

# Interfaces graphiques (II)

**Christina Boura, Stéphane Lopes**

[christina.boura@uvsq.fr](mailto:christina.boura@uvsq.fr),  
[stephane.lopes@uvsq.fr](mailto:stephane.lopes@uvsq.fr)

18 avril 2016



# Agenda du jour

Événements liés à une fenêtre

Gérer des listes de choix

Utiliser une zone de texte

Gérer les clics de la souris

## Les événements concernant une fenêtre

Les événements concernant une fenêtre sont modélisés par la classe [WindowEvent](#).

Cette classe sera instanciée pour les **sept événements** suivant qui peuvent concerner une fenêtre :

- ouverture
- fermeture par l'utilisateur
- fermeture par le programme à l'aide de la méthode `dispose` de la fenêtre
- activation
- désactivation
- iconification
- désiconification

## WindowListener

Si un événement se produit dans une fenêtre (variable `fenetre`), le gestionnaire d'événements du programme "regarde" s'il y a un (ou plusieurs) écouteurs pour ce type d'événements enregistré(s) sur `fenetre`.

Un tel enregistrement se fait par l'instruction

```
fenetre.addWindowListener(unWindowListener);
```

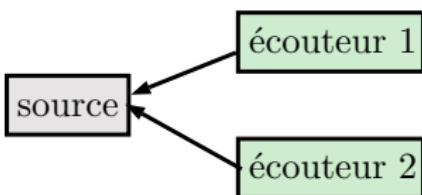
où `unWindowListener` est une instance d'une classe qui implémente l'interface `WindowListener`.

# Les méthodes de l'interface WindowListener

- public void windowOpened(WindowEvent e)
- public void windowClosing(WindowEvent e)
- public void windowClosed(WindowEvent e)
- public void windowActivated(WindowEvent e)
- public void windowDeactivated(WindowEvent e)
- public void windowIconified(WindowEvent e)
- public void windowDeiconified(WindowEvent e)

# Plusieurs écouteurs enregistrés sur une même source

On peut enregistrer plusieurs écouteurs sur un même objet.



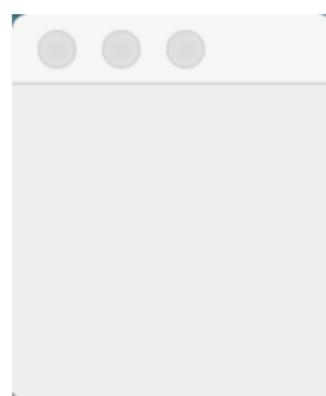
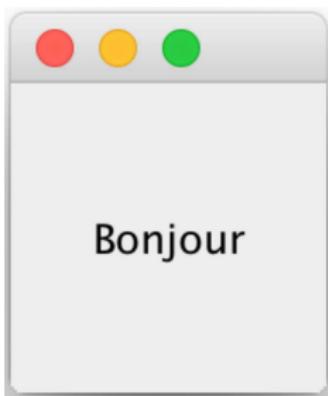
**Exemple :**

```
fenetre.addWindowListener(unWindowListener);  
fenetre.addWindowListener(unAutreWindowListener);
```

- Si un événement concernant **fenetre** se produit, les méthodes correspondantes de deux **écouteurs** sont effectuées successivement **mais** dans un ordre pas garanti.

## Application : Fenêtre active

- Lorsque la fenêtre est activée, le mot **Bonjour** s'y inscrit.
- Lorsque la fenêtre est désactivée, le mot **Bonjour** s'efface.



Active selon plateforme : Cliquer dessus ou y faire entrer l'icône de la souris.

## JLabel

Comment afficher le mot "Bonjour" dans la fenêtre ?

La classe **JLabel** permet d'afficher un texte ou une image non sélectionnable.

```
JLabel etiquette = new JLabel("Bonjour");
```

Pour placer l'étiquette **au milieu** de la fenêtre :

```
JLabel etiquette  
= new JLabel("Bonjour", SwingConstants.CENTER);
```

# Utilisation d'un WindowListener

```
public class ActionsFenetre implements WindowListener {
    JLabel etiquette;

    ActionsFenetre(JLabel etiquette) {
        this.etiquette = etiquette;
    }

    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }

    public void windowActivated(WindowEvent e) {
        etiquette.setText("Bonjour");
    }

    public void windowDeactivated(WindowEvent e) {
        etiquette.setText("");
    }

    public void windowOpened(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowClosed(WindowEvent e)
}
```

## Commentaires

- `System.exit(0);` : Terminer l'execusion du programme ; la valeur 0 indique une sortie dans les conditions normales.
- `SwingConstants.CENTER` : La classe `SwingConstants` définit un ensemble de constantes qui servent à effectuer des positionnements dans des composants.
- `addWindowListener(actions)` : Quand il y a un événement concernant la fenêtre, l'instance `actions` passée en paramètre agit avec la méthode correspondante au type d'événement.

## La classe FenetreActive

```
public class FenetreActive extends JFrame{  
  
    public FenetreActive() {  
        setSize(100,120);  
        JLabel etiquette = new JLabel("Bonjour", SwingConstants.CENTER);  
        add(etiquette, BorderLayout.CENTER);  
        ActionsFenetre actions = new ActionsFenetre(etiquette);  
        addWindowListener(actions);  
    }  
}
```

# Agenda du jour

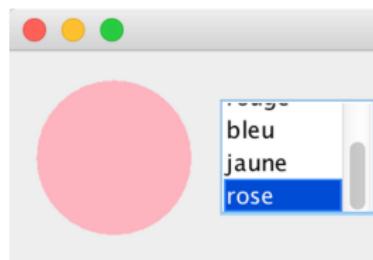
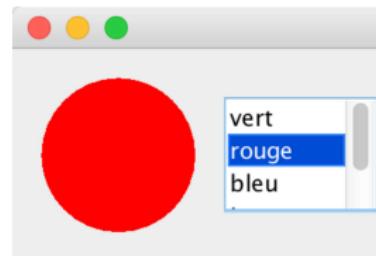
Événements liés à une fenêtre

Gérer des listes de choix

Utiliser une zone de texte

Gérer les clics de la souris

# Exercice



## La classe JScrollPane

- Une instance de la classe `JScrollPane` est un **conteneur intermédiaire** qui permet de munir un composant de **barres de défilement**.
- Ceci permet de visualiser des composants **plus grands** que l'espace dans lequel ils sont visualisés.



## Constructeurs de la classe JScrollPane

```
JScrollPane panneau = new JScrollPane();
```

- Constructeur qui prend en paramètre un **composant**.

```
JScrollPane panneau = new JScrollPane(Component  
view);
```

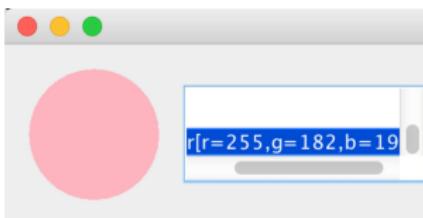
- Les **barres de défilement** ne sont visibles que si **nécessaire**.

## Créer notre propre classe Couleur

On peut toujours utiliser la classe **Color** de Java, mais la méthode **toString** de celle-ci n'est pas très sympathique ...

La méthode **toString** de la classe **Color** retourne la chaîne suivante pour la couleur **rose** :

```
java.awt.Color[r=255,g=175,b=175]
```



On veut **redéfinir** la méthode **toString** pour qu'elle retourne la chaîne **rose**.

# La classe Couleur

```
public class Couleur extends Color{  
  
    static final Couleur VERT = new Couleur(0,255,0);  
    static final Couleur ROUGE = new Couleur(255,0,0);  
    static final Couleur BLEU = new Couleur(0,0,255);  
    static final Couleur JAUNE = new Couleur(255,255,0);  
    static final Couleur ROSE = new Couleur(255,182,193);  
  
    public Couleur(int r, int v, int b) {  
        super(r,v,b);  
    }  
  
    public String toString() {  
        if (equals(VERT)) return "vert";  
        if (equals(ROUGE)) return "rouge";  
        if (equals(BLEU)) return "bleu";  
        if (equals(JAUNE)) return "jaune";  
        if (equals(ROSE)) return "rose";  
        return super.toString();  
    }  
}
```

# La classe Panneau pour dessiner

```
public class Panneau extends JPanel {  
  
    Couleur couleur;  
  
    public Panneau() {  
        setPreferredSize(new Dimension(100,100));  
    }  
  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        g.setColor(couleur);  
        g.fillOval(10, 10, 80, 80);  
    }  
}
```

## La classe JList

La classe `JList<E>` est un composant qui permet d'afficher une liste d'objets et qui permet à l'utilisateur de sélectionner un ou plusieurs éléments.

- Les instances de la classe `JList` sont souvent utilisées comme des composants des objets `JScrollPane`.

Un constructeur bien utile :

```
JList(Vector<E> liste)
```

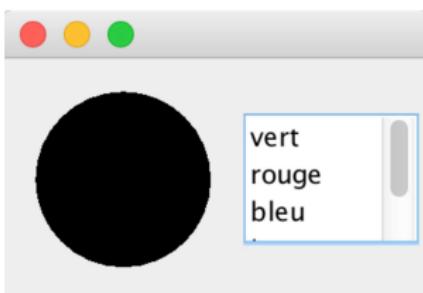
# Créer une liste de couleurs

```
JList<Couleur> liste;  
  
Vector<Couleur> listeItems = new Vector<Couleur>();  
listeItems.add(Couleur.VERT);  
listeItems.add(Couleur.ROUGE);  
listeItems.add(Couleur.BLEU);  
listeItems.add(Couleur.JAUNE);  
listeItems.add(Couleur.ROSE);  
  
liste = new JList<Couleur>(listeItems);  
liste.setSelectedIndex(0);
```

## La méthode setselectedIndex

Cette méthode fait en sorte qu'au démarrage c'est le **premier item** de la liste qui sera **sélectionné**.

**Sans** cette méthode, au démarrage on obtiendrait le résultat suivant :



## L'interface ListSelectionListener

```
public interface ListSelectionListener extends  
EventListerner
```

Cette interface modélise un **écouteur** qui est notifié quand l'utilisateur **choisit** un élément dans une liste.

Une **seule** méthode à implémenter :

```
void valueChanged(ListSelectionEvent e)
```

# Implémenter la fenêtre principale

```
public class Fenetre extends JFrame implements ListSelectionListener {  
  
    Panneau panneau = new Panneau();  
    JList<Couleur> liste;  
  
    public Fenetre() {  
        setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
        add(panneau);  
  
        Vector<Couleur> listeItems = new Vector<Couleur>();  
        listeItems.add(Couleur.VERT);  
        listeItems.add(Couleur.ROUGE);  
        listeItems.add(Couleur.BLEU);  
        listeItems.add(Couleur.JAUNE);  
        listeItems.add(Couleur.ROSE);  
  
        liste = new JList<Couleur>(listeItems);  
        liste.setSelectedIndex(0);  
  
        JScrollPane listeAvecAscenseur = new JScrollPane(liste);  
        listeAvecAscenseur.setPreferredSize(new Dimension(80, 60));  
        add(listeAvecAscenseur);  
        liste.addListSelectionListener(this);  
  
        panneau.couleur = (Couleur) liste.getSelectedValue();  
        pack();  
        setLocation(100, 100);  
        setVisible(true);  
    }  
}
```

## La méthode valueChanged

```
public void valueChanged(ListSelectionEvent e) {  
    panneau.couleur = (Couleur)liste.getSelectedValue();  
    panneau.repaint();  
}
```

- **ListSelectionEvent** : Modélise un événement lié à un changement de choix dans une liste.
- **getSelectedValue()** : Cette méthode de la classe **JList** renvoie l'objet sélectionné.

# Agenda du jour

Événements liés à une fenêtre

Gérer des listes de choix

Utiliser une zone de texte

Gérer les clics de la souris

# Une application



## Insérer une image

On veut insérer une image au panneau qui se trouve en haut de la fenêtre.

- Utiliser la classe `ImageIcon` :

```
 ImageIcon icone = new ImageIcon(getClass().getResource("world.png"));
```

- Plein de formats supportés (`png`, `jpeg`, `gif`, ...)
- L'image doit se trouver au même endroit que les fichiers `.class`.



## Associer l'image à un JLabel

Une étiquette, instance de la classe `JLabel`, peut être une **image**.

```
JLabel afficheImage = new JLabel(icone);
```

- Autre façon de faire :

```
JLabel afficheImage = new JLabel();  
afficheImage.setIcon(icone);
```

## La classe JTextField

Un objet **JTextField** est un **composant texte** qui permet d'**entrer du texte** sur une seule ligne.



On peut indiquer la largeur de ce composant, en précisant le **nombre de colonnes**.

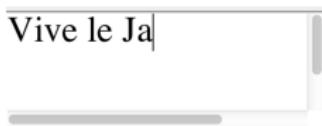
```
JTextField champTexte = new JTextField();  
champTexte.setColumns(10);
```

ou directement en utilisant le **constructeur** dédié :

```
JTextField champTexte = new JTextField(10);
```

## La classe JTextArea

La classe `JTextArea` est une classe qui permet d'**entrer**, ou d'**afficher** du texte (sur une ou plusieurs lignes).



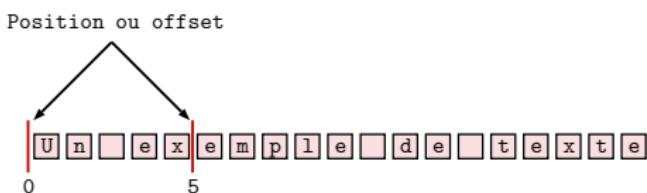
### Quelques constructeurs

- `JTextArea()`
- `JTextArea(int l, int c)` : Construit un objet avec **l** lignes et **c** colonnes qui sont utilisées pour déterminer la taille du composant, et éventuellement faire apparaître les barres de défilement, si un `JScrollPane` est utilisé.
- `JTextArea(String texte)` : Construit un objet avec un texte initial.
- `JTextArea(Document doc)` : Construit un objet à partir du Document **doc**.

## L'interface Document

Un objet de la classe **Document** permet de réaliser un composant pour contenir du texte qui sert de modèle pour des composantes **Swing** de texte.

Se déplacer dans le texte :



Quelques méthodes :

- `getLength()`
- `getText(int, int)`

## La classe Font

La classe **Font** permet d'utiliser une **police de caractères** particulière pour afficher un texte.

Un **constructeur** utile :

```
Font(String nom, int style, int taille)
```

- **nom** : Le nom de la fonte (**Helvetica**, **TimesRoman**, **Courier**, ...)
- **style** : Le style de la fonte (**BOLD**, **ITALIC**, **PLAIN**)
  - Possibilité de combiner les styles. Par exemple :  
`Font.BOLD + Font.ITALIC`
- **taille** : La taille de la police.

**Exemple :**

```
Font fonte = new Font("TimesRoman ", Font.PLAIN, 20);
```

# Événements qui peuvent se produire

Trois types d'événements peuvent se produire :

- L'utilisateur peut **changer** la taille de la fonte dans l'objet **JTextField**.
- L'utilisateur peut **saisir** un texte dans l'objet **JTextArea**.
- L'utilisateur peut **déplacer le curseur** dans l'objet **JTextArea**.

Ajouter trois types d'**écouteurs** :

- Un écouter associé aux changements dans l'objet **JTextField**.
  - **ActionListener**
- Un écouter associé aux déplacements du curseur et la sélection de texte dans l'objet **JTextArea**.
  - **CaretListener**
- Un écouter associé à l'insertion et la suppression de caractères dans l'objet **JTextArea**.
  - **DocumentListener**

## L'interface CaretListener

- Les événements de type **curseur** se produisent dans des composants texte lorsque **le curseur se déplace** ou lorsque la **sélection est modifiée**.
- Il est possible d'enregistrer un **CaretListener** à une instance de n'importe quel composant texte héritant de la classe **JTextComponent** à l'aide de la méthode **addCaretListener**.

L'interface **CaretListener** ne possède qu'une seule méthode :

```
void caretUpdate(CaretEvent e)
```

# Les événements liés aux objets Document

- Un composant texte **Swing** utilise un objet **Document** pour stocker et éditer le texte.
- Les **événements relatifs aux documents** ont lieu lorsque se produit n'importe quel type de modification sur le document.
- On peut enregistrer un **DocumentListener** au document d'un composant texte, plutôt qu'au composant texte lui-même.

## L'interface DocumentListener

L'interface **DocumentListener** contient les trois méthodes suivantes :

- void **insertUpdate**(DocumentEvent e) : Invoquée lorsque du **texte est inséré** dans le document observé.
- void **removeUpdate**(DocumentEvent e) : Invoquée lorsque du **texte est effacé** du document observé.
- void **changedUpdate**(DocumentEvent e) : Invoquée lorsqu'il y a un **changement de style** sur le texte.
  - Ce type d'événement n'est produit que par les **StyledDocument**, un **PlainDocument** n'en produit pas.

## La classe EcouteDocument

```
import javax.swing.JLabel;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

public class EcouteDocument implements DocumentListener {
    JLabel etiquette;
    EssaiTexte panneau;

    public EcouteDocument(EssaiTexte panneau) {
        this.panneau = panneau;
        etiquette = this.panneau.commentaire;
    }

    public void insertUpdate(DocumentEvent e) {
        etiquette.setText("Insertion du " + (panneau.position + 1) + " eme caractere");
    }

    public void removeUpdate(DocumentEvent e) {
        etiquette.setText("Suppresion du " + panneau.position + " eme caractere");
    }

    public void changedUpdate(DocumentEvent e) {}

}
```

# La classe EssaiTexte

```
public class EssaiTexte extends JFrame implements ActionListener, CaretListener {  
  
    JLabel commentaire = new JLabel("Vous pouvez écrire");  
    JTextField saisieTaille = new JTextField(5);  
    JTextArea grandeZone = new JTextArea(4,15);  
    int position;  
  
    public EssaiTexte() {  
        JPanel panneauTaille = new JPanel();  
  
        ImageIcon icone = new ImageIcon(getClass().getResource("world.png"));  
        JLabel afficheImage = new JLabel(icone, SwingConstants.LEFT);  
        panneauTaille.add(afficheImage);  
  
        JLabel etiquetteTaille = new JLabel(" Taille de la fonte : ", SwingConstants.RIGHT);  
        panneauTaille.add(etiquetteTaille);  
        saisieTaille.setText("20");  
        panneauTaille.add(saisieTaille);  
        add(panneauTaille, BorderLayout.NORTH);  
        saisieTaille.addActionListener(this);  
  
        int tailleFonte = Integer.parseInt(saisieTaille.getText());  
        Font fonte = new Font("TimesRoman", Font.PLAIN, tailleFonte);  
        grandeZone.setFont(fonte);  
        grandeZone.setText("Essayez d'écrire quelque chose");  
        JScrollPane texteAsc = new JScrollPane(grandeZone);  
        add(texteAsc, BorderLayout.CENTER);  
        grandeZone.addCaretListener(this);  
        Document document = grandeZone.getDocument();  
        document.addDocumentListener(new EcouteDocument(this));  
  
        add(commentaire, BorderLayout.SOUTH);  
        pack();  
    }  
}
```

# Les méthodes actionPerformed et caretUpdate

```
public void actionPerformed(ActionEvent e) {
    int taillePolice;

    try {
        taillePolice = Integer.parseInt(saisieTaille.getText());
        grandeZone.setFont(new Font("TimesRoman", Font.PLAIN, taillePolice));
    }
    catch(NumberFormatException exception) {

    }
}

public void caretUpdate(CaretEvent e) {
    if(e.getMark() != e.getDot()) {
        commentaire.setText("Selection de caracteres");
    }
    else position = e.getDot();
}
```

## Les méthodes getDot et getMark

- int `getDot()` : Renvoie la **position courante du curseur**. Si une partie du texte est sélectionnée, la valeur correspond à la position de la fin de la sélection.
- int `getMark()` : Renvoie **l'autre valeur de la sélection**. Si il n'y a pas de sélection, la valeur renournée est identique à celle renvoyée par la méthode `getDot`.

## La classe de test

```
public class TestFenetreFonte {  
  
    public static void main(String[] args) {  
        JFrame fenetre = new EssaiTexte();  
        //fenetre.setSize(800, 800);  
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        fenetre.setLocation(100, 100);  
        fenetre.setVisible(true);  
    }  
}
```

# Agenda du jour

Événements liés à une fenêtre

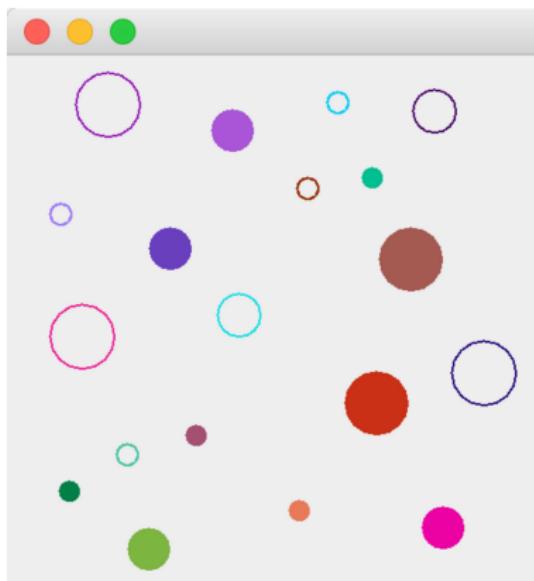
Gérer des listes de choix

Utiliser une zone de texte

Gérer les clics de la souris

## Exercice

Dessiner et effacer des cercles de couleur aléatoire avec un clic de la souris.



## Effets désirés

- Clic gauche : Dessiner un petit cercle.
  - Clic roulette : Dessiner un cercle moyen.
  - Clic droit : Dessiner un grand cercle.
- 
- Quand le curseur se trouve sur un cercle déjà désigné, le curseur se transforme en croix et en cliquant on efface le cercle.
  - Par défaut les cercles sont vides (juste le contour).
  - Si l'utilisateur garde appuyée la touche Shift les cercles dessinés sont pleins.

## Une classe représentant un cercle

Un objet de cette classe sera représenté par **cinq attributs** :

- Les coordonnées **x** et **y** du point haut et gauche du carré qui entoure le cercle.
- Son rayon
- Sa couleur
- Une variable booléenne indiquant si le cercle est plein ou vide.

Deux **méthodes** :

- `public boolean estDedans(int a, int b)` : Décider si le point aux coordonnées (a,b) se trouve dans le cercle ou pas.
- `public void dessiner(Graphics g)` : Dessiner un cercle vide ou plein selon la valeur de l'attribut booléen.

# La classe CercleGraphique

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

public class CercleGraphique {

    int x,y;
    Color couleur;
    int rayon;
    boolean plein;

    public CercleGraphique(int rayon, int x, int y, Color couleur, boolean plein) {
        this.rayon = rayon;
        this.x = x;
        this.y = y;
        this.couleur = couleur;
        this.plein = plein;
    }

    public boolean estDedans(int a, int b) {
        if ((a - this.x)*(a - this.x) + (b - this.y)*(b - this.y) < rayon*rayon) return true;
        else return false;
    }

    public void dessiner(Graphics g) {
        g.setColor(couleur);
        if(plein)
            g.fillOval(x-rayon, y-rayon, 2*rayon, 2*rayon);
        else
            g.drawOval(x-rayon, y-rayon, 2*rayon, 2*rayon);
    }
}
```

# Générer des nombres aléatoires

La classe `Random` permet de générer des nombres `pseudo-aléatoires`.

- Constructeur

```
Random alea = new Random();
```

- La méthode `nextInt()` retourne un entier compris entre `Integer.MIN_VALUE` (-2147488348) et `Integer.MAX_VALUE` (2147488347).

## Decider quelle touche de la souris a été appuyée

La classe **SwingUtilities** possède les trois méthodes **statiques** suivantes qui permettent de déterminer laquelle des trois touches de la souris a été appuyée :

- static boolean **isLeftMouseButton**(MouseEvent e)
- static boolean **isMiddleMouseButton**(MouseEvent e)
- static boolean **isRightMouseButton**(MouseEvent e)

## Changer l'apparence du curseur de la souris

La classe `Cursor` permet d'encapsuler la représentation graphique du curseur de la souris.

Deux méthodes qui nous seront utiles :

- static Cursor `getDefaultCursor()` : Renvoie le curseur par default.
- static Cursor `getPredefinedCursor(int type)` : Renvoie un curseur du type spécifié en paramètre. (Regardez la Javadoc de la classe `Cursor` pour tous les types possibles.)

**Exemple** : static int `CROSSHAIR_CURSOR` : forme de `croix`.

La méthode

```
public void setCursor(Cursor cursor)
```

de la classe `java.awt.Component` instancie l'image du curseur à celle passée en paramètre.

## Gérer les événements liés à la souris

Les événements liés à la souris sont modélisés par la classe [java.awt.event.MouseEvent](#).

Deux interfaces sont définies :

- L'interface [MouseListener](#)
- L'interface [MouseMotionListener](#)

## L'interface MouseListener

L'interface **MouseListener** déclare cinq méthodes :

- `public void mousePressed(MouseEvent e)` : Gérer l'événement "le bouton est **enfoncé**".
- `public void mouseReleased(MouseEvent e)` : Gérer l'événement "le bouton est **relâché**".
- `public void mouseClicked(MouseEvent e)` : Gérer l'événement "le bouton est **enfoncé puis relâché** sans que le curseur n'ait été **déplacé**".
- `public void mouseEntered(MouseEvent e)` : Gérer l'événement "**le curseur est entré dans le composant**".
- `public void mouseExited(MouseEvent e)` : Gérer l'événement "**le curseur est sorti du composant**".

# L'interface MouseMotionListener

L'interface **MouseMotionListener** déclare deux méthodes :

- `public void mouseDragged(MouseEvent e)` : Gérer l'événement "le curseur de la souris se déplace dans le composant **alors qu'un bouton de la souris est enfoncé**".
- `public void mouseMoved(MouseEvent e)` : Gérer l'événement "le curseur de la souris se déplace dans le composant **sans qu'un bouton de la souris soit enfoncé**".

# La classe EssaiClics (I)

```
public class EssaiClics extends JPanel{
    ArrayList<CercleGraphique> memoire = new ArrayList<CercleGraphique>();
    Random alea = new Random();
    CercleGraphique cercleCourant;

    public EssaiClics() {
        setPreferredSize(new Dimension(250,250));
        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                if(cercleCourant != null) {
                    memoire.remove(cercleCourant);
                    repaint();
                    return;
                }
                int rayon = 5;
                boolean plein = false;
                if(SwingUtilities.isMiddleMouseButton(e)) rayon = 10;
                else if(SwingUtilities.isRightMouseButton(e)) rayon = 15;
                if((e.getModifiers() & MouseEvent.SHIFT_MASK) != 0) plein = true;
                cercleCourant = new CercleGraphique(rayon, e.getX(), e.getY(), new Color(alea.nextInt(), plein));
                memoire.add(cercleCourant);
                repaint();
                curseurCarre();
            }
        });
        addMouseMotionListener(new MouseMotionAdapter() {
            public void mouseMoved(MouseEvent e) {
                cercleCourant = leCercle(e.getX(), e.getY());
                if(cercleCourant != null) curseurCarre();
                else setCursor(Cursor.getDefaultCursor());
            }
        });
    }
}
```

## La classe EssaiClics (II)

```
public void curseurCarre() {
    setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
}

public CercleGraphique leCercle(int a, int b) {
    CercleGraphique trouve = null;
    for(CercleGraphique c : memoire)
        if(c.estDedans(a, b)) trouve = c;
    return trouve;
}

public void paintComponent(Graphics g) {
    super.paintComponent(g);
    for(CercleGraphique c : memoire) c.dessiner(g);
}
```

## La classe de test

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