Interfaces graphiques

"Plus un ordinateur possède de RAM,
plus vite il peut générer un message d'erreur."

Dave Barry

Interfaces graphiques

- Historique
  - AWT
  - SWING
- Composants
  - Composants primaires
  - Composants secondaires
- Gestionnaires de placement
- Gestion des évènements

Historique

- Sun propose la première version de Java (1.0) en 1995
- A cette époque, les interfaces graphiques sont déjà très répandues. Java doit donc proposer une bibliothèque qui permette de construire des GUIs.
- La première tentative s'appelle AWT : *Abstract Windows Toolkit*.

AWT

- Les composants de AWT sont des "reflets" des composants du système d'exploitation ("composants lourds").
- Le système d'exploitation est chargé d'afficher les composants.
- Les composants sont des objets Java héritiers de la classe Component.
- Le package java.awt contient tous les éléments de AWT.

AWT

- AWT permet de construire des interfaces graphiques "à peu près" portables...
- En effet, le principe utilisé pour AWT limite le nombre de composants possibles (certains composants existent sur une architecture mais pas sur une autre)
- AWT est gourmand en ressources et donc lent.
- Toutefois, AWT propose des idées intéressantes comme les gestionnaires de placement, les gestionnaires d'événements,...

AWT -> SWING

- En 1998 Sun propose Java 1.2 ("Java 2"), une nouvelle bibliothèque de construction d'interfaces graphiques est ajoutée : **Swing**.
- Reprenant les éléments intéressants de AWT, elle propose une approche totalement différente pour la gestion des composants.
  - Les composants sont désormais gérés par la JVM ce qui assure une légèreté et une rapidité du système.
  - Seuls quelques éléments restent "lourds" par nécessité.

7

SWING

- La construction d'une application se fait en quatre parties :
  - **Construction des composants**: création via le constructeur, puis personnalisation à l'aide des différentes méthodes
  - **Ajout des composants à un conteneur**: composants placés dans un conteneur (c'est un composant capable d'en accueillir d'autres, comme une fenêtre).
  - **Placement des composants**: composants rarement placés de manière absolue, on préfère utiliser un gestionnaire de placement pour obtenir un affichage cohérent sur tous les systèmes.
  - **Gestion des évènements**: les évènements sont des objets qui transitent pour représenter les actions de l'utilisateur. Les auditeurs d'évènements les traitent pour modifier le comportement de l'application.

8

SWING

- La plupart des interfaces graphiques peuvent être programmées avec différents langages :
  - L'API Windows peut être programmée en C++ (Visual C++, Borland C++Builder), en Pascal (Borland Delphi), en Basic (Visual Basic),...
  - GTK/GTK+ peut être programmé en C/C++, en Perl, en ADA, en PHP, ...
- Swing ne peut être programmé qu'en Java et profite donc au maximum des avantages de ce langage.
- Les composants Swing sont groupés dans le package **javax.swing** et leurs noms commencent par J.

9

Principales propriétés des composants SWING

- Les classes mères
  - Tous les composants de Swing descendent de la classe *JComponent*.
  - La classe JComponent descend de la classe *Component* de AWT.
  - Les composants ont ainsi de très nombreuses méthodes communes.

10

Principales propriétés des composants SWING

- Relation entre les composants
  - Tous les composants sont capables d'accueillir d'autres composants car ils héritent de la classe *Container*.
  - On ajoute des composants à l'aide de la méthode *add*.
  - La méthode *getParent* renvoie le conteneur d'un composant.

11

Principales propriétés des composants SWING

- Propriétés géométriques
  - Dimensionnement d'un composant: méthode *setSize*.
  - Taille préférentielle d'un composant: *getPreferedSize*.
  - Position du composant: *setLocation* (le coin supérieur gauche du conteneur parent sert de référence).
  - Cacher un composant: *setVisible.*
  - Désactiver un composant (il devient grisé): méthode *setEnabled*.

12

Principales propriétés des composants SWING

- Les bordures
  - Les composants peuvent être entourés de bordures.
  - Les bordures peuvent être en creux, en relief, avec du texte, ...
  - La méthode *setBorder* permet de spécifier une bordure de type Border.
    - monComposant. setBorder(BorderFactory.createTitleBorder(" Une bordure"));

Principales propriétés des composants SWING

13

- Les bordures
  - La méthode *setBorder* permet de spécifier une bordure de type Border.
    - monComposant. setBorder(BorderFactory.createTitleBorder(" Une bordure"));

Principales propriétés des composants SWING

14

- Les infobulles
  - Les infobulles sont des aides qui apparaissent lorsque le curseur reste au dessus d'un composant.
  - La méthode *setToolTipText* permet de créer une infobulle.
    - monComposant. setToolTipText(" ceci est l'infobulle ");

Les conteneurs primaires

15

- Toute application Swing doit être basée sur au moins un conteneur primaire.
- Il assure le lien entre l'application Java et le système d'exploitation.
- Les conteneurs primaires sont les seuls composants graphiques de Swing construit par le système d'exploitation. (on parle de "composants lourds")

Les conteneurs primaires

16

- Propriétés communes
  - Les quatre conteneurs primaires (les applets, les fenêtres (JWindow et JFrame) et les boites de dialogues) sont tous basés sur le même modèle.
  - Ils contiennent un conteneur *contentPane* qui reçoit les composants graphiques. Par défaut ce conteneur est un JPanel.
  - Ils peuvent avoir une barre de menu (sauf JWindow) spécifiée par la méthode *setJMenuBar*.

Les conteneurs primaires

17

- Ajout d'un composant
  - Le conteneur est accessible via la méthode *getContentPane*.
  - Pour ajouter des composants dans un conteneur, on utilise la méthode *add*.
    - JFrame fen = new JFrame ();
      fen. getContentPane().add(monComposant);
  - Définir un nouveau conteneur: méthode *setContentPane*

Les conteneurs primaires

18

- Les fenêtres
  - Swing propose 2 types de fenêtres :
    - JWindow
    - JFrame
  - Elles sont toutes les deux héritières de Window (classe de AWT), elles partagent donc de nombreuses méthodes.
    - JFrame fen = new JFrame ();

19

Les conteneurs primaires

- Les fenêtres
  - Lors de la création une fenêtre n'est pas visible, pour la rendre visible: méthode *setVisible*.
  - Placer et redimensionner: *setLocation* et *setSize*.
  - Pour fixer la taille d'une fenêtre on peut utiliser la méthode pack qui fixe la fenêtre à sa taille idéale (vis-à-vis des composants qui sont à l'intérieur).
  - Libérer une fenêtre: méthode *dispose*. Elle est alors libérée par le système d'exploitation puis détruite par le garbage collector.
  - Une application graphique est terminée lorsque toutes ses fenêtres sont libérées.

20

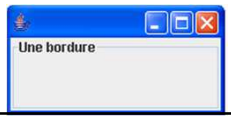
Les conteneurs primaires

- Fenêtres JWindow
  - Les fenêtres JWindow sont des fenêtres simples de l'interface graphique utilisé (Windows, X, ...).
  - Elles n'ont ni bordure, ni titre, ni icone.

21

Les conteneurs primaires

- Fenêtres JFrame
  - Les fenêtres JFrame sont des fenêtres plus complètes.
  - Elles ont une bordure, un titre et des icônes (en fonction de l'interface graphique)
  - Les applications graphiques écrites en Java sont généralement construites à partir d'une JFrame.



22

Les conteneurs primaires

- Fenêtres JFrame
  - Icône de fermeture d'une JFrame: elle est **cachée**.
  - On peut modifier ce comportement en utilisant la méthode *setDefaultCloseOperation*
    - Plusieurs comportement sont possible, ils sont définis par des constantes de la classe JFrame.
    - Souvent, on préfère fermer la fenêtre en utilisant le comportement JFrame.EXIT ON CLOSE.
  - Une JFrame a un titre, modifiable avec la méthode *setTitle*, ou en utilisant une version surchargée du constructeur.

23

Les conteneurs primaires

- Les boîtes de dialogue
  - Une boite de dialogue est modale (bloque l'application en cours).
  - La classe *JOptionPane* permet de construire des boîtes de dialogue très facilement.
  - Il n'est pas nécessaire d'instancier la classe, les principales méthodes étant statiques.
  - Afficher un message: méthode *showMessageDialog* (plusieurs versions surchargées).
    - JOptionPane. showMessageDialog(null ,"Un message");

24

Les conteneurs primaires

- Les boîtes de dialogue
  - La classe JOptionPane propose plusieurs constantes représentant différents cas comme :
    - ERROR MESSAGE pour les message d'erreur,
    - INFORMATION MESSAGE pour les messages informatifs,
    - WARNING MESSAGE pour les messages d'alerte.
  - Possibilité d'ajout de titre (voir surcharges méthode)
  - Boîte de dialogue "booléenne": méthode *showConfirmDialog*.

25

Les conteneurs primaires

- Les boîtes de dialogue
  - Boîtes de dialogue "booléenne": exemple
    - int rep = JOptionPane. showConfirmDialog(fen ," Aimez-vous Java ?" , " Examen de Java" , JOptionPane.YES_NO_OPTION);
      if (rep== JOptionPane. YES_OPTION){
      // Choix "oui"
      }
      if (rep== JOptionPane. NO_OPTION){
      // Choix "non"
      }

26

Les conteneurs secondaires

- Les conteneurs secondaires sont des composants de Swing destinés à être placés dans un conteneur primaire.
- Ce sont des composants légers (ils sont dessinés par la JVM).
- Les conteneurs peuvent être imbriqués les uns dans les autres sans aucune limitation (sauf les ressources du système...).

27

Les conteneurs secondaires

- Le panneau: JPanel
  - C'est le conteneur par défaut des composants JFrame et JWindow.
  - Un panneau est une surface d'affichage destinée à recevoir des composants (à l'aide de la méthode add).
    - JFrame fen=new JFrame();
      fen.setSize(200,100);
      fen.setVisible(true);
      JPanel panel=new JPanel();
      panel.setBorder(BorderFactory.createTitledBorder("Une bordure"));
      fen.add(panel);

28

Les conteneurs secondaires

- Panneau à onglet: JTabbedPanel
  - Composé de différentes fiches construites par la méthode *addTab*. Cette méthode admet deux paramètres :
    - une chaîne de caractères qui sera affichée sur l'onglet,
    - le composant qui va occuper la feuille.
  - Chaque fiche ne peut accueillir qu'un seul composant, généralement un JPanel.
  - Le système numérote les fiches **à partir de 0**.

29

Les conteneurs secondaires

- Label: JLabel
  - Un label est un élément statique d'un interface, on l'utilise pour afficher du texte ou une image.
  - Afficher une chaîne de caractères: métode *setText* ou version surchargée du constructeur.
  - Possibilité d'afficher une image
    - Utilisation du composant intermédiaire ImageIcon
    - Formats JPEG, GIF, PNG.
      - JLabel unLabel = new JLabel (new ImageIcon (
        panneau .add (unLabel);
        panneau.add(new JLabel ("toto"));

30

Les conteneurs secondaires

- Les boutons: classe JAbstractButton
  - Plusieurs éléments de Swing descendent de la classe abstraite *JAbstractButton*.
  - On retrouve ainsi de nombreuses méthodes communes entre les boutons, les cases à cocher, les boutons radios,...
  - Définition d'un texte: *setText*, d'une icône: *setIcon*.
  - Définir une mnémonique (forme ALT+touche) pour activer l'élement.
    - La méthode setMnemonic permet de choisir une mnémonique (sous la forme d'un paramètre int).
    - La classe KeyEvent propose les constantes pour toutes les touches du clavier (de la forme KeyEvent.VK T pour "t") ;

31

Les conteneurs secondaires

- Les boutons: JButton, JToggleButton
  - Bouton simple
    - classe JButton
    - Définition du texte et icône via le constructeur
  - Bouton bistable
    - classe JToggleButton
    - Etat du bouton: récup: méthode *isSelected*; modif: *setSelected*

32

Les conteneurs secondaires

- Les cases à cocher: JCheckBox
  - Constructeur permet l'initialisation du texte et de l'état
  - Comme pour les *JToggleButton*, l'état de la case peut être modifié avec *setSelected* et lu avec *isSelected*.

33

Les conteneurs secondaires

- Les "radio" boutons: JRadioButton
  - Constructeur permet l'initialisation du texte et de l'état
  - Comme pour les *JToggleButton*, l'état de la case peut être modifié avec *setSelected* et lu avec *isSelected*.
  - Pour que les boutons "radio" aient un comportement exclusif, ils doivent être groupés dans un *ButtonGroup*.
    - Ils sont ajoutés en utilisant la méthode add.

34

Les conteneurs secondaires

- Les "radio" boutons: JRadioButton
  - Pour que les boutons "radio" aient un comportement exclusif, ils doivent être groupés dans un *ButtonGroup*.
    - Ils sont ajoutés en utilisant la méthode add.

Sans grouper les radio boutons, ils peuvent tous être actif

Impossible en les groupant

35

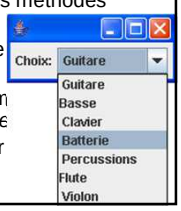
Les conteneurs secondaires

- Liste de choix: JList
  - Permet de choisir un (ou plusieurs) élément(s) parmi un ensemble.
  - Utilisent un tableau d'élements pour représenter les choix possible.
  - Définir la liste: méthode *setListData*, ou le constructeur.
  - Différents modes de sélection (méthode *setSelectionMode*)
    - ListSelectionModel.SINGLE SELECTION: sélectionner 1 seul élément
    - ListSelectionModel.SINGLE INTERVALS ELECTION: sélectionner plusieurs éléments formant un seul intervalle,
    - ListSelectionModel.MULTIPLE INTERVAL SELECTION: aucune restriction.
  - Accès aux indices des éléments (*getSelectedIndex* et *getSelectedIndices*) ou directement aux éléments (*getSelectedValue* et *getSelectedValues*).

36

Les conteneurs secondaires

- Liste "combo": JComboBox
  - Les boites "combo" permettent de choisir un élément parmi un ensemble.
  - Comme les listes de choix, elles utilisent un tableau d'éléments pour représenter les choix possibles.
  - Les objets sont ajoutés/supprimés avec les méthodes *addItem* et *removeItem*.
  - Il est toutefois possible de passer une liste constructeur.
  - L'élément sélectionné est renvoyé par la m *getSelectedItem*, son indice par *getSelecte*
  - Possibilité de rendre la liste modifiable par méthode *setEditable*.



37

Les conteneurs secondaires

- Les composants textuels
  - De nombreux composants peuvent être utilisés pour afficher ou saisir du texte.
  - La plupart descendent de *JTextComponent*.
  - Accès au texte: méthodes *getText* et *setText*.

38

Les conteneurs secondaires

- Champ de texte: JTextField
  - Permet de saisir une seule ligne de texte.
  - Forcer la largeur du champ de saisie: *setColumns*.
  - Une version modifiée de *JTextField* permet de saisir des mots de passe *JPasswordTextField*.
- Zone de texte: JTextArea
  - Permet de saisir/afficher du texte sur plusieurs lignes.
  - La largeur et la hauteur de la zone de texte peuvent être fixées avec *setColumns* et *setRows*.

39

Les conteneurs secondaires

- Autres composants
  - Panneaux
    - Panneau à défilement: *JScrollPane*
    - Panneau divisé: *JSplitPane*
    - Barre d'outils: *JToolBar*
    - Glissières: *JSlider*
  - Menus
    - *JMenu*, *JMenuBar*, *JMenuItem*,…
    - *JRadioButtonMenuItem*, *JCheckBoxMenuItem*

40

Gestionnaires de placement

- Un gestionnaire de placement assure le placement des composants à l'intérieur d'un conteneur.
- Lors de la creation d'un conteneur, un gestionnaire de placement lui est automatiquement associé. Exemples:
  - FlowLayout pour les classes JPanel et JApplet,
  - BorderLayout pour les classes JFrame et JWindow.
- Accès au gestionnaire d'un conteneur: méthode *getLayout*
- Spécifier un nouveau gestionnaire: méthode *setLayout*

41

Gestionnaires de placement

- Les gestionnaires de placement implementent l'interface *LayoutManager*.
- Principe:
  - Ils interrogent les composants à partir des méthodes *getPreferedSize* et/ou *getMinimumSize*.
  - Ensuite, ils calculent les tailles et positions des composants en fonction de ces informations.
  - Les méthodes *setSize* et *setLocation* sont ensuite utilisées pour dimensionner et placer les composants.

42

Gestionnaires de placement

- Positionnement absolu
  - Le positionnement absolu est peu recommandé, mais quelques fois utilisé.
  - Pour le forcer, on utilise la méthode *setLayout(null)*.
  - Ensuite, on place les composants avec *setLocation* et on les dimensionne avec *setSize*.

43

Gestionnaires de placement

- Gestionnaire FlowLayout
  - Le gestionnaire *FlowLayout* place les composants de gauche à droite puis de haut en bas comme un éditeur de texte.
  - L'espacement des composants peut être fixé avec *setHgap* et *setVgap*, par défaut les valeurs = 5 pixels.

44

Gestionnaires de placement

- Gestionnaire FlowLayout
  - Le gestionnaire *FlowLayout* place les composants de gauche à droite puis de haut en bas comme un éditeur de texte.
  - L'espacement des composants peut être fixé avec *setHgap* et *setVgap*, par défaut les valeurs = 5

```
Exemple:
JPanel panneau=new JPanel();
panneau . setLayout( new FlowLayout());
panneau . add( new JButton(" Un" ));
panneau . add( new JButton(" Deux " ));
panneau . add( new JButton(" Trois " ));
panneau . add( new JButton(" Quatre " ));
```

Fenêtre redim.

45

Gestionnaires de placement

- Gestionnaire GridLayout
  - Le gestionnaire *GridLayout* propose de placer les composants sur une grille régulièrement espacée.
  - Comme précédemment, les composants sont disposés de gauche à droite et de haut en bas.
  - Le constructeur permet de fixer le nombre de lignes ou le nombre de colonnes.

```
Exemple:
panneau . setLayout( new GridLayout(3,0));
panneau . add( new JButton(" Un" ));
panneau . add( new JButton(" Deux " ));
...
panneau . add( new JButton(" Sept " ));
```

46

Gestionnaires de placement

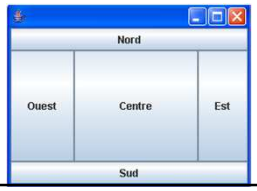
- Gestionnaire BorderLayout
  - Le gestionnaire BorderLayout sépare le conteneur en cinq zones : est, ouest, nord, sud et centre.
  - Pour placer des composants dans un conteneur géré par un *BorderLayout* on utilise une version surchargée de *add*.
    - le deuxième paramètre est un objet qui represente la position voulue,
    - des constantes sont définies dans la classe BorderLayout
      - BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST et BorderLayout.CENTER.
  - Il n'est pas obligatoire de spécifier cinq composants dans le conteneur.

47

Gestionnaires de placement

- Gestionnaire BorderLayout

```
panneau . setLayout( new BorderLayout());
panneau . add( new JButton(" Nord " ), BorderLayout.NORTH );
panneau . add( new JButton(" Sud " ), BorderLayout.SOUTH );
panneau . add( new JButton(" Est " ), BorderLayout.EAST);
panneau . add( new JButton(" Ouest " ), BorderLayout.WEST);
panneau . add( new JButton(" Centre " ), BorderLayout.CENTER);
```



48

Gestionnaires de placement

- Gestionnaire GridBagLayout
  - Le gestionnaire GridBagLayout est le plus souple et le plus complet des gestionnaires.
  - Il est basé sur une grille dans laquelle les composants sont placés.
  - Ils peuvent occuper plusieurs lignes et/ou plusieurs colonnes.
  - Les composants sont ajoutés en utilisant *add* avec comme deuxième paramètre un objet de type *GridBagConstraints*.
  - Les variables membres de la classe *GridBagConstraints* sont publiques pour en faciliter l'accès.
    - Ce sont les variables membres *gridx* et *gridy* qui représentent la position d'un composant sur la grille.
    - Le composant le plus large/haut qui fixe la largeur/hauteur d'une colonne/ligne.

49

Gestionnaires de placement

- Gestionnaire GridBagLayout

```
panneau . setLayout( new GridBagLayout());
GridBagConstraints contraintes = new GridBagConstraints();
contraintes . gridx =0;
contraintes . gridy =0;
panneau . add( new JButton("Un "), contraintes );
contraintes . gridx =1;
contraintes . gridy =0;
panneau . add( new JButton(" Deux "), contraintes );
contraintes . gridx =0;
contraintes . gridy =1;
panneau . add( new JButton(" Trois "), contraintes );
contraintes . gridx =1;
contraintes . gridy =1;
panneau . add( new JButton(" Quatre "), contraintes );
contraintes . gridx =2;
contraintes . gridy =2;
panneau . add( new JButton(" Cinq "), contraintes );
```

50

Gestionnaires de placement

- Gestionnaire GridBagLayout
  - Par défaut, les composants ne remplissent pas complètement leurs cellules (sauf le plus haut/large de chaque ligne/colonne).
  - La variable membre *fill* permet de définir le comportement de remplissage à l'aide de constantes :
    - GridBagConstraints.NONE : aucun remplissage, comportement par défaut,
    - GridBagConstraints.HORIZONTAL : remplissage dans le sens horizontal,
    - GridBagConstraints.VERTICAL : remplissage dans le sens vertical,
    - GridBagConstraints.BOTH : remplissage dans les deux sens.

51

Gestionnaires de placement

- Gestionnaire GridBagLayout

```
panneau . setLayout( new GridBagLayout());
GridBagConstraints contraintes = new GridBagConstraints();
contraintes.fill=GridBagConstraints.BOTH;
contraintes . gridx =0;
contraintes . gridy =0;
panneau . add( new JButton("Un "), contraintes );
contraintes . gridx =1;
contraintes . gridy =0;
panneau . add( new JButton(" Deux "), contraintes );
...
```

52

Gestionnaires de placement

- Gestionnaire GridBagLayout
  - L'une des caractéristiques les plus appréciées de *GridBagLayout* est la possibilité donnée aux composants d'occuper plusieurs cellules.
  - Le nombre de cases occupées est défini par les variables *gridwidth* et *gridheight*.
  - Un composant peut alors occuper indifféremment plusieurs lignes et/ou plusieurs colonnes.

```
...
contraintes . gridx =1; contraintes . gridy =0;
contraintes . gridwidth = 2;
contraintes . gridheight = 1;
panneau . add( new JButton(" Deux "), contraintes );
...
```

53

Gestionnaires de placement

- Gestionnaire GridBagLayout
  - Le poids des composants gère la manière dont les composants vont se répartir l'espace libre.
  - Les variables *weightx* et *weighty* définissent le poids d'un composant.
  - L'espace libre est partagé entre les différents composants, ceux ayant un poids le plus important occupant le plus d'espace.
  - Le poids est évalué de manière relative.

54

Gestionnaires de placement

- Gestionnaire GridBagLayout

```
panneau . setLayout( new GridBagLayout());
GridBagConstraints contraintes = new GridBagConstraints();
contraintes . fill = GridBagConstraints.BOTH;
contraintes . weighty = 1.0;
contraintes . gridx = 0;
contraintes . gridy = 0;
contraintes . weightx = 1.0;
panneau . add( new JButton("Un "), contraintes );
contraintes . gridx =1;
contraintes . gridy =0;
contraintes . weightx = 2.0;
panneau . add( new JButton(" Deux "), contraintes );
contraintes . gridx =2;
contraintes . gridy =0;
contraintes . weightx = 4.0;
panneau . add( new JButton(" Trois "), contraintes );
```

55

Les événements

- Principes: Généralités
  - La programmation par évènements est utilisée pour faire communiquer deux entités de manière non bloquante.
  - Elle peut être utilisée pour échanger des données entre deux objets.
  - L'objet qui émet un événement est nommé *source*.
  - Un objet qui reçoit un événement est un *auditeur*.
  - Il est possible d'avoir plusieurs auditeurs pour une seule source.

56

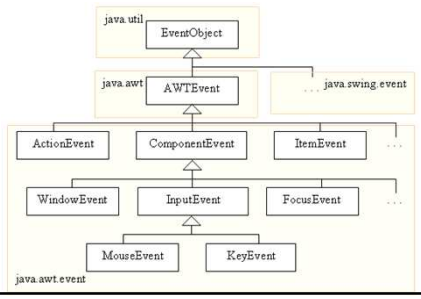
Les événements

- Evénements dans SWING
  - Les composants graphiques génèrent des événements en fonction des actions de l'utilisateur.
  - Les événements sont des descendants de *EventObject* et respectent la syntaxe *XXXEvent*.
  - Dans les composants, on trouve des méthodes *AddXXXListener* pour pouvoir enregistrer les auditeurs.
  - L'auditeur doit implémenter une interface de la forme *XXXListener*.

57

Les événements

- Evénements dans SWING: hiérarchie



58

Les événements

- Exemples d'événements: focusEvent
  - L'événement *focusEvent* est créé lorsqu'un composant prend ou perd le focus.
  - L'interface correspondante est *FocusListener* qui contient deux méthodes :
    - *focusGain(FocusEvent e)* qui est appelée quand le composant reçoit le focus,
    - *focusLost(FocusEvent e)* qui est appelée quand le composant perd le focus.
  - L'événement *FocusEvent* comprend une méthode *getOppositeComponent* qui renvoie l'autre composant impliqué dans le changement de focus.

59

Les événements

- Exemples d'événements: actionEvent
  - L'événement *actionEvent* correspond à l'action élémentaire d'un composant (click sur un bouton ou un élément de menu, touche [ENTREE] dans un champ de texte,...)
  - L'interface correspondante est *ActionListener* qui ne contient qu'une méthode *actionPerformed(ActionEvent e)*.

```
interface ActionListener{
    void actionPerformed(ActionEvent e);
}
```

60

Les événements

- Exemples d'événements: mouseEvent
  - L'événement *mouseEvent* correspond à l'action sur la souris (click,...).
  - L'interface correspondante est *MouseListener*.

```
interface MouseListener{
    void mouseClicked(MouseEvent e);
    void mouseEntered(MouseEvent e);
    void mouseExited(MouseEvent e);
    void mousePressed(MouseEvent e);
    void mouseReleased(MouseEvent e);
}
```

61

Les événements

- Exemples d'événements: keyEvent
  - L'événement *keyEvent* correspond à l'action sur une touche du clavier.
  - L'interface correspondante est *KeyListener*.

```
interface MouseListener{
   void keyTyped(KeyEvent e);
   void keyPressed(KeyEvent e);
   void KeyReleased(KeyEvent e);
}
```

62

Les événements

- D'autres types d'auditeurs
  - *ComponentListener*: pour réagir quand un composant change de taille, de position ou de visibilité.
  - *MouseMotionListener* pour réagir au déplacement de la souris au dessus du composant.

63

Les événements

- Exemple

```
public class TestBouton extends JPanel{
   private int n;
   private JButton bouton;
   private JLabel label;

   public TestBouton() {
      bouton=new JButton("Cliquer ici !!!");
      label=new JLabel("Nb clics: "+n);
      n=0;
      this.add(label);
      this.add(bouton);
   }
}
```

aucune intéraction possible

```
public static void main(String[] args) {
   JFrame fen=new JFrame();
   fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   fen.setVisible(true);
   TestBouton test=new TestBouton();
   fen.add(test);
   fen.pack();
}
```

64

Les événements


```
public class TestBouton extends Jpanel implements ActionListener {
   private int n;
   private JButton bouton;
   private JLabel label;

   public TestBouton() {
      bouton=new JButton("Cliquer ici !!!");
      bouton.addActionListener(this);
      label=new JLabel("Nb clics: "+n);
      n=0;
      this.add(label);
      this.add(bouton);
   }
   public void actionPerformed(ActionEvent e) {
      n++;
      label.setText("Nb clics: "+n);
   }
}
```

permet de signifier au bouton, l'objet qui va gérer un événement de type actionEvent qu'il créera

implémentation obligatoire, due à

65

Les événements

- Gérer plusieurs événements

66

```
public class TestBouton extends JPanel implements ActionListener{
   private int n;
   private JButton bouton1,bouton2;
   private JLabel label;
   public TestBouton() {
      bouton1=new JButton("Bouton +");
      bouton2=new JButton("Bouton -");
      label=new JLabel("Nb clics: "+n);
      n=0;
      this.add(label);this.add(bouton1);this.add(bouton2);
      bouton1.addActionListener(this);
      bouton2.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e) {
      if(e.getSource()==bouton1){
         n++;
      }
      if(e.getSource()==bouton2){
         n--;
      }
      label.setText("Nb clics: "+n);
   }
}
```

67

Les événements

- Gestion de plusieurs auditeurs

68

```
public class TestBouton extends JPanel implements ActionListener,MouseListener{
   private int n;    private JButton bouton1,bouton2;    private JLabel label;
   public TestBouton() {
      bouton1=new JButton("Bouton +");
      bouton2=new JButton("Bouton -");
      ...
      bouton1.addActionListener(this);
      bouton2.addActionListener(this);
      bouton1.addMouseListener(this);
   }
   public void actionPerformed(ActionEvent e) {
      ...
   }
   public void mouseClicked(MouseEvent e) {
     if(e.getSource()==bouton1){
        JOptionPane. showMessageDialog(null ,"Bouton 1 cliqué");
     }
   }
   public void mousePressed(MouseEvent e) {   }
   public void mouseReleased(MouseEvent e) {   }
   public void mouseEntered(MouseEvent e) {   }
   public void mouseExited(MouseEvent e) {   }
}
```

69

Aspects graphiques

- Chaque composant a un contexte graphique
- Un objet *Graphics* possède des méthodes permettant de dessiner du texte, des formes géométriques,…
  - drawLine
  - drawRect
  - drawString
  - fillRect
  - …

70

Aspects graphiques

- Exemple
  - Pour tracer dans un JPanel, il faut redéfinir la méthode paintComponent

```
class MyPanel extends JPanel {
      …
      public void paintComponent (Graphics g) {
            super.paintComponent(g);
            g.drawLine(10,10,10,80);
            g.drawLine(10,80,110,80);
            g.drawLine(110,80,10,10);
      }
}
```

paintComponent est la méthode exécutée à la création, quand on l'appelle ou quand on appelle repaint().