



Cours Programmation Orientée Objet : JAVA 2

Enseignant : **Malek BEN SALEM**
mal.bensalem@gmail.com

2^{ème} année Licence en Informatique

AU 2019/2020

1



Plan du Cours

1. Chapitre 1 : Les interfaces graphiques
2. Chapitre 2 : Les entrées sorties
3. Chapitre 3 : L'accès aux bases de données
4. Chapitre 4 : Les threads

2



Chapitre 1: Les interfaces graphiques

Enseignant : **Malek BEN SALEM**
mal.bensalem@gmail.com

2^{ème} année Licence en Informatique

AU 2019/2020

3

Interfaces graphiques

- Une interface graphique (fenêtres, boutons, menus...) nécessite beaucoup de composants et de lignes de code.
- Il est intéressant de bien séparer la partie interface graphique du reste du programme. L'approche objet est bien adaptée.
- Java offre 2 systèmes pour créer des interfaces graphiques :
 - le plus ancien s'appelle **AWT** (Abstract Window Toolkit ou «bibliothèque abstraite de fenêtrage»); package java.awt
 - le plus récent (depuis Java 2) s'appelle **Swing**. C'est une amélioration de AWT; package javax.swing
- **Swing** est un ensemble de classes permettant de créer des objets représentant des composants graphiques.

4

AWT

- **AWT: Abstract Window Toolkit**
 - Package: java.awt
- Utilise des composants **lourds** utilisant beaucoup les ressources système
- Liés directement aux possibilités de l'interface utilisateur graphique de la plateforme locale.
- Le plus petit dénominateur Commun
 - Si non disponible en natif sur l'une des plate-formes Java, alors non disponible sur toutes les plate-formes Java
- Ensemble de Composants Simples
- Affichage différent en fonction de la plateforme

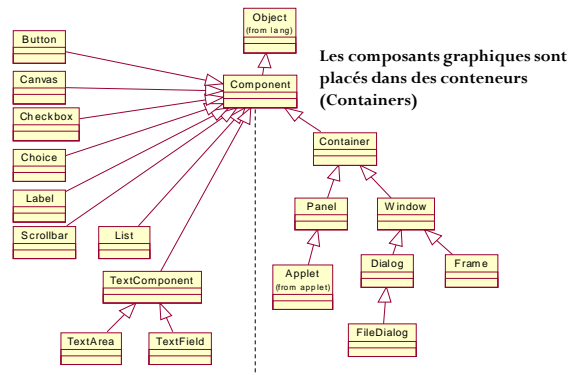
5

Conteneurs et composants

- Une interface graphique en Java est un assemblage **conteneurs (Container)** et de **composants (Component)**.
- **Un composant est une partie "visible" de l'interface utilisateur Java.**
 - C'est une sous-classe de la classe abstraite **java.awt.Component**.
 - Exemple: les boutons, les zones de textes ou de dessin, etc.
- **Un conteneur est un espace dans lequel on peut positionner plusieurs composants.**
 - Sous-classe de la classe **java.awt.Container**.
 - La classe Container est elle-même une sous-classe de la classe Component
 - Exemple: les fenêtres, les applets, etc.

6

Hiérarchie d'héritage des principaux éléments des interfaces graphiques en Java



7

Conteneurs et composants

- Les deux conteneurs les plus courants sont le **Frame** et le **Panel**.
- Un **Frame** représente une fenêtre de haut niveau avec un titre, une bordure et des angles de redimensionnement.
 - ♦ La plupart des applications utilisent au moins un Frame comme point de départ de leur interface graphique.
- Un **Panel** n'a pas une apparence propre et ne peut pas être utilisé comme fenêtre autonome.
 - ♦ Les Panels sont créés et ajoutés aux autres conteneurs de la même façon que les composants tels que les boutons
 - ♦ Les Panels peuvent ensuite redéfinir une présentation qui leur soit propre pour contenir eux-mêmes d'autres composants.

8

Swing

- La bibliothèque **Swing** est une nouvelle bibliothèque de composants graphiques pour Java.
 - Swing est intégré à Java 1.2. (Le Java 2)
 - Swing peut être téléchargé séparément pour une utilisation avec des versions de Java antérieures (1.1.5+)
- Cette bibliothèque s'ajoute à celle qui était utilisée jusqu'alors (**AWT**) pour des raisons de compatibilité.
 - Swing fait cependant double emploi dans beaucoup de cas avec **AWT**.
 - L'ambition de Sun est que, progressivement, les développeurs réalisent toutes leurs interfaces avec Swing et laissent tomber les anciennes **API** graphiques.

9

9

Swing

- Un composant graphique lourd (*heavyweight GUI component*) s'appuie sur le gestionnaire de fenêtres local, celui de la machine sur laquelle le programme s'exécute.
 - AWT ne comporte que des composants lourds.

10

10

Swing

- Exemple :
 - ♦ Un bouton de type `java.awt.Button` intégré dans une application Java sur la plate-forme Unix est représenté grâce à un vrai bouton Motif (appelé son pair - *peer* en anglais).
 - ♦ Java communique avec ce bouton Motif en utilisant la Java Native Interface. Cette communication induit un coût.
 - ♦ C'est pourquoi ce bouton est appelé composant lourd.

11

11

Swing

- Un composant graphique léger (en anglais, *lightweight GUI component*) est un composant graphique indépendant du gestionnaire de fenêtre local.
 - ♦ Un composant léger ressemble à un composant du gestionnaire de fenêtre local mais n'en est pas un : un composant léger émule les composants de gestionnaire de fenêtre local.
 - ♦ Un bouton léger est un rectangle dessiné sur une zone de dessin qui contient une étiquette et réagit aux événements souris.
 - ♦ Tous les composants de Swing, exceptés **JApplet**, **JDialog**, **JFrame** et **JWindow** sont des composants légers.

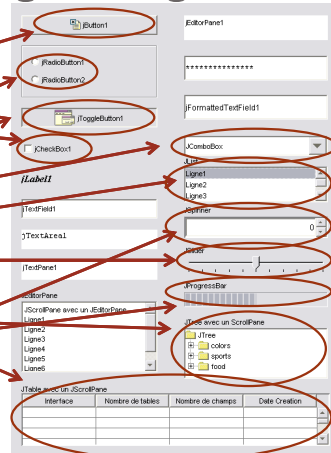
12

12

Vue des paquetages Swing

• Composants Basiques

- JApplet
- JButton
- JCheckBox
- JRadioButton
- JToggleButton
- JComboBox
- JList
- JSlider
- JTable
- JTree
- JProgressBar
- JSpinner



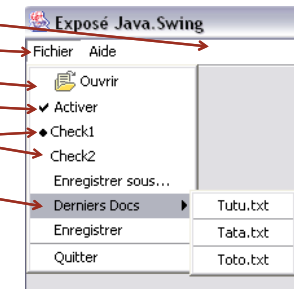
13

13

Vue des paquetages Swing

• Menus, Bar d'outils et les Info Bulles

- JMenuBar
- JMenu
- JMenuItem
- JCheckBoxMenuItem
- JRadioButtonMenuItem
- JPopupMenu
- JToolBar
- JToolTip



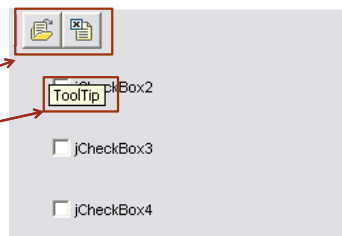
14

14

Vue des paquetages Swing

• Menus, Bar d'outils et Info Bulles

- JMenuBar
- JMenu
- JMenuItem
- JCheckBoxMenuItem
- JRadioButtonMenuItem
- JPopupMenu
- JToolBar
- JToolTip



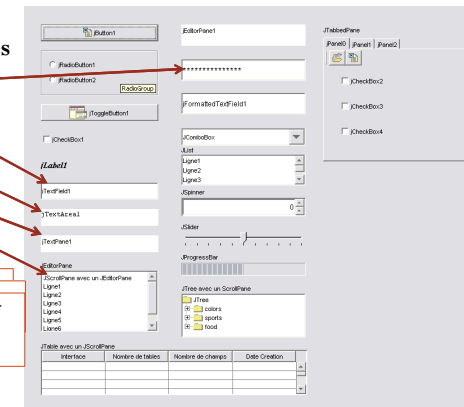
15

15

Vue des paquetages Swing

• Composants Textes

- JPasswordField
- JTextField
- JTextArea
- JTextPane
- JEditorPane



16

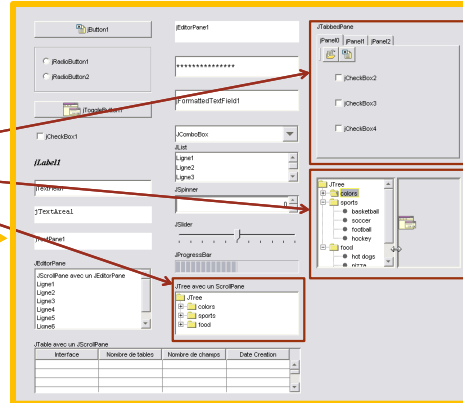
16

JEditorPane permet d'éditer des contenus de natures différentes: Text, HTML, etc

Vue des paquetages Swing

Containers

- JOptionPane
- JDialog
- JTabbedPane
- JSplitPane
- JScrollPane
- JFrame
- JInternalFrame
- JDesktopPane
- JWindow



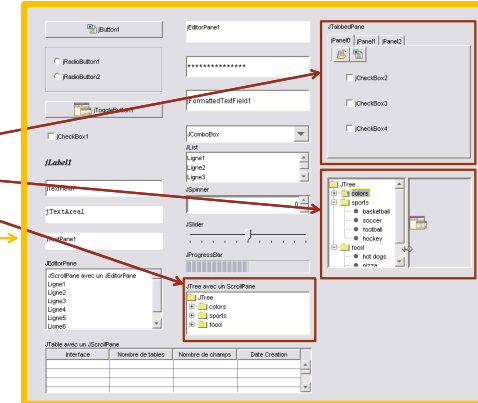
17

17

Vue des paquetages Swing

Containers

- JOptionPane
- JDialog
- JTabbedPane
- JSplitPane
- JScrollPane
- **JFrame**
- JInternalFrame
- JDesktopPane
- JWindow

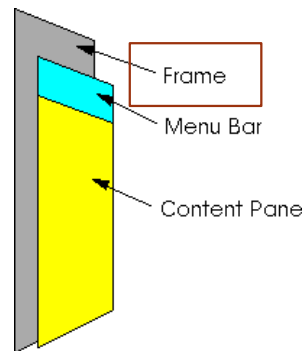
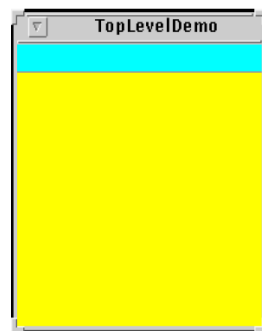


18

18

Créer une interface

La fenêtre JFrame



19

19

La fenêtre JFrame

- Pour créer une fenêtre graphique, on dispose, dans le paquetage *javax.swing*, une classe standard nommée **JFrame** possédant:
 - Un constructeur sans arguments:
 - `JFrame fen = new JFrame();`
 - Si on se limite à cela, **rien n'apparaîtra à l'écran.**
 - Il faut demander l'affichage de la fenêtre en appelant l'une de ces deux méthodes:
 - `fen.setVisible(true);`
 - `fen.show();`

20

20

Diapositive 20

BSm1

dans la séance 1, on est arrivé à la diapo 20.

BEN SALEM malek; 18/09/2019

La fenêtre JFrame

- **Autres fonction indispensable pour la création d'une fenêtre:**

- Par défaut, la taille d'une fenêtre est nulle. Pour définir les dimensions au paravent, il faut utiliser la fonction:

- `fen.setSize(l,h);`
 - h: la hauteur en pixels;
 - L: la largeur en pixels;

- **Exemple:**

`fen.setSize(300,150);`

21

21

La fenêtre JFrame

- **Autres fonction indispensable pour la création d'une fenêtre:**

- En général, on choisira d'afficher un texte précis dans la barre de titre.

- On utilise alors:

- `fen.setTitle(String Message);`

- **Exemple:**

- `fen.setTitle("Ma première fenetre");`

22

22

La fenêtre JFrame

- L'utilisateur peut manipuler cette fenêtre comme n'importe qu'elle fenêtre graphique d'un logiciel:

- La retailler
- La déplacer
- La réduire à une icône.

- **Attention**, la fermeture d'une fenêtre de type *JFrame* ne met pas fin au programme, mais rend simplement la fenêtre invisible (comme si on appelait la méthode *setVisible (false)*).

23

23

La fenêtre JFrame

- Pour que la fermeture de la fenêtre met fin au programme il faut ajouter :

- `setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE) ;`

- Pour fermer la fenêtre sans mettre fin au programme il faut ajouter:

- `setDefaultCloseOperation (JFrame.DISPOSE_ON_CLOSE) ;`

- L'effet de cette instruction n'est visible que si vous avez ouvert plus qu'une fenêtre dans votre programme.

24

24

La fenêtre JFrame

- **Création d'une classe fenêtre personnalisée:**

- Dans l'exemple précédent, nous avons simplement créé un objet de type *JFrame* et nous avons utilisé les fonctionnalités présentes dans cette classe.
- Pour que notre programme présente un intérêt, il faut lui associer des fonctionnalités supplémentaires : la fenêtre devra pouvoir réagir à certains événements.
- Pour cela, nous faudra définir notre propre classe dérivée de *JFrame* et créer un objet de ce nouveau type.

25

25

La fenêtre JFrame

- **Création d'une classe fenêtre personnalisée:**

```
import javax.swing.*;

public class Fenetre extends JFrame {
    public Fenetre()
    {
        setSize(300,150);
        setTitle("Ma première fenêtre");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        Fenetre premFen = new Fenetre();
        premFen.setVisible(true);
    }
}
```

26

26

La fenêtre JFrame

- **Repositionner et redimensionner une fenêtre :**

- On peut utiliser la fonction:
 - `setBounds(px,py,l,h)`
 - Avec:
 - px et py: les coordonnées du pixel qui présente la nouvelle position de la fenêtre.
 - h et l: les nouvelles dimensions de la fenêtre.

27

27

La fenêtre JFrame

- **Repositionner et redimensionner une fenêtre :**

```
import javax.swing.*;

public class Fenetre extends JFrame {
    public Fenetre()
    {
        setSize(300,150);
        setTitle("Ma première fenêtre");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        Fenetre premFen = new Fenetre();
        premFen.setVisible(true);
        premFen.setBounds(250,250,300,200);
    }
}
```

28

28

La fenêtre JFrame

- **Repositionner et redimensionner une fenêtre :**
- Si on veut positionner la fenêtre par rapport au dimension de l'écran:
 - On doit utiliser un trousse à outil (Toolkit) permettant de charger les dimensions, les icones et tous les outils nécessaires à partir du Système d'exploitation. (package Java.awt).

29

29

La fenêtre JFrame

- **Repositionner et redimensionner une fenêtre :**

• **Exemple:**

```
import javax.swing.*;

public class Fenetre extends JFrame {
    public Fenetre()
    {
        setTitle("Ma première fenêtre");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        Fenetre premFen = new Fenetre();
        premFen.setVisible(true);
        Toolkit kit = Toolkit.getDefaultToolkit();
        Dimension screenSize = kit.getScreenSize();
        int H = screenSize.height; // Hauteur
        int W = screenSize.width; // Largeur
        premFen.setBounds(W/4, H/4, W/2, H/2);
        Image img = kit.getImage("icon.gif");
        this.setIconImage(img);
    }
}
```

Repositionnement et redimensionnement par rapport à l'écran

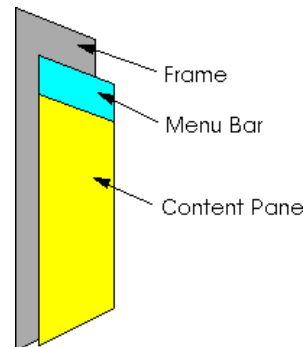
Modification de l'icône de la fenêtre

30

30

Le Panneau JPanel

- Il **n'est pas possible** d'introduire directement un composant dans une fenêtre (**JFrame**).
- **JFrame** possède une structure complexe. Il est formé d'une superposition de plusieurs éléments, en particulier une racine, un contenu, etc .
- La méthode `getContentPane` de la classe **JFrame** fournit la référence à ce contenu de type Container :
 - `Container c = getContentPane();`
 - Ou bien
 - `Jpanel c = new JPanel();`
`c=(JPanel)getContentPane();`



31

31

Le Panneau JPanel: Ajout de composants

- les différents composants sont à ajouter dans la partie contenu avec méthode **add** .
 - **c.add(composant);**
 - **c.remove(composant);** pour la suppression.
- Mais la méthode **add** ajoute toujours le **composant au même endroit dans le conteneur**.
- Ainsi lorsqu'on veut ajouter plusieurs composants dans le panel, **seul le dernier composant apparaît** .

32

32

Le Panneau Jpanel: Ajout de composants

```
class Fenetre extends JFrame
{
    private JButton bouton1, bouton2;
    private JLabel label1, label2;
    private JTextField text1, text2;

    public Fenetre()
    {
        setSize(400,300);
        setTitle("Une fenêtre avec composants");

        // création des composants
        bouton1 = new JButton("Bouton 1");
        bouton2 = new JButton("Bouton 2");
        label1 = new JLabel("Label 1");
        label2 = new JLabel("Label 2");
        text1 = new JTextField("Zone de texte 1");
        text2 = new JTextField("Zone de texte 2");
    }
}
```

33

33

Le Panneau Jpanel: Ajout de composants

```
JPanel c = (JPanel) getContentPane();

// on ajoute les composants
c.add(bouton1);
c.add(label1);
c.add(text1);
c.add(bouton2);
c.add(label2);
c.add(text2);

// fin de constructeur de la classe Fenetre

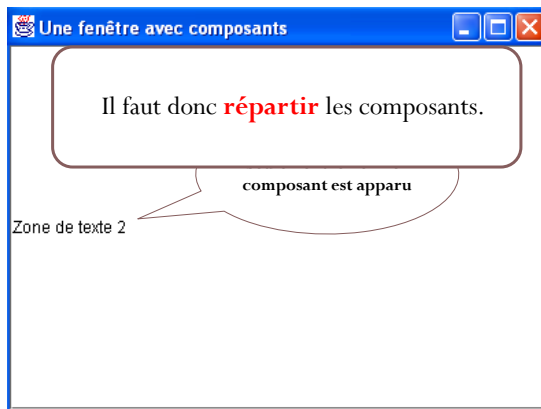
public static void main(String args[])
{
    Fenetre premFen = new Fenetre ();
    premFen.show();
}

// fin de la classe Fenetre
```

34

34

Le Panneau Jpanel: Ajout de composants



35

35

Positionnement Absolu

- Pour placer les composants à **un endroit précis**, il faut indiquer qu'on n'utilise pas de *LayoutManager*.

ObjetConteneur.setLayout(null);

- Et on indique les coordonnées et la taille de chaque composant **Composant.setBounds(x, y, larg, haut);**

- x est le déplacement par rapport à la gauche du conteneur
- y est le déplacement par rapport au haut du conteneur

36

36

Gestionnaires de présentation

- A chaque conteneur est associé un gestionnaire de présentation (**GP**) ou gestionnaire de disposition (*layout manager*)
- Le GP contrôle l'emplacement et la taille des composants chaque fois qu'ils sont affichés.
- Le réagencement des composants dans un conteneur a lieu lors de :
 - la modification de sa taille,
 - le changement de la taille ou le déplacement d'un des composants.
 - l'ajout, l'affichage, la suppression ou le masquage d'un composant.

37

37

Gestionnaires de présentation

- Tout conteneur possède un GP par défaut.
- Toute instance de **Container** référence une instance de la classe **LayoutManager** du package **java.awt**.
- Il est possible de le changer grâce à la méthode **setLayout()**.
- Tous les GPs sont des classes de **java.awt** qui héritent de **LayoutManager**.

38

38

Gestionnaires de présentation

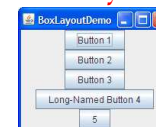
- Pour positionner un composant, nous avons plusieurs positions prédéfinies. Ces positions qui sont proposés par Java sont:
 - BorderLayout
 - BoxLayout
 - CardLayout
 - FlowLayout
 - GridBagLayout
 - GridLayout
- Les GPs par défaut sont :
 - Le BorderLayout pour JFrame et ses descendants
 - Le FlowLayout pour JPanel et ses descendants (JApplet, etc.)

39

39

Gestionnaires de présentation

BoxLayout



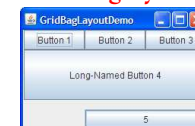
BorderLayout



GridLayout



GridBagLayout



40

40

FlowLayout

- FlowLayout organise les composants en lignes, de gauche à droite, puis de haut en bas, en utilisant la taille préférée (preferredSize) de chaque composant.
- FlowLayout place en ligne autant de composants que possible, avant de recommencer sur une nouvelle ligne.
- L'alignement dans un **FlowLayout** peut être à gauche (FlowLayout.LEFT), à droite (FlowLayout.RIGHT) ou centrée (FlowLayout.CENTER)
 - par défaut, les composants sont centrés à l'intérieur de la zone qui leur est allouée.
 - un espacement horizontal ou vertical entre deux composants est défini à l'aide des méthodes `setHgap()` et `setVgap()`.

41

41

FlowLayout

- Le **FlowLayout** cache réellement et effectivement les composants qui ne rentrent pas dans le cadre.
- Le **FlowLayout** n'a d'intérêt que lorsqu'il y a peu de composants dans le conteneur.
- **Exemple:**
 - `JPanel p= (JPanel) this.getContentPane();`
`p.setLayout(new FlowLayout(FlowLayout.LEFT));`

42

42

FlowLayout

- L'appel de la méthode `pack()` pour une fenêtre (JFrame) lui demande de calculer sa taille (preferredSize), en fonction des composants qu'elle contient, puis de se redimensionner elle-même en adoptant cette taille. Cela a généralement pour effet de la rendre la plus petite possible tout en respectant la preferredSize des composants contenus.
- La méthode `pack` donne automatiquement à la fenêtre une taille minimale en préservant l'aspect de tous ses composants et ses sous-conteneurs.

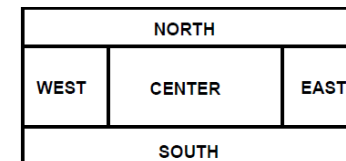


43

43

BorderLayout

- BorderLayout divise son espace de travail en cinq zones géographiques : NORTH, SOUTH, EAST, WEST et CENTER.
- Les composants sont ajoutés par nom à ces zones (un seul composant par zone).
- **Exemple:**
 - `add(new Button("Le bouton nord !"), BorderLayout.NORTH);`
 - Si une des zones de bordure ne contient rien, sa taille est 0.

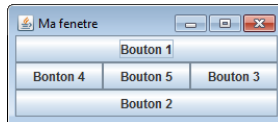


44

44

BorderLayout

- Stratégie de disposition du BorderLayout
- Les composants situés dans NORTH et SOUTH adoptent leur hauteur préférée et s'étalent sur toute la largeur du conteneur.
- Les composants situés dans EAST et WEST adoptent leur largeur préférée et s'étalent verticalement sur tout l'espace restant entre les zones NORTH et SOUTH.
- Le composant situé dans CENTER remplit tout l'espace restant.



45

45

BorderLayout

- **BorderLayout** est un gestionnaire de disposition utile pour les grands conteneurs d'une interface utilisateur.
- En imbriquant un panneau dans chaque zone du conteneur **BorderLayout**, puis en remplissant chacun de ces panneaux avec d'autres panneaux de diverses dispositions, on peut créer des interfaces utilisateur riches.

46

46

GridLayout

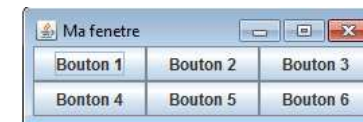
- **GridLayout** place les composants sur une grille de cellules en lignes et en colonnes.
- Il agrandit chaque composant de façon qu'il remplisse l'espace disponible dans la cellule.
- Toutes les cellules sont exactement de même taille et la grille est uniforme.
- Suite à un redimensionnement du conteneur, **GridLayout** change la taille des cellules de sorte qu'elles soient aussi grandes que possible étant donné l'espace disponible dans le conteneur.
- Construction d'un **GridLayout** : `new GridLayout(3, 2);`

47

47

GridLayout

- Le nombre de lignes ou de colonnes peut être nul, mais pas les deux à la fois. Il faut indiquer au moins une valeur pour que le **GridLayout** puisse calculer la seconde.
- **GridLayout** est utilisé pour concevoir un conteneur dont tous les composants doivent avoir la même taille, par exemple un pavé numérique ou une barre d'outils.



48

48

GridLayout

- Le nombre de pixels entre les cellules est spécifié en utilisant les méthodes `setHgap()` et `setVgap()`. L'espace horizontal et l'espace vertical sont nuls par défaut.
- Ces deux méthodes appartiennent à la classe `GridLayout`;



49

49

Exercice

- Ecrire un programme *JAVA* qui permet de calculer la moyenne de 2 entiers entrés comme paramètres de la classe, et l'afficher sur une fenêtre.



50

50

Exercice

```
package fentre2;
import javax.swing.*;
import java.awt.*;
public class Fentre2 extends JFrame {

    private JLabel l1,l2;

    public Fentre2(String t, String a, String b)
    {
        super(t);
        setLocation(500,270);
        setSize(200,70);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        l1=new JLabel("La moyenne est =");
```

51

51

Exercice

```
JPanel p= (JPanel) this.getContentPane();
FlowLayout fl=new FlowLayout(FlowLayout.LEFT);
p.setLayout(fl);

double x= Double.parseDouble(a);
double y= Double.parseDouble(b);
Double res=new Double((x+y)/2);
l2=new JLabel(res.toString());
String s1= "la moyenne de " + a + " et " + b;

l2.setToolTipText(s1);
l2.setForeground(Color.red);

p.setBackground(Color.WHITE);

p.add(l1);
p.add(l2);
}
```

52

52

Exercice

```
public static void main(String[] args) {
    Fenetre2 f=new Fenetre2("Fenetre 2", args[0],args[1]);

    f.show();
}
}
```

53

53

La gestion des événements

- Quand l'utilisateur effectue une action au niveau de l'interface graphique (clic souris, sélection d'un item, frappe d'une touche au clavier, etc), alors un événement est émis.
- Un événement est l'instance d'une classe XXXEvent du package java.awt.event.
- **Exemple:**
 - Suite à un clic souris dans une fenêtre, celle-ci envoie un événement de type MouseEvent.
 - Un bouton qui reçoit un clic de souris émet un événement instance de la classe ActionEvent.

54

54

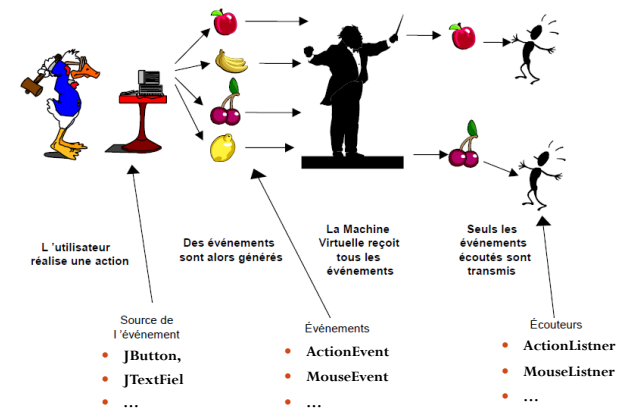
La gestion des événements

- Le composant émettant l'événement est la **source de l'événement**. Il transmet cet événement à un ou plusieurs autres objets, jouant le rôle d'**écouteur (Listener)**.
- Chaque composant **SWING** est conçu pour être la source d'un ou plusieurs types d'événements particuliers.
- Pour traiter un événement de type XXXEvent, un écouteur doit implémenter l'interface XXXListener
- Tout écouteur d'un composant doit s'inscrire auprès de ce composant grâce à la méthode **addXXXListener()**.
- Un composant peut être son propre écouteur.

55

55

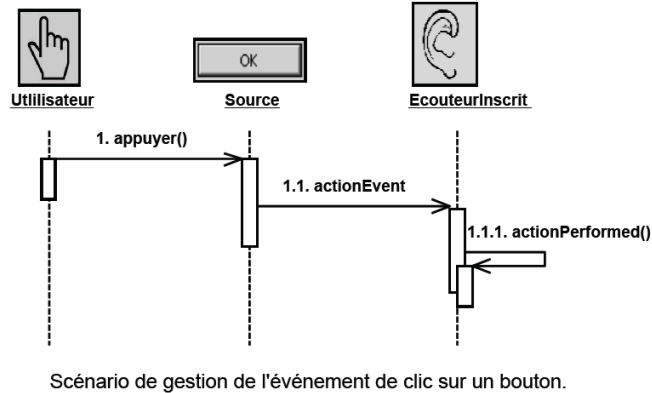
La gestion des événements



56

56

La gestion des événements



57

57

La gestion des événements

Exemple 1:

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Fenetre6 extends JFrame
implements ActionListener {
    private JButton b;
    private JTextField t= new JTextField();

    public Fenetre6() {
        JPanel c=(JPanel)this.getContentPane();
        c.setLayout(new BorderLayout());
        b = new JButton ("OK");
        c.add(b , BorderLayout.SOUTH);
        c.add(t, BorderLayout.NORTH);
        b.addActionListener(this);
    }
  
```

Fenetre6 est un écouteur de b. Elle doit donc implémenter ActionListener

Fenetre6 (this) s'inscrit comme écouteur de b.

58

La gestion des événements

Exemple 1:

```

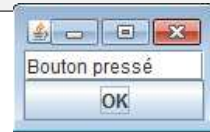
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Fenetre6 extends JFrame
implements ActionListener {
    private JButton b;
    private JTextField t= new JTextField();

    public Fenetre6() {
        JPanel c=(JPanel)this.getContentPane();
        c.setLayout(new BorderLayout());
        b = new JButton ("OK");
        c.add(b , BorderLayout.SOUTH);
        c.add(t, BorderLayout.NORTH);
        b.addActionListener(this);
    }

    public void actionPerformed (ActionEvent e)
    {
        t.setText("Bouton pressé");
    }

    public static void main (String args [])
    {
        Fenetre6 f = new Fenetre6();
        f.pack ();
        f.setVisible(true) ;
    }
  
```



implémenter

59

59

La gestion des événements

Exemple 2: Utilisation des classes internes

- Il s'agit d'une deuxième solution pour gérer les événements.
- On crée des **classes internes** dont les instances ne serviront qu'à écouter les événements.
- Il y a ainsi **une classe par composant** dédiée à la gestion des événements que ce composant peut générer.

60

60

La gestion des événements

```
package fenetre;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Fenetre7 extends JFrame {
    private JButton b1;
    private JTextField t;

    public Fenetre7 ()
    {
        setLocation(250,250);
        setTitle("Fenetre ");
        setSize(200,100);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel p=(JPanel)this.getContentPane();
        setLayout(new GridLayout(2,1));
        b1=new JButton("OK");
        t=new JTextField("Zonne de Text");
        b1.addActionListener(new OkListener());
        p.add(t);
        p.add(b1);
    }
}

class OkListener implements ActionListener {

    public void actionPerformed(ActionEvent e)
    {
        t.setText("Bouton OK pressé");
    }
}
```

Création d'une instance de la classe interne

61

Catégories d'événements

- Plusieurs types d'événements sont définis dans le package *java.awt.event*.
- Pour chaque catégorie d'événements, il existe une interface qui doit être définie par toute classe souhaitant écouter cette catégorie d'événements.
- Cette interface exige aussi qu'une ou plusieurs méthodes soient définies.
- Ces méthodes sont appelées lorsque les événements de cette catégorie surviennent.

62

Catégories d'événements

- **ActionListener**
 - Ecoute un clic sur un bouton, le retour chariot dans une zone de texte, etc.
 - Nécessite l'implémentation de *actionPerformed(ActionEvent e)*
- **TextListener**
 - Ecoute le changement de valeur dans une zone de texte
 - Nécessite l'implémentation de *textValueChanged(TextEvent e)*
- **ItemListener**
 - Ecoute la sélection d'un item dans une liste
 - Nécessite l'implémentation de *itemStateChanged(ItemEvent e)*
- **FocusListener**
 - Ecoute le gain ou la perte de focus par un composant.
 - Nécessite l'implémentation de *focusGained(FocusEvent e)* et *focusLost(FocusEvent e)*

63

63

La gestion des événements

- L'objet événement envoyé aux écouteurs et passé en paramètres des fonctions correspondantes peut contenir des paramètres intéressants pour l'application.
- Par exemple, *getX()* et *getY()* sur un *MouseEvent* retournent les coordonnées de la position du pointeur de la souris.
- L'une des informations généralement utile pour le traitement d'un événement est la source de cet événement. On obtient cette information en appelant la méthode *getSource()* pour cet événement.
- La méthode *Object getSource()* est présente dans toutes les classes événements

64

64

La gestion des événements

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

public class Fenetre8 extends JFrame
implements ActionListener {
    private JButton b1 = new JButton ("OK");
    private JButton b2 = new JButton ("Quitter");
    private JTextField t = new JTextField();

    public Fenetre8() {
        Container c=this.getContentPane();
        c.setLayout(new BorderLayout());
        c.add(b1 , BorderLayout.CENTER);
        c.add(b2 , BorderLayout.SOUTH);
        c.add(t, BorderLayout.NORTH);
        b1.addActionListener(this);
        b2.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e)
    {
        if ((JButton) e.getSource()==b1)
            t.setText("Bouton OK pressé");
        else
            t.setText("Bouton QUITTER
            pressé");
    }

    public static void main (String args [])
    {
        Fenetre8 f = new Fenetre8();
        f.pack ();
        f.setVisible(true) ;
    }
}
```

e.getSource() renvoie l'objet source de l'événement

Les 2 boutons ont le même écouteur (la fenêtre)

65

La gestion des événements


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

public class Fenetre8 extends JFrame
implements ActionListener {
    private JButton b1 = new JButton ("OK");
    private JButton b2 = new JButton ("Quitter");
    private JTextField t = new JTextField();

    public Fenetre8() {
        Container c=this.getContentPane();
        c.setLayout(new BorderLayout());
        c.add(b1 , BorderLayout.CENTER);
        c.add(b2 , BorderLayout.SOUTH);
        c.add(t, BorderLayout.NORTH);
        b1.addActionListener(this);
        b2.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e)
    {
        if ((JButton) e.getSource()==b1)
            t.setText("Bouton OK pressé");
        else
            t.setText("Bouton QUITTER
            pressé");
    }

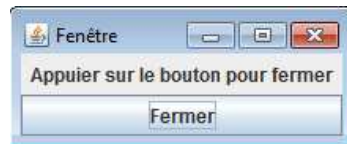
    public static void main (String args [])
    {
        Fenetre8 f = new Fenetre8();
        f.pack ();
        f.setVisible(true) ;
    }
}
```



66

Exercice

- Ecrire un programme qui permet d'afficher une fenêtre disposant d'un bouton. Quand on clique sur le bouton, on ferme la fenêtre et le programme s'arrête.



67

Solution

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

public class Fenetre9 extends JFrame implements ActionListener {
    private JLabel l1;
    private JButton b;

    public Fenetre9 ()
    {
        setLocation(250,250);
        setTitle("Fenêtre ");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) ;
        JPanel p=(JPanel)this.getContentPane();
        setLayout(new GridLayout(2,1));
        b=new JButton("Fermer");
        l1=new JLabel(" Appuyer sur le bouton pour fermer ");
        b.addActionListener(this);
        add(l1);
        add(b);
    }

    public void actionPerformed(ActionEvent e)
    {
        System.exit(0);
    }

    public static void main(String[] args) {
        Fenetre9 fen = new Fenetre9();
        fen.setVisible(true);
        fen.pack();
    }
}
```

68

Exercice

- Ecrire un programme qui permet d'afficher une fenêtre sur laquelle on voit affiché le nombre de clics effectués sur la fenêtre.



69

69

Exercice

Indication:



- Clic sur bouton, déplacement du pointeur.
- Méthodes de l'interface `MouseListener`
 - `public void mouseClicked(MouseEvent e)`
 - `public void mouseEntered(MouseEvent e)`
 - `public void mouseExited(MouseEvent e)`
 - `public void mousePressed(MouseEvent e)`
 - `public void mouseReleased(MouseEvent e)`
- Evénements générés par *Component*
- Méthode d'enregistrement `addMouseListener`

70

70

Solution

```
public class Fenetre10 extends JFrame implements MouseListener{
    private JLabel l1,l2;
    private static int nb=0;
    public Fenetre10 ()
    {
        setLocation(250,250);
        setTitle("Compteur ");
        setSize(300,70);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel p=(JPanel)this.getContentPane();
        setLayout(new FlowLayout(FlowLayout.LEFT));

        l1=new JLabel("Nombre de clics de la souris : ");
        l2=new JLabel("0");
        add(l1);
        add(l2);

        addMouseListener(this);
    }
}
```

71

71

Solution

```
public void mouseClicked(MouseEvent e)
{
    String s="";
    s+=" "+nb;
    l2.setText(s);
}
```

// Insertion des prototypes sans la définition de leurs corps.

```
public void mouseEntered(MouseEvent e){};
public void mouseExited(MouseEvent e){};
public void mousePressed(MouseEvent e){};
public void mouseReleased(MouseEvent e){};
```



72

72

Les adaptateurs

- Rappel: une classe qui implémente une interface doit redéfinir toutes les méthodes de l'interface
- Or certaines interfaces `Listener` ont beaucoup de méthodes, ce qui oblige à redéfinir des méthodes dont on n'a aucune utilité
- Pour remédier à cela Java propose des classes `Adapter` pour certaines interfaces `Listener`

73

73

Les adaptateurs

- Au lieu d'**implémenter l'interface** il suffira d'**hériter de la classe `Adapter`**.
- Dans ce cas le développeur **ne redéfinit que** les méthodes qui l'intéresse.
- Java propose ainsi un **ensemble** de classe `Adapter`
- Le principe de nommage est simple: une classe nommée **`XXXAdapter`** correspond à l'interface écouteur **`XXXListener`**

74

74

Les adaptateurs

- **`WindowAdapter`**
- **`MouseAdapter`**
- **`MouseMotionAdapter`**
- **`KeyAdapter`**
- **`FocusAdapter`**
- **`ComponentAdapter`**
- **`ContainerAdapter`**

75

75

Exercice

- **Solution 2: Créer une classe interne qui hérite de la classe `MouseAdapter`**

```
public class Fenetre11 extends JFrame {
    private JLabel l1,l2;
    private static int nb=0;
    public Fenetre11 ()
    {
        setLocation(250,250);
        setTitle(" Compteur ");
        setSize(300,70);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel p=(JPanel)this.getContentPane();
        p.setLayout(new FlowLayout(FlowLayout.LEFT));
        l1=new JLabel("Nombre de clics");
        l2=new JLabel("0");
        add(l1);
        add(l2);

        addMouseListener(new EcouterMouse());
    }
}

public class EcouterMouse extends MouseAdapter {
    public void mouseClicked(MouseEvent e)
    {
        String s="";
        s += ++nb;
        l2.setText(s);
    }
}
```

76

76

Exercice

- **Solution 3: Créer une classe anonyme qui hérite de `MouseListener`**

```
public class Fenetre12 extends JFrame{
    .....
    l1=new JLabel("Nombre de clics de la souris : ");
    l2=new JLabel("0");
    add(l1);
    add(l2);

    addMouseListener(new MouseAdapter() {public void
mouseClicked(MouseEvent)
    {
        String s=" ";
        s+=++nb;
        l2.setText(s);
    }
    }
    ...
}
```

77

77

Exercice

- Ecrire un programme qui permet d'afficher une fenêtre sur laquelle se trouvent 2 champs.
 - Le premier est précédé d'une étiquette Euros
 - Le deuxième d'une étiquette Dinars.
- Lorsque l'utilisateur entre un nombre dans le premier champ, sa conversion en Dinars apparaît simultanément dans le deuxième champ.



78

78

Solution

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.DecimalFormat;

public class Fenetre13 extends JFrame{
    private JLabel l1,l2,l3;
    private JTextField t1,t2;
    public Fenetre13 ()
    {
        setLocation(250,250);
        setTitle("Convertisseur ");
        setSize(300,60);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel p=(JPanel)this.getContentPane();
        setLayout(new GridLayout(1,5));
        l1=new JLabel("Euros");
        l2=new JLabel(" = ");
        l3=new JLabel("Dinars");
        t1=new JTextField();
        t2=new JTextField();
        add(l1);    add(t1);    add(l2);    add(l3);    add(t2);
    }
}
```

79

79

Solution

```
t1.addActionListener(new ActionListener() {public void actionPerformed(ActionEvent
e){
    double d =Double.parseDouble(t1.getText());
    String s="";
    d=d*2.2;
    DecimalFormat df = new DecimalFormat("###.###");

    s+=d;
    t2.setText(df.format(d));
    }});

    }
    public static void main(String[] args) {
        Fenetre13 fen = new Fenetre13();
        fen.setVisible(true);
    }
}
```

80

80

Des méthodes utiles de la classe Component

- Les méthodes suivantes de la classe Component permettent de gérer l'aspect d'un composant (y compris les fenêtres) :
 - public void setVisible (boolean b)** : Montrer et cacher un composant
 - public void setEnabled (boolean b)** : Activer et désactiver un composant
 - public boolean isEnabled ()** : Connaître l'état (activé ou non) d'un composant
 - public void setBackground (Color c)** : Modifier la couleur de fond d'un composant
 - public void setSize (int largeur , int hauteur)** : Modifier la taille d'un composant
 - public void setBounds (int x , int y , int largeur , int hauteur)** : Modifier la position et la taille d'un composant
 - public Dimension getSize ()** : retourner les dimensions d'un composant. La classe Dimension du paquetage java.awt contient deux champs publics : un champ height (hauteur) et un champ width (largeur).

81

81

Le Composant JButton

- Un **JButton** est un bouton qui peut contenir un intitulé de texte, une icône, ou une combinaison des deux.
- Parmi les constructeurs disponibles :
 - JButton (String text)**
 - JButton (Icon icône)**
 - JButton (String text , Icon icône)**
- Les événements envoyés par un bouton :
 - java.awt.event.ActionEvent** : est généré lorsqu'un bouton est pressé.
 - java.awt.event.ChangeEvent** : est généré quand l'état interne du bouton change, par exemple, quand le pointeur de la souris arrive sur le bouton.

82

82

Le JCheckBox

- Un **JCheckBox** est une case à cocher. Il permet à l'utilisateur d'effectuer un choix de type oui/non.
- Parmi les constructeurs disponibles :
 - JCheckBox (String texte)**
 - JCheckBox (Icon icône)**
 - JCheckBox (String texte , boolean sélectionné)**
- Parmi les méthodes disponibles :
 - public void setSelected (boolean b)**
 - public boolean isSelected ()**
- Événement envoyé par une case à cocher :
 - Java.awt.ItemEvent**, envoyé lorsque la case passe de "cochée" à "décochée" ou inversement.

83

83

Le JCheckBox

- Exemple:

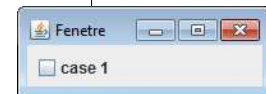
```
public class Fenetre14 extends JFrame implements ItemListener {

    private JCheckBox c;

    public Fenetre14()
    {
        .....
        JPanel p=(JPanel) getContentPane();
        p.setLayout(new FlowLayout(FlowLayout.LEFT));
        c=new JCheckBox("case 1");
        p.add(c);
        c.addItemListener(this);
    }

    public void itemStateChanged ( ItemEvent e )
    {
        int etat = e.getStateChange ( ) ;

        if ( etat == 1 )
            System.out.println( "La case est cochée" ) ;
        else
            System.out.println( "La case est décochée" ) ;
    }
}
```



84

84

Le JRadioButton

- Un **JRadioButton** est un bouton radio. Il permet à l'utilisateur d'effectuer un choix de type oui/non.
- Parmi les constructeurs disponibles :
 - JRadioButton (String texte)
 - JRadioButton (Icon icône)
 - JRadioButton(String texte , boolean sélectionné)
- Parmi les méthodes disponibles :
 - public void setSelected(boolean b)
 - public boolean isSelected ()
- Les boutons radio ne sont pas exclusifs. Pour les rendre exclusifs, il faut les grouper dans un **ButtonGroup**.

85

85

Le JRadioButton

- Événement envoyé par un JRadioButton :
- Java.awt.**ItemEvent** envoyé lorsque l'état du bouton radio change de "sélectionné" à "désélectionné" et inversement.

86

86

Le JRadioButton

Exemple:

```
public class Fenetre15 extends JFrame implements
ItemListener {
    private ButtonGroup grp ;
    private JRadioButton r1, r2, r3 ;
    public Fenetre15()
    {
        .....
        grp = new ButtonGroup ( ) ;
        r1 = new JRadioButton ( "Java" ) ;
        r2 = new JRadioButton ( "C#" ) ;
        r3 = new JRadioButton ( "C++" ) ;
        grp.add(r1); grp.add(r2); grp.add(r3);
        p.add(r1); p.add(r2); p.add(r3);
        r1.addItemListener(this);
        r2.addItemListener(this);
        r3.addItemListener(this);
    }

    public void itemStateChanged ( ItemEvent e ) {
        JRadioButton r = ( JRadioButton ) e.getItemSelectable ( ) ;
        String option = r.getText ( ) ;
        System.out.println( "L'option "+option+" est l'origine de
l'événement" ) ;

        boolean etat = r.isSelected ( ) ;
        if ( etat ) {
            System.out.println( "L'option "+option+" est
sélectionnée" ) ; }
        else { System.out.println( "L'option " + option + " n'est
plus sélectionnée" ) ; }
    }
}
```



87

Le JLabel

- Les JLabel ou les étiquettes (ou libellés) sont des composants qui contiennent du texte. Ils ne possèdent pas d'ornements, telle une bordure, et ne réagissent pas aux entrées de l'utilisateur (pas d'événements envoyés).
- Une étiquette permet d'identifier des composants qui n'ont pas de libellé, comme les champs de saisie, les listes, etc..
- Parmi les constructeurs disponibles :
 - JLabel (String texte)
 - JLabel (String texte , int alignementHorizontal)
 - L'alignement peut avoir l'une des valeurs : SwingConstants.LEFT et SwingConstants.RIGHT.

88

88

Le JLabel

- Parmi les méthodes disponibles :
 - `public void setHorizontalAlignment (int alignement)` : change l'alignement de l'étiquette.
 - `public void setText (String text)` : modifie le texte de l'étiquette.
 - `public String getText ()` : renvoie le texte de l'étiquette.

89

89

Le JLabel

- Exemple:

```
public class Fenetre16 extends JFrame {
    private JLabel l1,l2;
    public Fenetre16()
    {
        .....
        JPanel p=(JPanel)getContentPane();
        p.setLayout(new BorderLayout());
        l1=new JLabel("label à droite",SwingConstants.RIGHT);
        l2=new JLabel("label à gauche",SwingConstants.LEFT);
        p.add(l1,BorderLayout.NORTH);
        p.add(l2,BorderLayout.SOUTH);
    }
}
```



90

90

Le JTextField

- Un JTextField permet à l'utilisateur d'entrer et d'éditer une ligne unique de texte simple.
- Parmi les constructeurs disponibles :
 - `JTextField (String text)`
 - `JTextField (int colonnes)` : le nombre de colonnes ne limite pas le nombre de caractères que l'utilisateur peut taper. Il peut introduire des chaînes plus longues ; toutefois, la vue de l'entrée défile lorsque le texte dépasse la longueur du champ souhaitée.
 - `JTextField (String text, int colonnes)`

91

91

Le JTextField

- Parmi les méthodes disponibles :
 - `public void setText (String text)` : modifie le contenu du champ de texte
 - `public void setColumns (int colonnes)` : donne le nombre de caractères dans le champ. Si le gestionnaire de mise en forme a besoin d'agrandir ou de réduire le champ de texte, il peut ajuster la taille.
 - `public String getText ()` : renvoie tout ce que l'utilisateur a entré dans le champ, y compris les espaces en tête et en fin de chaîne.
 - `public void setFont()` : permet de spécifier la fonte dans laquelle le texte est affiché.
- Événement envoyés par un JTextField :
 - `java.awt.ActionEvent` envoyé aux écouteurs de type `ActionListener` quand l'utilisateur tape sur la touche "Entrée" du clavier.

92

92

Le JTextField

- Le champ de mot de passe *JPasswordField* est un type spécial de champ de texte (sous-classe de *JTextField*).
- Il est conçu pour saisir des mots de passe.
- Les caractères tapés ne sont pas affichés dans ce champ. Un caractère d'écho est utilisé à la place.
- la méthode *setEchoChar()* permet de choisir le caractère d'écho à faire apparaître au lieu des caractères entrés par l'utilisateur.
- getPassword()* permet de récupérer le mot de passe dans un tableau de caractères. *getText()* aussi permet de récupérer le mot de passe).

93

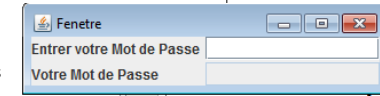
93

Le JTextField

Exemple:

```
public class Fenetre17 extends JFrame implements ActionListener {
    private JLabel l1,l2; private JTextField t1; private JPasswordField t2;

    public Fenetre17()
    {
        .....
        JPanel p=(JPanel) getContentPane();
        p.setLayout(new BorderLayout());
        JPanel p1=new JPanel();
        JPanel p2=new JPanel();
        p1.setLayout(new GridLayout(1,2));
        p2.setLayout(new GridLayout(1,2));
        l1=new JLabel("Votre Mot de Passe");
        t1=new JTextField();
        t1.setEditable(false);
        p1.add(l1);p1.add(t1);
        l2=new JLabel("Entrez votre Mot de Passe");
        t2=new JPasswordField("",50);
        p2.add(l2);p2.add(t2);
        t2.addActionListener(this);
        p.add(p2,BorderLayout.NORTH);p.add(p1,BorderLayout.SOUTH);
    }
}
```



```
public void actionPerformed(ActionEvent e)
{
    t1.setText(t2.getText());
}
```

94

Le JFormattedTextField

- La classe *JFormattedTextField* possède un constructeur prenant en paramètre un *Format*. Cela permet de spécifier un format au champ de saisie.
- Classes dérivées de *Format* :
 - DateFormat*
 - MessageFormat*
 - NumberFormat*
- Exemple


```
// champ de saisie de pourcentages (importer le package java.text)
JFormattedTextField cp =new
JFormattedTextField(NumberFormat.getPercentInstance());
```

95

95

Le JFormattedTextField

```
//champ de saisie de dates
JFormattedTextField champDate = new
JFormattedTextField(DateFormat.getDateInstance());
// Dec 31, 2000
JFormattedTextField(DateFormat.getDateInstance(DateFormat.SHORT));
// 12/31/2000
```

- Un *MaskFormatter* convient aux motifs à taille fixe (n° de téléphone, n° de sécurité sociale, n° de série...).
- Liste des symboles :
 - # => Un chiffre
 - A => Un chiffre ou une lettre
 - ? => Une lettre
 - U => Une lettre (les minuscules sont changées en majuscules)
 - * => Tout caractère
 - L => Une lettre (les majuscules sont changées en minuscules)
 - H => Tout caractère hexadécimal (0-9, a-f ou A-F)

96

96

Le JFormattedTextField

- Exemple : Pour un n° de tél. en Tunisie
 - `new MaskFormatter("## ## ## ## ##");`
(Importer le package `javax.text`)
- On peut limiter les caractères valides en appelant l'une des méthodes suivantes : `setValidCharacters()`, `setInvalidCharacters()`.

- Exemple :

```
// Champ de saisie d'une case d'un échiquier (A1 à H8)
try {
    MaskFormatter mask = new MaskFormatter("U#");
    mask.setValidCharacters("ABCDEFGH12345678");
    JFormattedTextField case = new JFormattedTextField(mask);
}
catch (ParseException e) { }
```

97

97

Le JList

- La boîte de liste de type **JList** permet de choisir une ou plusieurs valeurs dans une liste prédéfinie. Initialement, aucune valeur n'est sélectionnée dans la liste.
- Parmi les constructeurs disponibles :
 - `JList ()`
 - `JList (Object [] liste)`
 - `JList(Vector<?> liste)`

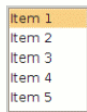
98

98

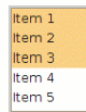
Le JList

- Il existe trois modes de sélection pour une boîte de liste. est donc nécessaire, une fois que l'objet `JList` est construit, de choisir le mode de sélection au travers de la méthode `setSelectionMode()` :

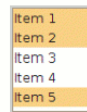
Valeur du paramètre de type	Type de sélection
<code>static int ListSelectionModel.SINGLE_SELECTION</code>	une seule valeur (a)
<code>static int ListSelectionModel.SINGLE_INTERVAL_SELECTION</code>	une plage de valeurs (b)
<code>static int ListSelectionModel.MULTIPLE_INTERVAL_SELECTION</code> (Valeur par défaut)	sans restriction (c)



(a)



(b)



(c)

99

99

Le JList

- Par défaut, une boîte de liste affiche tous les éléments présents dans la liste dans la mesure de la capacité du conteneur.
- On peut modifier ce comportement initial pour que la visualisation de la liste corresponde à l'apparence souhaitée :
 - en mettant la liste dans un panneau de défilement `JScrollPane` :
`JList nl=new JList (new Integer[] { 100,200,300,400,500,700,800,900 });`
`JScrollPane jsp = new JScrollPane(nl);`
 - en définissant le nombre d'éléments à afficher :
`nl.setVisibleRowCount(3);`
 - en proposant une couleur particulière pour la sélection :
`nl.setSelectionBackground(Color.black);`
`nl.setSelectionForeground(Color.red);`



100

100

Le JList

- Méthodes disponibles pour récupérer les valeurs sélectionnées :
 - Liste à sélection simple :
 - `public Object getSelectedValue()`. Elle fournit le (seul) objet sélectionné. Il faudra parfois procéder à une conversion explicite pour utiliser cet objet.
 - Liste à sélections multiples
 - `public Object [] getSelectedValues ()`
 - Il est possible de récupérer la position des objets sélectionnés à la place de leurs valeurs :
 - `public int getSelectedIndex ()`
 - `public int [] getSelectedIndices ()`

101

101

Le JList

- Evénement envoyé par un JList :
 - `Javax.swing.event.ListSelectionEvent` envoyé lorsqu'un élément de la liste est sélectionné.
 - Les écouteurs de cet événement doivent implémenter l'interface `Javax.swing.event.ListSelectionListener` qui contient une seule méthode:
 - `public void valueChanged()`.

102

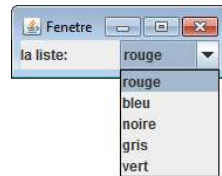
102

Le JComboBox

- Un **JComboBox** est une boîte de liste combinée (boîte combo). Elle associe un champ de texte et une boîte de liste à sélection simple. Tant que le composant n'est pas sélectionné, seul le champ de texte s'affiche.



- Lorsque l'utilisateur sélectionne le champ de texte, la boîte de liste s'affiche
- Parmi les constructeurs :
 - `JComboBox ()`
 - `JComboBox (Object [] éléments)`
 - `JComboBox (Vector<?> éléments)`



103

103

Le JComboBox

- Il est possible d'insérer ou de supprimer de nouveaux éléments à une liste déroulante :
 - `addItem(Object e)` : ajouter un nouvel élément à la fin de la liste
 - `insertItemAt(Object e, int pos)` : insère un nouvel élément à un endroit spécifique.
 - `removeAllItems()` : supprimer tous les éléments
 - `removeItem(Object e)` : supprimer un élément spécifique
 - `removeItemAt(int pos)` : supprime un élément en spécifiant sa position.
 - `getItemCount()` : permet de connaître le nombre de rubriques déjà présentes.

104

104

Le JComboBox

- Il est possible de *forcer la sélection* :
 - d'un élément de rang donné par `setSelectedIndex(int)`
 - d'une valeur particulière au travers de `setSelectedItem(Object)`
- Pour récupérer l'élément sélectionné :
 - `public Object getItem()` : fournit la valeur sélectionnée, qu'il s'agisse d'une valeur prédéfinie ou saisie dans le champ de texte associé.
 - `public int getSelectedIndex()` fournit aussi le rang de la valeur sélectionnée (-1 lorsque l'utilisateur entre un nouvel élément).
- Par défaut un JComboBox n'est pas éditable.
 - La méthode `setEditable()` permet de le rendre modifiable.
 - La saisie d'un élément ne modifie que l'élément courant et ne change pas le contenu de la liste.

105

105

Le JComboBox

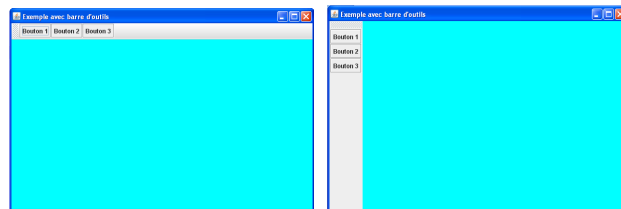
- Événement envoyé par un JComboBox:
 - `Java.event.ActionEvent` lors d'une sélection d'une valeur dans la liste ou lors de la validation du champ de texte (lorsqu'il est éditable).
 - `Java.event.ItemEvent` à chaque modification de la sélection

106

106

La barre d'outils (JToolBar)

- Une **JToolBar** peut être placée sur les bords (et **pas au centre!**) d'un container muni d'un **BorderLayout**
- Ne rien mettre d'autre qu'un composant (JPanel par exemple) au centre et la **JToolBar**

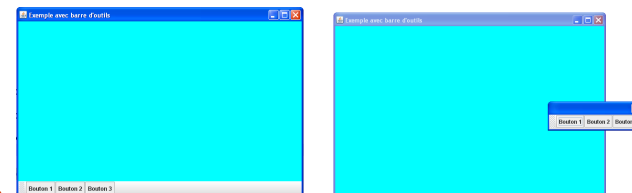


107

107

La barre d'outils (JToolBar)

- Utilise un `BoxLayout`
- Peut se détacher si elle est floatable (`setFloatable(boolean b);`)
- Quand on ferme une JToolBar flottante, elle retourne à sa dernière position.



108

108

La barre d'outils (JToolBar)

```
import java.awt.*;
import javax.swing.*;

public class TestBarreOutils {
    public static void main(String[] args) {
        JFrame f = new JFrame("Exemple avec barre d'outils");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setPreferredSize(new Dimension(600, 400));
        f.setLocation(100, 100);
        JPanel p = (JPanel) f.getContentPane();

        JPanel pCentre = new JPanel();
        pCentre.setBackground(Color.CYAN);
        p.add(pCentre, BorderLayout.CENTER);
    }
}
```

109

109

La barre d'outils (JToolBar)

```
JToolBar barreOutils = new JToolBar(JToolBar.HORIZONTAL);
//barreOutils.setFloatable(false);
JButton b1 = new JButton("Bouton 1");
barreOutils.add(b1);
JButton b2 = new JButton("Bouton 2");
barreOutils.add(b2);
JButton b3 = new JButton("Bouton 3");
barreOutils.add(b3);

p.add(barreOutils, BorderLayout.NORTH);

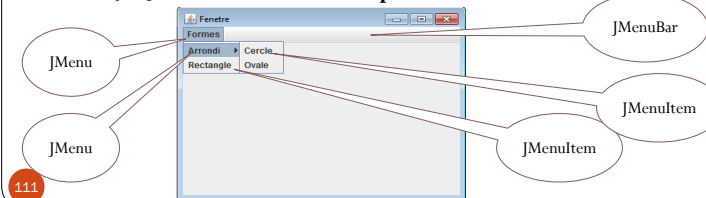
f.pack();
f.setVisible(true);
}
```

110

110

Le JMenuBar, le JMenu et le JMenuItem

- Un **JMenu** est un menu déroulant standard dont le nom est fixé. Les menus peuvent contenir d'autres menus en (sous-menus), ce qui permet de mettre en place des structures de menus complexes.
- Un **JMenuBar** est un composant qui héberge des menus dans une barre horizontale. Les barres de menu peuvent être placées à tout endroit d'un conteneur
- Un objet **JMenuItem** est une option de menu.



111

111

Le JMenuBar, le JMenu et le JMenuItem

- La méthode `setEnabled(boolean)` de la classe **JComponent** permet d'activer ou de désactiver un **menu** ou une **option de menu**.
- **Les étapes de création de menu :**
 - Création d'une barre de menu instance de **JMenuBar**
 - Adjonction de la barre à la fenêtre directement (**et non au contenu**).
 - Création des menus instances de **JMenu**
 - Création des options de chaque menu instances de **JMenuItem**
 - Attacher une action à chaque option de menu

112

112

Le JMenuBar, le JMenu et le JMenuItem

- Constructeur de **JMenuBar** :
 - `JMenuBar ()`
- Constructeur de **JMenu** :
 - `JMenu(String s)`
- Constructeurs de **JMenuItem** :
 - `JMenuItem (Icon icône)`
 - `JMenuItem (String texte)`
 - `JMenuItem (String texte , Icon icône)`
 - `JMenuItem (String texte , int mnémonique)`

113

113

Le JMenuBar, le JMenu et le JMenuItem

- Parmi les méthodes de **JMenuBar** disponibles:
 - `public JMenu add (JMenu c)`
 - `public JMenu getMenu (int index)`
 - `public int getMenuCount ()`
- Parmi les méthodes de **JMenu** disponibles:
 - `public JMenuItem add (JMenuItem menuItem)`
 - `public JMenuItem add (String s)`
 - `public void addSeparator ()`
 - `public JMenuItem getItem (int pos)`
 - `public void remove (JMenuItem item)`
- Parmi les méthodes de **JMenuItem** disponibles :
 - `public void setEnabled (boolean b)`
 - `public boolean isEnabled ()`

114

114

Le JMenuBar, le JMenu et le JMenuItem

- Événements envoyés:
 - La sélection d'une option (JMenuItem) génère un événement de type `java.event.ActionEvent`
 - On peut recourir à la méthode `getActionCommand()` de la classe `ActionEvent`, qui comme pour un bouton, fournit la chaîne de commande associée à l'option. Par défaut, il s'agit tout simplement du nom de l'option (fourni au constructeur de JMenuItem).

115

115

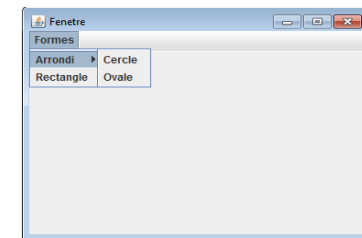
Exemple:

```
public class Fenetre13 extends JFrame {
    JMenuBar barreMenus = new JMenuBar ( );
    JMenu formes = new JMenu ( "Formes" );
    JMenu arrondi = new JMenu ( "Arrondi" );
    JMenuItem cercle = new JMenuItem ( "Cercle" );
    JMenuItem ovale = new JMenuItem ( "Ovale" );
    JMenuItem rectangle = new JMenuItem ( "Rectangle" );
    public Fenetre13 ( ) {
        ...
        setJMenuBar(barreMenus);
        barreMenus.add ( formes );
        formes.add ( arrondi );
        formes.add ( rectangle );

        arrondi.add ( cercle );
        arrondi.add ( ovale );
    }
}
```

116

116



Le JMenuBar, le JMenu et le JMenuItem

- Les **JMenu** peuvent contenir:
 - des boutons spéciaux: **JMenuItem**,
 - des sous-menus de type **JMenu**
 - JRadioButtonMenuItem**,
 - JCheckBoxMenuItem**
 - des séparateurs: **addSeparator()**

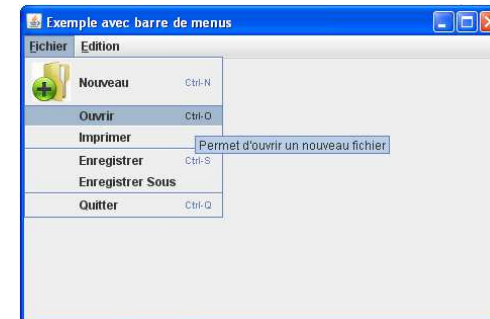


117

117

Le JMenuBar, le JMenu et le JMenuItem

- 3 modes de sélection:
 - clic de souris
 - validation clavier avec Entrée
 - mnémonique (caractère souligné) ou bien raccourci clavier



118

118

Le JMenuBar, le JMenu et le JMenuItem

- pour installer un Mnémonique :
 - `Fichier.setMnemonic('F')`
- pour créer un raccourci clavier :

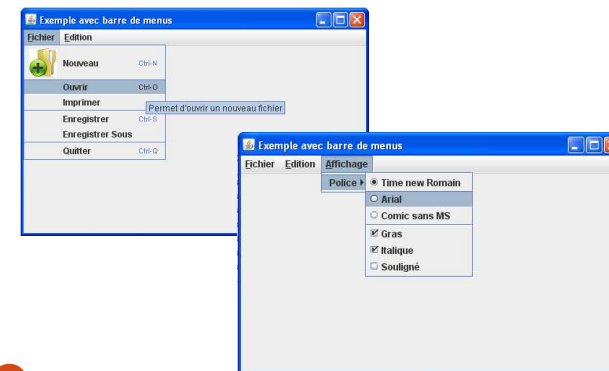

```
miNouveau.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, Event.CTRL_MASK));
```
- Icône: `Icon img = new ImageIcon("new.jpg");`
`miNouveau.setIcon(img);`
 - Il faut parfois ajouter l'URL de l'icône pour qu'elle apparaisse.
- Bulle d'information: `miOuvrir.setToolTipText("Permet d'ouvrir un nouveau fichier");`

119

119

Exercice

- Implémenter l'interface suivante:



120

120

Solution

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

public class TestMenus extends JFrame implements ActionListener
{
    public TestMenus() {
        super("Exemple avec barre de menus");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setPreferredSize(new Dimension(600, 400));
        setLocation(100, 100);
        JPanel p = (JPanel) getContentPane();
```

121

121

Solution

```
JMenuBar mb = new JMenuBar();
JMenu mFichier = new JMenu("Fichier");
mFichier.setMnemonic('F');
JMenuItem miNouveau = new JMenuItem("Nouveau");

miNouveau.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N
, Event.CTRL_MASK));
Icon img = new ImageIcon("icomes/new.jpg");
miNouveau.setIcon(img);
mFichier.add(miNouveau);

JMenuItem miOuvrir = new JMenuItem("Ouvrir");

miOuvrir.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
Event.CTRL_MASK));
miOuvrir.setToolTipText("Permet d'ouvrir un nouveau
fichier");
```

122

122

Solution

```
miOuvrir.addActionListener(this);
mFichier.add(miOuvrir);

JMenuItem miImprimer = new JMenuItem("Imprimer");
mFichier.add(miImprimer);

mFichier.add(new JSeparator(JSeparator.VERTICAL));
//mFichier.addSeparator();

JMenuItem miEnregistrer = new JMenuItem("Enregistrer");
miEnregistrer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
Event.CTRL_MASK));
miEnregistrer.addActionListener(this);
mFichier.add(miEnregistrer);
```

123

123

Solution

```
JMenuItem miEnregistrerSous = new
JMenuItem("Enregistrer Sous");
mFichier.add(miEnregistrerSous);
mFichier.add(new JSeparator(JSeparator.HORIZONTAL));
JMenuItem miQuitter = new JMenuItem("Quitter");

miQuitter.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q
, Event.CTRL_MASK));
miQuitter.addActionListener(this);
mFichier.add(miQuitter);

mb.add(mFichier);
JMenu mEdition = new JMenu("Edition");
mEdition.setMnemonic('E');

mb.add(mEdition);
```

124

124

Solution

```
JMenu mAffichage = new JMenu("Affichage");
mAffichage.setMnemonic('A');
JMenu mPolice = new JMenu("Police");
ButtonGroup bg = new ButtonGroup();
JRadioButtonMenuItem police1 = new JRadioButtonMenuItem("Time
new Romain");
JRadioButtonMenuItem police2 = new
JRadioButtonMenuItem("Arial");
JRadioButtonMenuItem police3 = new JRadioButtonMenuItem("Comic
sans MS");
bg.add(police1);
bg.add(police2);
bg.add(police3);
mPolice.add(police1);
mPolice.add(police2);
mPolice.add(police3);
```

125

125

Solution

```
mPolice.add(new JSeparator(JSeparator.HORIZONTAL));

JCheckBoxMenuItem font1 = new JCheckBoxMenuItem("Gras");
JCheckBoxMenuItem font2 = new JCheckBoxMenuItem("Italique");
JCheckBoxMenuItem font3 = new JCheckBoxMenuItem("Souligné");
mPolice.add(font1);
mPolice.add(font2);
mPolice.add(font3);

mAffichage.add(mPolice);
mb.add(mAffichage);
setJMenuBar(mb);
```

126

126

Les boîtes de dialogue

- Une boîte de dialogue est un conteneur. Elle permet de regrouper n'importe quels composants dans une sorte de fenêtre qu'on fait apparaître ou disparaître.
- Java propose un certain nombre de boîtes de dialogue standard obtenues à l'aide de méthodes de la classe *JOptionPane* : boîtes de message, boîtes de confirmation, boîtes de saisie et boîtes d'options.
 - La classe **JDialog** permet de construire des boîtes de dialogue personnalisées.

127

127

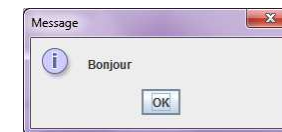
Les boîtes de dialogue

- Les boîtes de message
 - Une boîte de message fournit à l'utilisateur un message qui reste affiché tant que l'utilisateur n'agit pas sur le bouton OK. Elle est construite à l'aide de la méthode de classe


```
static void showMessageDialog ( parent, message )
```

 de la classe *JOptionPane*.
 - Exemple


```
JOptionPane.showMessageDialog ( null , "Bonjour" );
```







128

128

Les boîtes de dialogue

- Il existe une variante de la méthode `showMessageDialog()` qui permet de choisir le titre de la boîte et le type d'icône parmi la le éléments du tableau suivant (les paramètres sont des constantes entières de la classe `JOptionPane`).

Paramètre	Type d'icône
<code>static int JOptionPane.ERROR_MESSAGE</code>	Erreur 
<code>static int JOptionPane.INFORMATION_MESSAGE</code>	Information 
<code>static int JOptionPane.WARNING_MESSAGE</code>	Avertissement 
<code>static int JOptionPane.QUESTION_MESSAGE</code>	Question 
<code>static int JOptionPane.PLAIN_MESSAGE</code>	Aucune icône

129

129

Les boîtes de dialogue

- Exemple
 - `JOptionPane.showMessageDialog(fenetreParent, contenu, titre, type_icone);`
 - `JOptionPane.showMessageDialog(null, " avertissement " , « titre de la boîte", JOptionPane.WARNING_MESSAGE);`



130

130

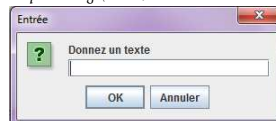
Les boîtes de dialogue

- Les boîtes de saisie
 - Une boîte de saisie permet à l'utilisateur de fournir une information sous la forme d'une chaîne de caractères.
 - Elle est construite à l'aide de la méthode de classe

`Static String showInputDialog (parent , message)`

de la classe `JOptionPane`.

- Exemple
 - `JOptionPane.showInputDialog (null , "Donnez un texte ");`



- La valeur de retour de la méthode `showInputDialog` est soit un objet de type **String** contenant le texte fournit par l'utilisateur, soit la valeur null si l'utilisateur n'a pas confirmé sa saisie par le bouton OK.

131

131

Les boîtes de dialogue

- Il existe une variante de la méthode `showInputDialog` qui permet aussi de choisir le titre de la boîte et le type d'icône.
- Exemple
 - `JOptionPane.showInputDialog (null, "Entrez votre nom", "Contrôle d'identité ", JOptionPane.WARNING_MESSAGE);`



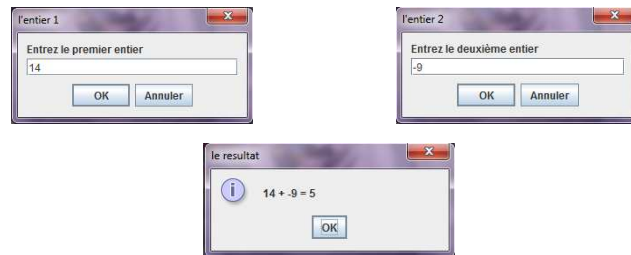
132

132

Les boîtes de dialogue

• Exercice:

- Ecrire un programme permettant de demander à l'utilisateur de donner deux entiers et lui afficher leur somme par la suite.
- **NB:** Utiliser les boîtes de dialogues pour répondre à l'exercice.



133

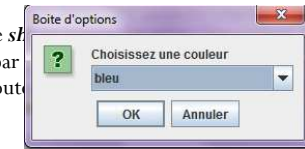
133

Les boîtes de dialogue

• Les boîtes d'options :

- Une boîte d'options permet à l'utilisateur de choisir une valeur parmi une liste de valeurs, par l'intermédiaire d'une boîte combo. Elle est construite à l'aide de la méthode de classe
 - `static Object showInputDialog (arg1,arg2,arg3,arg4,arg5,arg6,arg7)` de la classe `JOptionPane`.

- La valeur de retour de la méthode `showInputDialog` contenant la valeur sélectionnée par l'utilisateur, si l'utilisateur n'a pas confirmé sa saisie par le bouton "Annuler", la méthode retourne `null`.



• Exemple

```
String[] couleurs = { "rouge", "bleu", "gris", "vert", "jaune", "noir" };
JOptionPane.showInputDialog(null, "Choisissez une couleur", "Boîte d'options",
    JOptionPane.QUESTION_MESSAGE, null, couleurs, couleurs [1] );
```

134

134

Les boîtes de dialogue

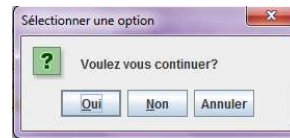
• Les boîtes de confirmation

- Une boîte de confirmation offre à l'utilisateur un choix de type oui/non/annuler.
- Elle est construite à l'aide de la méthode de classe :

`static int showConfirmDialog (fenetreParent, message)`

de la classe `JOptionPane`.

```
JOptionPane.showConfirmDialog ( null , "Voulez-vous continuer ?" );
```



135

135

Les boîtes de dialogue

- Il existe une variante de la méthode `showConfirmDialog()` qui permet de choisir le titre de la boîte et la nature des boutons qui s'y trouvent parmi les éléments du tableau suivant (les paramètres sont des constantes entières de la classe `JOptionPane`).

Paramètre	Type de boîte
<code>JOptionPane.DEFAULT_OPTION</code>	Erreur
<code>JOptionPane.YES_NO_OPTION</code>	boutons YES et NO
<code>JOptionPane.YES_NO_CANCEL_OPTION</code>	boutons YES, NO et CANCEL
<code>JOptionPane.OK_CANCEL_OPTION</code>	boutons OK et CANCEL

136

136

Les boîtes de dialogue

- Exemple

- `int n = JOptionPane.showConfirmDialog (null, "Voulez-vous continuer ?", "Incident majeur", JOptionPane.YES_NO_OPTION) ;`



- La valeur de retour de la méthode `showConfirmDialog` précise l'action effectuée par l'utilisateur sous la forme d'un entier ayant l'une des constantes suivantes :
 - `JOptionPane.YES_OPTION (0)`,
 - `JOptionPane.OK_OPTION (0)`,
 - `JOptionPane.NO_OPTION (1)`,
 - `JOptionPane.CANCEL_OPTION (2)`,
 - `JOptionPane.CLOSED_OPTION (-1)`.

137