

Introduction aux interfaces graphiques

Christina Boura, Stéphane Lopes

christina.boura@uvsq.fr,
stephane.lopes@uvsq.fr

11 avril 2016

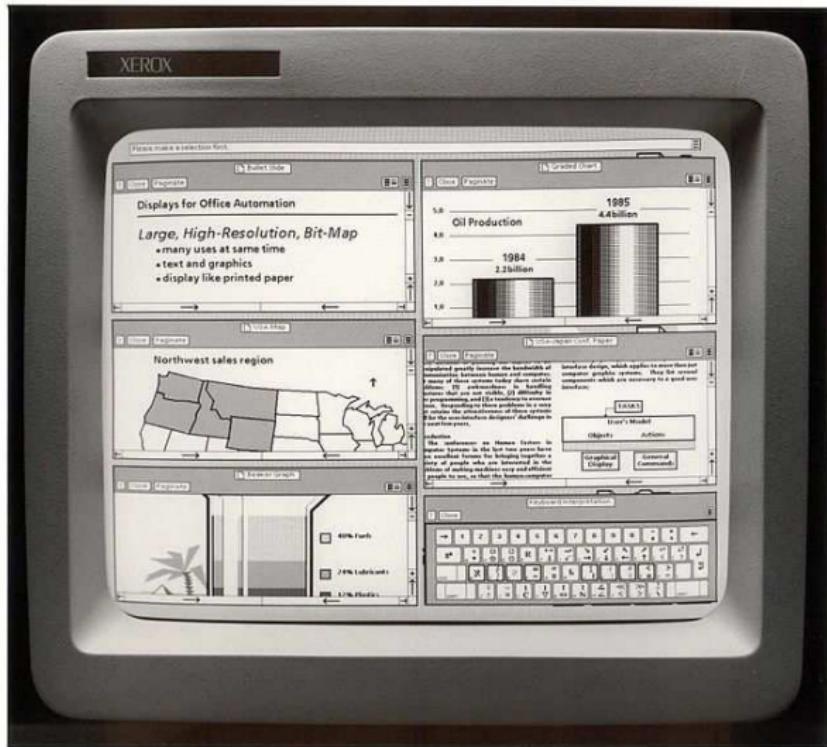


Qu'est-ce que c'est une interface graphique ?

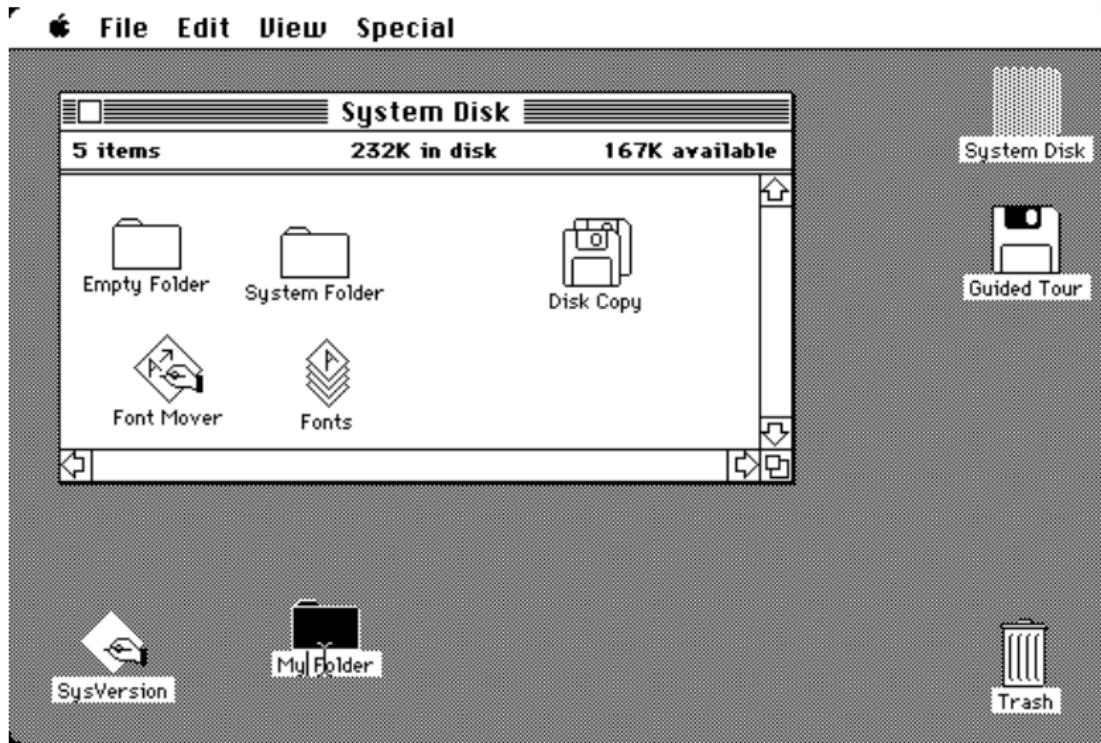
Une interface graphique (GUI en anglais) est un intermédiaire de communication entre l'utilisateur et la machine.

- Créé par les ingénieurs du Xerox PARC à la fin des années 1970, pour remplacer les interfaces en ligne de commande.
- Développée et commercialisée par Apple en 1984.

Xerox Star Workstation (1981)



Apple Macintosh Desktop (1984)



Interface en ligne de commande

Le sentiment des utilisateurs dans les années 70 :

Les usagers de nouveaux ordinateurs étaient souvent frustrés et déçus par de lourdes procédures de manipulation, des messages d'erreurs obscurs, des systèmes intolérants et confus au comportement incompréhensible, mystérieux et intimidant.

Paul A. Booth, *An Introduction to Human-Computer Interaction*



A screenshot of a Mac OS X desktop showing a terminal window titled "Terminal". The window displays a file listing from the directory "/Users/charles/Desktop". The output shows several files and their details:

```
Desktop -- bash -- 116x41
charles@MacBook-Pro:~$ ls -l
total 424
drwxr-xr-x  3 charles  staff   1628 Apr  7 12:28 Images
-rw-r--r--  1 charles  staff    1.5K Apr  9 12:43 config18.motd
-rw-r--r--  1 charles  staff    11K Apr  9 12:43 config18.log
-rw-r--r--  1 charles  staff    11K Apr  9 12:43 config18.toc
-rw-r--r--  1 charles  staff   434K Apr  9 12:43 config18.htm
-rw-r--r--  1 charles  staff    10K Apr  9 12:43 config18.pdf
-rw-r--r--  1 charles  staff    98K Apr  9 12:43 config18.xls
-rw-r--r--  1 charles  staff    10K Apr  9 12:43 config18.ppt
-rw-r--r--  1 charles  staff    4.5K Apr  9 12:43 config18.syncron.gr
-rw-r--r--  1 charles  staff    43K Apr  9 12:43 config18.toc
-rw-r--r--  1 charles  staff    10K Apr  9 12:43 config18.toc
charles@MacBook-Pro:~$ cd Desktop/
charles@MacBook-Pro:~/Desktop$ ls -l
total 18656
drwxr-xr-x  3 charles  staff   82K Mar 28 10:10 PublicIdentifier.jpg
-rw-r--r--  1 charles  staff  102K Mar 28 10:10 PublicIdentifier.pdf
-rw-r--r--  1 charles  staff  2.3M Dec 14 12:22 26c_3421.jpg
-rw-r--r--  1 charles  staff  1.1M Dec 14 12:22 26c_3421.pdf
-rw-r--r--  1 charles  staff  825K Apr  9 12:48 Screen Shot 2013-04-09 at 12.48.50.png
-rw-r--r--  1 charles  staff  1.1M Apr  9 12:48 Screen Shot 2013-04-09 at 12.48.50.pdf
-rw-r--r--  1 charles  staff  157K Jun  9 08:28 syncron.xls
charles@MacBook-Pro:~/Desktop$ rm -f former.pdf Screen Shot 2013-04-09 at 12.48.50.pdf ext_17igneocommandes.pdf
charles@MacBook-Pro:~/Desktop$ rm -f former.jpg PublicIdentifier.jpg
charles@MacBook-Pro:~/Desktop$ rm -f former.pdf PublicIdentifier.pdf
charles@MacBook-Pro:~/Desktop$ rm -f former.jpg PublicIdentifier.jpg
charles@MacBook-Pro:~/Desktop$ rm -f former.pdf Screen Shot 2013-04-09 at 12.48.50.pdf
charles@MacBook-Pro:~/Desktop$ rm -f former.pdf 17igneocommandes.xls
charles@MacBook-Pro:~/Desktop$
```

Interface graphique

Une **interface graphique** est formée d'une ou plusieurs **fenêtres** qui contiennent divers **composants graphiques** tels que

- boutons
- listes déroulantes
- menus
- champs texte
- etc.



Comment composer une interface graphique

- Une interface graphique est **composée d'un ensemble de composants graphiques** (boutons, zones de texte, menus,...)
- Certains composants peuvent contenir d'autres composants.

On peut

- indiquer les **dimensions** et la **position** d'un sous-composant dans un composant.
- employer des **gestionnaires de répartition** pour positionner les sous-composants dans les composants.

Programmation avec interface graphique

- L'utilisateur peut **interagir** à tout moment avec plusieurs objets graphiques : bouton, liste déroulante, menu, champ texte, etc.
- Ces actions peuvent **modifier** totalement le cheminement du programme
- L'**ordre d'exécution** des instructions **ne peut être prévu** à l'avance, notamment à l'écriture du code.

Librairies pour les interfaces graphiques en Java

- **JDK \leq 1.1** : AWT (Abstract Window Toolkit)
 - Package `java.awt`
 - Toujours supporté mais n'est plus utilisé pour de nouveaux développements
 - Certains modules d'**AWT** sont encore utilisés, en particulier `java.awt.event` pour la gestion des événements.
- **JDK \geq 1.2** : Introduction de nouveaux composants graphiques, dits **swing**.
 - Package `javax.swing`
 - Les noms de composants **swing** commencent (souvent) par **J**

Composants *légers* vs composants *lourds*

- Composants **lourds** (heavyweight) : **Dépendent** du système de fenêtrage local et utilisent les ressources du système hôte.
 - Les composants graphiques de **AWT** sont des composants lourds.
- Composants **légers** (lightweight) : **Ne dépendent pas** du système de fenêtrage sur lequel l'application s'exécute.
 - Composants écrits uniquement en **Java**.
 - La plupart des composants **swing** (pas tous) sont des composants légers.

Structure générale d'une application Swing

On peut distinguer dans une application Swing deux types d'objets :

- Les **conteneurs**, destinés, à contenir les objets du second type nommés **composants**.
- Les conteneurs sont également des composants.
- Ces objets héritent de la même classe `javax.swing.JComponent`.
- Possible de placer des conteneurs dans des conteneurs.

Application Swing et fenêtres

- La fenêtre principale (main window ou frame) d'une application est un **conteneur de premier niveau**.
- Ce ne sont pas des sous classes de **JComponent**.
- Un conteneur de premier niveau peut contenir des objets d'une des sous-classes de **JComponent**, mais n'est pas destiné à être contenu dans un autre conteneur de premier niveau.
- Les principaux conteneurs de premier niveau sont **JFrame**, **JDialog** et **JApplet**.
- Ce sont les seuls composants **Swing** dépendants du système de fenêtrage local (heavyweight component).
- Toute application **Swing** doit comporter **au moins un** conteneur de premier niveau.

La classe JFrame

Un *cadre* est un **conteneur de premier niveau** qu'on obtient en instanciant la classe `JFrame`.

- Un des rares composants `swing` construit à partir de système de fenêtrage local.
 - Son **apparence** (barre de titre et contour) correspond à celle des fenêtres de la plate-forme utilisée.
- Peut posséder une barre des menus (par défaut il y en a pas).
- Dispose d'un **conteneur** qui recouvre tout l'intérieur de la fenêtre.

Créer une première fenêtre

- Un **frame** est une fenêtre avec un titre et une bordure et qui peut supporter des boutons.
- Un **frame** en Java est une instance de la classe **JFrame**.

```
import javax.swing.JFrame;  
  
public class MaPremiereFenetre {  
  
    public static void main(String[] args){  
        JFrame fenetre = new JFrame();  
    }  
}
```

Rendre la fenêtre visible

Utiliser la méthode

```
public void setVisible(boolean b)
```

de la classe [Window](#).

```
import javax.swing.JFrame;  
  
public class MaPremiereFenetre {  
  
    public static void main(String[] args){  
        JFrame fenetre = new JFrame();  
        fenetre.setVisible(true);  
    }  
}
```



Rendre la fenêtre un peu plus utile

```
import javax.swing.JFrame;

public class MaPremiereFenetre {

    public static void main(String[] args){
        JFrame fenetre = new JFrame();

        /* Donner un titre à notre fenêtre */
        fenetre.setTitle("Voici une fenêtre");

        /* Définir sa taille : 300 pixels de large et 200 pixels de haut */
        fenetre.setSize(300, 200);

        /* Positionner la fenêtre au centre de l'écran */
        fenetre.setLocationRelativeTo(null);

        /* Terminer l'application lorsqu'on clique sur la croix rouge */
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Rendre la fenêtre visible */
        fenetre.setVisible(true);
    }
}
```

Le résultat

```
new JFrame();  
  
    à r à f à * /  
    /oïc (red) (yellow) (green) Voici une fenêtre  
  
Le : */  
Ø, 2  
  
Fené  
nRel  
  
icat  
Clos  
  
re v  
[true],
```

Fermer une fenêtre

```
public void setDefaultCloseOperation(int  
                                operation)
```

Indiquer l'opération qui doit se produire quand l'utilisateur **ferme** la fenêtre.

Quatre choix possible :

- **DO NOTHING ON CLOSE**.
- **HIDE ON CLOSE** : render la fenêtre invisible (mode **par défaut**).
- **DISPOSE ON CLOSE** : cacher la fenêtre et libérer toutes les ressources associées ; elles seront réallouées si la fenêtre redevient visible.
- **EXIT ON CLOSE** : terminer le programme.

Créer notre propre classe Fenêtre

Créer une classe **Fenetre** pour pouvoir créer facilement autant de fenêtres qu'on veut.

```
import javax.swing.JFrame;

public class Fenetre extends JFrame {

    public Fenetre(String titre, int large, int haut) {

        this.setTitle(titre);
        this.setSize(large, haut);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
}
```

Hiérarchie de composition

- Chaque composant **Swing** d'une application fait partie d'une (et une seule) **hiérarchie de composition**.
- Chaque hiérarchie de composition est un **arbre** qui a pour racine un conteneur de premier niveau.
- Chaque conteneur de premier niveau contient
 - un **panneau de contenu** (content pane) qui est un **conteneur intermédiaire**.
 - un menu positionné dans le conteneur de 1er niveau mais en dehors du panneau de contenu (**optionnel**).
- Les principaux conteneurs intermédiaires sont **JPanel**, **JScrollPane** (panneaux défilants) et **JTabbedPane** (panneaux à onglets).
- Les composants tels que les boutons ou labels sont appelés **composants atomiques**.
- Il existe de nombreux composants (**JButton**, **JLabel**, ...)

La classe JPanel

La classe Panel est un **conteneur intermédiaire** dont le rôle est d'**accueillir d'autres objets** de même type ou des objets de type composant.

- On peut utiliser une instance de la classe **JPanel** pour faire un dessin dedans.
- On appelle une telle instance **panneau**.
- Il est déconseillé d'utiliser les deux fonctionnalités (placer des objets et dessiner) en même temps.

Instancier la classe JPanel

```
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Fenetre extends JFrame {

    public Fenetre(String titre, int large, int haut) {
        this.setTitle(titre);
        this.setSize(large, haut);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Instanciation d'un objet JPanel */
        JPanel conteneur = new JPanel();

        /* Definir une couleur pour notre conteneur */
        conteneur.setBackground(Color.CYAN);

        /* Notre objet JPanel sera le conteneur de notre fenêtre */
        this.getContentPane(conteneur);
        this.setVisible(true);
    }
}
```

Une fenêtre . . bleue



Dessiner dans un composant (I)

La classe **Component** possède la méthode

```
public Graphics getGraphics()
```

qui renvoie un objet de type **Graphics** correspondant au composant concerné.

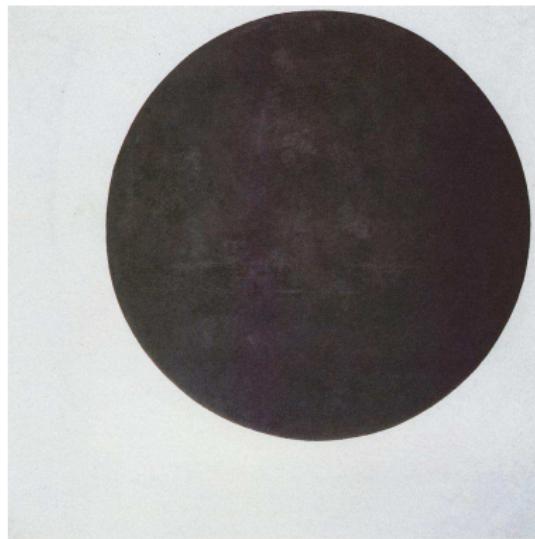
- La classe **Graphics** modélise un crayon destiné à dessiner dans un composant qui saurait dessiner un ensemble de formes géométriques.
- Méthodes pour tracer des droites, des rectangles, des ovales pleins ou non, du texte, des images ...

Dessiner dans un composant (II)

- Le système de coordonnées d'un composant s'étend de `(0,0)` à `(largeur - 1, hauteur - 1)` pixels.
- La `bordure` d'un composant est prise à l'intérieur du composant.
- La méthode `paintComponent` reçoit un paramètre de type `Graphics` qui fournit à la fois un `contexte graphique` et les `méthodes de dessin`.
- Un `contexte graphique` regroupe un ensemble d'informations pour l'affichage (couleurs de dessin, ...).
- Quelques méthodes de dessin
 - `drawLine`, `drawRect`, `fillRect`, `drawOval`, `fillOval`,
`drawString` ...

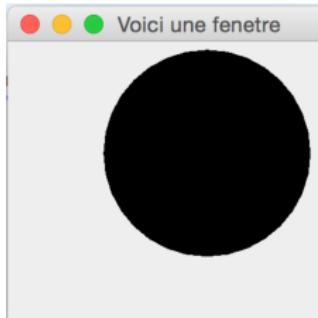
Faire de l'art moderne avec Java

Kazimir Malevich
Black circle -1923



Faire de l'art moderne avec Java

```
import java.awt.Graphics;  
  
import javax.swing.JPanel;  
  
public class Panneau extends JPanel {  
  
    public void paintComponent(Graphics g){  
  
        g.fillOval(60, 5, 130, 130);  
  
    }  
}
```



Dessiner des rectangles et écrire du texte

```
import java.awt.Color;
import java.awt.Graphics;

import javax.swing.JPanel;

public class Panneau extends JPanel {

    public void paintComponent(Graphics g){

        g.drawRoundRect(10, 10, 50, 100, 10, 10);
        g.fillRoundRect(155, 65, 135, 30, 5, 5);
        g.setColor(Color.red);
        g.drawString("Deux rectangles !", 100, 150);

    }
}
```



La classe JButton

- Un bouton est un élément graphique sur lequel l'utilisateur peut cliquer pour déclencher une action.
- Le bouton ne fait aucune action tant que l'utilisateur n'a pas cliqué dessus.
- Instancier la classe JButton.

Instancier avec un libellé

```
JButton bouton1 = new JButton("Un bouton");
```

Instancier puis définir le libellé

```
JButton bouton2 = new JButton();  
bouton2.setText("Un autre bouton");
```

Un premier bouton

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Fenetre extends JFrame{

    private JPanel panneau = new JPanel();
    private JButton bouton = new JButton("Un bouton");

    public Fenetre(){
        this.setTitle("Une fenêtre avec un bouton");
        this.setSize(300, 200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocationRelativeTo(null);

        panneau.add(bouton);
        this.setContentPane(panneau);
        this.setVisible(true);
    }
}
```



Placer des composants : les Layout Managers

Positionner des composantes dans un panneau sans utiliser des unités de distance.

Plusieurs choix en Java :

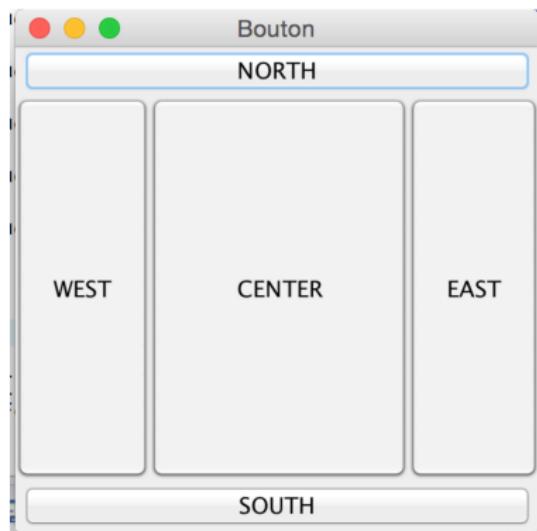
- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- GroupLayout
- SpringLayout

Positionner plusieurs boutons à l'aide d'un BorderLayout

```
public class Fenetre extends JFrame{  
  
    public Fenetre(){  
        this.setTitle("Un bouton");  
        this.setSize(300, 300);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setLocationRelativeTo(null);  
  
        this.setLayout(new BorderLayout());  
  
        add(new JButton("Button 1"), BorderLayout.CENTER);  
        add(new JButton("Button 2"), BorderLayout.NORTH);  
        add(new JButton("Button 3"), BorderLayout.SOUTH);  
        add(new JButton("Button 4"), BorderLayout.WEST);  
        add(new JButton("Button 5"), BorderLayout.EAST);  
  
        this.setVisible(true);  
    }  
}
```

L'objet BorderLayout

- Placer les composants par rapport à une position cardinale du conteneur (**NORTH**, **SOUTH**, **WEST**, **EAST**, **CENTER**).
- Définir le layout à utiliser avec la méthode **setLayout()** de la classe **JFrame**.



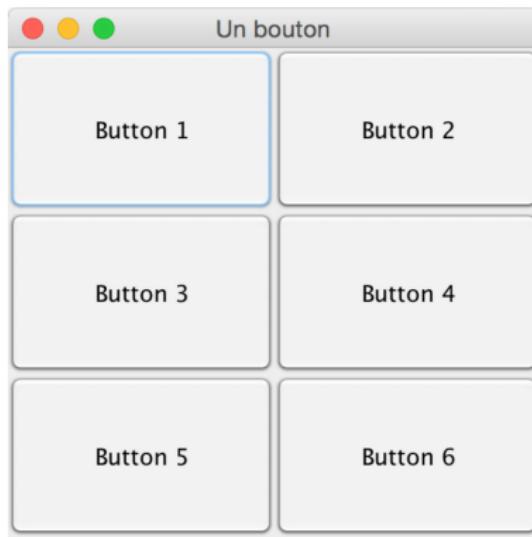
L'objet GridLayout

- Ajouter des composants suivant une grille définie par un nombre de lignes et de colonnes.
- Les composants sont placés à partir de la case située en haut à gauche.

```
this.setLayout(new GridLayout(3,2));  
  
add(new JButton("Button 1"));  
add(new JButton("Button 2"));  
add(new JButton("Button 3"));  
add(new JButton("Button 4"));  
add(new JButton("Button 5"));  
add(new JButton("Button 6"));
```

Une grille 3×2

Placer 6 boutons dans 3 lignes et 2 colonnes.



L'objet FlowLayout

- Le **layout** le plus facile à utiliser.
- Son rôle : **Centrer** les composants dans le conteneur.
- Passe à la ligne suivante dès qu'il n'y plus de place.

```
this.setLayout(new FlowLayout());  
  
add(new JButton("Button 1"));  
add(new JButton("Button 2"));  
add(new JButton("Button 3"));  
add(new JButton("Button 4"));  
add(new JButton("Button 5"));  
add(new JButton("Button 6"));  
add(new JButton("Button 7"));  
add(new JButton("Button 8"));
```

Position des composants en utilisant FlowLayout



Créer une Classe Bouton

```
import javax.swing.JButton;  
  
public class Bouton extends JButton {  
  
    private String nom;  
  
    public Bouton(String nom){  
        super(nom);  
        this.nom = nom;  
    }  
}
```

- Un seul champ représentant le nom du bouton.

L'objet Graphics2D

- Amélioration de l'objet **Graphics**
- **Utilisation** : Caster l'objet **Graphics** en **Graphics2D**
 - `g2d = (Graphics2D) g`
- Peindre des objets avec des **dégradés** de couleurs.
 - Classe **GradientPaint**.

Utiliser GradientPaint

```
GradientPaint gp = new GradientPaint(0, 0,  
Color.RED, 40, 40, Color.ORANGE, true);
```

- **1^{er} paramètre** : Coordonnée *x* où commence la **première** couleur
- **2^{ième} paramètre** : Coordonnée *y* où commence la **première** couleur
- **3^{ième} paramètre** : **Première** couleur
- **4^{ième} paramètre** : Coordonnée *x* où commence la **seconde** couleur
- **5^{ième} paramètre** : Coordonnée *y* où commence la **seconde** couleur
- **6^{ième} paramètre** : **Seconde** couleur
- **7^{ième} paramètre** : Un **booléen** indiquant si le dégradé doit se répéter

Personnaliser notre bouton

```
public class Bouton extends JButton {  
  
    private String name;  
    public Bouton(String str){  
        super(str);  
        this.name = str;  
    }  
  
    public void paintComponent(Graphics g){  
  
        Graphics2D g2d = (Graphics2D)g;  
        GradientPaint gp = new GradientPaint(0, 0, Color.RED, 0, 20, Color.ORANGE, true);  
        g2d.setPaint(gp);  
        g2d.fillRect(0, 0, this.getWidth(), this.getHeight());  
        g2d.setColor(Color.white);  
        g2d.drawString(this.name, this.getWidth()/2 - this.getWidth()/4, (this.getHeight()/2) + 5);  
    }  
}
```



Attribuer une action à un bouton

- On doit indiquer ce qui doit se passer **en cas de clic**.

Principe des **événements** et des **écouteurs**.

- Lors du clic sur un bouton, un **événement** se produit.
- Cet événement est envoyé à tous les **écouteurs** du bouton.
- Mettre un écouteur sur ce bouton et **réagir** en cas d'événement.

Programmation événementielle

Programmation séquentielle

- Un programme est une succession d'instructions de contrôle et d'entrées/sorties.
- Le programme passe son temps à attendre une entrée.
- C'est le programmeur qui définit le flux d'exécution.

Programmation événementielle

- Le programme ne fait rien jusqu'à ce que l'utilisateur fasse quelque chose.
- Le flux d'exécution du programme est contrôlé par l'utilisateur.
- On ne peut pas supposer que l'utilisateur va suivre un chemin d'exécution prédéfini.

Gestion des événements (I)

- Un **événement** est ce qui est renvoyé par le système d'exploitation supportant les interfaces graphiques au programme lorsque une pression sur les touches du clavier ou sur les boutons de la souris est effectuée.
- Des objets **sources d'événements** (bouton, barre de défilement, ...) transmettent les événements à des objets **écouteurs** d'événements.
- Un **objet écouteur** d'événements est une instance d'une classe qui implémente une interface appelée **interface écouteur**.
- Un **objet source** d'événements permet de recenser les objets écouteurs et de leur envoyer des objets événements.
- Lorsqu'un événement se produit, la source d'événement envoie l'objet événement à tous les écouteurs recensés.

Gestion des événements (II)

L'**écouteur d'événement** d'un composant d'interface, est souvent le **conteneur** de ce composant.

- Un **objet écouteur** (par exemple un `JPanel`) contient un à plusieurs **objets source d'événements** (par exemple des `JButton`)
- L'écouteur s'enregistre auprès des sources d'événements afin de pouvoir les écouter (Le `JPanel` doit implémenter l'interface `ActionListener`)
- Lorsqu'un événement se produit, l'écouteur reçoit un objet **ActionEvent**.
- La méthode `getSource()` (héritée de `EventObject` superclasse des classes événements) permet alors de déterminer quel est l'objet source de l'événement.

L'interface ActionListener

```
public interface ActionListener extends  
EventListerner
```

Cette interface déclare une seule méthode

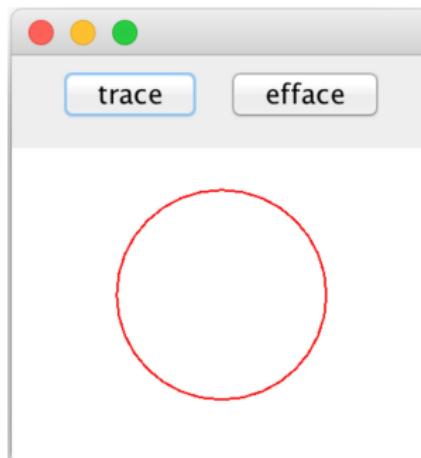
```
void actionPerformed(ActionEvent)
```

- Cette méthode doit indiquer ce qui doit se passer quand un événement se produit.

Exercice

Créer une fenêtre avec deux boutons.

- Si le bouton **trace** est appuyé, un cercle **rouge** se dessine sur la fenêtre.
- Si le bouton **efface** est appuyé, le cercle s'efface.



Première étape : créer et configurer le panneau

```
public class Ardoise extends JPanel{  
  
    boolean cercle;  
    public Ardoise() {  
        setBackground(Color.WHITE);  
        setPreferredSize(new Dimension(200,150));  
    }  
  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        if(cercle) {  
            g.setColor(Color.RED);  
            g.drawOval(50,20,100,100);  
        }  
    }  
}
```

Méthodes utilisées (I)

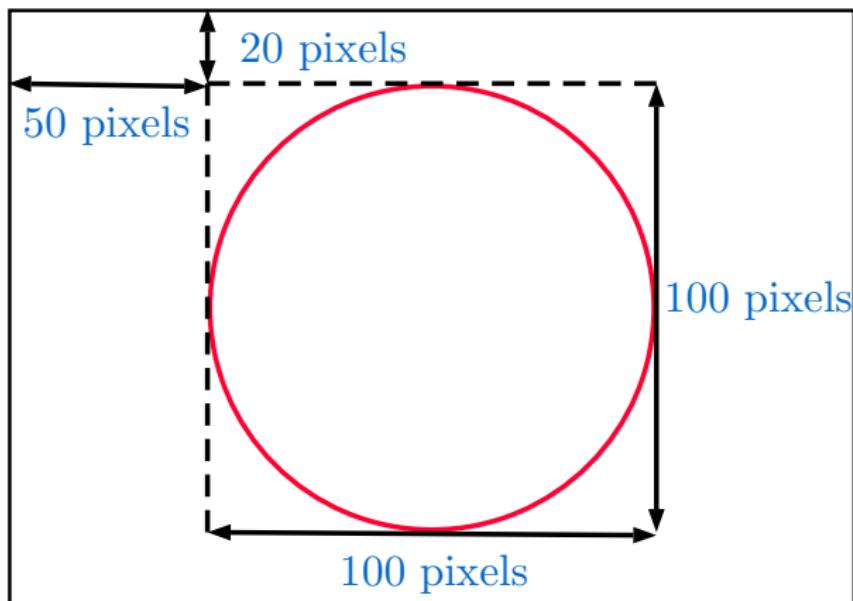
La classe `Ardoise` correspond au panneau (la partie blanche).

- `setBackground(Color.WHITE);` : L'arrière-plan du composant sera blanc.
- `setPreferredSize(new Dimension(200, 150));` : Le panneau aura 200 pixels de large et 150 pixels de haut.
 - Cette méthode de la classe `javax.swing.JComponent` permet de choisir la taille d'un composant graphique.
 - La classe `Dimension` appartient au package `java.awt`.

Méthodes utilisées (II)

- `paintComponent(Graphics g);` : Méthode appelée automatiquement quand le composant doit être dessiné ou redessiné.
 - Elle "paint" la couleur du fond du composant.
 - Si on veut ajouter des spécificités, cette méthode doit être redéfinie.
- `super.paintComponent(g);` : Nécessaire quand on redéfinit la méthode `paintComponent` de faire appel à la même méthode de la superclasse pour qu'elle effectue son propre travail (coloriage du fond).
- `g.setColor(Color.RED);` : Les tracés effectués avec g seront rouges.

Les coordonnées du cercle



Créer et configurer la fenêtre

```
public class Boutons extends JFrame implements ActionListener{  
  
    JButton trace = new JButton("trace");  
    JButton efface = new JButton("efface");  
    Ardoise ardoise = new Ardoise();  
  
    public Boutons() {  
        setLayout(new BorderLayout(5,5));  
        JPanel lesBoutons = new JPanel();  
        lesBoutons.add(trace);  
        lesBoutons.add(efface);  
        add(lesBoutons, BorderLayout.NORTH);  
        add(ardoise, BorderLayout.CENTER);  
  
        trace.addActionListener(this);  
        efface.addActionListener(this);  
        pack();  
    }  
}
```

Commentaires (I)

- `setLayout(new BorderLayout(5,5))` : Le conteneur de la fenêtre disposera d'un gestionnaire de répartition du type `BorderLayout`.
 - Laisser une marge de `5 pixels` à droite et à gauche de chaque sous-composant ajouté.
- `lesBoutons.add(trace);` : Ajouter le bouton `trace` au panneau `lesBoutons`.
- Utilisation implicite du gestionnaire de réparation `FlowLayout`.
 - Les deux boutons sont centrés.

Commentaires (II)

- `add(lesBoutons, BorderLayout.NORTH);` : Le panneau contenant les boutons est ajouté au conteneur de la fenêtre positionné au nord du composant.
- `pack();` : Dimensionner la fenêtre concernée en fonction de composants qu'elle contient et des gestionnaires de répartition.

Commentaires (III)

- `trace.addActionListener(this);` : Établir une correspondance entre le bouton trace et l'`ActionListener` indiqué en argument.

De façon générale , la méthode

```
addActionListener(ActionListener listener)
```

permet d'effectuer une correspondance entre le clic d'un utilisateur sur le bouton et une action que le programme doit effectuer suite à ce clic.

- `this` est un `ActionListener` et dispose donc de la méthode `actionPerformed` :
 - Lorsque on clique sur le bouton cette méthode est exécutée.

La méthode actionPerformed

```
public void actionPerformed(ActionEvent e) {  
    if(e.getSource() == trace) {  
        ardoise.cercle = true;  
    }  
    else ardoise.cercle = false;  
    ardoise.repaint();  
}
```

Commentaires

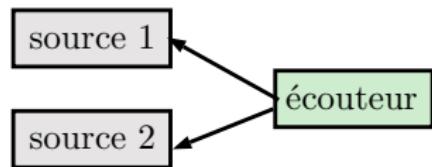
- La méthode `actionPerformed` est appelée lorsqu'un clic sur un des boutons `trace` ou `efface` s'est produit.
- Lors d'un clic, le système avertit les différents programmes en cours d'exécution.
 - Le programme note certaines informations liées à l'événement dont il est averti dans un objet héritant de la classe `java.awt.event.ActionEvent`.
 - La classe `ActionEvent` hérite de cette classe et encapsule les informations concernant le clic.
- `e.getSource()` : Renvoie l'objet qui est à l'origine de l'événement.
- `ardoise.repaint()` : Appel à la méthode `paintComponent` du composant concerné.

La classe de test

```
public class EssaiBoutons1 {  
  
    public static void main(String[] args) {  
        JFrame fenetre = new Boutons();  
  
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        fenetre.setLocation(100,100);  
        fenetre.setVisible(true);  
    }  
}
```

- L'ordre des méthodes `setLocation` et `setVisible` est important.

Un écouteur enregistré sur plusieurs sources



Rendre un bouton disponible ou non



- Utiliser la méthode `setEnabled(boolean b)` en lui passant le paramètre `true` ou `false`.

Utiliser des classes internes

Une **classe interne** est une classe à l'intérieur d'une classe ou à l'intérieur d'un bloc quelconque.

Avantage : Les attributs de la classe **externe** sont accessibles dans la classe **interne**.

- Approche très populaire lorsqu'on travaille avec des **interfaces graphiques**.

Solution avec des classes internes

```
public class Boutons2 extends JFrame {  
  
    public Boutons2() {  
  
        final Ardoise ardoise = new Ardoise();  
        add(ardoise, BorderLayout.CENTER);  
  
        JPanel lesBoutons = new JPanel();  
        final JButton trace = new JButton("trace");  
        final JButton efface = new JButton("efface");  
        lesBoutons.add(trace);  
  
        class EcouteTrace implements ActionListener {  
  
            public void actionPerformed(ActionEvent e) {  
                ardoise.cercle = true;  
                ardoise.repaint();  
                trace.setEnabled(false);  
                efface.setEnabled(true);  
            }  
        }  
        trace.addActionListener(new EcouteTrace());  
  
        lesBoutons.add(efface);  
        efface.setEnabled(false);  
        efface.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                ardoise.cercle = false;  
                ardoise.repaint();  
                trace.setEnabled(true);  
                efface.setEnabled(false);  
            }  
        });  
  
        add(lesBoutons, BorderLayout.NORTH);  
        pack();  
    }  
}
```

Commentaires (I)

- **Rappel** : Afin qu'une action se produise lorsqu'on clique sur un bouton, il faut enregistrer un **écouteur** de type **ActionListener** sur ce bouton.
- **Avantage** de l'approche avec des classes internes : Décrire la conduite à tenir à l'endroit où l'écouteur est enregistré.

Les classes internes sont utilisées de **deux façons**.

- Bouton **trace** : Définir une classe interne au constructeur de la classe **Boutons2**.
 - Les champs et les variables locales de la classe **Boutons2** **sont visibles** de la classe interne.
- Un clic sur le bouton **trace** conduira à utiliser la méthode **actionPerformed** de la classe **EcouteTrace**.

Commentaires (II)

- Bouton **trace** : Raccourci de la solution précédente en utilisant une classe interne **anonyme**.
- **Classe anonyme** : Classe créée dans une instruction sans nom.
- La variable **ardoise** est déclarée **final** : On ne peut utiliser que des variables locales constantes à l'intérieur d'une classe interne.