ISI5
Développement d'interfaces Homme-Machine

Manuele Kirsch Pinheiro
Maître de conférences en Informatique
Centre de Recherche en Informatique
Université Paris 1 – Panthéon Sorbonne
Manuele.Kirsch-Pinheiro@univ-paris1.fr
http://mkirschp.free.fr

Présentation

- **Objectif** :
  - Comment concevoir et réaliser des interfaces Homme-Machine
- **Organisation** :
  - 5 séances de 6h
  - ½ concepts + ½ travaux pratiques
- **Évaluation** :
  - Contrôle continu : travaux en cours + projet
  - Examen

Présentation

- **Contenu prévisionnel**
  - IHM : les enjeux
  - Programmation événementiel avec AWT
  - Modèle MVC
  - Java Swing
  - Internationalisation
  - Programmation en 3 couches

Présentation

- **Projet « Scrabble »**
  - Création jeu « scrabble »
    - On propose un mot dont les lettres sont dans le désordre
    - L'utilisateur doit trouver le mot correct
  - Programmation « incrémentale »
    - À chaque séance, on améliore le jeu
    - À la fin, on rend le jeu abouti
      - Interface graphique sympa et bien construite
      - Accès à une base de mots (dictionnaire)
      - …

Présentation

- **Bibliographie**
  - Cay S. Horstmann, Gary Cornell, « **Au cœur de Java** », Volume I – Notions Fondamentales, Pearson Education
  - Cay S. Horstmann, Gary Cornell, « **Core Java** », Volume II – Advanced Features, Pearson Education
  - R. Eckestein, M. Loy, D. Wood, « **Java Swing** », O'Reilly
- **Sur le Web**
  - API : http://java.sun.com/javase/6/docs/api/
  - Tutorial Sun : http://java.sun.com/docs/books/tutorial/
  - Cours :
    - http://java.developpez.com/cours/
    - http://www.jmdoudoux.fr/java/dej/

IHM
Les enjeux

Interface Homme-Machine

- **Interface** :
  - Dispositif technique de **communication** avec l'utilisateur
  - **Interaction** entre l'utilisateur (**personne**) et l'application (**système**)
- **Le succès d'une application dépend de son interface**
  - **Utiles** : en adéquation aux besoins
  - **Utilisables :** en adéquation aux capacités de l'utilisateur
  - Dans son **contexte** : en adéquation à l'environnement physique, à la plate-forme d'interaction…

Interface Homme-Machine

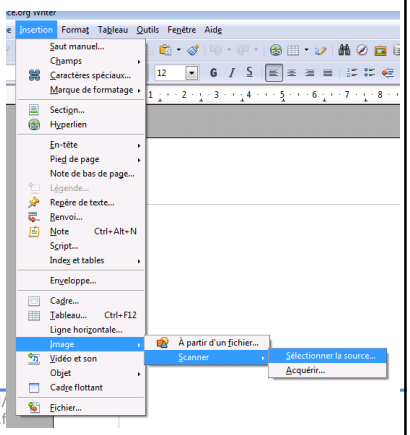
- Conception centrée sur l'**utilisateur**
  - L'interface doit servir l'utilisateur, pas le concepteur
- Plusieurs **aspects** à prendre en compte :
  - **Type d'utilisateur**
    - novice, expérimenté, expert
  - **Planification des tâches**
    - tâches ⊕ utilisées sont ⊕ accessibles
  - **Ergonomie**
    - Couleurs
    - Manipulation
    - Culture (lecture gauche ↔ droite, haut ↔ bas…)
  - . . .
- Plusieurs méthodes sont disponibles

Conseils (in)utiles

- Quelques **conseils** au moment de concevoir l'interface d'une application
- **Correspondance** avec le monde « réel »
  - Établir une correspondance directe entre objets conceptuels et d'interaction
  - Métaphore du monde réel
    - Ex: la corbeille
  - Intuitive : *affordance*

Conseils (in)utiles

- Menus & formulaires
  - Extensions de la mémoire à court terme
  - Ordre logique, fréquence, alphabétique
  - Profondeur & fréquence, profondeur & cohérence



Conseils (in)utiles

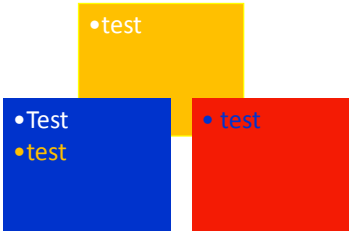
- Considérations syntaxiques & lexicales
  - Structure cohérente des commandes
    - « *cmd -opt arg* » X « *cmd --opt arg* » X « *cmd /opt arg* »
  - Précision et cohérence des termes
    - Congruence des termes
      - « créer / détruire » au lieu de « créer / supprimer »
    - Mélanges verbes-substantifs à éviter
      - « créer / destruction » ☹
    - Mélanges majuscules-minuscules aussi
      - « Créer / détruire » 😐

Conseils (in)utiles

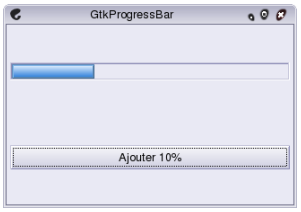
- Considérations articulatoires
  - Mémoire musculaire
    - Stabilité de l'écran, des dispositifs d'entrée…
  - Efficacité
    - Raccourcis, refaire-défaire, valeurs par défaut
- Personnalisation

Conseils (in)utiles

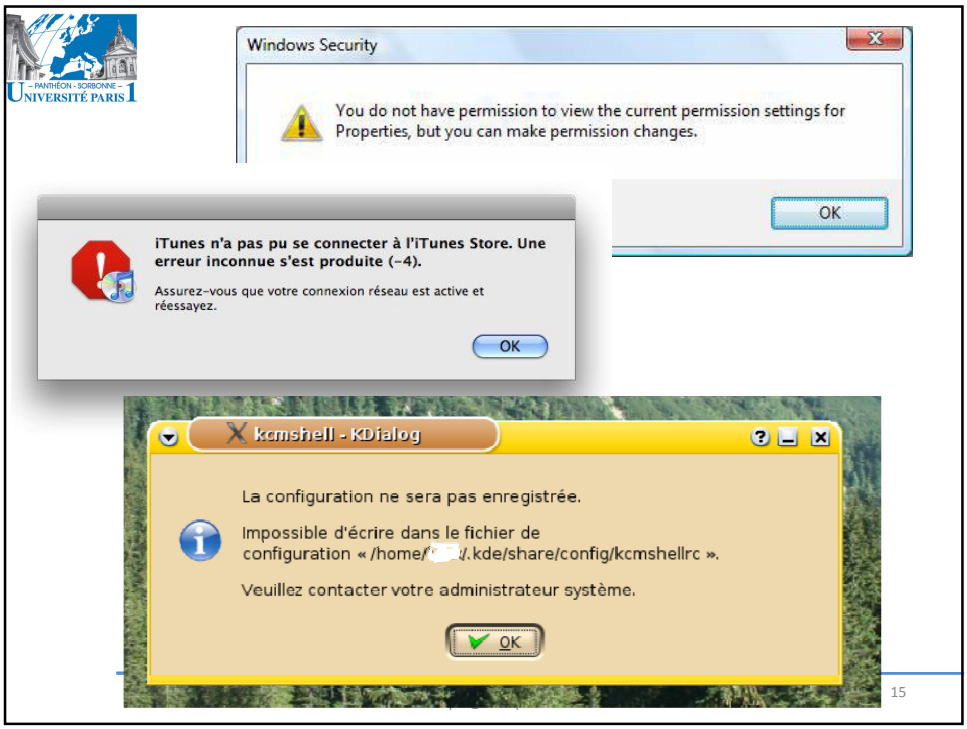
- Usage des **couleurs**
  - **Mémorisation**, recherche, localisation ☺
  - Véhicule de traits **sémantique** ☺
  - Effet « sapin de Nöel » ☹
  - Déficiences visuels 😐
  - Codage arbitraire ou en contradiction avec l'usage ☹
    - Rouge : danger, chaud (chimie)
  - **Association** des couleur 😐
    - Saturés
    - Aux extrémités du spectre
  - **Portabilité** 😐



Conseils (in)utiles



- Feedback
  - Rassurer l'utilisateur
- Traitement d'erreurs
  - Les erreurs sont inévitables
  - Faciliter la détection et la correction
  - Informer la cause et proposer des solutions



15

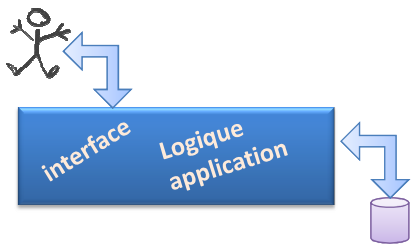
Conseils (in)utiles

- ***Usability*** (usabilité ou utilisabilité)
  - Capacité d'un système à permettre à l'utilisateur d'atteindre ses objectifs avec efficacité, en tout confort et sécurité (la sienne et celle des autres)

Conception : les architectures

Architectures

- Préhistoire : « 1 couche »
  - Pas de séparation entre l'interface et la logique d'application (métier)
  - Pas (ou peu) de réutilisation possible



Exemple

Outil de conversion °C → °F

```
package ihmexamples;

import java.util.Scanner;
import java.util.InputMismatchException;

/**
 * Outil de conversion Celsius - Fahreneit
 * @author kirsch
 */
public class CelsiusConverter {

    static double convertCtoF (double celc) {
        return (celc*9)/5 + 32;
    }

    static double readTemperature () {...}

    static void showTemperature (double fahr) {...}

    /**
     * Conversion entre Celsius et Fahrneit
     * @param args pas d'argument sur la ligne de commande
     */
    public static void main(String[] args) {
        double celc, fahr;

        celc = readTemperature();
        fahr = convertCtoF(celc);
        showTemperature(fahr);
    }
}
```


Exemple

Outil de conversion °C → °F

```
package ihmexamples;

import java.util.Scanner;
import java.util.InputMismatchException;

    static double readTemperature () {
        double celc = 0;
        Scanner in = new Scanner (System.in);

        System.out.println ("Temperature (° C) : ");

        try {
            celc = in.nextDouble();
        } catch (InputMismatchException ipe) {
            System.out.println("Sorry! I'm waiting for a float number.");
        }

        return celc;
    }

    static void showTemperature (double fahr) {
        System.out.printf("Temperature (° F) : %.2f \n", fahr);
    }

    public static void main(String[] args) {
        double celc, fahr;

        celc = readTemperature();
        fahr = convertCtoF(celc);
        showTemperature(fahr);
    }
}
```

Exemple

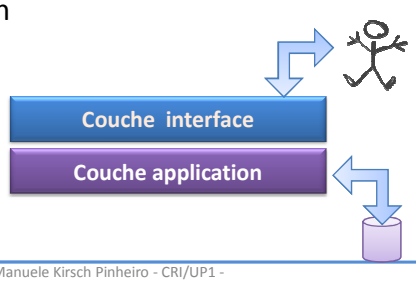
Outil de conversion °C → °F

```
package ihmexamples;

import java.util.Scanner;
import java.util.InputMismatchException;

    static double readTemperature () {
        double celc = 0;
        Scanner in = new Scanner (System.in);

        System.out.println ("Temperature (° C) : ");

        try {
            celc = in.nextDouble();
        } catch (InputMismatchException ipe) {
            System.out.println("Sorry! I'm waiting for a float number.");
        }

        return celc;
    }

    static void showTemperature (double fahr) {
```

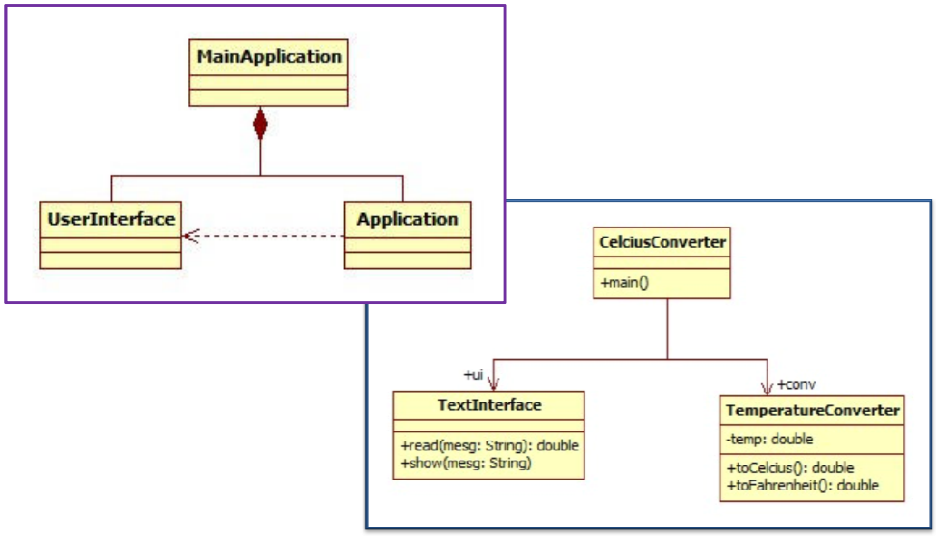
- Interface en mode texte
- Peu de réutilisation possible
  - Héritage ?
  - Délégation ?
- Changement d'interface demande réécriture quasi-totale

Architectures

- 2-couches
  - Séparation entre interface et logique d'application
  - Réutilisation possible, mais limitée
    - Dépendance application → interface
    - Changement d'interface demande des modifications sur l'application



Architecture en 2 couches



Exemple

```
package ihmexamples.tempconverter;

/**...*/
public class TemperatureConverter {
    private double temp;

    public TemperatureConverter(double temp) {
        this.temp = temp;
    }

    public double getTemp() {...}

    public void setTemp(double temp) {...}

    /**...*/
    public double toCelcius () {
        return ((this.temp-32)*5)/9;
    }

    /**...*/
    public double toFahrenheit () {
        return (this.temp*9)/5 + 32;
    }

}
```

```
package ihmexamples.tempconverter;

import java.util.Scanner;
import java.util.InputMismatchException;

/**...*/
public class TextInterface {

    /**...*/
    public double read(String mesg, String error) {
        Scanner in = new Scanner(System.in);
        double temp=0;

        System.out.print(mesg);
        try {
            temp = in.nextDouble();
        } catch (InputMismatchException ipe) {
            System.out.println(error);
        }

        return temp;
    }

    /**...*/
    public void show (String mesg, double temp) {
        System.out.printf("%s %.2f\n", mesg, temp);
    }
}
```

```
C:\Users\kirsch\Documents\NetBeansProjects\IHMExamples\build\classes>
java ihmexamples.tempconverter.CelciusConverter
Temperature (°C): -2.5
Temperature (°F): 27,50
```

```
package ihmexamples.tempconverter;

/**...*/
public class TemperatureConverter {
    private double temp;
```

```
package ihmexamples.tempconverter;

import java.util.Scanner;
import java.util.InputMismatchException;

/**...*/
public class TextInterface {
```

```
package ihmexamples.tempconverter;

/**
 * Tool for converting a temperature in Celcius (°C) into Fahrneheit (°F)
 * @author kirsch
 */
public class CelciusConverter {

    public static void main(String[] args) {
        double celc;
        TemperatureConverter conv;
        TextInterface ui = new TextInterface();

        celc = ui.read("Temperature (°C): ",
                       "Sorry! I'm waiting for a float number.");
        conv = new TemperatureConverter(celc);
        ui.show("Temperature (°F):", conv.toFahrenheit());
    }
}
```

Architectures

- Vers l'indépendance entre interface et logique d'application…
  - Modèle MVC
  - Architecture en 3-couches

(à venir…)

On y reviendra. . .

IHM & Java :
Programmation événementiel en AWT

AWT

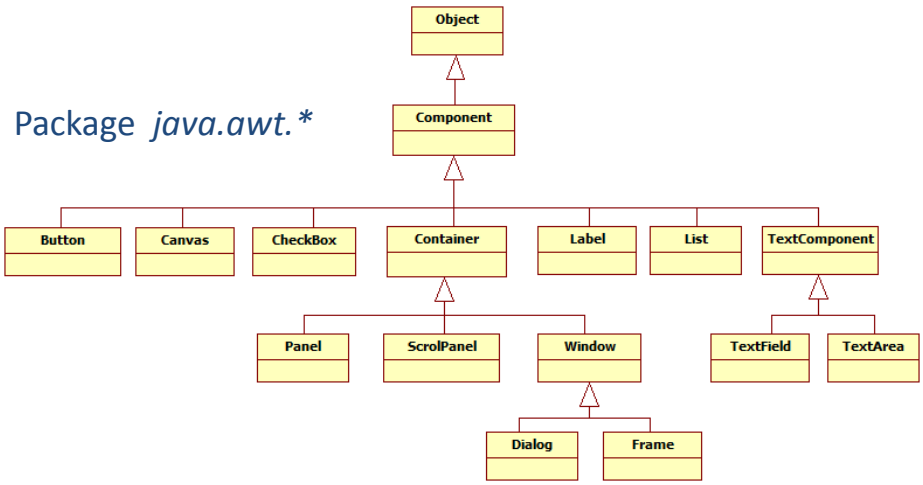
- AWT *: **Abstract Windowing Toolkit***
- *Toolkit* permettant la réalisation d'applications graphiques
  - Composants **interface utilisateurs** (fenêtres, menus, boutons, labels, zone de texte…)
  - Éléments **graphique 2D** (lignes, polygones…)
- Implémentation native
  - *Look & feel* du système hôte
  - Comportement pouvant varier selon la plateforme

AWT

- Historique **AWT**
  - Proposé dès JDK 1.0
  - Reformulé au JDK 1.1
    - Traitement d'événements : changement de *pattern*
      - de *Chain of responsibility* à *Observer*
  - JDK 1.2 : **Swing**
  - Depuis il s'améliore à chaque version…

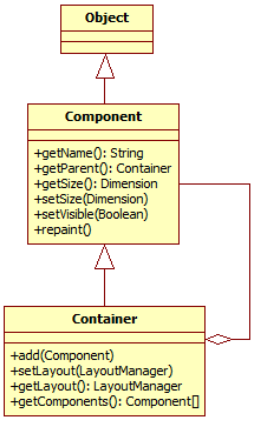
Hiérarchie de classes

Package *java.awt.**



Hiérarchie de classes

- **Component**
  - *"A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user."* (Java API)
- **Container**
  - *"A generic Abstract Window Toolkit (AWT) container object is a component that can contain other AWT components."* (Java API)

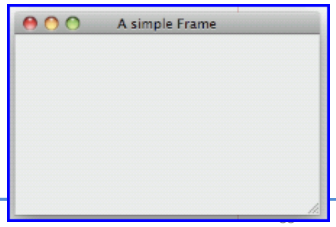


Containers

- Un *container* peut contenir d'autres *composants*
  - Méthode **add (Component c)** pour ajouter des nouveaux composants
- Un conteneur dispose d'un **gestionnaire de placement** (*LayoutManager*)
  - **LayoutManager** prend en charge de la disposition des composants dans le container
- Principaux containers :
  - Window
  - Frame
  - Panel
  - Dialog

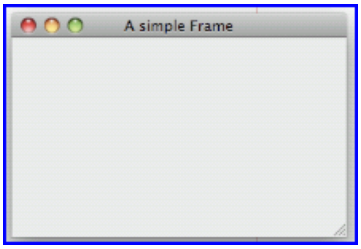
Containers

- **Window**
  - "*A Window object is a top-level window with no borders and no menubar.*" (Java API)
  - Toujours attaché à un autre Windows ou Frame (*parent*)
- **Frame**
  - "A Frame is a top-level window with a title and a border." (Java API)
  - Propose des méthodes tels que :
    - set/getResizable
    - set/getTitle

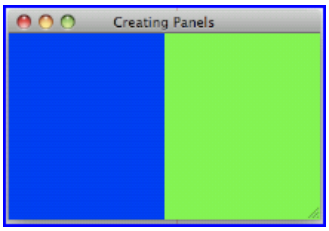


Containers

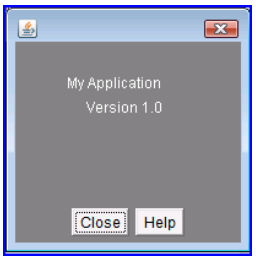
```
import java.awt.*;

public class NewFrame extends Frame
{
    // "final" variables are constants
    static final int H_SIZE = 300;
    static final int V_SIZE = 200;

    public NewFrame()
    {
        setTitle("A simple Frame");

        pack();
        setSize(H_SIZE, V_SIZE);
        setVisible(true);
    }

    public static void main(String args[])
    {
        new NewFrame();
    }
}
```



Containers

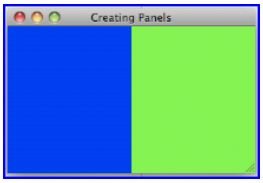


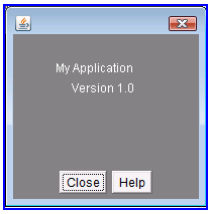
- **Panel**
  - "*A panel provides space in which an application can attach any other component, including other panels.*" (Java API)
- **Dialog**
  - A Dialog *"is typically used to take some form of input from the user."* (Java API)
  - Modality : set/getModalityType



Containers

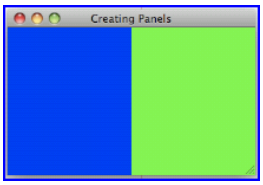
```
public NewPanel()
{
    setTitle("Creating Panels");
    initComponents();
    pack();
    setVisible(true);


    setSize(H_SIZE, V_SIZE);
    setLayout(new GridLayout()); // change the Frame layout

    // create a simple Panel (container) blue colored
    Panel p = new Panel();
    p.setSize(100, 110);
    p.setBackground(Color.BLUE);

    // create a simple Panel (container) green colored
    Panel q = new Panel();
    q.setSize(100, 110);
    q.setBackground(Color.GREEN);

    // add both panels to the Frame
    add(p);
    add(q);
}
```

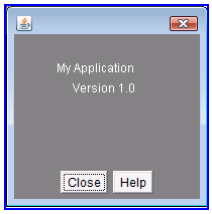




Containers

```
import java.awt.*;

class NewDialog extends Dialog {

    static int H_SIZE = 200;
    static int V_SIZE = 200;

    public NewDialog(Frame parent) {
        // Calls the parent telling it this
        // dialog is modal(i.e true)
        super(parent, true);
        setBackground(Color.gray);
        setLayout(new BorderLayout());

        // Two buttons "Close" and "Help"
        Panel p = new Panel();
        p.add(new Button("Close"));
        p.add(new Button("Help"));
        add("South", p);
        setSize(H_SIZE, V_SIZE);
    }

    public boolean action(Event evt, Object arg) {...}

    public void paint(Graphics g) {...}

}
```

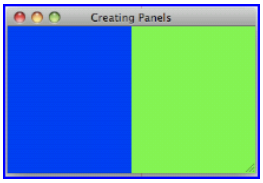


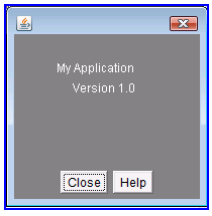


37

Containers

```
import java.awt.*;

class NewDialog extends Dialog {

    static int H_SIZE = 200;
    static int V_SIZE = 200;

    public NewDialog(Frame parent) {
        // Calls the parent telling it this
        // dialog is modal(i.e true)
        super(parent, true);
        setBackground(Color.gray);
        setLayout(new BorderLayout());

        // Two buttons "Close" and "Help"
        Panel p = new Panel();

    public static void main(String args[])
    {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                NewPanel panel = new NewPanel();
                panel.setVisible(true);
                new NewDialog(panel).setVisible(true);
            }
        });
    }
```

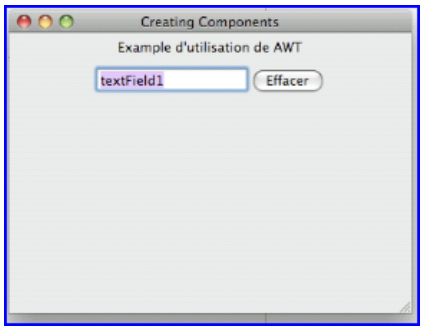




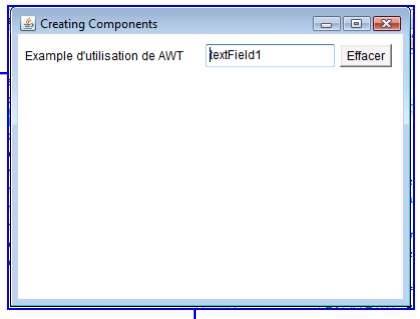
38

Components

- Component représente les composants d'interface utilisateur
  - Observation des différents événements (actions)
  - Différents méthodes de base
    - set/getSize
    - set/getVisible
    - repaint
- Quelques composants
  - Button
  - Label
  - TextField
  - Canvas



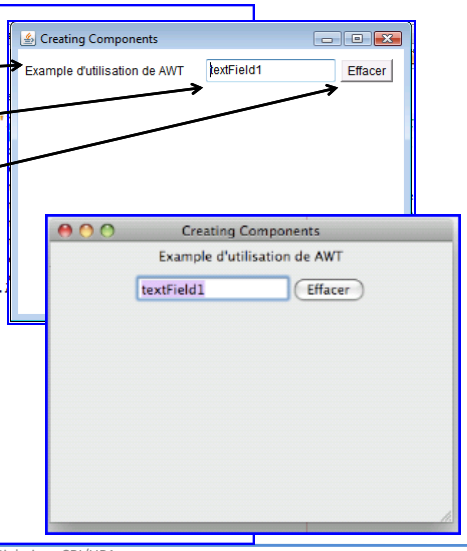
Components




```
// create a simple OK Button
Panel p = new Panel();
Panel bg = new Panel();

label1 = new java.awt.Label();
label1.setText("Example d'utilisation de AWT");

textField1 = new java.awt.TextField();
textField1.setColumns(15);
textField1.setText("textField1");

button1 = new java.awt.Button();
button1.setLabel("Effacer");
button1.addActionListener(new java.awt.event.ActionListener() {...});

p.add("North",label1);
p.add("Center",bg);

bg.add(textField1);
bg.add(button1);

bg.setVisible(true);
p.setVisible(true);

add("Center",p);
```

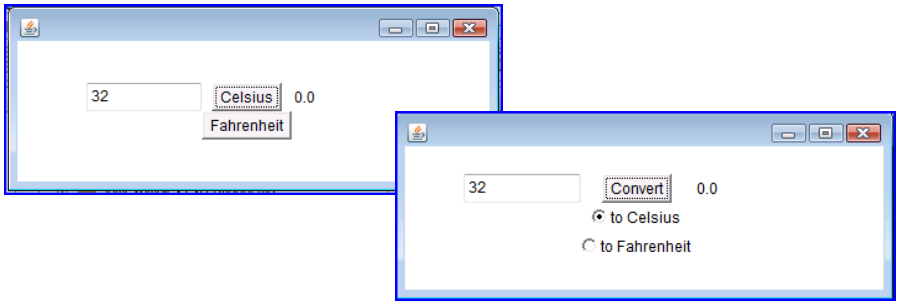
Components

```
// create a simple OK Button
Panel p = new Panel();
Panel bg = new Panel();

label1 = new java.awt.Label();
label1.setText("Example d'utilisation de AWT");

textField1 = new java.awt.TextField();
textField1.setColumns(15);
textField1.setText("textField1");

button1 = new java.awt.Button();
button1.setLabel("Effacer");
button1.addActionListener(new java.awt.event.

p.add("North",label1);
p.add("Center",bg);

bg.add(textField1);
bg.add(button1);

bg.setVisible(true);
p.setVisible(true);

add("Center",p);
```



Exemple

- Convertisseur de température
  - Réutilisation de la classe *TemperatureConverter*
  - Création d'une interface utilisateur avec AWT



Exemple

Création d'un container sous-classe de Frame

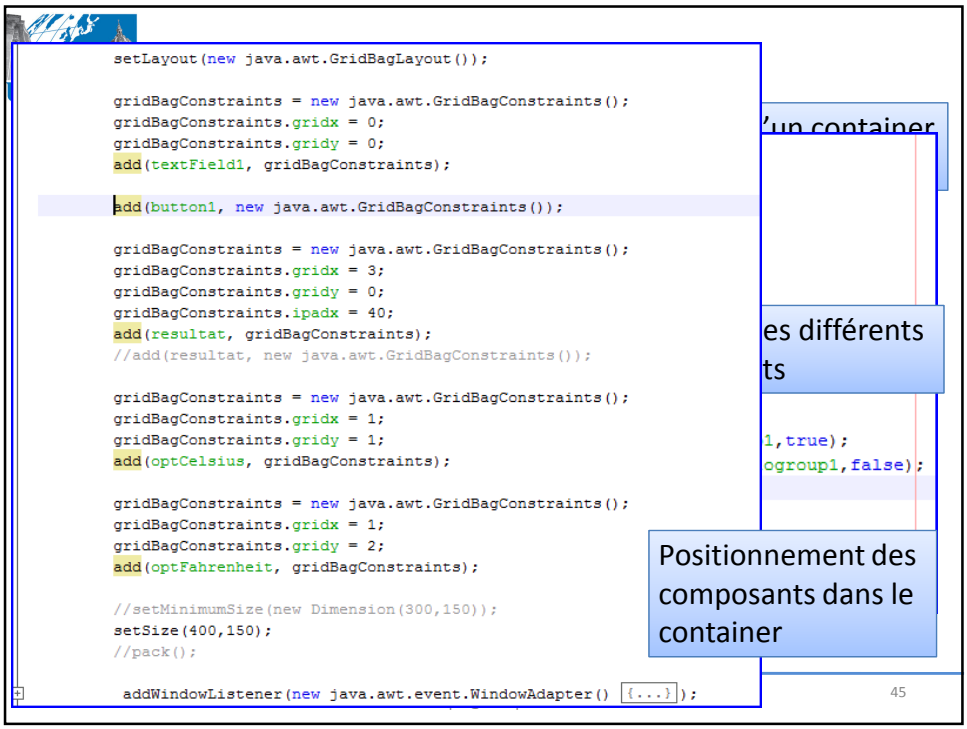
```
import java.awt.*;
import ihmexamples.tempconverter.*;

public class TempConverterAwtGUI extends java.awt.Frame {

    private java.awt.Button button1;
    private java.awt.CheckboxGroup radiogroup1;
    private java.awt.Checkbox optCelsius;
    private java.awt.Checkbox optFahrenheit;
    private java.awt.Label resultat;
    private java.awt.Panel panel1;
    private java.awt.TextField textField1;


    /** Creates new form TempConverterAwtGUI */
    public TempConverterAwtGUI() {
        initComponents();
    }
```

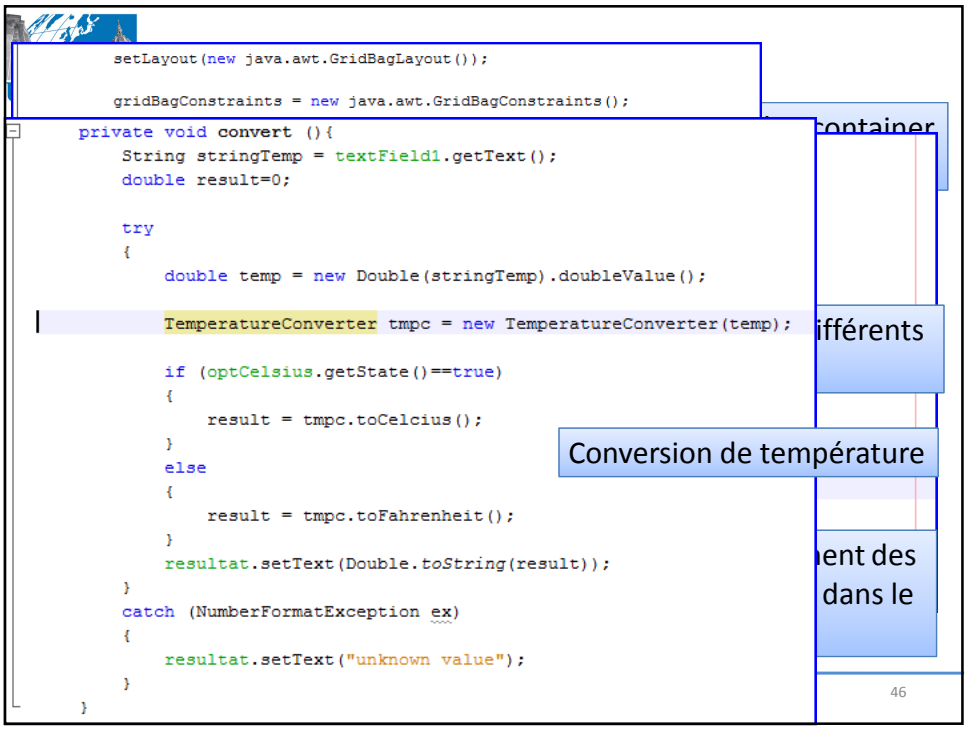

Exemple

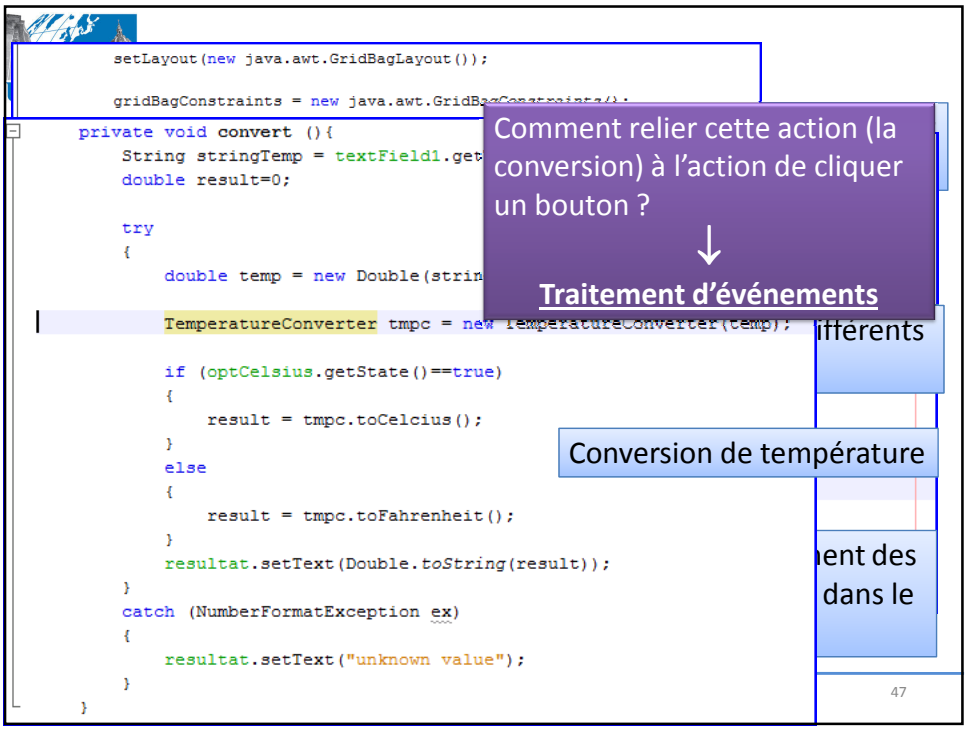
Création d'un container

Création des différents composants

```
    private void initComponents() {

        java.awt.GridBagConstraints gridBagConstraints;

        panel1 = new java.awt.Panel();
        panel1.setLayout(new java.awt.BorderLayout());

        textField1 = new java.awt.TextField();
        textField1.setText("");
        textField1.setColumns(10);

        radiogroup1 = new java.awt.CheckboxGroup();
        optCelsius = new java.awt.Checkbox("to Celsius",radiogroup1,true);
        optFahrenheit = new java.awt.Checkbox("to Fahrenheit",radiogroup1,false);
        button1 = new java.awt.Button();
        button1.setLabel("Convert");

        resultat = new java.awt.Label();
        resultat.setText("                ");
```


setLayout(new java.awt.GridBagLayout());
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
add(textField1, gridBagConstraints);
add(button1, new java.awt.GridBagConstraints());
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 0;
gridBagConstraints.ipadx = 40;
add(resultat, gridBagConstraints);
//add(resultat, new java.awt.GridBagConstraints());
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
add(optCelsius, gridBagConstraints);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 2;
add(optFahrenheit, gridBagConstraints);
//setMinimumSize(new Dimension(300,150));
setSize(400,150);
//pack();
addWindowListener(new java.awt.event.WindowAdapter() {...});
Positionnement des composants dans le container
45


setLayout(new java.awt.GridBagLayout());
gridBagConstraints = new java.awt.GridBagConstraints();
private void convert (){
String stringTemp = textField1.getText();
double result=0;
try
{
double temp = new Double(stringTemp).doubleValue();
TemperatureConverter tmpc = new TemperatureConverter(temp);
if (optCelsius.getState()==true)
{
result = tmpc.toCelcius();
}
else
{
result = tmpc.toFahrenheit();
}
resultat.setText(Double.toString(result));
}
catch (NumberFormatException ex)
{
resultat.setText("unknown value");
}
}
Conversion de température
46



Traitement d'événements

- **Objectif**
  - Gérer l'interaction avec l'utilisateur
- Chaque action génère un **événement**
  - Click souris
  - Click bouton
  - Fermer une fenêtre
  - …
- Pour traiter un type d'action donné, on doit souscrire au type d'événement lui correspondant
  - Définition des *listeners*
  - Design pattern *Observer*

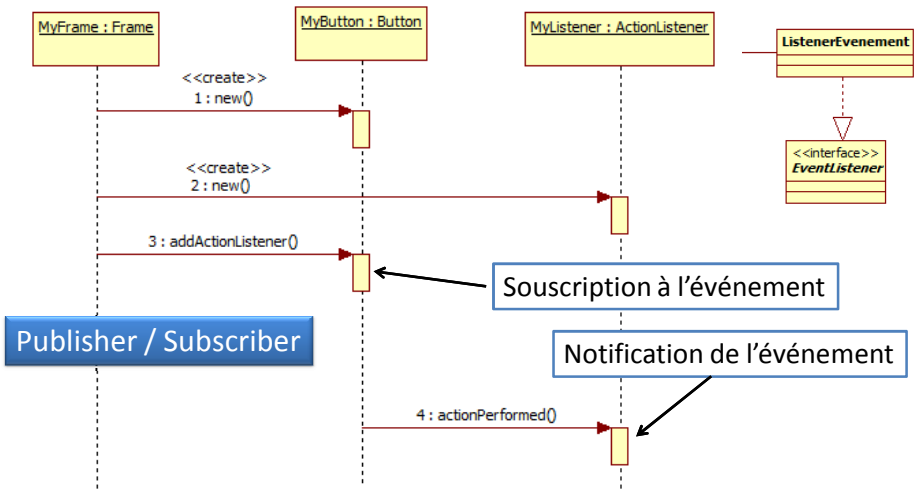
Traitement d'événements

- Les écouteurs (*listeners*) souscrivent aux événements auprès des sources potentiels des événements

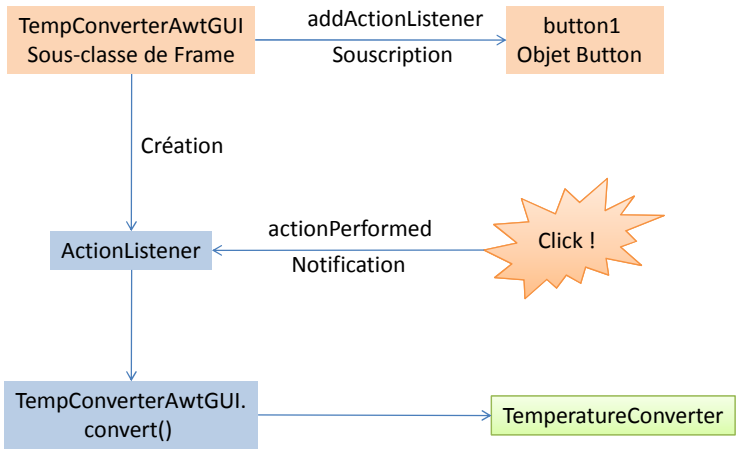
- Les sources informent les *listeners* de l'occurrence de l'événement

Source: SUN Java Tutorial, « Introduction to Event Listeners »

Traitement d'événements

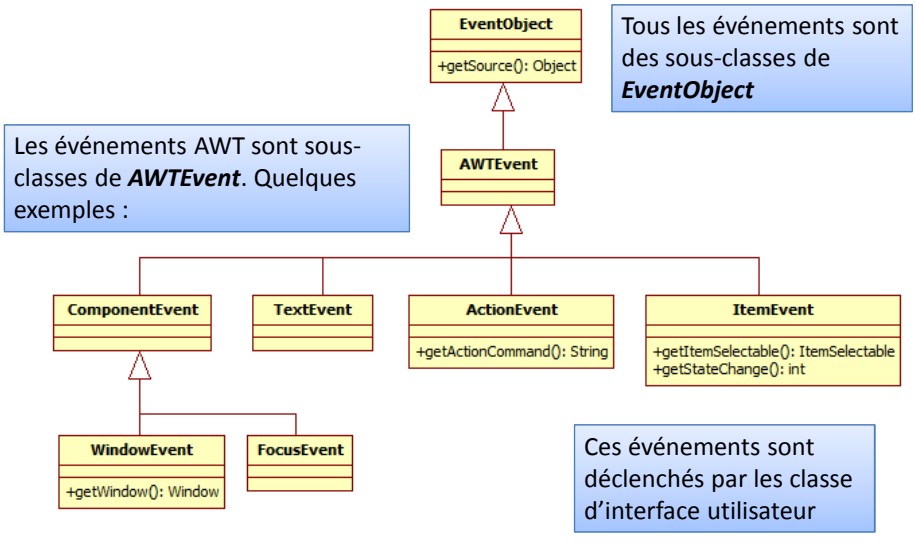


Exemple




```
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        textField1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                textField1ActionPerformed(evt);
            }
        });

        button1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                button1ActionPerformed(evt);
            }
        });
    }

    private void convert (){...}

    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt) {...}

    private void button1ActionPerformed(java.awt.event.ActionEvent evt) {
        convert();
    }
```

Hiérarchie d'événements



Tous les événements sont des sous-classes de ***EventObject***

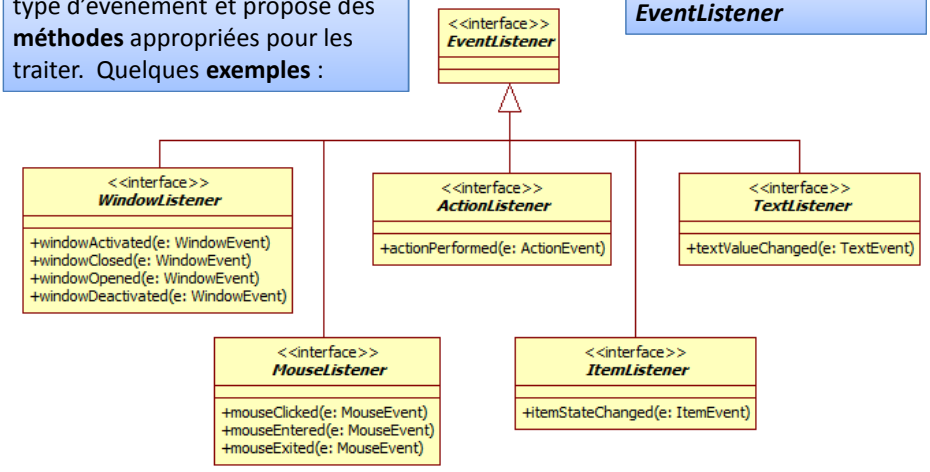
Les événements AWT sont sous-classes de ***AWTEvent***. Quelques exemples :

Ces événements sont déclenchés par les classe d'interface utilisateur

Hiérarchie des *listeners*

Chaque ***listener*** observe un certain type d'événement et propose des **méthodes** appropriées pour les traiter. Quelques **exemples** :

Les *listeners* spécialisent ***EventListener***



Définition d'un *listener*

- Implémentation de l'interface XXXXListener
- Possibilités :
  - dans la propre classe
  - dans une classe anonyme / inner class
  - dans une classe dédiée / séparée

Traitement dans la propre classe

- Le *container* qui contient le composant implémente l'interface *XXXXListener* approprié

```
public class ClassAsListener extends Frame implements
    ActionListener, WindowListener { ... }
```

- Le container registre lui-même comme *listener* auprès du composant

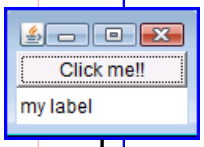
```
addWindowListener(this);
button1.addActionListener(this);
```

- Il implémente les méthodes de l'interface *XXXXListener*

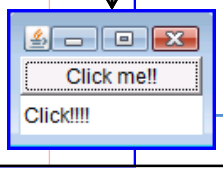
```
public void actionPerformed(ActionEvent e) { … }
public void windowClosing(WindowEvent e) { … }
```

Exemple dans la propre classe

```
public class ClassAsListener extends Frame implements ActionListener, WindowListener  {

    /** Creates new form ClassAsListener */
    public ClassAsListener() {
        myinitComponents();
    }

    private void myinitComponents() {
        addWindowListener(this);

        button1 = new java.awt.Button();
        label1 = new java.awt.Label();

        button1.setLabel("Click me!!");
        add(button1, java.awt.BorderLayout.CENTER);
        button1.addActionListener(this);

        label1.setText("my label");
        add(label1, java.awt.BorderLayout.SOUTH);

        pack();
    }

    public void actionPerformed(ActionEvent e) {
        label1.setText("Click!!!!");
    }

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



ClassAsListener.
actionPerformed()



Traitement dans une classe anonyme

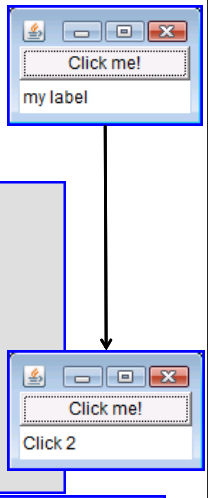
- Utilisation d'une **classe interne anonyme** pour traiter les événements
  - Classes définies localement à l'intérieur d'une méthode

```
button1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        button1ActionPerformed(evt);
    }
});
```

- Principe souvent utilisé par les IDEs

Exemple dans une classe anonyme

```
public class AnonymousListener extends java.awt.Frame {

    /** Creates new form AnonymousListener */
    public AnonymousListener() {
        this.count = 0;
        initComponents();
    }

        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        button1.setLabel("Click me!");
        button1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                button1ActionPerformed(evt);
            }
        });
        add(button1, java.awt.BorderLayout.CENTER);

    private void button1ActionPerformed(java.awt.event.ActionEvent evt) {
        label1.setText("Click "+ (++this.count));
    }
```



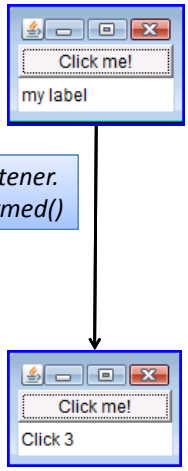
Traitement dans une *inner class*

- Utilisation d'une **classe interne** non-anonyme **(*inner class*)** pour traiter les événements

```
public class InnerAsListener extends Frame {
    public InnerAsListener() { …
        button1.addActionListener(new MyActionListener());
    …
    private class MyActionListener implements ActionListener {
        public void actionPerformed(ActionEvent e) { … }
    }
} // fin InnerAsListener
```

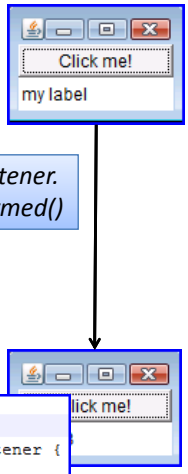
Exemple dans une *inner class*

```
public class InnerAsListener extends java.awt.Frame {

    /** Creates new form InnerAsListener */
    public InnerAsListener() {
        this.count = 0;
        myInitComponents();
    }

    private void myInitComponents() {

        button1 = new java.awt.Button();
        label1 = new java.awt.Label();

        addWindowListener(new java.awt.event.WindowAdapter() {...});

        button1.setLabel("Click me!");
        add(button1, java.awt.BorderLayout.CENTER);
        button1.addActionListener(new MyActionListener());

        label1.setText("my label");
        add(label1, java.awt.BorderLayout.SOUTH);

        pack();
    }// </editor-fold>
```



Exemple dans une *inner class*

```
public class InnerAsListener extends java.awt.Frame {

    /** Creates new form InnerAsListener */
    public InnerAsListener() {
        this.count = 0;
        myInitComponents();
    }

    private void myInitComponents() {

        button1 = new java.awt.Button();
        label1 = new java.awt.Label();

        addWindowListener(new java.awt.event.WindowAdapter() {...});

        button1.setLabel("Click me!");
        add(button1, java.awt.BorderLayout.CENTER);
        button1.addActionListener(new MyActionListener());

    private int count;

    private class MyActionListener implements java.awt.event.ActionListener {
        public void actionPerformed(ActionEvent e) {
            label1.setText("Click "+ (++count));
        }
    }
}
```



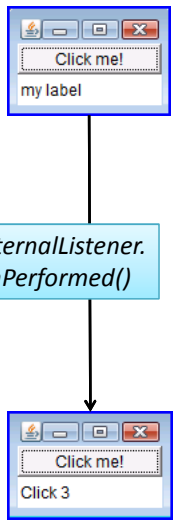
Traitement dans une classe dédiée

- Utilisation d'une **classe** à part (extérieure), entièrement **dédiée** au traitement des **événements**
- Cette classe implémente les interfaces listeners souhaitées
- La **communication** entre les classes doit être gérée
  - **Classes d'interface → classe de traitement**

Exemple dans une classe dédiée

Classe externe

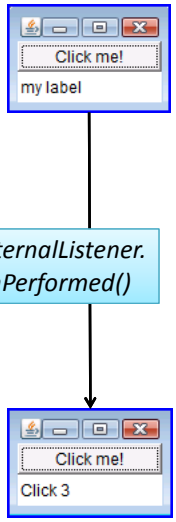
```
public class MyExternalListener implements ActionListener {
   int count;
   ExternalListener frame;

   public MyExternalListener (ExternalListener parent) {
      this.frame = parent;
      this.count = 0;
   }
   public void actionPerformed(ActionEvent e) {
      frame.setLabel("Click "+ (++count));
   }
}
```



Exemple dans une classe dédiée

L'interface utilisateur

```
public class ExternalListener extends Frame {
    public ExternalListener() {  . . .
        listener = new MyExternalListener(this);
        button1 = new java.awt.Button();
        button1.setLabel("Click me!");
        button1.addActionListener(listener);
        add(button1, java.awt.BorderLayout.CENTER);
    … }
  public void setLabel (String text) {
     label1.setText(text);
  }

  private MyExternalListener listener;
```



Exercices

Exercices

1) Implémenter un convertisseur °C ↔ °F en utilisant des composants AWT
   a) Utiliser un listener anonyme
   b) Utiliser une classe anonyme
   c) Utiliser une classe dédiée

2) Implémenter une calculatrice avec les 4 opérations de base (+, -, *, /)

3) Jeu de scrabble : version 1
   - Architecture en 2 couches
   - Éléments : interface, dictionnaire statique, application

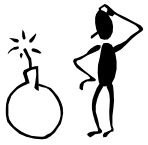
mkirschpin@univ-paris1.fr

Concepts supplémentaires

Exceptions

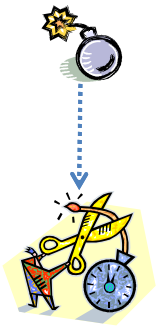
Design Pattern

Exceptions



- Exception : situation exceptionnelle
- Traitement d'exceptions en Java

```
try {
    //code pouvant lancer l'exception
    . . .
} catch (Exception ex) {
    //code traitement le problème
    . . .
}
```

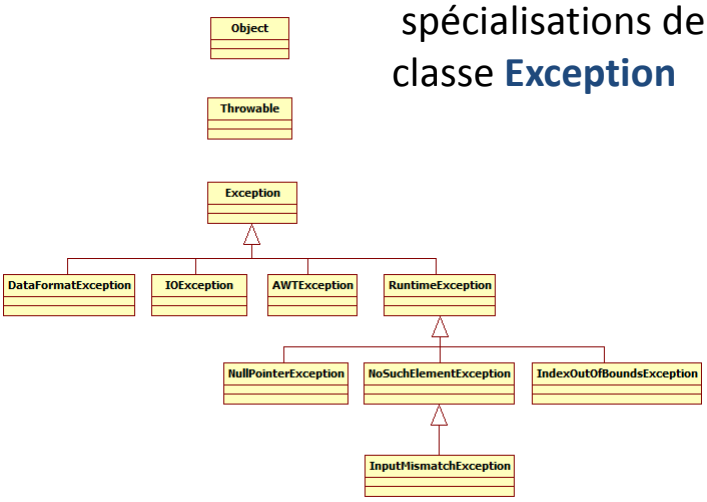


Exceptions

- En cas des problème, on lance une exception
  - Création d'un objet Exception lui correspondant
  - Lancement : throw
    - **throw new NullPointerException ();**
- Le flux d'exécution est abandonné
  - Observation des exceptions : try
  - Capture d'une exception : catch
  - Et si personne observe / capture l'exception ?

Hiérarchie d'exceptions

- Toutes des exceptions qu'on traite sont des spécialisations de la classe **Exception**



71

Exemple

Définition d'une nouvelle exception
« NegativeFactorialException »

```
class NegativeFactorialException extends ArithmeticException {

    public NegativeFactorialException(String s) {
        super(s);
    }

    public NegativeFactorialException (int x) {
        super (new String("Factorial of " + x + " impossible"));
    }

    public NegativeFactorialException() {
    }
}
```

On n'est pas obligé à toujours créer ses propres exceptions.

72

Exemple

```
class NegativeFactorialE

    public NegativeFacto
        super(s);
    }

    public NegativeFacto
        super (new Strin
    }

    public NegativeFacto
    }
}
```

```
public class ExceptionDemo {
    public int factorial (int n) throws NegativeFactorialException {
        int f = 1;

        if (n < 0)
            throw new NegativeFactorialException (n);

        for (int i=n; i>0; i--) {
            f *= n;
        }
        return f;
    }

    public static void main (String args[]) {
        int fa=0, fb=0;
        ExceptionDemo ed = new ExceptionDemo();

        try {
            fa = ed.factorial(-4);
            fb = ed.factorial(5);   //jamais atteint
        }
        catch (NegativeFactorialException nfe) {
            System.out.println ("Exception! " + nfe.getMessage());
            nfe.printStackTrace(System.out);
        }

        System.out.println ("fa = " + fa + " fb = " + fb);
    }
}
```

Exemple

```
Exception! Factorial of -4 impossible
ihmexamples.exceptiondemo.NegativeFactorialException: Factorial of -4 impossible
        at ihmexamples.exceptiondemo.ExceptionDemo.factorial(ExceptionDemo.java:20)
        at ihmexamples.exceptiondemo.ExceptionDemo.main(ExceptionDemo.java:34)
fa = 0 fb = 0
```


```
class NegativeFactorialE

    public NegativeFacto
        super(s);
    }

    public NegativeFacto
        super (new Strin
    }

    public NegativeFacto
    }
}
```

```
public class ExceptionDemo {
    public int factorial (int n) throws NegativeFactorialException {
        int f = 1;

        if (n < 0)
            throw new NegativeFactorialException (n);

        for (int i=n; i>0; i--) {
            f *= n;
        }
        return f;
    }

    public static void main (String args[]) {
        int fa=0, fb=0;
        ExceptionDemo ed = new ExceptionDemo();

        try {
            fa = ed.factorial(-4);
            fb = ed.factorial(5);   //jamais atteint
        }
        catch (NegativeFactorialException nfe) {
            System.out.println ("Exception! " + nfe.getMessage());
            nfe.printStackTrace(System.out);
        }

        System.out.println ("fa = " + fa + " fb = " + fb);
    }
}
```

Design patterns

- **Définition d'un pattern :**
  - Une règle en trois parties exprimant une relation entre un contexte donné, un problème récurrent dans ce contexte et une solution
  - Réutilisation d'un certain savoir faire
  - Un problème récurrent pour lequel un décrit une solution reconnue
- **Design pattern**
  - Patron de conception logicielle
  - Gang of Four 94 :
    - Gamma, Helm, Johnson, Vlissides, « Design Patterns: Elements of reusable object-oriented software », Adison-Wesley, 1994
- Autres types sont possibles
  - Architecturaux
  - Organisationnels
  - . . .