D[i] | Département Informatique

UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

Faculté Sciences et Ingénierie

Licence Informatique

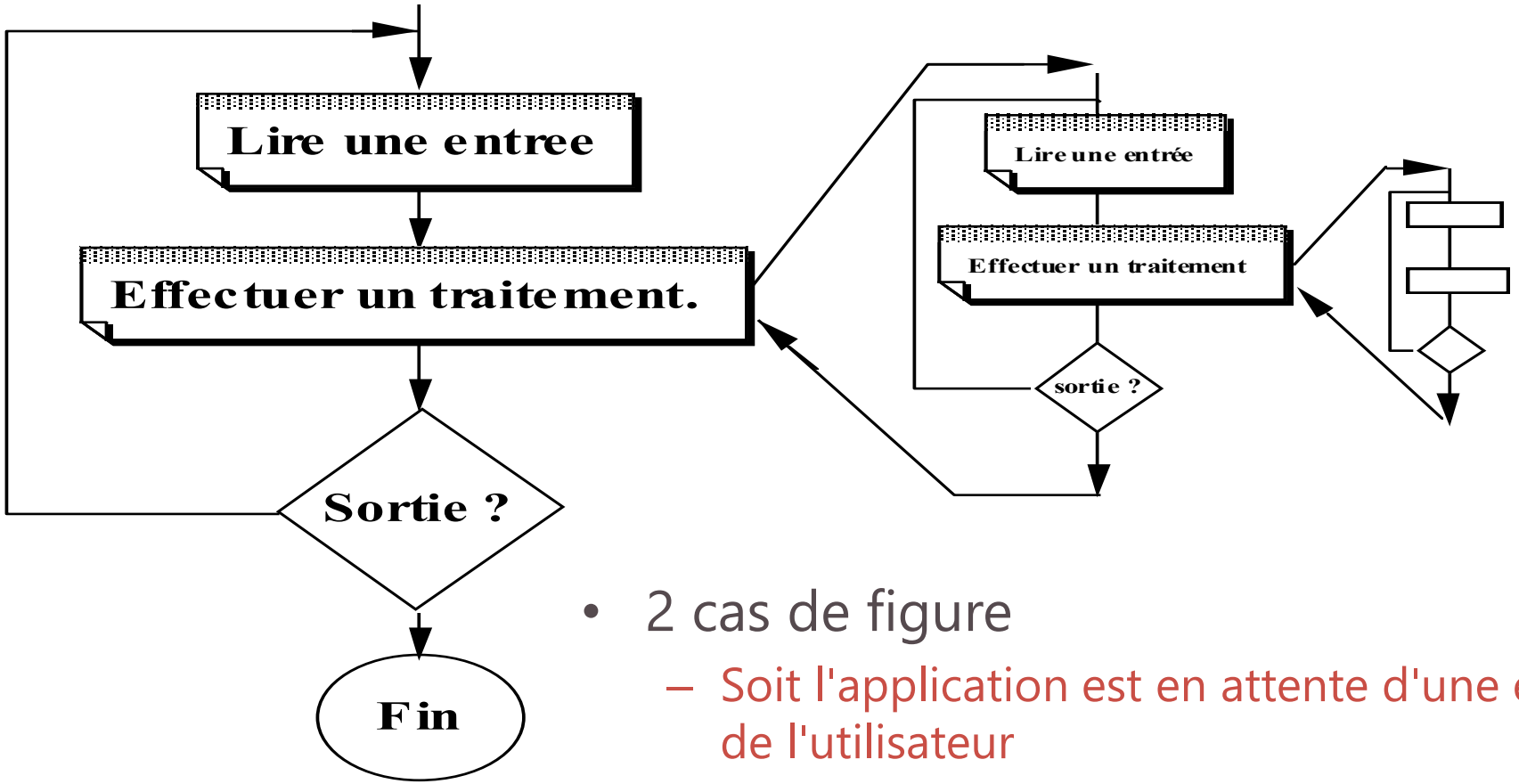
Librairie Java Swing & Programmation événementielle

Mathieu RAYNAL

mathieu.raynal@irit.fr

http://www.irit.fr/~Mathieu.Raynal

Vos applications actuelles …

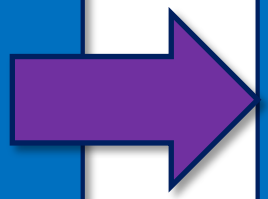


- 2 cas de figure
  - Soit l'application est en attente d'une entrée de l'utilisateur
  - Soit elle fait des calculs et l'utilisateur doit attendre

… un fonctionnement séquentiel

- Structure d'un programme séquentiel

```
Début
choix = '1';
Tantque choix <> '9' faire
      affiche-menu;
      lire(choix);
      case choix of
      1 : ajouter;
      2 : modifier;
      3 : supprimer;
      9 : Quitter;
      Fin Case
FintTantque
Fin
```



```
Procédure Ajouter;
début
rep = 'o';
Tantque rep <> 'n';
      dessin-écran;
      lire(nom);
      lire(prenom);
      …
      écrire('voulez-vous
      continuer ?');
      lire(rep)
FinTantque
Fin
```

Fonctionnement par événements

- Evénement = Action extérieure au programme se produisant sur l'interface
  - Généralement effectué par l'utilisateur

- Les sources
  - Matériel
    - Clavier,
    - Souris,
    - Ecran tactile,
    - …
  - Logiciel (composants graphiques)
    - Fenêtre,
    - Bouton,
    - Liste,
    - …

Les événements

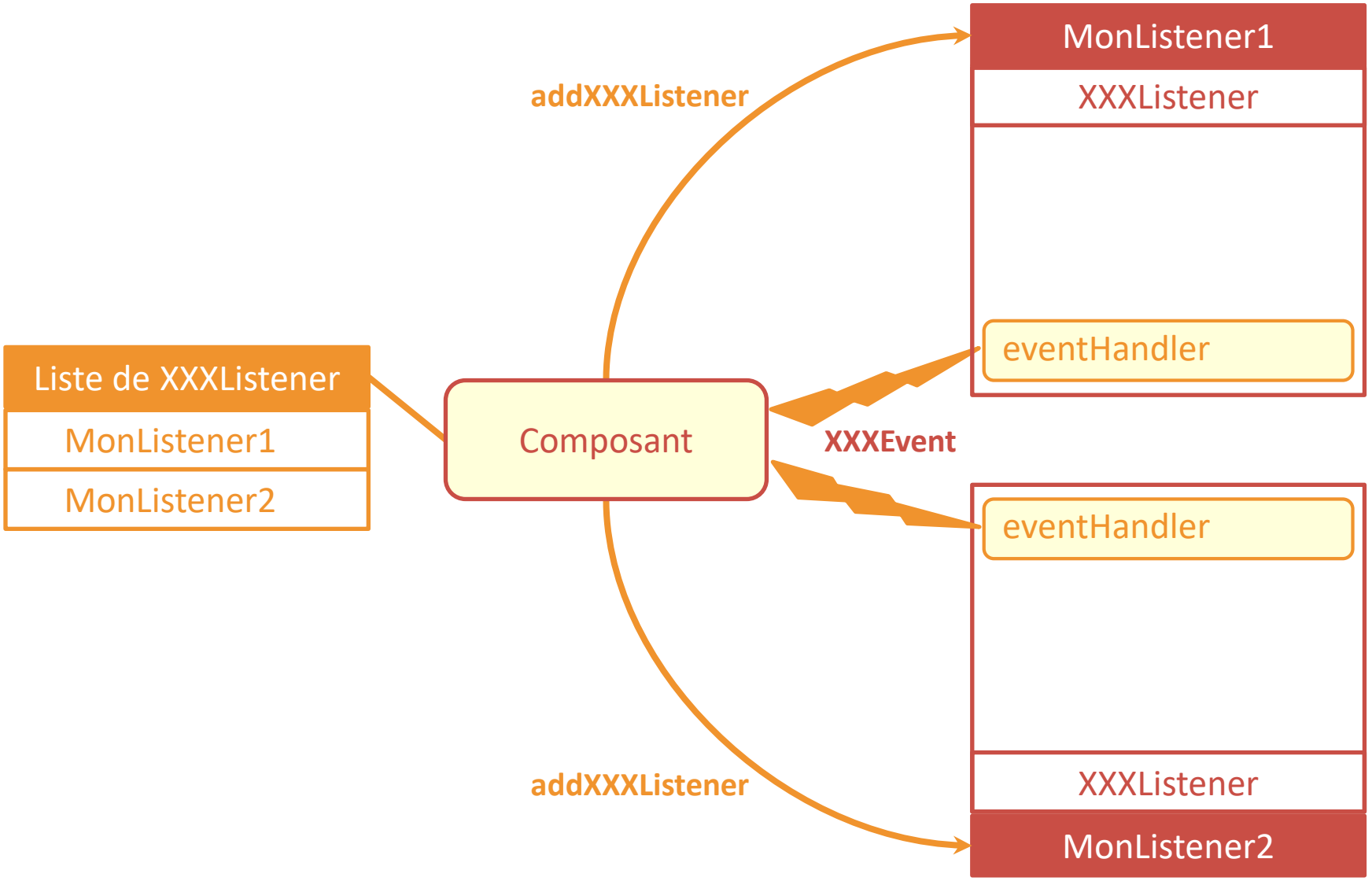
- Deux niveaux d'événements
  - Bas niveau
    - Déplacer la souris
    - Appuyer sur un bouton d'un dispositif
    - Taper sur une touche du clavier
  - Sémantique
    - Appuyer sur un bouton de l'interface
    - Prendre ou perdre le focus
    - Entrer ou sortir d'un composant

Les événements

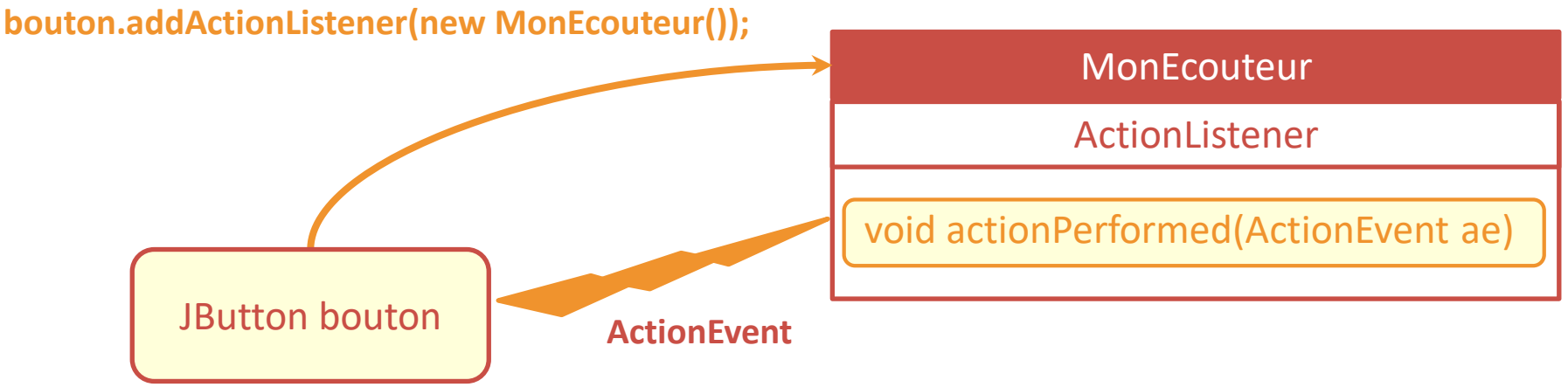
- Les événements sont caractérisés par des classes
  - Chaque type d'événement a une classe spécifique
    - Clic bouton : **ActionEvent**
    - Evénements liés à la souris : **MouseEvent**
    - Evénements liés au clavier : **KeyEvent**
  - Cette classe contient les caractéristiques de l'événement
    - Position, moment, …

- Les événements sont gérés par les composants graphiques
  - Un composant peut gérer plusieurs types d'événements
  - Un type d'événement peut se produire sur différents types de composants

Gestion des événements

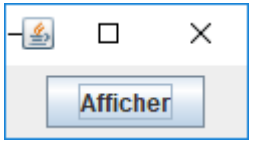


Cas du bouton



Exemple 1 – gestion d'un bouton

- Afficher Bonjour dans la console suite à un appui sur le bouton

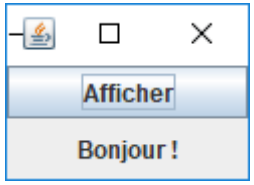



```
6  public class MonEcouteur implements ActionListener{
7      @Override
8      public void actionPerformed(ActionEvent e) {
9          System.out.println("Bonjour !");
10     }
11 }
```

```
17         bouton = new JButton("Afficher");
18         bouton.addActionListener(new MonEcouteur());
```

Exemple 2 – gestion d'un bouton

- Afficher Bonjour dans un label sur l'interface suite à un appui sur le bouton

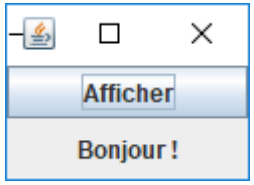



```
14 public class ExempleEvenementBouton1 implements ActionListener{
15    JLabel label;
16    JButton bouton;
17
18    ExempleEvenementBouton1() {
19       JFrame frame = new JFrame("Gestion événement bouton 1");
20       frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
21
22       label = new JLabel("",JLabel.CENTER);
23
24       bouton = new JButton("Afficher");
25       bouton.addActionListener(this);
```

```
45    @Override
46    public void actionPerformed(ActionEvent e) {
47       label.setText("Bonjour !");
48    }
```

Exemple 3 – gestion d'un bouton

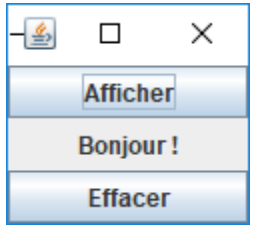
- Afficher Bonjour dans un label sur l'interface et dans la console suite à un appui sur le bouton




```
24      bouton = new JButton("Afficher");
25      bouton.addActionListener(this);
26      bouton.addActionListener(new MonEcouteur());
```

Exemple 4 – gestion d'un bouton

- 2 boutons
  - Afficher Bonjour dans un label sur l'interface suite à un appui sur le bouton « Afficher »
  - Effacer le contenu du label suite à un appui sur le bouton « Effacer »

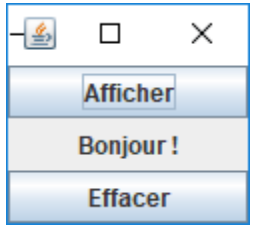



```
22      boutonAfficher = new JButton("Afficher");
23      boutonAfficher.addActionListener(this);
24
25      boutonEffacer = new JButton("Effacer");
26      boutonEffacer.addActionListener(this);
```

```
47    public void actionPerformed(ActionEvent e) {
48       if(e.getActionCommand().equals("Afficher"))
49          label.setText("Bonjour !");
50
51       if(e.getActionCommand().equals("Effacer"))
52          label.setText("");
53    }
```

Exemple 4 – gestion d'un bouton

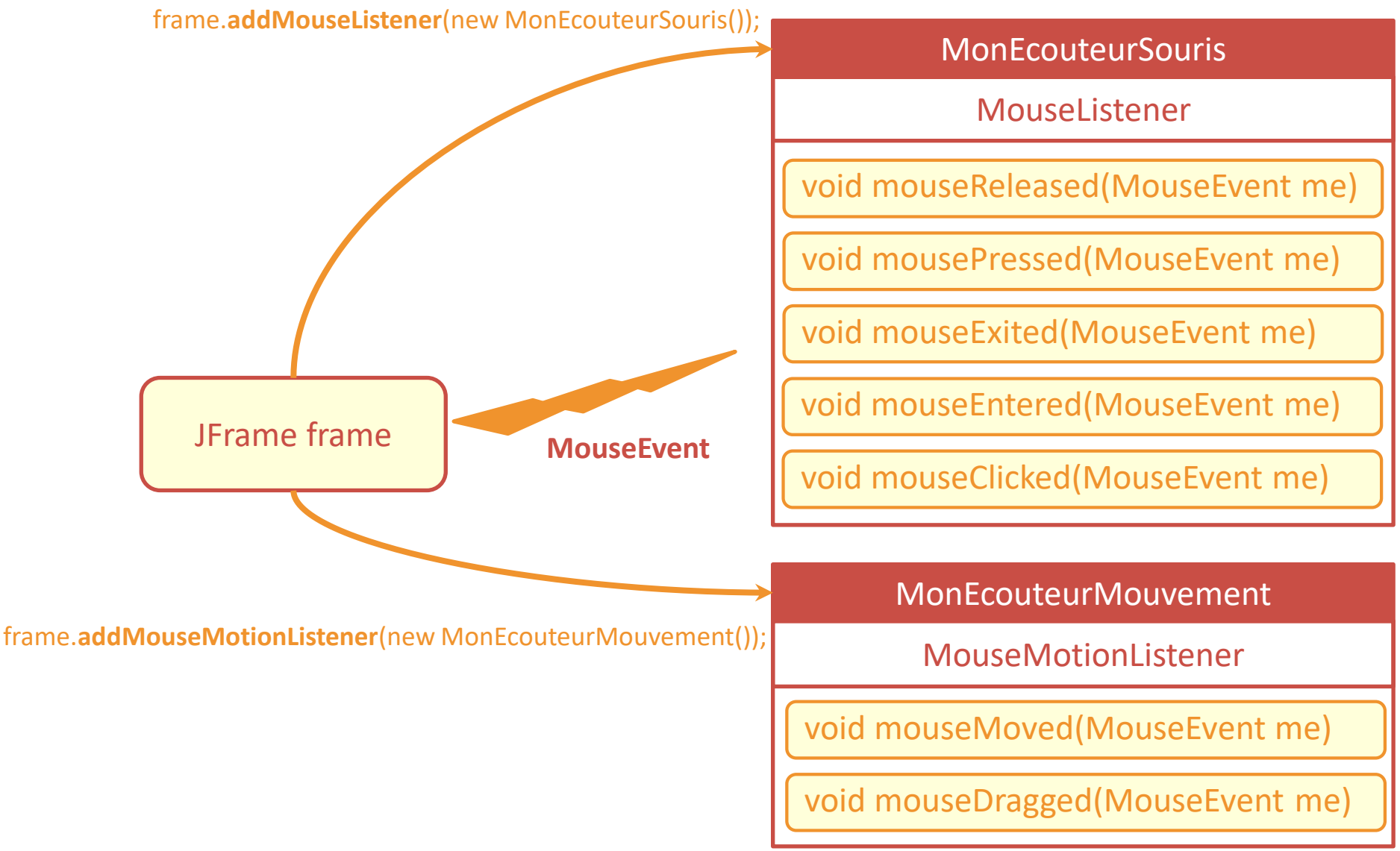
- 2 boutons
  - Afficher Bonjour dans un label sur l'interface suite à un appui sur le bouton « Afficher »
  - Effacer le contenu du label suite à un appui sur le bouton « Effacer »




```
22      boutonAfficher = new JButton("Afficher");
23      boutonAfficher.addActionListener(new ActionListener() {
24          @Override
25          public void actionPerformed(ActionEvent arg0) {
26              afficher();
27          }
28      });
```

```
51      public void afficher()
52      {
53          label.setText("Bonjour !");
54      }
```

Cas de la souris



Listener versus Adapter

- **Listener** : Interface
  - Obligation d'avoir toutes les méthodes de l'interface !
- **Adapter** : Classe abstraite qui implémente le Listener
  - On ne masque que les méthodes qui nous intéressent

  → Attention de masquer correctement la méthode

Exemple – gestion de la souris

- Afficher les coordonnées du pointeur au moment où l'on presse un bouton de la souris

```
25      frame.addMouseListener(new MouseAdapter() {
26          @Override
27          public void mousePressed(MouseEvent e) {
28              System.out.println("Pression du bouton de la souris en ("+e.getX()+", "+e.getY()+")");
29          }
30      });
```

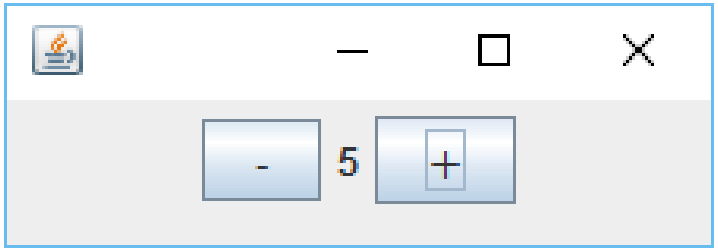
En résumé - Gestion des événements

- Lorsqu'un événement se produit sur un **composant**
  - Création d'une instance de l'événement
  - Envoi à tous ceux qui sont intéressés

- Qui est « intéressé » par un événement **XXXEvent** ?
  - Les écouteurs (**XXXListener**)
    - Spécifiques à chaque type d'événement
  - Les écouteurs doivent se faire connaitre auprès du composant
    - **addXXXListener**(XXXListener)

Exercice

- Les boutons + et – permettent d'incrémenter et décrémenter le compteur



Utilisation du Timer

Javax.swing.Timer

- Envoie un événement de type ActionEvent toutes les N millisecondes
  - Evènements gérés de la même manière que pour les boutons
- Constructeur

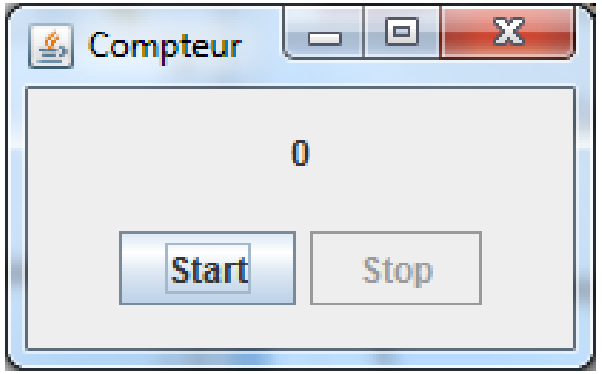
Timer(int N, ActionListener listener)

- Principales méthodes

void **setDelay**(int N)
void **start()**
void **stop()**

Exercice : Compteur

- Réalisez le compteur suivant
  - Au démarrage seul le bouton Start est activé et le compteur est initialisé à 0
  - Après un clic sur le bouton Start
    - Le compteur est réinitialisé à 0 si ce n'est pas le cas
    - le bouton Stop est activé
    - le bouton Start est désactivé
  - Jusqu'à l'appui sur Stop, le compteur est incrémenté de 1 toutes les secondes
  - Après un clic sur le bouton Stop
    - Le compteur est arrêté
    - le bouton Stop est désactivé
    - le bouton Start est activé



Dessiner sur un composant

Dessiner en JAVA

- Où ?
  - Sur le composant Canvas en AWT
  - Sur tous les composants SWING héritant de JComponent

- Comment faire en SWING ?
  - Créer sa propre classe qui étend de **Jcomponent**
    - Avoir un constructeur par défaut qui appelle celui de **JComponent**
  - Pour SWING, redéfinir la méthode

    void **paintComponent**(Graphics g)

    - Commencer cette méthode en appelant la méthode **paintComponent** de la classe **JComponent**

La classe Graphics

- Permet de dessiner sur le composant
- Dessiner les contours

void **draw3DRect**(int x, int y, int width, int height, boolean raised)
void **drawArc**(int x, int y, int width, int height, int startAngle, int arcAngle)
void **drawLine**(int x1, int y1, int x2, int y2)
void **drawOval**(int x, int y, int width, int height)
void **drawPolygon**(int[] xPoints, int[] yPoints, int nPoints)
void **drawPolygon**(Polygon p)
void **drawPolyline**(int[] xPoints, int[] yPoints, int nPoints)
void **drawRect**(int x, int y, int width, int height)
void **drawRoundRect**(int x, int y, int width, int height, int arcWidth, int arcHeight)

- Remplir la forme
  - Même méthode avec fill à la place de draw dans le nom de la méthode

Appliquer un style à un objet graphique

- Changer la couleur

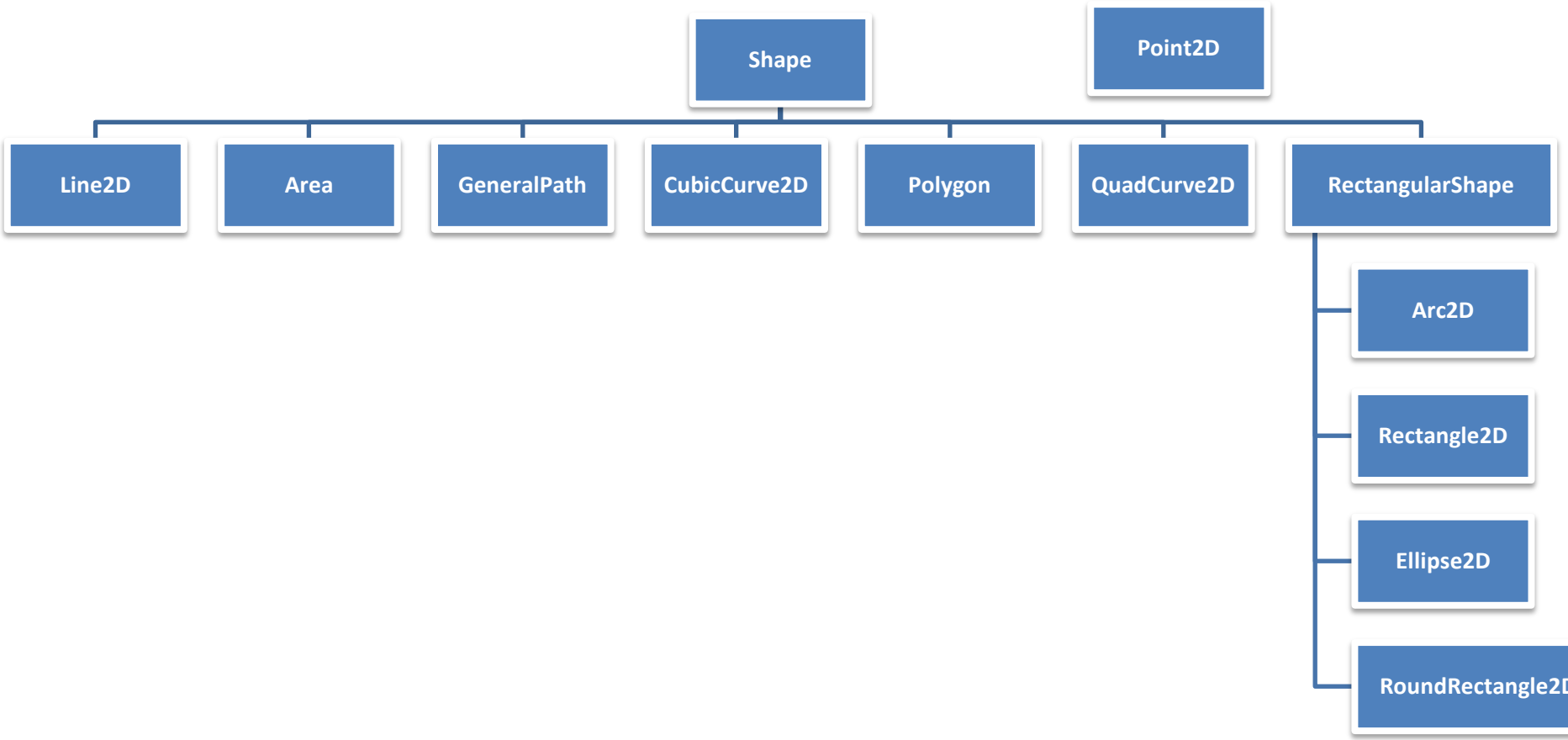
```
void setColor(Color couleur)
```

- Déplacer l'objet

```
void translate(int x, int y)
```

La classe Graphics2D

- Affichage de Shape
  - void draw(Shape s)
  - void fill(Shape s)



Méthodes de Graphics2D

- Fonctionnalités supplémentaires avec Graphics2D
  - Propriété d'affichage : RenderingHints

    void **setRenderingHint**(RenderingHints.Key hintKey, Object hintValue)

  - Style de trait : Stroke

    void **setStroke**(Stroke s)

  - Outil de remplissage : Paint

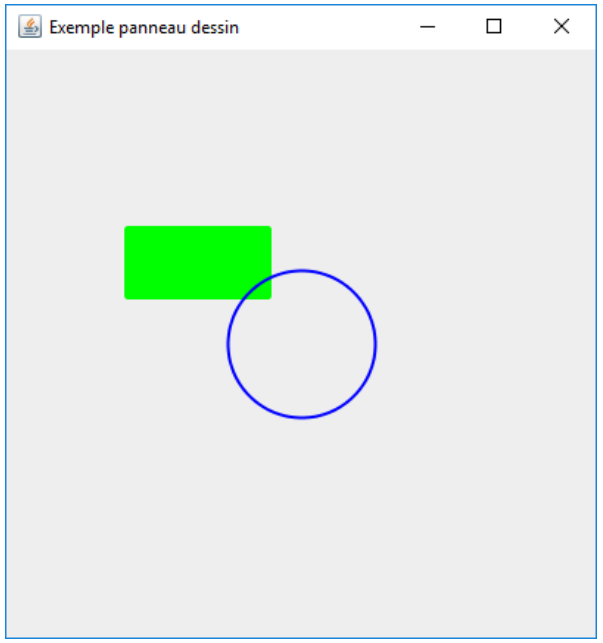
    void **setPaint**(Paint paint)

  - Transformation géométrique : AffineTransform

    void **setTransform**(AffineTransform Tx)

Exemple

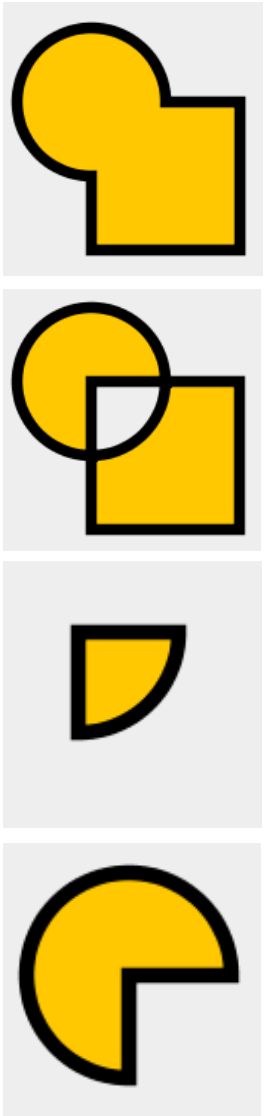
```
17 public class PanneauDessin extends JComponent{
18    int diametre;
19    Ellipse2D.Double rond;
20    RoundRectangle2D.Double rect;
21
22    public PanneauDessin()
23    {
24       super();
25       rect = new RoundRectangle2D.Double(80, 120, 100, 50, 5, 5);
26       diametre = 100;
27       rond = new Ellipse2D.Double(200-diametre/2, 200-diametre/2, diametre, diametre);
28       setPreferredSize(new Dimension(400, 400));
29    }
30
31    public void paintComponent(Graphics g)
32    {
33       super.paintComponent(g);
34       Graphics2D g2 = (Graphics2D)g;
35       g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
36             RenderingHints.VALUE_ANTIALIAS_ON);
37       g2.setColor(Color.GREEN);
38       g2.fill(rect);
39       g2.setColor(Color.BLUE);
40       g2.setStroke(new BasicStroke(2.0f));
41       g2.draw(rond);
42    }
43 }
```



Gestion des Aires

- La class Area
  - Area(Shape s)

- Méthodes de combinaison de forme
  - add(Area rhs)
  - exclusiveOr(Area rhs)
  - intersect(Area rhs)
  - subtract(Area rhs)



Exercice : tracé de ligne

- Sur une interface ne contenant qu'un JComponent
  - afficher la trace du déplacement du pointeur de la souris quand le bouton gauche de la souris est enfoncé