

Chapitre 5

Interfaces graphiques

Les APIs pour les GUIs

- AWT (Abstract Windows Toolkit)
 - JDK 1.1
 - classes Frame, Panel, Button, Label, ...
- Swing
 - JDK 1.2
 - classes JFrame, JPanel, JButton, JLabel, ...

Plan

- AWT et Swing
- Créer un cadre
- Premiers composants
- Boîtes de dialogue
- Bordures
- Layout managers
- Programmation événementielle
- Types d'événements
- Autres composants Swing
- Jeu de la vie

Swing ou AWT

- Swing est construit au dessus de AWT
- Swing est plus riche que AWT
- AWT utilise des composants lourds, i.e., des composants utilisant des ressources systèmes.
- Swing utilise des composants légers sauf pour les containers primaires.
- Swing est plus lent que AWT.

Interface graphique

- Une interface graphique est formée de fenêtres contenant divers composants graphiques tels que :
 - boutons
 - champs texte
 - listes déroulantes
 - menus
 - ...
- Les interfaces graphiques sont souvent appelées GUI (Graphical User Interface).

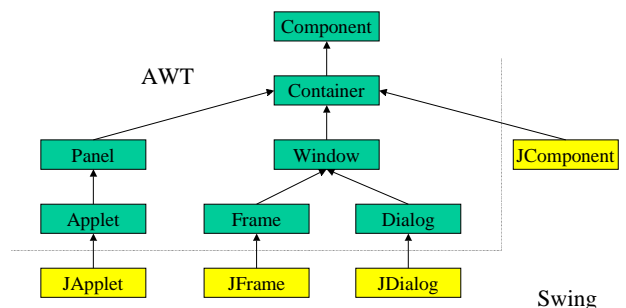
Packages

- API AWT
 - java.awt
 - java.awt.event
 - autres sous-packages de java.awt.
- API Swing
 - javax.swing
 - javax.swing.event
 - autres sous-packages de javax.swing.

AWT et Swing



Hierarchie AWT/Swing



Composants et containers

- **Component**
 - Tout objet ayant une représentation graphique et pouvant interagir avec l'utilisateur
- **Container**
 - Tout composant pouvant contenir d'autres composants

05/12/03

Interfaces graphiques

page 9

Containers primaires

- Il existe 3 principales classes de containers primaires (« top-level ») :
 - JFrame, JDialog et JApplet.
- Chaque container primaire possède un « content pane » qui détient les composants à afficher.
- Il est possible d'ajouter une barre de menu à un container primaire.

05/12/03

Interfaces graphiques

page 13

Fenêtres

- **Window**
 - Toute fenêtre primaire (top-level) sans bordure et sans barre de titre et menu.
- **JFrame**
 - Tout cadre primaire avec une bordure et une barre de titre.
- **JDialog**
 - Toute boîte de dialogue sous la responsabilité d'un cadre ou d'une autre boîte de dialogue.

05/12/03

Interfaces graphiques

page 10

Containers intermédiaires

- Ces composants servent à regrouper et/ou présenter d'autres composants.
- Quelques containers intermédiaires :
 - JPanel
 - JScrollPane
 - JSplitPane
 - JTabbedPane
 - JToolBar

05/12/03

Interfaces graphiques

page 14

Composants Swing

- **Composants lourds**
 - Composants utilisant les ressources (fenêtres) fournies par le système d'exploitation
 - JFrame, JDialog, JApplet (et JWindow)
- **Composants légers**
 - Composants issus (essentiellement) de sous-classes de la classe JComponent

05/12/03

Interfaces graphiques

page 11

Composants atomiques

- Ces composants ne sont pas composés d'autres composants.
- Quelques composants atomiques :
 - JLabel
 - JButton
 - JTextField
 - JComboBox
 - JTable

05/12/03

Interfaces graphiques

page 15

Types de composants

- On peut distinguer 3 types de composants Swing :
 - containers primaires (lourds)
 - containers intermédiaires (légers)
 - composants atomiques (légers)

05/12/03

Interfaces graphiques

page 12

Créer un cadre

05/12/03

Interfaces graphiques

page 16

JFrame

- Une instance de la classe JFrame (Frame pour AWT) correspond à la fenêtre principale (ou cadre) d'une application autonome.
- Il ne peut y avoir qu'un seul cadre par application.

05/12/03

Interfaces graphiques

page 17

Quitter l'application

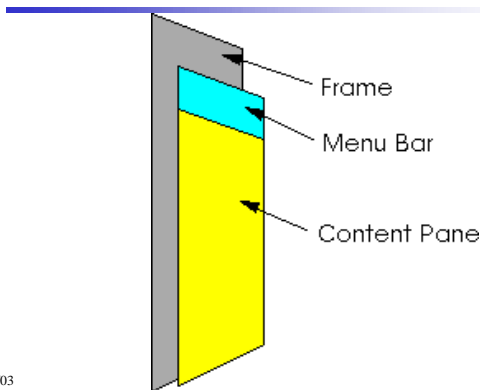
- Depuis JDK 1.3
Utilisation de `setDefaultCloseOperation` avec le paramètre `JFrame.EXIT_ON_CLOSE`
 - Avec JDK1.2
Utilisation d'un « window listener »
`addWindowListener(new MyWindowListener());`
+ une classe interne qui étend `WindowAdapter`
- Exemple : [CadreVide2.java](#)

05/12/03

Interfaces graphiques

page 21

Structure d'un cadre



05/12/03

page 18

setDefaultCloseOperation

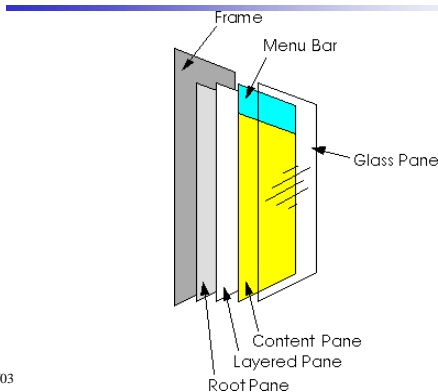
- Lorsque l'utilisateur décide de fermer un cadre (ou une boîte de dialogue), il est possible d'avoir un comportement particulier.
- Pour cela, on utilise la méthode `setDefaultCloseOperation` avec l'un des arguments suivants :
 - `DO_NOTHING_ON_CLOSE`
 - `HIDE_ON_CLOSE`
 - `DISPOSE_ON_CLOSE`
 - `EXIT_ON_CLOSE`

05/12/03

Interfaces graphiques

page 22

Structure complète d'un cadre



05/12/03

page 19

Classe Toolkit

- Il s'agit d'une classe qui permet de gérer des informations dépendant du système.
- Par exemple,
 - la récupération de la taille de l'écran
 - la récupération d'une image
- On récupère une boîte à outils par défaut avec : `Toolkit.getDefaultToolkit()`
- Exemple : [CadreDimensionne.java](#)

05/12/03

Interfaces graphiques

page 23

Définir un cadre

- Il suffit d'écrire une classe qui hérite de `JFrame` et d'intégrer un constructeur qui permet de :
 - spécifier un titre : `setTitle("coucou");`
 - spécifier une taille : `setSize(400,200);`
 - spécifier une position : `setLocation(20,50);`
 - rendre visible le cadre : `setVisible(true);`
- Exemple : [CadreVide1.java](#)

05/12/03

Interfaces graphiques

page 20

Ajouter des composants

- On souhaite ajouter des composants dans la fenêtre (cadre).
- Pour ajouter des composants, on peut procéder :
 - en utilisant le « content pane » par défaut. On utilise alors la méthode `getContentPane()`.
 - en définissant son propre « content pane ». On utilise alors la méthode `setContentPane()`.

05/12/03

Interfaces graphiques

page 24

Utiliser getContentPane

- Pour ajouter un composant comp (objet instance de Component), on écrit :
`getContentPane().add(comp);`
- Exemple: [HelloWorld1.java](#)
- Il est possible d'ajouter plusieurs composants à la suite (mais attention à la logique de mise en place).

05/12/03

Interfaces graphiques

page 25

Premiers Composants

- Les premiers composants (atomiques) que nous allons étudier sont des objets instances de :
 - JLabel
 - JButton
 - JTextField

05/12/03

Interfaces graphiques

page 29

Utiliser setContentPane

- Une façon classique de procéder :
 - créer une classe interne héritant de JPanel
 - déclarer un champ de cette classe
 - ajouter dans le constructeur du cadre l'instruction `setContentPane(nomChamp);`
- Un container intermédiaire JPanel est généralement utilisé pour regrouper (contenir) d'autres composants ou pour dessiner.
- Exemple : [HelloWorld2.java](#)

05/12/03

Interfaces graphiques

page 26

ToolTips

- Il est facile d'ajouter une information contextuelle (tool tip text) associée à n'importe quel objet instance de JComponent.
- Il suffit d'utiliser la méthode : `setToolTipText(...)`

05/12/03

Interfaces graphiques

page 30

Dessiner

- Pour dessiner dans un composant, il faut définir une classe héritant d'une sous-classe de JComponent et intégrer la méthode :

```
public void paintComponent(Graphics g)
{
    super.paintComponent(g)
    ...
}
```
- En utilisant le contexte graphique (variable de type Graphics), il est possible de dessiner.
- Exemple : [CadreDessin.java](#)

05/12/03

Interfaces graphiques

page 27

ImageIcon

- Il est possible de décorer un objet JLabel et JButton avec une image (icône).
- Pour charger une image de décoration (objet de classe ImageIcon), il est possible entre autres d'utiliser un constructeur prenant en argument :
 - un nom de fichier (gif ou jpeg)
 - une URL
 - un objet de la classe Image

05/12/03

Interfaces graphiques

page 31

Premiers Composants

JLabel

- Un objet qui représente un message et/ou une icône non sélectionnable.
- Il est possible de préciser l'alignement (LEFT, CENTER, RIGHT, ...) à la création.
- Exemple : [TestLabels.java](#)

05/12/03

Interfaces graphiques

page 28

05/12/03

Interfaces graphiques

page 32

Positionnement et raccourcis

- Il est possible de positionner un message par rapport à une icône lorsque ceux-ci figurent sur un objet JLabel ou JButton :
setVerticalTextPosition(...);
setHorizontalTextPosition(...);
- Il est possible d'associer un raccourci clavier à un bouton :
setMnemonic(...);

05/12/03

Interfaces graphiques

page 33

Interface ActionListener

- Dans cette interface ne figure qu'une seule méthode :
public void actionPerformed(ActionEvent e)
- Il suffit d'implémenter cette interface pour créer un « listener » qui sera capable d'apporter une réponse à toute « action » :
 - click pour un bouton
 - « Entrée » pour un champ de saisie

05/12/03

Interfaces graphiques

page 37

JButton

- Un objet que l'on peut activer (click de souris en général).
- Il est possible de placer un message (un intitulé) et/ou une icône sur un bouton.
- Exemple : [TestButtons1.java](#)

05/12/03

Interfaces graphiques

page 34

Exemple

- Lorsque l'utilisateur clique sur le premier bouton, le bouton central devient accessible.
- Lorsque l'utilisateur clique sur le troisième bouton, le bouton central devient inaccessible.
- Exemple : [TestButtons2.java](#)

05/12/03

Interfaces graphiques

page 38

Gérer les événements

- Il est possible de gérer tous les événements que reçoivent les différents composants placés sur une interface.
- Pour cela, il faut mettre en place des objets qui apportent une réponse à ces événements.
- Ces objets sont appelés des « listener ».

05/12/03

Interfaces graphiques

page 35

JTextField

- Un composant qui permet l'édition d'une simple ligne de texte.
- Les fonctions d'accès au contenu du champ de saisie sont :
 - setText(...)
 - getText()
- On peut indiquer la largeur désirée lors de la construction (en nombre de caractères).

05/12/03

Interfaces graphiques

page 39

Réponse à un événement "Action"

- Lorsque l'utilisateur clique sur un bouton, un événement ActionEvent est généré.
- Pour prendre en compte ce type d'événements, il faut :
 - définir une classe implémentant ActionListener
 - effectuer le lien entre le bouton et un objet de cette nouvelle classe à l'aide de la méthode addActionListener(...).

05/12/03

Interfaces graphiques

page 36

Document

- Sous-jacent à tout champ de saisie (une vue) existe un document (le modèle).
- Pour récupérer le document sous-jacent, on utilise la méthode getDocument().
- Il est possible de gérer les événements
 - directement sur la vue (champ de saisie) en utilisant (entre autres) l'interface ActionListener
 - indirectement sur le modèle (document) en utilisant l'interface DocumentListener

05/12/03

Interfaces graphiques

page 40

DocumentListener

- Dans cette interface figurent trois méthodes :
 - public void insertUpdate(DocumentEvent e)
 - public void removeUpdate(DocumentEvent e)
 - public void changedUpdate(DocumentEvent e)
- Les deux premières méthodes sont appelées lorsque des caractères ont été insérés ou supprimés.
- La troisième méthode n'est pas appelée pour les champs de saisie.

05/12/03

Interfaces graphiques

page 41

Types de boîtes de dialogue

- Il est possible de créer des boîtes de dialogue « standard » à partir des classes suivantes :
 - JOptionPane
 - JColorChooser
 - JFileChooser
 - ProgressMonitor
- Il est possible de créer des boîtes de dialogue personnalisées à partir de la classe suivante :
 - JDialog

05/12/03

Interfaces graphiques

page 45

Exemple

- Lorsque l'utilisateur valide le premier champ, le texte est positionné dans le troisième.
- Lorsque l'utilisateur modifie le second champ, le texte est positionné dans le troisième.
- Exemple : [TestTextField.java](#)

05/12/03

Interfaces graphiques

page 42

JOptionPane

- Un objet JOptionPane représente une boîte de dialogue standard simple (toujours modale) pour informer ou interroger l'utilisateur.
- De nombreuses méthodes de classes existent dans JOptionPane permettant d'ouvrir une boîte :
 - demandant une confirmation : showConfirmDialog(...)
 - proposant une saisie élémentaire : showInputDialog(...)
 - présentant un message : showMessageDialog(...)
 - proposant plusieurs options : showOptionDialog(...)

05/12/03

Interfaces graphiques

page 46

Boîtes de dialogue

- Les paramètres que l'on retrouve généralement pour ouvrir les différentes boîtes sont :
 - le composant responsable de la boîte
 - si le composant désigne une référence non « null » alors la boîte est ouverte et positionnée par rapport à ce composant.
 - sinon la boîte est ouverte et centrée par rapport à l'écran
 - le message à afficher dans la boîte
 - la chaîne de caractères à afficher dans la barre de titre

05/12/03

Interfaces graphiques

page 43

05/12/03

Interfaces graphiques

page 47

Boîte de dialogue





- Une boîte de dialogue est une fenêtre qui est dépendante d'un cadre ou d'une autre boîte de dialogue.
- Pour gérer cette dépendance, il est nécessaire de préciser lors de la création d'une boîte de dialogue quel est le cadre ou boîte propriétaire.
- Une boîte de dialogue peut être modale ou non modale.
- Une boîte de dialogue modale ne permet pas l'accès, tant qu'elle n'est pas fermée, au cadre et autres boîtes de dialogue du programme.

05/12/03

Interfaces graphiques

page 44

Icônes prédéfinies sur une boîte

- Un paramètre « messageType » permet de sélectionner une icône. Les valeurs possibles sont :
 - ERROR_MESSAGE 
 - INFORMATION_MESSAGE 
 - WARNING_MESSAGE 
 - QUESTION_MESSAGE 
 - PLAIN_MESSAGE

05/12/03

Interfaces graphiques

page 48

Les boutons sur une boîte

- Un paramètre « optionType » permet de définir les boutons apparaissant sur une boîte. Les valeurs possibles sont : :
 - DEFAULT_OPTION
 - YES_NO_OPTION
 - YES_NO_CANCEL_OPTION
 - OK_CANCEL_OPTION
- Il est également possible de passer sa propre liste de boutons en paramètre.

05/12/03

Interfaces graphiques

page 49

Valeurs de retour

- Les valeurs de retour possibles après l'ouverture de l'une de ces boîtes de dialogue sont :
 - JFileChooser.CANCEL_OPTION
 - JFileChooser.APPROVE_OPTION
 - JFileCHooser.ERROR_OPTION

05/12/03

Interfaces graphiques

page 53

Valeurs de retour d'une boîte

- Les valeurs de retour possibles pour une boîte (retournant une valeur) sont :
 - YES_OPTION
 - NO_OPTION
 - OK_OPTION
 - CANCEL_OPTION
 - CLOSED_OPTION

05/12/03

Interfaces graphiques

page 50

Fichier et/ou répertoire

- La méthode setFileSelectionMode permet d'indiquer si l'utilisateur a la possibilité de sélectionner des fichiers et/ou des répertoires.
- Cette méthode prend en paramètre l'une des valeurs suivantes :
 - JFileChooser.FILES_ONLY
 - JFileChooser.DIRECTORIES_ONLY
 - JFileChooser.FILES_AND_DIRECTORIES
- La première est la valeur par défaut.

05/12/03

Interfaces graphiques

page 54

JColorChooser

- Cette classe possède une méthode statique showDialog qui permet d'ouvrir une boîte prédéfinie proposant l'ensemble des couleurs 24 bits possibles.
- La méthode prend en paramètre :
 - le composant responsable de la boîte,
 - une chaîne de caractère à placer dans la barre de titre
 - la couleur sélectionnée initialementet retourne :
 - la couleur sélectionnée par l'utilisateur.

05/12/03

Interfaces graphiques

page 51

JDialog

- La classe JDialog permet de créer une boîte de dialogue personnalisée.
- La seule différence fondamentale entre un objet JDialog et un objet JFrame est que le premier dépend nécessairement d'un second.
 - si le second est fermé, le premier le sera également
 - si le second est réduit, le premier le sera également
 - ...

05/12/03

Interfaces graphiques

page 55

JFileChooser

- La classe JFileChooser offre 2 méthodes permettant d'ouvrir une boîte de dialogue proposant la sélection d'un fichier ou d'un répertoire.
- Ces méthodes sont respectivement
 - public int showOpenDialog(Component parent) throws HeadlessException
 - public int showSaveDialog(Component parent) throws HeadlessException

05/12/03

Interfaces graphiques

page 52

Création d'une boîte de dialogue

- Lors de la création d'un objet JDialog, il est nécessaire de passer en argument le composant responsable de la boîte créée. Celui-ci doit être nécessairement issu de JFrame ou de JDialog.
- Il est par ailleurs possible de préciser la chaîne à afficher dans la barre de titre ainsi que le mode de la boîte.

05/12/03

Interfaces graphiques

page 56

Ajouter des composants

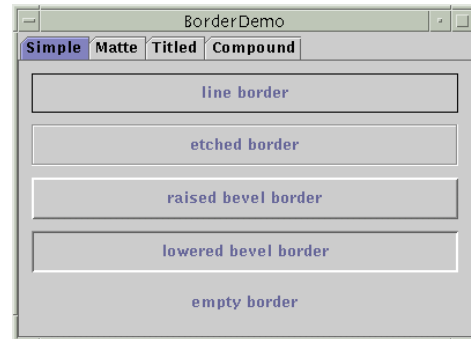
- Un objet `JDialog` est un container primaire (de la même façon qu'un objet `JFrame`).
- Il faut donc utiliser un panneau spécial pour ajouter les composants. On utilise donc les méthodes (voir section sur la création d'un cadre) :
 - `getContentPane`
 - `setContentPane`

05/12/03

Interfaces graphiques

page 57

Exemple



05/12/03

Interfaces graphiques

page 61

Bordures

05/12/03

Interfaces graphiques

page 58

Bordures « Matte »

- On peut créer une bordure en spécifiant une icône ou une couleur. On indique la largeur souhaitée en pixels dans l'ordre : top, left, bottom, and right.
- Par exemple :

```
BorderFactory.createMatteBorder(-1,-1,-1,-1,icon);
BorderFactory.createMatteBorder(1,5,1,1,Color.red);
BorderFactory.createMatteBorder(0,20,0,0,icon);
```
- Exemple : [TestMatteBorders.java](#)

05/12/03

Interfaces graphiques

page 62

Bordures

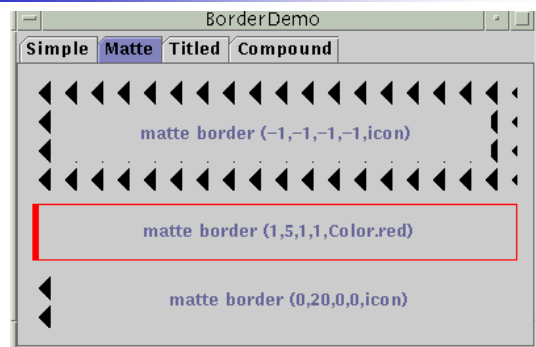
- Il est possible de placer une bordure autour de chaque composant.
- Pour cela, il suffit d'utiliser la méthode `setBorder` et de placer en argument un objet d'une classe implémentant l'interface `Border`.
- La classe `BorderFactory` fournit de nombreuses méthodes « static » retournant des objets `Border`.

05/12/03

Interfaces graphiques

page 59

Exemple



05/12/03

Interfaces graphiques

page 63

Bordures simples

- Voici 5 types de bordures simples :

```
Border blackline, etched, raisedbevel, loweredbevel, empty;
blackline = BorderFactory.createLineBorder(Color.black);
etched = BorderFactory.createEtchedBorder();
raisedbevel = BorderFactory.createRaisedBevelBorder();
loweredbevel = BorderFactory.createLoweredBevelBorder();
empty = BorderFactory.createEmptyBorder();
```
- Exemple : [TestSimpleBorders.java](#)

05/12/03

Interfaces graphiques

page 60

Bordures avec titre

- Voici 5 exemples de bordures avec titre :

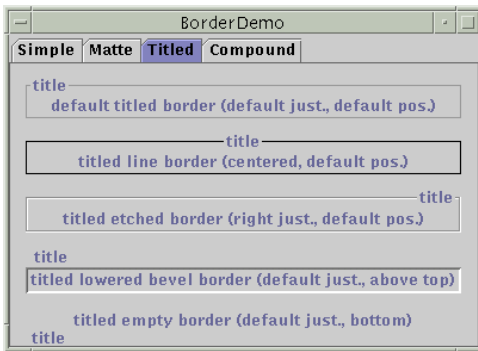
```
TitledBorder tb1, tb2, tb3, tb4, tb5;
t1 = BorderFactory.createTitledBorder("title");
t2 = BorderFactory.createTitledBorder(blackline, "title");
t2.setTitleJustification(TitledBorder.CENTER);
t3 = BorderFactory.createTitledBorder(etched, "title");
t3.setTitleJustification(TitledBorder.RIGHT);
t4 = BorderFactory.createTitledBorder(loweredbevel, "title");
t4.setTitlePosition(TitledBorder.ABOVE_TOP);
t5 = BorderFactory.createTitledBorder(empty, "title");
t5.setTitlePosition(TitledBorder.BOTTOM);
```

05/12/03

Interfaces graphiques

page 64

Exemple



05/12/03

Interfaces graphiques

page 65

Créer son propre style de bordure

- Généralement, on crée une sous-classe de `AbstractBorder` dans laquelle apparaît :
 - au moins un constructeur
 - la méthode `paintBorder` contenant le code qui sera exécuté pour dessiner la bordure
 - la méthode `getBorderInsets` qui spécifie l'espace nécessaire de chaque côté pour la bordure.

05/12/03

Interfaces graphiques

page 69

Bordures composées

- Les bordures composées se conçoivent en combinant 2 bordures (extérieure et intérieure).
- Voici 3 exemples de bordures composées :

```
Border redline = BorderFactory.createLineBorder(Color.red);
Border compound1, compound2, compound3;
compound1 = BorderFactory.createCompoundBorder(
    raisedbevel, loweredbevel);
compound2 = BorderFactory.createCompoundBorder(
    redline, compound1);
compound3 = BorderFactory.createTitledBorder(
    compound2, "title", TitledBorder.CENTER,
    TitledBorder.BELOW_BOTTOM);
```

05/12/03

Interfaces graphiques

page 66

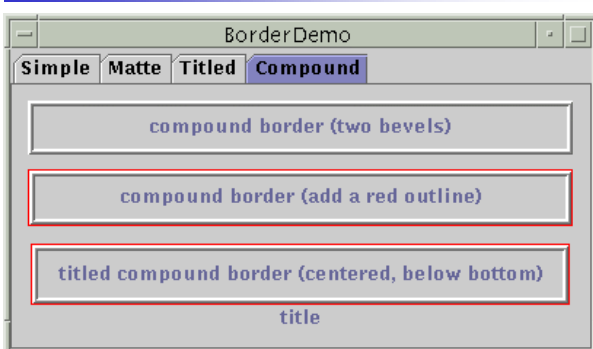
Layouts

05/12/03

Interfaces graphiques

page 70

Exemple



05/12/03

Interfaces graphiques

page 67

Positionnement des composants

- Il est judicieux d'utiliser une logique de mise en place des composants dans un container car un ajustement (re-disposition) s'effectue automatiquement lorsque un paramètre change
 - taille du container
 - police utilisée
 - apparence (Look-and-feel)
- But if a container holds components whose size isn't affected by the container's size or by font and look-and-feel changes, then absolute positioning might make sense.

05/12/03

Interfaces graphiques

page 71

Ajouter une seconde bordure

- Il suffit
 - de créer une nouvelle bordure
 - de récupérer la bordure existante avec `getBorder()`
 - d'utiliser une bordure composée
- Par exemple, pour ajouter un espace de 20 pixels en haut du composant component, on écrira :

```
Border b1 = BorderFactory.createEmptyBorder(20,0,0,0);
Border b2 = component.getBorder();
Border b = BorderFactory.createCompoundBorder(b1, b2);
component.setBorder(b);
```

05/12/03

Interfaces graphiques

page 68

Positionnement absolu

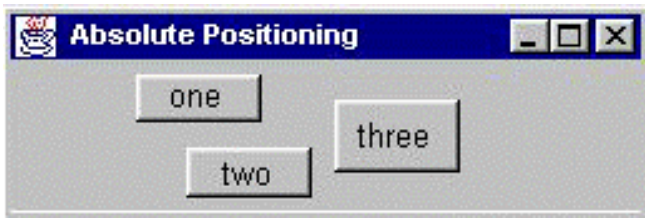
- Pour utiliser un positionnement absolu, il faut :
 - l'indiquer : `setLayout(null)`;
 - placer de façon absolue chaque composant dans le container : `setBounds(int x, int y, int width, int height)`
- On peut récupérer la taille de chaque bordure du container (méthode `getInsets()`) afin d'éviter de déborder dans celles-ci.
- Exemple : [TestAbsolute.java](#)

05/12/03

Interfaces graphiques

page 72

Illustration



05/12/03

Interfaces graphiques

page 73

Illustration



05/12/03

Interfaces graphiques

page 77

Layout managers

- Chaque container possède une logique de mise en forme (« layout manager ») par défaut.
- Cette logique est utilisée pour placer (visuellement) les composants dans le container.
- Une logique de mise en forme est un objet qui implémente l'interface `LayoutManager`.
- Il est possible de changer de logique de mise en forme à l'aide de la méthode `setLayout()`.

05/12/03

Interfaces graphiques

page 74

GridLayout

- Cette logique de mise en forme permet de construire une grille (ensemble de cellules) dans laquelle on peut placer des composants.
- Chaque composant utilise toute la place nécessaire de sa cellule.
- A la création, il est nécessaire d'indiquer le nombre de lignes et le nombre de colonnes.
- Il est également possible d'indiquer l'espace horizontal et vertical (en pixels) désiré entre chaque cellule.
- Exemple : [TestGridLayout.java](#)

05/12/03

Interfaces graphiques

page 78

Layouts managers existants

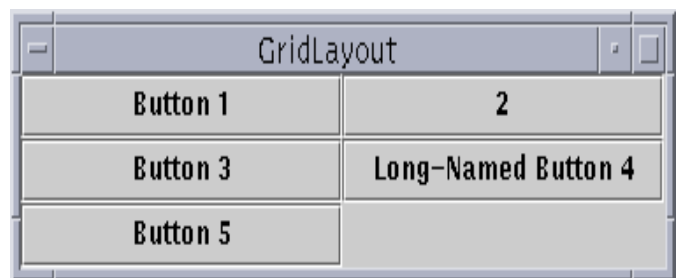
- Java (SDK 1.3) propose plusieurs logiques de mise en forme. Ces logiques sont :
 - simples : `FlowLayout` et `GridLayout`
 - spécialisées : `BorderLayout` et `CardLayout`
 - souples : `BoxLayout` et `GridBagLayout`
- Quand on ajoute des composants dans un container, les arguments que l'on passe à la méthode `add()` dépendent du « layout manager » utilisé.

05/12/03

Interfaces graphiques

page 75

Illustration



05/12/03

Interfaces graphiques

page 79

FlowLayout

- Cette logique de mise en forme permet de placer les composants selon leur taille préférée les uns à la suite des autres de gauche à droite et de haut en bas.
- Lorsque la place est insuffisante sur une ligne, une nouvelle ligne est considérée.
- Au sein d'une ligne, les composants sont centrés (par défaut), alignés à gauche ou à droite selon la valeur spécifiée à la création.
- `FlowLayout` est la logique de mise en forme par défaut pour les objets `JPanel`.
- Exemple : [TestFlowLayout.java](#)

05/12/03

Interfaces graphiques

page 76

BorderLayout

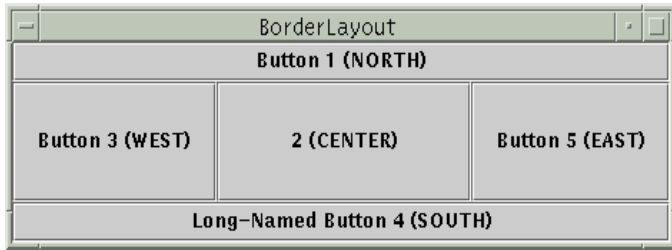
- Cette logique de mise en forme permet de placer les composants selon leur taille préférée dans cinq zones différentes : north, south, east, west, et center.
- Le composant placé au centre (si il existe) prend tout l'espace non utilisé par les autres composants.
- Il faut préciser lorsqu'on ajoute un composant la zone dans laquelle on souhaite l'établir.
- `BorderLayout` est la logique de mise en forme par défaut pour tout « content-pane ».
- Exemple : [TestBorderLayout.java](#)

05/12/03

Interfaces graphiques

page 80

Illustration



05/12/03

Interfaces graphiques

page 81

Taille des composants

- Chaque composant possède une taille minimum, préférée et maximum.
- La taille est donnée par un objet Dimension.
- Il est possible de modifier les tailles :
`comp.setPreferredSize(new Dimension(50,25));`
`comp.setMaximumSize(comp.getPreferredSize());`
- On peut également redéfinir les méthodes `getXxxSize()` pour modifier les tailles si on a défini une classe pour le composant.

05/12/03

Interfaces graphiques

page 85

CardLayout

- Cette logique de mise en forme permet d'utiliser le même espace pour afficher différents composants.
- Les composants sont organisés en « cartes » (généralement des objets `JPanel`) et une seule d'entre elles est visible à un moment donné.
- L'utilisation d'un onglet (objet `TabbedPane`) est généralement avantageux.
- Exemple : [TestCardLayout.java](#)

05/12/03

Interfaces graphiques

page 82

Alignement des composants

- Chaque composant possède une propriété d'alignement horizontale et verticale.
- L'alignement est donnée par une constante de `JComponent`.
- Exemple : [TestSizesAndAlignments.java](#)
- Il est possible de modifier l'alignement :
`comp.setAlignmentX(JComponent.LEFT_ALIGNMENT);`
`comp.setAlignmentY(JComponent.TOP_ALIGNMENT);`
- Exemple : [TestBoxLayout2.java](#)

05/12/03

Interfaces graphiques

page 86

Changer de « carte »

- On peut changer de « carte » visible en considérant :
 - la première ou la dernière
 - la précédente ou la suivante
 - une carte donnée en utilisant son nom
- Il faut pour cela récupérer l'objet `CardLayout` du conteneur puis appeler l'une des méthodes en passant le conteneur en paramètre.

05/12/03

Interfaces graphiques

page 83

Rigid area

- Ce composant invisible permet de placer un espace de taille fixe entre 2 composants.
- Exemple :
`container.add(firstComponent);`
`container.add(Box.createRigidArea(new Dimension(5,0)));`
`container.add(secondComponent);`



05/12/03

Interfaces graphiques

page 87

BoxLayout

- Cette logique de mise en forme permet de placer les composants en ligne ou en colonne.
- Contrairement à d'autres logiques, `BoxLayout` respecte les tailles minimum, préférée et maximum des composants ainsi que leurs propriétés d'alignement.
- Tout espace non utilisé est placé au bout de la ligne ou en bas de la colonne.
- Exemple : [TestBoxLayout1.java](#)

05/12/03

Interfaces graphiques

page 84

Exemple : [TestBoxLayout3.java](#)

Glue

- Ce composant invisible est utilisé pour représenter l'espace non utilisé.
- Exemple :
`container.add(firstComponent);`
`container.add(Box.createHorizontalGlue());`
`container.add(secondComponent);`

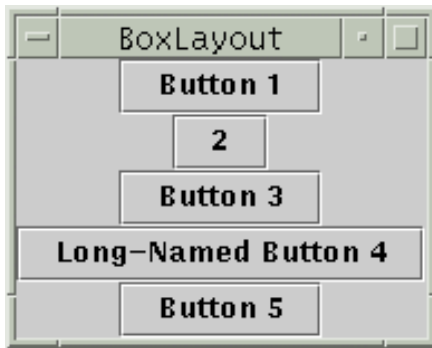


05/12/03

Interfaces graphiques

page 88

Illustration

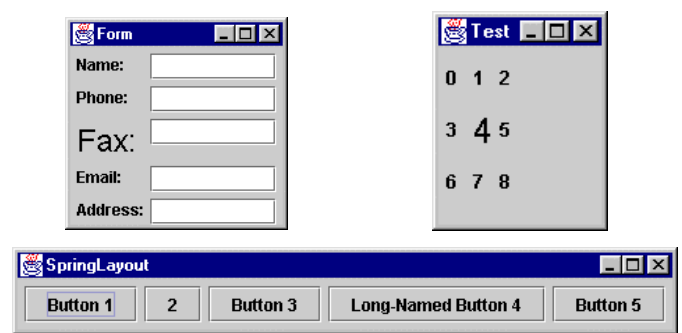


05/12/03

Interfaces graphiques

page 89

Illustration



05/12/03

Interfaces graphiques

page 93

GridBagLayout

- Cette logique de mise en forme est la plus souple et la plus complexe.
- Les composants sont placés dans une grille et peuvent s'étendre sur plusieurs cellules.
- Pas d'exemple.

05/12/03

Interfaces graphiques

page 90

Programmation événementielle

05/12/03

Interfaces graphiques

page 94

Illustration



05/12/03

Interfaces graphiques

page 91

Événements

- Lorsque l'utilisateur interagit avec une interface graphique (en appuyant sur une touche du clavier ou un bouton de souris par exemple), un (ou plusieurs) événement est généré.
- La programmation événementielle consiste à associer des réponses aux différents événements
 - pouvant se produire
 - et intéressant l'utilisateur

05/12/03

Interfaces graphiques

page 95

SpringLayout

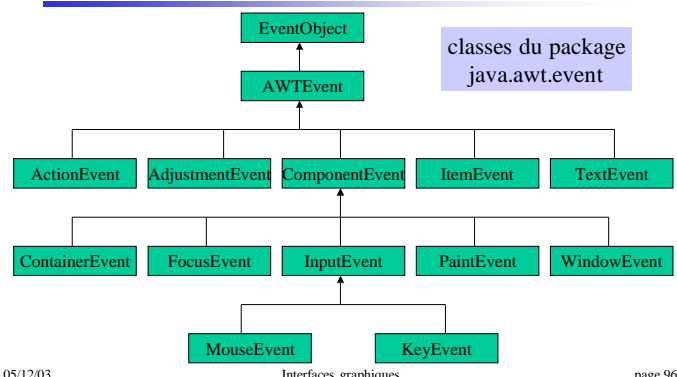
- Cette logique de mise en forme a été ajoutée dans le SDK 1.4.
- Elle permet de placer les composants de façon similaire au positionnement absolu mais réagit de façon plus appropriée aux changements.
- Pas d'exemple

05/12/03

Interfaces graphiques

page 92

Hierarchie d'événements



05/12/03

Interfaces graphiques

page 96

Formes d'événements

- En Java, on peut distinguer deux formes d'événements :
 - événements de bas niveau : ce sont les événements qui sont générés par des actions élémentaires (« physiques ») de l'utilisateur
 - événements de haut niveau (sémantiques) : ce sont les événements qui sont générés par des actions logiques de l'utilisateur

05/12/03

Interfaces graphiques

page 97

Programmation événementielle

- Il est possible d'apporter une réponse automatique à tout événement. Il suffit de lier la source de l'événement à une (ou plusieurs) cible(s).
- Source : composant qui « crée » l'événement (généralement en conséquence d'une action de l'utilisateur)
- Cible : objet qui traite l'événement en exécutant une méthode en rapport avec la nature précise de celui-ci.

05/12/03

Interfaces graphiques

page 101

Evénements de bas niveau

- Ce sont les événements définis à partir de `ComponentEvent`. Ils correspondent à des actions sur :
 - le clavier (`KeyEvent`)
 - la souris (`MouseEvent`)
 - la focalisation (`FocusEvent`)
 - une fenêtre (`WindowEvent`)
 - un composant (`ComponentEvent`)
 - un container (`ContainerEvent`)

05/12/03

Interfaces graphiques

page 98

Source

- Une source d'événements en Java est un composant graphique, i.e., tout objet d'une classe issue de `Component`.
- En fait, un composant ne génère des événements qu'en réaction à une intervention extérieure (comme celle de l'utilisateur).
- Tous les composants ne peuvent pas générer les mêmes événements.

05/12/03

Interfaces graphiques

page 102

Evénements sémantiques

- Ce sont des événements qui peuvent se produire de différents manières pour différents composants.
- Par exemple, un `ActionEvent` peut être produit par :
 - un click de souris sur un bouton
 - un raccourci clavier (pour un bouton)
 - la validation (touche « entrée ») sur un champ de saisie
 - la sélection d'un article de menu
- Il est judicieux de préférer (quand cela est possible) les événements sémantiques à ceux de bas niveau. Cela garantit un code plus robuste et plus portable.

05/12/03

Interfaces graphiques

page 99

Source Component et Container

- Tout objet `Component` peut produire des événements. Ceux-ci sont :
 - `ComponentEvent` : changements de taille, position ou visibilité
 - `FocusEvent` : gain ou perte de la focalisation
 - `KeyEvent` : frappes au clavier
 - `MouseEvent` : clicks et mouvements de souris
- Tout objet (utilisé en tant que) `Container` peut produire des événements
 - `ContainerEvent` : ajout et suppression d'un composant

05/12/03

Interfaces graphiques

page 103

Liens entre événements

- En général, à un événement sémantique correspond plusieurs événements de bas niveau.
- Par exemple, en imaginant que le pointeur de souris se trouve à l'extérieur d'un bouton et que l'on décide de cliquer dessus :
 - à l'événement sémantique `ActionEvent`
 - correspond les événements `MouseEvent` suivants :
 - `MOUSE_MOVED`
 - `MOUSE_ENTERED`
 - `MOUSE_PRESSED`
 - `MOUSE_RELEASED`

05/12/03

Interfaces graphiques

page 100

Source spécifique

- Toute composant `Swing` peut par ailleurs produire des événements spécifiques.
- Par exemple :
 - tout objet `JButton`, `JTextField`, ... peut produire des événements `ActionEvent`
 - tout objet `JComboBox` peut produire des événements `ItemEvent`
 - tout objet `JFrame` ou `JDialog` peut produire des événements `WindowEvent`

05/12/03

Interfaces graphiques

page 104

Cible

- Une cible est un objet « écouteur » (listener) d'événements.
- Un tel objet est instance d'une classe qui implémente une interface étendant `EventListener`.
- La plupart de ces interfaces possède plusieurs méthodes (en rapport avec la nature précise de l'événement).

05/12/03

Interfaces graphiques

page 105

Rôle de la cible

- Lorsqu'un lien est établi entre une source et une cible, on dit que la cible est enregistrée en tant que « listener » de la source.
- Cela signifie que ce « listener » sera systématiquement mis à contribution lorsqu'un événement (du type de ceux que le « listener » peut traiter) sera généré par la source.

05/12/03

Interfaces graphiques

page 109

Interfaces étendant `EventListener`

- Interfaces du package `java.awt.event` :
 - `ActionListener` `AdjustmentListener`
 - `ComponentListener` `ContainerListener`
 - `FocusListener` `ItemListener`
 - `KeyListener` `MouseListener`
 - `MouseMotionListener`
 - `TextListener` `WindowListener`

05/12/03

Interfaces graphiques

page 106

Problème

- On décide de créer une interface avec 3 boutons, un champ de saisie et un label.
- Lorsqu'on clique sur un bouton, l'intitulé de celui-ci s'affiche dans le label.
- Lorsqu'on valide un champ de saisie (par « Entrée »), l'intitulé de celui-ci s'affiche également dans le label.

05/12/03

Interfaces graphiques

page 110

Classes Adapter

- Lorsqu'une interface `XxxListener` possède plusieurs méthodes, une classe `XxxAdapter` implémentant cette interface est généralement disponible dans le JDK.
- Cette implémentation consiste à donner un corps vide à chaque méthode.
- Lorsqu'un nombre limité de méthodes d'une interface intéresse l'utilisateur, celui-ci préférera alors étendre la classe `Adapter` qu'implémenter l'interface `Listener`.

05/12/03

Interfaces graphiques

page 107

Solution 1

- Considérer comme « listener » le container (objet `JPanel`) des boutons.
- Il est donc nécessaire d'effectuer des tests afin de savoir quel est le bouton qui a généré l'événement.
- On peut utiliser la méthode `getSource` définie dans `EventObject`.
- Exemple : [TestActionEvent1.java](#)

05/12/03

Interfaces graphiques

page 111

Lier Source et Cible

- Lier la source `c` d'un événement à une cible `l` consiste à utiliser une méthode du type :
`c.addXxxListener(l);`
- Il est possible de créer :
 - plusieurs liens avec la même source
 - plusieurs liens avec la même cible
- Il est également possible de dénouer un lien donné avec une méthode du type :
`c.removeXxxListener(l);`

05/12/03

Interfaces graphiques

page 108

Critique

- Positive
 - rapide à mettre en œuvre (si peu de composants) :-)
- Négative
 - il n'est pas très clair et logique que le container gère les événements de ses composants :-)
 - le code de réponse associé aux différents composants est mélangé :-)
 - il est nécessaire de déterminer quel composant a généré l'événement (avec des successions de tests) :-)

05/12/03

Interfaces graphiques

page 112

Solution 2

- Créer une classe interne au container dédiée à la gestion des événements.
- Créer un objet de cette classe dans le container.
- Considérer cet objet comme le « listener » des différents composants.
- Exemple : [TestActionEvent2.java](#)

05/12/03

Interfaces graphiques

page 113

Solution 4

- Créer une classe interne anonyme par composant.
- On associe alors une instance de cette classe anonyme lors de l'enregistrement d'un « listener ».
- Exemple : [TestActionEvent4.java](#)

05/12/03

Interfaces graphiques

page 117

Critique

- Positive
 - On ne considère plus que le container gère les événements :-)
- Négative
 - le code de réponse associé aux différents composants est mélangé :-)
 - il est nécessaire de déterminer quel composant a généré l'événement (avec des successions de tests) :-)

05/12/03

Interfaces graphiques

page 114

Critique

- Positive
 - il n'est plus nécessaire de nommer les classes.
 - le code est proche du composant
- Négative
 - clarté ?

05/12/03

Interfaces graphiques

page 118

Solution 3

- Créer une classe interne par composant afin de gérer les événements produits par celui-ci.
- Associer à chaque composant, i.e., considérer comme « listener », une instance de la classe qui lui est associée.
- Exemple : [TestActionEvent3.java](#)

05/12/03

Interfaces graphiques

page 115

Types d'événements

05/12/03

Interfaces graphiques

page 119

Critique

- Positive
 - le code de réponse associé aux différents composants n'est plus mélangé :-)
 - il n'est plus nécessaire de déterminer quel composant a généré l'événement (avec des successions de tests) :-)
- Négative
 - le code de réponse(s) reste séparé du composant

05/12/03

Interfaces graphiques

page 116

Focalisation

- Un composant détient la focalisation si il peut recevoir les frappes du clavier.
- Un seul composant peut détenir la focalisation à un moment donné.
- Il est possible de passer la focalisation à un autre composant :
 - avec le clavier (tabulation en général)
 - avec la souris
 - par programmation (méthodes requestFocus et transferFocus)

05/12/03

Interfaces graphiques

page 120

Composants « focusable »

- Certains composants tels que les objets JLabel ou JPanel ne peuvent obtenir la focalisation (par défaut).
- Il est possible de modifier cet état en
 - (JDK 1.4) utilisant la méthode setFocusable()
 - (JDK 1.3) en redéfinissant la méthode isFocusTraversable

05/12/03

Interfaces graphiques

page 121

Événements de niveaux différents

- L'événement KEY_TYPED correspond à l'entrée d'un caractère. C'est un événement de plus haut niveau que les deux autres.
- Par exemple, la frappe d'un A engendre :
 - 4 événements de bas niveau
 - Appui (KEY_PRESSED) sur la touche Shift
 - Appui (KEY_PRESSED) sur la touche A
 - Relâchement (KEY_RELEASED) de la touche A
 - Relâchement (KEY_RELEASED) de la touche Shift
 - 1 événement de plus haut niveau
 - Frappe (KEY_TYPED) du caractère 'A'

05/12/03

Interfaces graphiques

page 125

FocusListener

- Cette interface contient 2 méthodes :

```
public void focusGained(FocusEvent e)
public void focusLost(FocusEvent e)
```
- Il existe une classe FocusAdapter
- Exemple : [TestFocus.java](#)

05/12/03

Interfaces graphiques

page 122

MouseListener

- Cette interface comprend 5 méthodes :
 - 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)
- Il existe une classe MouseAdapter.

05/12/03

Interfaces graphiques

page 126

Perte de focalisation d'un champ

- Il est parfois intéressant de s'assurer qu'un champ c a été correctement saisi :

```
public void focusLost(FocusEvent e) {
    if (! e.isTemporary() && ! isCorrect(c.getText()))
        c.requestFocus();
}
```
- isTemporary retourne vrai si on est sur de récupérer la focalisation (par exemple, parce qu'on a sélectionné une autre fenêtre).

05/12/03

Interfaces graphiques

page 123

MouseEvent

- Il est possible d'utiliser les méthodes suivantes :

```
public int getClickCount()
public int getX()
public int getY()
```
- Exemple: [TestMouseEvent.java](#)

05/12/03

Interfaces graphiques

page 127

KeyListener

- Cette interface comprend les méthodes :

```
public void keyTyped(KeyEvent e)
public void keyPressed(KeyEvent e)
public void keyReleased(KeyEvent e)
```
- Il existe une classe KeyAdapter.
- Exemple : [TestKeyEvent1.java](#)

05/12/03

Interfaces graphiques

page 124

MouseMotionListener

- Cette interface comprend 2 méthodes :

```
public void mouseDragged(MouseEvent e)
public void mouseMoved(MouseEvent e)
```
- Il existe une classe MouseMotionAdapter

05/12/03

Interfaces graphiques

page 128

Pourquoi 2 interfaces ?

- Il semble étrange qu'il y ait deux interfaces traitant des événements souris.
- En fait, les événements traités par un « listener » `MouseEvent` :
 - intéressent peu d'applications
 - et se produisent fréquemment (ils utilisent des ressources)

05/12/03

Interfaces graphiques

page 129

Description

- Les boutons sont des composants graphiques sur lesquels l'utilisateur clique pour amorcer une action bien déterminée.
- Un bouton à bascule (`JToggleButton`) est un bouton à deux états. Il en existe deux versions spécialisées :
 - la boîte à cocher (`JCheckBox`)
 - le bouton radio (`JRadioButton`)

05/12/03

Interfaces graphiques

page 133

WindowListener

- Cette interface comprend 7 méthodes :

```
public void windowOpened(WindowEvent e)
public void windowClosing(WindowEvent e)
public void windowClosed(WindowEvent e)
public void windowIconified(WindowEvent e)
public void windowDeiconified(WindowEvent e)
public void windowActivated(WindowEvent e)
public void windowDeactivated(WindowEvent e)
```
- Il existe une classe `WindowAdapter`

05/12/03

Interfaces graphiques

page 130

AbstractButton

- Pour tout objet instance de `AbstractButton`, il est possible de gérer des événements de type :
 - `ActionEvent`
 - `ChangeEvent`
 - `ItemEvent`
- Plusieurs alternatives de codage se présentent alors parfois.

05/12/03

Interfaces graphiques

page 134

Autres composants Swing

05/12/03

Interfaces graphiques

page 131

JCheckBox

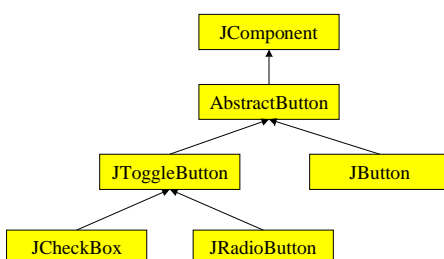
- La boîte à cocher possède deux états.
- Exemple : [TestCheckBox.java](#)
- Bien que sur cet exemple, on utilise un `ItemListener`, il était tout à fait possible d'utiliser un `ActionListener`.

05/12/03

Interfaces graphiques

page 135

Les composants « boutons »



05/12/03

Interfaces graphiques

page 132

ItemListener

- Cette interface contient une seule méthode :

```
public void itemStateChanged(ItemEvent e)
```
- Il n'existe donc pas de classe `ItemAdapter`.
- Il est possible, pour un objet `ItemEvent`, d'utiliser la méthode `getStateChange()` qui retourne :
 - `ItemEvent.SELECTED`
 - `ItemEvent.DESELECTED`

05/12/03

Interfaces graphiques

page 136

JRadioButton

- Les boutons radio se gèrent en groupe (à l'aide d'un objet ButtonGroup).
- Il suffit d'ajouter (méthode add) les différents boutons radios au groupe.
- Un seul des boutons peut être sélectionné à un moment donné.
- Exemple : [TestRadioButtons.java](#)

05/12/03

Interfaces graphiques

page 137

Utiliser setAction

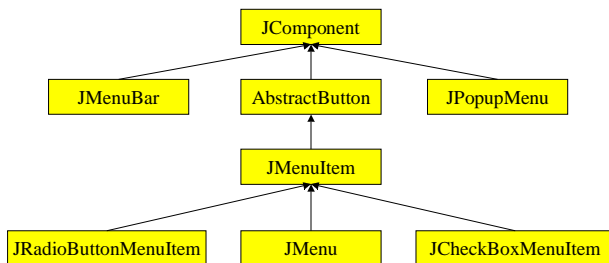
- Il est possible de définir des objets Action et de les associer à des composants (JButton ou JMenuItem par exemple) avec la méthode setAction (depuis JDK 1.3).
- Exemple : [TestMenu2.java](#)

05/12/03

Interfaces graphiques

page 141

Les composants « menus »



05/12/03

Interfaces graphiques

page 138

Ajouter des objets Action

- Il est possible d'ajouter directement des objets Action
 - à un objet JMenu
 - à un objet Jtoolbar
- Exemple : [TestMenu3.java](#)

05/12/03

Interfaces graphiques

page 142

Ajouter une barre de menu

- Il est possible d'ajouter une barre de menu (objet JMenuBar) à un cadre, d'ajouter des menus (objets JMenu) à une barre de menu et des articles de menus (objets JMenuItem) à un menu.
- Exemple : [TestMenu1.java](#)

05/12/03

Interfaces graphiques

page 139

JToolBar

- Un objet JToolBar est un container qui regroupe plusieurs composants en une ligne ou une colonne.
- En règle générale, les composants sont des boutons avec une icône qui fournissent un accès rapide à des fonctionnalités qui existent aussi dans la barre de menu.

05/12/03

Interfaces graphiques

page 143

Action

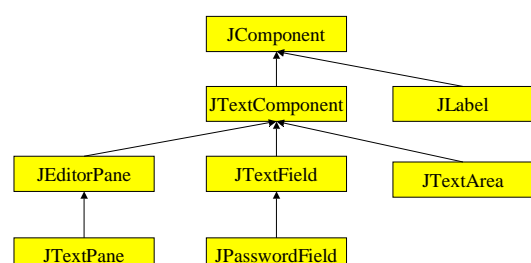
- Si plusieurs composants réalisent la même action, il est intéressant de considérer l'interface Action qui étend ActionListener.
- Utiliser des objets Action permet de centraliser, entre autres, la gestion des libellés et icônes.
- La classe AbstractButton implémente cette interface.

05/12/03

Interfaces graphiques

page 140

Les composants « texte »



05/12/03

Interfaces graphiques

page 144

JTextArea

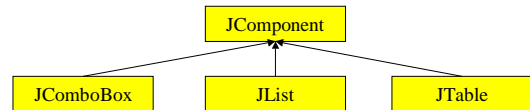
- Un objet JTextArea permet d'afficher et d'éditer plusieurs lignes de texte.
- Il est possible d'utiliser n'importe quelle police (mais une seule à la fois).
- Exemple : [TestTextArea.java](#)
- Pour gérer simultanément plusieurs polices, il faut utiliser des objets JEditorPane et JTextPane.

05/12/03

Interfaces graphiques

page 145

Les composants « liste »



05/12/03

Interfaces graphiques

page 149

JScrollPane

- Un objet JScrollPane fournit une vue « scrollable » d'un composant.
- Cela signifie qu'un ascenseur (barre de défilement) peut être utilisé lorsque la taille du composant est plus grande que celle qui lui est alloué.
- Pour utiliser un ascenseur avec un composant donné, il suffit de passer celui-ci en argument d'un constructeur de JScrollPane.

05/12/03

Interfaces graphiques

page 146

JComboBox

- Un objet JComboBox représente une liste déroulante dans laquelle l'utilisateur peut effectuer une sélection.
- On peut créer une telle liste en passant en arguments un tableau d'objets.
- L'indice de l'élément sélectionné à un moment donné est donné par la méthode `getSelectedIndex()`
- Exemple : [TestComboBox.java](#)
- Une « combo box » peut être éditable.

05/12/03

Interfaces graphiques

page 150

Politique de défilement

- Il est possible à la création d'un objet JScrollPane de préciser
 - la politique de défilement vertical
 - `JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED`
 - `JScrollPane.VERTICAL_SCROLLBAR_NEVER`
 - `JScrollPane.VERTICAL_SCROLLBAR_ALWAYS`
 - la politique de défilement horizontal
 - `JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED`
 - `JScrollPane.HORIZONTAL_SCROLLBAR_NEVER`
 - `JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS`

05/12/03

Interfaces graphiques

page 147

JList

- Un objet JList représente une liste à sélection multiple.
- Il est possible de préciser le mode de sélection avec `setSelectionMode` et l'une des constantes :
 - `ListSelectionModel.SINGLE_SELECTION`
 - `ListSelectionModel.SINGLE_INTERVAL_SELECTION`
 - `ListSelectionModel.MULTIPLE_INTERVAL_SELECTION`
- Il est possible de préciser le nombre de lignes visibles avec la méthode `setVisibleRowCount`.
- Exemple : [TestList.java](#)

05/12/03

Interfaces graphiques

page 151

JPasswordField

- Un objet JPasswordField est un composant qui permet à l'utilisateur d'éditer une ligne de texte sans montrer les caractères tapés.
- Exemple : [TestPasswordField.java](#)

05/12/03

Interfaces graphiques

page 148

JTabbedPane

- Un objet JTabbedPane représente un panneau à onglets.
- Il est ainsi possible d'avoir plusieurs composants (en règle générale des panneaux) partageant le même espace.
- Il suffit d'ajouter ceux-ci au panneau à onglets avec la méthode `addTab`.
- La logique `CardLayout` offre sensiblement le même service.
- Exemple : [TestTabbedPane.java](#)

05/12/03

Interfaces graphiques

page 152

JSlider

- Un objet JSlider est une glissière que manipule l'utilisateur pour choisir une valeur dans une gamme de valeurs entières.
- Il est possible de la positionner horizontalement ou verticalement.
- Les traits de graduation peuvent être rendus visibles avec la méthode setPaintTicks.
- Exemple : [TestSlider.java](#)

05/12/03

Interfaces graphiques

page 153

Version "classique"

- Caractéristiques
 - un tableau de booléens représentant l'état des cellules
 - un contour artificiel
 - [programme](#)
- Amélioration possible
 - initialisations
 - cellules = objets
 - interface graphique

05/12/03

Interfaces graphiques

page 157

JSplitPane

- Un objet JSplitPane représente un panneau divisé en deux parties.
- A la création, il suffit d'indiquer le mode de division (horizontal ou vertical) et de fournir les références de deux composants.
- Il est possible de préciser l'emplacement de la division avec la méthode setDividerLocation.
- Exemple : [TestSplitPane1.java](#)

05/12/03

Interfaces graphiques

page 154

Première version "objet"

- Caractéristiques
 - une cellule = un objet
 - pas de contour artificiel
 - [programme](#)
- Amélioration possible
 - initialisations
 - interface graphique

05/12/03

Interfaces graphiques

page 158

Look and Feel

- Il est possible de modifier l'apparence d'une application en Java.
- Il est par exemple possible d'utiliser les apparences :
 - metal
 - motif
 - windows
- Exemple : [Plaf.java](#)

05/12/03

Interfaces graphiques

page 155

Seconde version "objet"

- Idée : définir un objet permettant d'effectuer l'initialisation des cellules.
- Comme il y a plusieurs initialisations possibles, il est intéressant de définir un objet spécifique pour chacune d'entre elles.
- Ces objets doivent alors être considérés comme des constantes.

05/12/03

Interfaces graphiques

page 159

Jeu de la vie

Classe Initializer

- Un objet "Initializer" est caractérisé par :
 - un identifiant (numéro)
 - un nom
 - un ensemble de coordonnées de cellules (devant être créées initialement).
- On crée autant d'objets statiques qu'il y a d'initialisations possibles.

05/12/03

Interfaces graphiques

page 156

05/12/03

Interfaces graphiques

page 160

Classe Initializer

- Tous les objets statiques sont placés dans une liste chaînée. On peut ainsi :
 - connaître le nombre d'objets
 - gérer automatiquement l'identifiant de chaque objet
- [Classe Initializer](#)

05/12/03

Interfaces graphiques

page 161

Dessiner

- Pour dessiner, il faut utiliser le contexte graphique (objet Graphics) passé automatiquement à la méthode paintComponent.
- Les instructions sont alors du type :
 - g.setColor(...);
 - g.drawLine(...);
 - g.fillOval(...);

05/12/03

Interfaces graphiques

page 165

Seconde version "objet"

- Caractéristiques
 - une cellule = un objet
 - un type d'initialisation = un objet
 - [programme](#)
- Amélioration possible
 - interface graphique

05/12/03

Interfaces graphiques

page 162

Dessiner

- Un composant est automatiquement repeint lorsque cela est nécessaire (après un déplacement, une réduction, ...).
- Il est possible de repeindre explicitement un composant en appelant la méthode repaint().
- Lorsqu'un composant est repeint, c'est la méthode paintComponent qui est exécutée.

05/12/03

Interfaces graphiques

page 166

Troisième version "objet"

- Un cadre (JFrame) avec deux panels.
- Le premier panel permet de sélectionner le nombre de lignes, le nombre de colonnes, l'initialisation et de lancer puis stopper le jeu.
- Le second panel permet d'afficher la grille avec les cellules.

05/12/03

Interfaces graphiques

page 163

Interface graphique

- [programme](#)
- Critique
 - pas de contrôle sur la saisie de l'utilisateur au niveau des champs.
 - composants accessibles alors qu'ils ne devraient pas l'être

05/12/03

Interfaces graphiques

page 167

Dessiner

- Pour dessiner sur un panel, il faut redéfinir la méthode paintComponent. On obtient quelque chose comme :

```
class MyPanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        ... //
    }
}
```

05/12/03

Interfaces graphiques

page 164

Thread

- Pour pouvoir gérer un thread, il faut définir une classe héritant de Thread, puis simplement écrire une fonction run().
- Pour créer un objet Thread, il suffit alors de créer une instance de la nouvelle classe.
- Pour lancer le thread, il suffit d'appeler la méthode start() sur l'objet créé.

05/12/03

Interfaces graphiques

page 168

Troisième version "objet"

- Le jeu (objet GameOfLife) est maintenant un thread qu'il suffit de lancer quand l'utilisateur clique sur le bouton start.
- Pour pouvoir stopper le thread, on utilise un booléen (et non la méthode stop()).
- [programme](#)

05/12/03

Interfaces graphiques

page 169

Un éditeur graphique

- [Exemple : GraphicEditor.java](#)

05/12/03

Interfaces graphiques

page 173

Lien entre l'interface et le jeu

- Il est indispensable de limiter au maximum le couplage entre l'interface et la logique de l'application.
- Cela facilite nettement l'extension et la maintenance de l'application.
- Généralement, l'interface (la vue) connaît la logique (le modèle) mais pas l'inverse.

05/12/03

Interfaces graphiques

page 170

Lien entre l'interface et le jeu

- Au niveau de l'interface, on constate la présence d'un champ GameOfLife et essentiellement la création d'un objet de ce type en réponse au click sur le bouton start.
- Au niveau de la logique, on constate (malheureusement) la présence d'un champ GraphicalUI. Cela est nécessaire car à chaque nouvelle génération le jeu doit informer l'interface de redessiner.

05/12/03

Interfaces graphiques

page 171

Quatrième version objet

- Améliorations
 - contrôle sur la saisie de l'utilisateur au niveau des champs.
 - composants inaccessibles quand ils ne doivent pas l'être.
 - un bouton pause.
- [programme](#)

05/12/03

Interfaces graphiques

page 172