fixer sa grandeur et le titre qui apparaît dans la barre du haut, en utilisant les paramètres que le constructeur a reçu et les méthodes `setSize` et `setTitle`. C'est aussi possible d'appeler la méthode `setLocationRelativeTo` pour placer la fenêtre au milieu de l'écran.

```
// ... Personnalisation de la fenêtre
this.setTitle(unTitre);
this.setSize(largeur, hauteur);
this.setLocationRelativeTo(null);
```

Il est à noter que le "**this.**" réfère ici aux méthodes disponibles dans notre classe et son parent. Ces deux méthodes sont en effet définies dans la classe `JFrame`.

3.2.3 Personnaliser les polices

Ici, nous allons utiliser la méthode statique **put** la classe ***UIManager*** pour appliquer des polices spécifiques sur différents types de composants, si celles qui sont utilisées par défaut ne nous conviennent pas.

```
// ... Personnalisation des polices
Font maPolice = new Font("Consolas", 0, 20);
UIManager.put("Label.font", maPolice);
UIManager.put("Button.font", maPolice);
UIManager.put("TextField.font", maPolice);
```

Pour obtenir la liste des polices disponibles dans votre environnement, on peut utiliser le code suivant. La liste des polices s'affichera dans la console.

```
String[] fonts = GraphicsEnvironment.getLocalGraphicsEnvironment()
                    .getAvailableFontFamilyNames();

for (String font : fonts)
    System.out.println(font);
```

3.2.4 Initialiser les données utiles, ajouter les composants et les écouteurs

Dans cette section, puisque ce code peut être plutôt long, nous allons invoquer trois nouvelles méthodes qui permettront d'effectuer ces tâches

```
// ... Initialisation des données utiles
this.initDonnees();
// ... Ajout des composants
this.ajouterComposants();
// ... Ajout des écouteurs
this.ajouterEcouteurs();
```

Ces méthodes seront décrites à la section 3.3.

3.2.5 Rendre l'application visible

Une fois que toutes les tâches d'initialisation sont terminées, il ne reste plus qu'à rendre l'application visible avec cette instruction provenant de la classe JFrame :

```
// ... Ajustement de visibilité de la fenêtre
this.setVisible(true);
```

3.3 Méthodes d'initialisation appelées par le constructeur

3.3.1 La méthode ajouterComposants

C'est ici qu'on crée, ajuste et ajoute les composants graphiques à la fenêtre.

Il est fortement conseillé de prendre le temps de planifier et de dessiner le résultat désiré sur une feuille de papier ou avec une application de dessin (PowerPoint, Visio, ...) avant de commencer à coder.

On divise d'abord la fenêtre en sections regroupant les composants qui ont un lien entre eux. Ensuite, pour chacune des sections de la fenêtre, il faut :

- **créer un panneau** JPanel
- **ajouter un schéma de disposition** au panneau
- au besoin, ajuster les propriétés du panneau (ex. couleur d'arrière-plan)
- **créer les composants** (JButton, JTextField, JLabel, JList, ...)
- au besoin, ajuster les propriétés des composants

- **ajouter les composants au panneau** dans l'ordre dans lequel vous voulez qu'ils apparaissent
- **ajouter le panneau à la fenêtre**

```
// ===== Méthode pour ajouter les composants =====
public void ajouterComposants(){
    // Créer et ajuster un panneau jaune pâle
    // avec bordure transparente de 20px
    JPanel panneau=new JPanel();
    panneau.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    panneau.setBackground(new Color(255,255,150));

    // Ajuster la disposition du panneau (grille 3 rangées x2 colonnes)
    panneau.setLayout(new GridLayout(3,2,20,10));

    // Créer les composants ui sont des attributs
    this.btnPlus = new JButton("+");
    this.txtNombre1 = new JTextField("0.0");
    this.txtNombre2 = new JTextField("0.0");
    this.txtSomme = new JTextField("0.0");
    // Modifier les composants (ex: ici on change la couleur du bouton)
    this.btnPlus.setBackground(Color.BLUE);
    this.btnPlus.setForeground(Color.WHITE);

    // Ajouter les composant au panneau
    // ... les étiquettes sont ajoutées directement (sans attributs
    // ... car on n'a plus besoins d'y accéder plus tard)
    panneau.add(new JLabel("Nombre1 : "));
    panneau.add(txtNombre1);
    panneau.add(new JLabel("Nombre2 : "));
    panneau.add(txtNombre2);
    panneau.add(btnPlus);
    panneau.add(txtSomme);

    // Ajouter le panneau à la fenêtre
    this.add(panneau);
}
```

Ce qui donne visuellement :

3.3.2 La méthode ajouterEcouteurs

Il reste maintenant à faire le lien entre les boutons et les méthodes à exécuter sur un clic de ce bouton. Pour ce faire, il suffit d'utiliser la méthode **addActionListener** sur le composant. Cette méthode doit recevoir un objet qui implémente l'interface **ActionListener**. Comme nous allons le voir bientôt, nous allons devoir créer une classe qui implémente cette interface.

Pour l'instant, notre méthode doit simplement créer une instance de cette classe et l'attacher au bouton. Dans notre exemple, il n'y a qu'un seul bouton qui sera attacher la méthode actionPerformed de la classe interne EcouteurPlus.

```
// ===== Méthode pour ajouter les écouteurs =====
public void ajouterEcouteurs(){
    this.ecouteur1= new EcouteurPlus();
    this.btnPlus.addActionListener(this.ecouteur1);
}
```

3.3.3 La méthode initDonnees

Comme nous l'avons vu plutôt, nous allons créer une fonction qui permettra d'initialiser les données utiles. Dans **l'exemple de la calculatrice**, cette **fonction sera vide**.

```
// ===== Méthode pour initialiser les données utiles =====
public void initDonnees() {
}
```

Par contre, si on reprend l'exemple du guichet automatique mentionné précédemment, on pourrait avoir :

```
public void initDonnees() {
    this.maBanque = new Banque("Crosemont");
    this.leCompte = this.maBanque.chercherCompteParNumero(1);
    ...
}
```

où maBanque et leCompte sont des attributs de notre application.

4 Classe internes et les écouteurs

4.1 Classe interne

Il est possible en Java, de créer une ou plusieurs classes à l'intérieur d'une autre classe. Si cette nouvelle classe, dite interne est privée, elle pourra être utilisée seulement par notre classe qui la contient, dite externe. Voici à quoi ressemblerait une telle structure :

```
public class UneClasseExterne {  
    ... attributs  
    ... méthodes  
    private class UneClasseInterneNo1 {  
        ... attributs  
        ... méthodes  
    }  
    ...  
    private class UneClasseInterneNo1 {  
        ... attributs  
        ... méthodes  
    }  
}
```

4.2 Principe de base des écouteurs

Chaque composant graphique a la capacité de générer un évènement lorsque l'utilisateur interagit avec celui-ci. Lors d'une action de l'utilisateur, le composant graphique regarde la liste de tous les objets qui lui ont été affecté comme écouteur, et appelle la méthode correspondant à l'action si applicable.

Pour ce faire, il faut d'abord :

- Créer une classe interne qui implémente une des interfaces dérivant d'**EventListener** du paquetage ***java.awt.event***.

C'est dans cette classe qu'on **place le code à effectuer lorsque l'évènement est déclenché**.

4.3 Interface ActionListener

Dans cette introduction aux interfaces graphiques, nous allons nous concentrer sur un seul type d'évènement **ActionEvent** et une seule interface **ActionListener**. Cette interface permet, entre autres, de réagir à un clic sur différent type de boutons.

Comme mentionné plus tôt, cette interface exige qu'on implémente une seule méthode, soit la méthode **actionPerformed** qui reçoit en paramètre un objet de type **ActionEvent**. Ce paramètre est fourni automatiquement par l'interpréteur java quand un événement arrive (... un peu comme avec les exceptions ...).

C'est dans cette méthode que l'on doit faire le travail à accomplir pour réagir à un tel événement. Dans la plupart des cas, on doit :

- On doit récupérer les données à partir des composants graphiques ou des données utiles.
- On doit traiter ces données et au besoin modifier les données utiles.
- On doit mettre à jour l'affichage des composants graphiques.

Pour l'exemple de la calculatrice, cela correspond à :

- Récupérer les données des zones de texte txtNombre1 et txtNombre2
- Convertir les données textes en nombre et en calculer la somme
- Convertir la somme en chaine de caractères et mettre à jour l'affichage de la zone de texte txtSomme



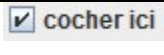

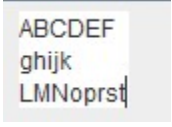

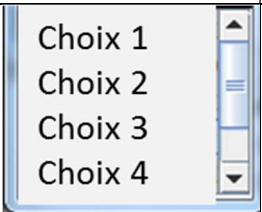
Ce qui donne le code suivant :

```
// ===== Classes internes pour les écouteurs =====
public class EcouteurX implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        double nombre1=Double.parseDouble(txtNombre1.getText());
        double nombre2=Double.parseDouble(txtNombre2.getText());
        double somme=nombre1+nombre2;
        txtSomme.setText(Double.toString(somme));
    }
}
```

Les méthodes **getText** et **setText** permettent d'échanger du contenu avec les zones de textes de type **JTextField**. Il est à noter qu'elles ne traitent que des String. Si on veut traiter d'autres types de données, il faut faire les conversions.

5 Quelques composants et méthodes utiles

5.1 Résumé des composants utilisés

CLASSE	APPARENCE	CONSTRUCTEUR	MÉTHODE UTILES
JLabel	Ligne txt	<code>lbl = new JLabel("Ligne txt");</code>	
JButton		<code>btn = new JButton("Bouton 1");</code>	
JRadioButton		<code>rad = new JRadioButton("Option2");</code>	<code>boolean isSelected() void setSelected(boolean b)</code>
JCheckBox		<code>chk= new JCheckBox("cocher ici");</code>	<code>boolean isSelected() void setSelected(boolean b)</code>
JTextField		<code>txf = new JTextField(10); // 10 colonnes de large</code>	<code>String getText() void setText(String txt)</code>
JTextArea		<code>txa = new JTextArea(3,5); // 3 rangées et // 5 colonnes</code>	<code>String getText() void setText(String txt)</code>
JComboBox <E>		<code>String[] s={"danse", "marche", "court"}; cmb=new JComboBox<String>(s);</code>	<code>void setSelectedIndex(int n) Object getSelectedItem() // vous devez convertir au // bon type exemple : // String s = (String) // getItemSelected();</code>
*JList<E> et JScrollPane		<code>String[] s={"Choix 1","Choix 2", "Choix 3","Choix 4", "Choix 5",}; lst=new JList<String>(s); sPan=new JScrollPane(lst);</code>	<code>void setSelectedIndex(int n) E getSelectedItem();</code>

*<E> est le type d'éléments dans la liste. Pour une liste de String on remplace E par String, pour une liste d'Employe on remplace E par Employe, ...

5.2 Méthodes utiles pour tous les composants

Méthode	Exemple
<code>setBackground(Color c)</code>	// Bouton 1 avec couleur arrière-plan bleue <code>btn1.setBackground(Color.BLUE);</code>
<code>setForeground(Color c)</code>	// Bouton 1 avec couleur de texte rouge <code>btn1.setForeground(Color.RED);</code>
<code>//add ListSelectionListener(... ecouteur..)</code> <code>//pour JList</code> <code>addActionListener(...ecouteur ...)</code>	// Ajoute un écouteur avec un objet qui // implémente Action Listener au bouton 1 <code>btn1.addActionListener(ecouteur1);</code>
<code>setBorder(Border b)</code>	// Ajoute un contour à un composant ou panneau

5.3 Autres classes utiles

Classe	Obtenir l'objet	Description
Color	<code>Color.RED;</code> <code>new Color(int r, int g, int b)</code>	Obtient la couleur rouge Crée une nouvelle couleur avec index rgb (de 0 à 255)
Border	<code>BorderFactory.createLineBorder(Color color)</code> <code>BorderFactory.createTitledBorder(String title)</code> ...	Contour ligné Contour avec titre
Button Group	<code>ButtonGroup btn = new ButtonGroup();</code> <code>JRadioButton r1 = new JRadioButton("A");</code> <code>JRadioButton r2 = new JRadioButton("B");</code> <code>JRadioButton r3 = new JRadioButton("C");</code> <code>btn.add(r1);</code> <code>btn.add(r2);</code> <code>btn.add(r3);</code>	Utilisé pour les groupes de boutons radio pour que leur sélection puisse être mutuellement exclusive. Il faut quand même ajouter les boutons un à un au panneau ...

5.4 Les organisateurs de disposition (LayoutManager)

Classe	Construction	Utilisation
FlowLayout	<code>disp = new FlowLayout();</code>	<code>pan.setLayout(disp);</code> <code>pan.add(btn1);</code> <code>pan.add(txf3);...</code>
Grid Layout	<code>disp = new GridLayout(r,c);</code> // r:range; c:colonne	<code>pan.setLayout(disp);</code> <code>pan.add(btn1);</code> <code>pan.add(txf3); ...</code>
BorderLayout	<code>disp = new BorderLayout();</code> <code>*d2 = new BorderLayout(hg,vg);</code>	<code>pan.setLayout(disp);</code> <code>pan.add(btn1,BorderLayout.NORTH);</code> <code>pan.add(txf3,BorderLayout.EAST);</code> <code>pan.add(lbl3,BorderLayout.WEST);</code> <code>pan.add(txf1,BorderLayout.CENTER);</code>

* la plupart des dispositions ont un constructeur qui permet de placer des espacements entre deux composants : `int hgap` et `int vgap` (espacement horizontal et vertical respectivement, en pixel)

6 Architecture MVC

Les exemples vus dans ce chapitre sont plutôt simples. La façon de procéder ne s'applique plus très bien quand la complexité augmente. Une des bonnes pratiques consiste à séparer notre code en 3 catégories (aussi appelées couches) en utilisant l'architecture MVC (modèle-vue-contrôleur) :

- **Modèle** : toutes les classes représentant les objets à manipuler (banque, centre de ski, équipes de soccer, ...) qui doivent être indépendante du mode d'affichage (elles doivent être utilisables en mode console, graphique ou même web, sans aucune modifications).
- **Vue** : représente toute ce qui est graphique. Les calculs et manipulations directes des données doivent être réduits au minimum.
- **Contrôleur** : cette catégorie s'occupe d'échanger les données entre la vue et le modèle.

Voici un des nombreux tutoriels ([lien](#)) qui traite de ce sujet.