Programmation Orientée Objet

6 – Interfaces Graphiques en Java: la bibliothèque Swing

P. Berthomé

INSA Centre Val de Loire
Département STI – 3 ème année

5 décembre 2017

Plan

Swing

- Les éléments graphiques
- Les interactions

Plan

Swing

- Les éléments graphiques
- Les interactions

Bibliothèques

- AWT (*Abstract Window Toolkit*) : premiers éléments *obsolètes*
  (`Frame`, `Button`, ...)
- Swing ; éléments basés sur AWT
  (`JFrame`, `Jbutton`, ...)
- SWT (*Standard Widget Toolkit*) développé par IBM

Premier exemple

```
public class Fenetre {
  public static void main(String[] args) {
    JPanel panneau = new JPanel();
    panneau.setBackground(Color.blue);
    panneau.setPreferredSize(new Dimension(250, 350));
```

Premier exemple

```
public class Fenetre {
  public static void main(String[] args) {
    JPanel panneau = new JPanel();
    panneau.setBackground(Color.blue);
    panneau.setPreferredSize(new Dimension(250, 350));

    JFrame cadre = new JFrame("Premier exemple");
    cadre.setLocation(400, 300);
    cadre.setContentPane(panneau);
```

Premier exemple

```
public class Fenetre {
  public static void main(String[] args) {
    JPanel panneau = new JPanel();
    panneau.setBackground(Color.blue);
    panneau.setPreferredSize(new Dimension(250, 350));

    JFrame cadre = new JFrame("Premier exemple");
    cadre.setLocation(400, 300);
    cadre.setContentPane(panneau);

    cadre.pack();
    cadre.setVisible(true);
    cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }}
```

Commentaires

javax.swing.JFrame

JFrame	fenêtre principale de l'application
méthode pack()	calcule la taille de la fenêtre avec ses composants
setVisible()	par défaut, la fenêtre créée est invisible

Commentaires

javax.swing.JFrame

JFrame	fenêtre principale de l'application
méthode pack()	calcule la taille de la fenêtre avec ses composants
setVisible()	par défaut, la fenêtre créée est invisible

javax.swing.JPanel

JPanel	conteneur permettant de décrire l'interface
setBackground(), ...	méthodes permettant de modifier l'aspect

Commentaires

javax.swing.JFrame

JFrame	fenêtre principale de l'application
méthode pack()	calcule la taille de la fenêtre avec ses composants
setVisible()	par défaut, la fenêtre créée est invisible

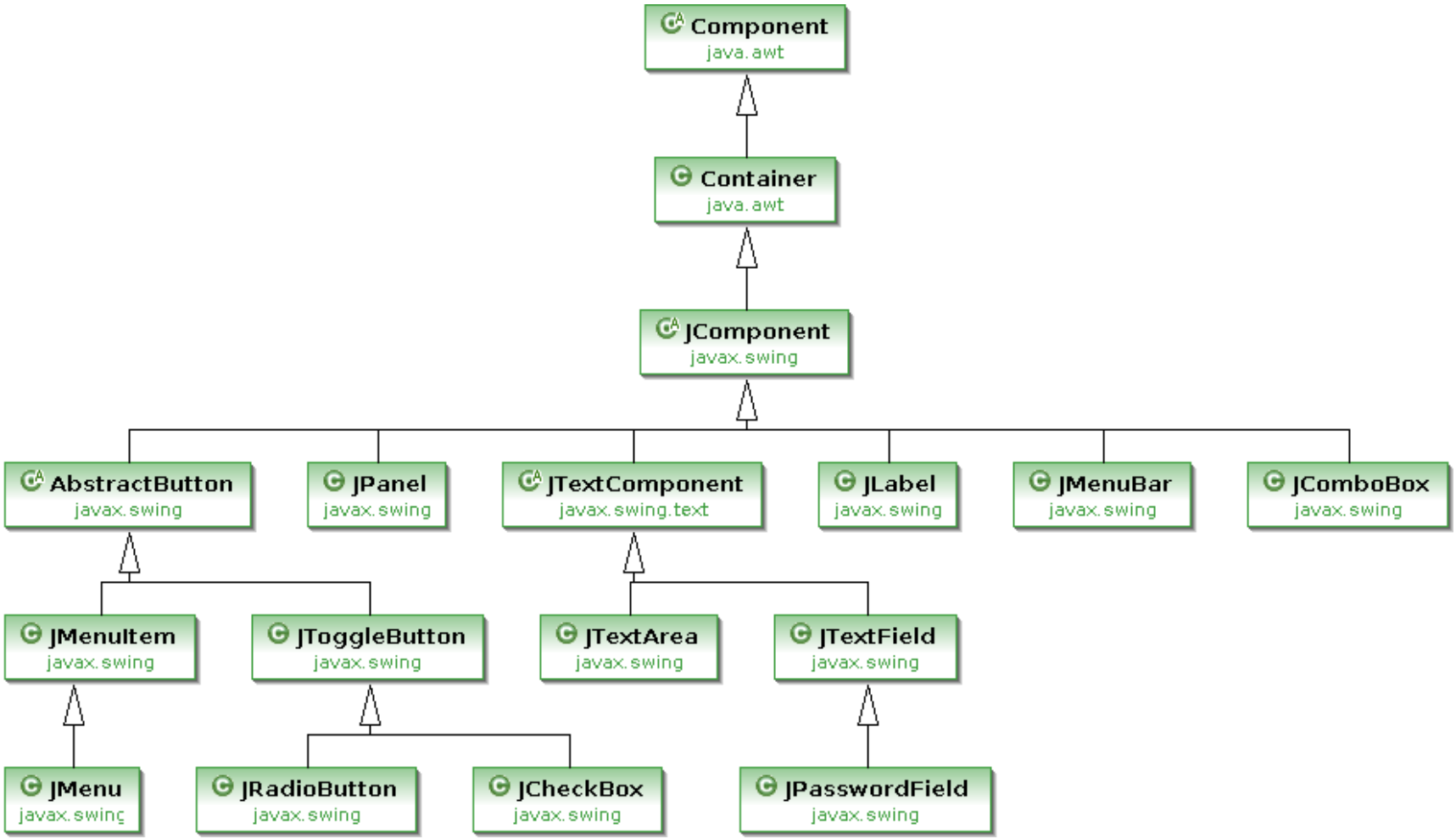
javax.swing.JPanel

JPanel	conteneur permettant de décrire l'interface
setBackground(), ...	méthodes permettant de modifier l'aspect

java.awt.Color et java.awt.Dimension

Définitions des couleurs et des points

Composants Swing – Hiérarchie (très) partielle



Quelques éléments graphiques

Taxonomie

Interacteurs Objets généraux interagissant avec l'utilisateur (`AbstractButton`, `JLabel`, `JComboBox`,)

Boutons Boutons classiques (`JButton`), cases à cocher (`JCheckBox`), Choix exclusif (`JRadioButton`)

Texte `JTextComponent` se dérivant en `JTextField`, `JtextArea` et `JEditorPane`

Conteneurs à peu près tous les composants, mais principalement `JPanel`, `JToolBar`

Quelques éléments graphiques

Taxonomie

Interacteurs Objets généraux interagissant avec l'utilisateur (`AbstractButton`, `JLabel`, `JComboBox`,)

Boutons Boutons classiques (`JButton`), cases à cocher (`JCheckBox`), Choix exclusif (`JRadioButton`)

Texte `JTextComponent` se dérivant en `JTextField`, `JtextArea` et `JEditorPane`

Conteneurs à peu près tous les composants, mais principalement `JPanel`, `JToolBar`

Fenêtres

JFrame Fenêtre principale de l'application

JDialog Fenêtre secondaire, dépendante de la JFrame, souvent temporaire et modale

Arbre d'instanciation

Hiérarchisation de l'interface

- Chaque objet *contient* ses enfants

Arbre d'instanciation

Hiérarchisation de l'interface

- Chaque objet *contient* ses enfants
- **Superposition** : enfants affichés au dessus des parents

Arbre d'instanciation

Hiérarchisation de l'interface

- Chaque objet *contient* ses enfants
- **Superposition** : enfants affichés au dessus des parents
- **Clipping** : enfants ne dépassent pas les parents

Arbre d'instanciation

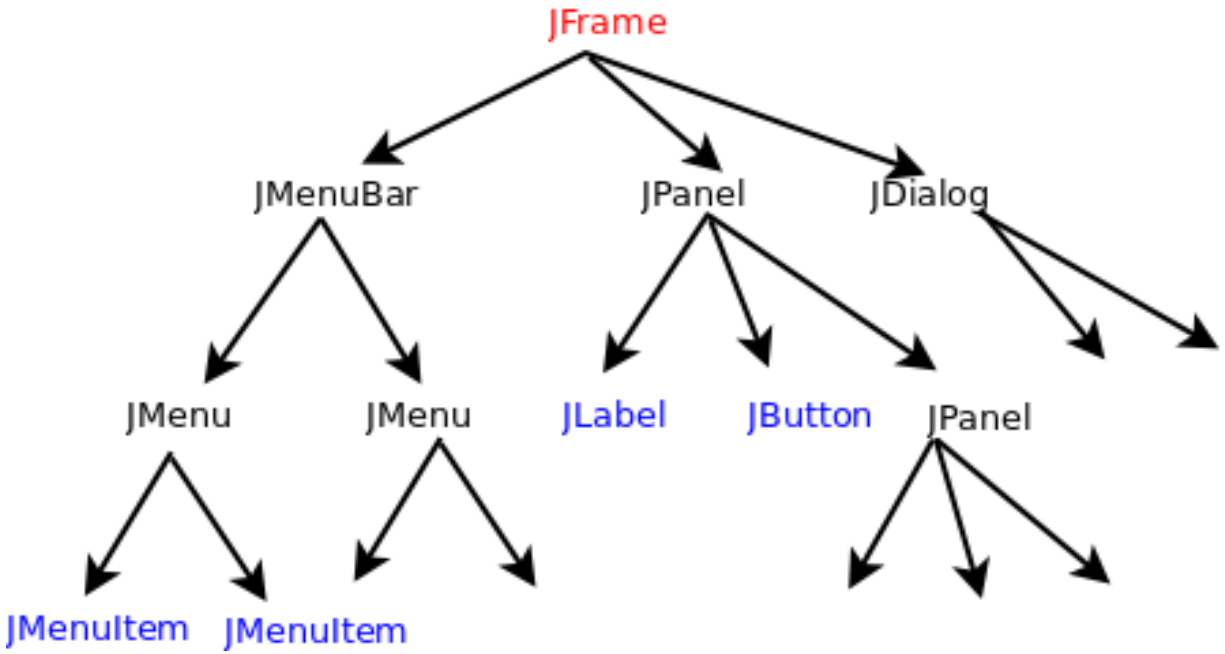
Hiérarchisation de l'interface

- Chaque objet *contient* ses enfants
- **Superposition** : enfants affichés au dessus des parents
- **Clipping** : enfants ne dépassent pas les parents

Arbre d'instanciation

Hiérarchisation de l'interface

- Chaque objet *contient* ses enfants
- **Superposition** : enfants affichés au dessus des parents
- **Clipping** : enfants ne dépassent pas les parents



Composition d'une interface graphique

Layout

- Sans *Layout*, le dernier composant se superpose au précédent

Composition d'une interface graphique

Layout

- Sans *Layout*, le dernier composant se superpose au précédent
- Différents types (voir `docs.oracle.com.javase/tutorial/uiswing/layout/index.html`)

Composition d'une interface graphique

Layout

- Sans *Layout*, le dernier composant se superpose au précédent
- Différents types (voir `docs.oracle.com.javase/tutorial/uiswing/layout/index.html`)

  BorderLayout 5 positions

Composition d'une interface graphique

Layout

- Sans *Layout*, le dernier composant se superpose au précédent
- Différents types (voir `docs.oracle.com.javase/tutorial/uiswing/layout/index.html`)

  BorderLayout 5 positions

  FlowLayout composants ajoutés les uns à la suite de autres

Composition d'une interface graphique

Layout

- Sans *Layout*, le dernier composant se superpose au précédent
- Différents types (voir `docs.oracle.com.javase/tutorial/uiswing/layout/index.html`)

  BorderLayout 5 positions

  FlowLayout composants ajoutés les uns à la suite de autres

  GridLayout(a,b) composants ajoutés dans un tableau virtuel $a \times b$

Composition d'une interface graphique

Layout

- Sans *Layout*, le dernier composant se superpose au précédent
- Différents types (voir `docs.oracle.com.javase/tutorial/uiswing/layout/index.html`)

  BorderLayout 5 positions

  FlowLayout composants ajoutés les uns à la suite de autres

  GridLayout(a,b) composants ajoutés dans un tableau virtuel $a \times b$

  GridBagLayout pour des interfaces plus complexes

Composition d'une interface graphique

Layout

- Sans *Layout*, le dernier composant se superpose au précédent
- Différents types (voir `docs.oracle.com.javase/tutorial/uiswing/layout/index.html`)

  BorderLayout 5 positions

  FlowLayout composants ajoutés les uns à la suite de autres

  GridLayout(a,b) composants ajoutés dans un tableau virtuel $a \times b$

  GridBagLayout pour des interfaces plus complexes
- On peut les composer.

Une nouvelle classe

```
public class Ardoise extends JPanel {
    private static final long serialVersionUID = 1L;
    private boolean possedeDisque = true;
```

Une nouvelle classe

```
public class Ardoise extends JPanel {
    private static final long serialVersionUID = 1L;
    private boolean possedeDisque = true;

    public Ardoise(){
        setBackground(Color.YELLOW);
        setPreferredSize(new Dimension(200, 150));}
```

Une nouvelle classe

```
public class Ardoise extends JPanel {
    private static final long serialVersionUID = 1L;
    private boolean possedeDisque = true;

    public Ardoise(){
        setBackground(Color.YELLOW);
        setPreferredSize(new Dimension(200, 150));}

    public void setPossedeDisque(boolean avec){
        possedeDisque = avec;}
```

Une nouvelle classe

```
public class Ardoise extends JPanel {
    private static final long serialVersionUID = 1L;
    private boolean possedeDisque = true;

    public Ardoise(){
        setBackground(Color.YELLOW);
        setPreferredSize(new Dimension(200, 150));}

    public void setPossedeDisque(boolean avec){
        possedeDisque = avec;}

    public void Dessiner(Graphics g){
        g.setColor(Color.RED); g.fillOval(60, 35, 80, 80);}
```

Une nouvelle classe

```
public class Ardoise extends JPanel {
    private static final long serialVersionUID = 1L;
    private boolean possedeDisque = true;

    public Ardoise(){
        setBackground(Color.YELLOW);
        setPreferredSize(new Dimension(200, 150));}

    public void setPossedeDisque(boolean avec){
        possedeDisque = avec;}

    public void Dessiner(Graphics g){
        g.setColor(Color.RED); g.fillOval(60, 35, 80, 80);}

    public void paintComponent(Graphics g){
        super.paintComponent(g); if (possedeDisque) Dessiner(g);}}
```

Utilisation de ce nouveau composant

```
public static void main(String[] args) {
    JFrame cadre = new JFrame("un disque");
    cadre.setContentPane(new Ardoise());
    cadre.setLocation(400, 300);
    cadre.pack();
    cadre.setVisible(true);
    cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
```

Utilisation de ce nouveau composant

```
public static void main(String[] args) {
    JFrame cadre = new JFrame("un disque");
    cadre.setContentPane(new Ardoise());
    cadre.setLocation(400, 300);
    cadre.pack();
    cadre.setVisible(true);
    cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
```

Utilisation de ce nouveau composant

```
public static void main(String[] args) {
    JFrame cadre = new JFrame("un disque");
    cadre.setContentPane(new Ardoise());
    cadre.setLocation(400, 300);
    cadre.pack();
    cadre.setVisible(true);
    cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
```

Utilisation d'un Layout

```
public static void main(String[] args) {
    JButton trace = new JButton("Trace");
    JButton efface = new JButton("Efface");
    Ardoise ardoise = new Ardoise();
```

Utilisation d'un Layout

```
public static void main(String[] args) {
    JButton trace = new JButton("Trace");
    JButton efface = new JButton("Efface");
    Ardoise ardoise = new Ardoise();

    JPanel pan_boutons = new JPanel();
    pan_boutons.add(trace);
    pan_boutons.add(efface);
```

Utilisation d'un Layout

```
public static void main(String[] args) {
    JButton trace = new JButton("Trace");
    JButton efface = new JButton("Efface");
    Ardoise ardoise = new Ardoise();

    JPanel pan_boutons = new JPanel();
    pan_boutons.add(trace);
    pan_boutons.add(efface);

    JPanel panneau = new JPanel(new BorderLayout(5,5));
    panneau.add(pan_boutons, BorderLayout.NORTH);
    panneau.add(ardoise, BorderLayout.CENTER);
```

Utilisation d'un Layout

```
public static void main(String[] args) {
    JButton trace = new JButton("Trace");
    JButton efface = new JButton("Efface");
    Ardoise ardoise = new Ardoise();

    JPanel pan_boutons = new JPanel();
    pan_boutons.add(trace);
    pan_boutons.add(efface);

    JPanel panneau = new JPanel(new BorderLayout(5,5));
    panneau.add(pan_boutons, BorderLayout.NORTH);
    panneau.add(ardoise, BorderLayout.CENTER);

    JFrame cadre = new JFrame("une interface");
    cadre.setContentPane(panneau);
    // la suite comme d'habitude
```

L'interface

```
public class UneInterface extends JPanel {
    private static final long serialVersionUID = 1L;
    protected JButton trace = new JButton("trace");
    protected JButton efface = new JButton("Efface");
    protected Ardoise ardoise = new Ardoise();
```

L'interface

```
public class UneInterface extends JPanel {
    private static final long serialVersionUID = 1L;
    protected JButton trace = new JButton("trace");
    protected JButton efface = new JButton("Efface");
    protected Ardoise ardoise = new Ardoise();

    public UneInterface(){
        JPanel pan_boutons = new JPanel();
        pan_boutons.add(trace); pan_boutons.add(efface);
        setLayout(new BorderLayout(5, 5));
        add(pan_boutons, BorderLayout.NORTH);
        add(ardoise, BorderLayout.CENTER);}
```

L'interface

```
public class UneInterface extends JPanel {
    private static final long serialVersionUID = 1L;
    protected JButton trace = new JButton("trace");
    protected JButton efface = new JButton("Efface");
    protected Ardoise ardoise = new Ardoise();

    public UneInterface(){
        JPanel pan_boutons = new JPanel();
        pan_boutons.add(trace); pan_boutons.add(efface);
        setLayout(new BorderLayout(5, 5));
        add(pan_boutons, BorderLayout.NORTH);
        add(ardoise, BorderLayout.CENTER);}

    public static void main(String [] args){
        JFrame IInterface = new JFrame("Une interface"); // la suite
```

Interactivité

Programmation événementielle

- Répondre aux interactions des utilisateurs

Gestion des événements

Interactivité

Programmation événementielle

- Répondre aux interactions des utilisateurs
- en Java, on utilise des *Listener*

Gestion des événements

Interactivité

Programmation événementielle

- Répondre aux interactions des utilisateurs
- en Java, on utilise des *Listener*
- Interfaces java donnant les comportements à définir

Gestion des événements

Interactivité

Programmation événementielle

- Répondre aux interactions des utilisateurs
- en Java, on utilise des *Listener*
- Interfaces java donnant les comportements à définir

Gestion des événements

- Boucle infinie :

Interactivité

Programmation événementielle

- Répondre aux interactions des utilisateurs
- en Java, on utilise des *Listener*
- Interfaces java donnant les comportements à définir

Gestion des événements

- Boucle infinie :
  - récupère les événements

Interactivité

Programmation événementielle

- Répondre aux interactions des utilisateurs
- en Java, on utilise des *Listener*
- Interfaces java donnant les comportements à définir

Gestion des événements

- Boucle infinie :
  - récupère les événements
  - appelle les fonctions de *callback* des composants graphiques

Interactivité

Programmation événementielle

- Répondre aux interactions des utilisateurs
- en Java, on utilise des *Listener*
- Interfaces java donnant les comportements à définir

Gestion des événements

- Boucle infinie :
  - récupère les événements
  - appelle les fonctions de *callback* des composants graphiques
- Lancée automatiquement à la fin de la fonction *main*

Première version

```
public class UneInterfaceReflexe1 extends UneInterface
     implements ActionListener {
    private static final long serialVersionUID = 1L;

    public UneInterfaceReflexe1() {
            trace.addActionListener(this);
            efface.addActionListener(this);}
```

Première version

```
public class UneInterfaceReflexe1 extends UneInterface
     implements ActionListener {
    private static final long serialVersionUID = 1L;

    public UneInterfaceReflexe1() {
            trace.addActionListener(this);
            efface.addActionListener(this);}

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

Différentes façons de résoudre le même problème

Deux types de boutons

Différentes façons de résoudre le même problème

Deux types de boutons

- Chacun spécifie son action

Différentes façons de résoudre le même problème

Deux types de boutons

- Chacun spécifie son action
- Doit posséder un lien (association) vers l'objet à modifier

Différentes façons de résoudre le même problème

Deux types de boutons

- Chacun spécifie son action
- Doit posséder un lien (association) vers l'objet à modifier

Un seul type de bouton

Différentes façons de résoudre le même problème

Deux types de boutons

- Chacun spécifie son action
- Doit posséder un lien (association) vers l'objet à modifier

Un seul type de bouton

- Différentier le comportement des deux boutons

Différentes façons de résoudre le même problème

Deux types de boutons

- Chacun spécifie son action
- Doit posséder un lien (association) vers l'objet à modifier

Un seul type de bouton

- Différentier le comportement des deux boutons

Utilisation de délégués

Différentes façons de résoudre le même problème

Deux types de boutons

- Chacun spécifie son action
- Doit posséder un lien (association) vers l'objet à modifier

Un seul type de bouton

- Différentier le comportement des deux boutons

Utilisation de délégués

- Écrire l'action au plus près de l'objet

Différentes façons de résoudre le même problème

Deux types de boutons

- Chacun spécifie son action
- Doit posséder un lien (association) vers l'objet à modifier

Un seul type de bouton

- Différentier le comportement des deux boutons

Utilisation de délégués

- Écrire l'action au plus près de l'objet
- Définition d'une classe interne

Deux types de boutons

```
public class BoutonTrace extends JButton implements
      ActionListener {
     private final Ardoise monArdoise;
```

Deux types de boutons

```
public class BoutonTrace extends JButton implements
      ActionListener {
     private final Ardoise monArdoise;

     public BoutonTrace(String nom, Ardoise ardoise) {
          super(nom);
          monArdoise = ardoise; addActionListener(this);}
```

Deux types de boutons

```
public class BoutonTrace extends JButton implements
      ActionListener {
     private final Ardoise monArdoise;

     public BoutonTrace(String nom, Ardoise ardoise) {
          super(nom);
          monArdoise = ardoise; addActionListener(this);}

     public void actionPerformed(ActionEvent e) {
          monArdoise.setPossedeDisque(true); monArdoise.repaint()
               ;}}
```

Deux types de boutons

```
public class BoutonTrace extends JButton implements
      ActionListener {
    private final Ardoise monArdoise;

    public BoutonTrace(String nom, Ardoise ardoise) {
        super(nom);
        monArdoise = ardoise; addActionListener(this);}

    public void actionPerformed(ActionEvent e) {
        monArdoise.setPossedeDisque(true); monArdoise.repaint()
            ;}}
```

```
public class UneInterfaceReflexe2 extends JPanel {
    protected Ardoise ardoise = new Ardoise();
    protected BoutonTrace trace = new BoutonTrace("trace",
        ardoise);
    protected BoutonEfface efface = new BoutonEfface("Efface",
```

Utilisation de délégués

```
public class Delegue1 extends JFrame {
    private final Ardoise ardoise = new Ardoise();
    private JButton trace = new JButton("Trace");
    private JButton efface = new JButton("Efface");
```

Utilisation de délégués

```
public class Delegue1 extends JFrame {
    private final Ardoise ardoise = new Ardoise();
    private JButton trace = new JButton("Trace");
    private JButton efface = new JButton("Efface");

// Version 1
    private class DelegueTrace implements ActionListener{
        public void actionPerformed(ActionEvent e) {
            ardoise.setPossedeDisque(true);
            ardoise.repaint();}}
```

Utilisation de délégués

```
public class Delegue1 extends JFrame {
    private final Ardoise ardoise = new Ardoise();
    private JButton trace = new JButton("Trace");
    private JButton efface = new JButton("Efface");

// Version 1
    private class DelegueTrace implements ActionListener{
        public void actionPerformed(ActionEvent e) {
            ardoise.setPossedeDisque(true);
            ardoise.repaint();}}

    public Delegue1(){
// Version 2
        class DelegueEfface implements ActionListener{
            public void actionPerformed(ActionEvent e) {
                ardoise.setPossedeDisque(false);
                ardoise.repaint();}}
```

Utilisation de délégués (suite)

```
    // suite constructeur Delegue1
JPanel pan_boutons = new JPanel();
        pan_boutons.add(trace);
        pan_boutons.add(efface);
```

Utilisation de délégués (suite)

```
    // suite constructeur Delegue1
JPanel pan_boutons = new JPanel();
        pan_boutons.add(trace);
        pan_boutons.add(efface);

        trace.addActionListener(new DelegueTrace());
        efface.addActionListener(new DelegueEfface());
// Fin du constructeur avec la mise en place du layout et packing
[...]
```

Utilisation de délégués (suite)

```
    // suite constructeur Delegue1
JPanel pan_boutons = new JPanel();
        pan_boutons.add(trace);
        pan_boutons.add(efface);

        trace.addActionListener(new DelegueTrace());
        efface.addActionListener(new DelegueEfface());
// Fin du constructeur avec la mise en place du layout et packing
[...]

    public static void main(String[] args) {
        new Delegue1(); }
```

Autre forme anonyme pour les délégués

```
efface.addActionListener(
      new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            ardoise.setPossedeDisque(false);
            ardoise.repaint();}
      }
                                    );
```

Autre forme anonyme pour les délégués

```
efface.addActionListener(
      new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            ardoise.setPossedeDisque(false);
            ardoise.repaint();}
      }
                                    );
```

Anonymisation

- Si on associe une action à un seul élément graphique
- Attention à la lisibilité du code !

Utilisation des délégués – une petite dernière

```
public class Delegue2 implements ActionListener {
    private final Ardoise ardoise;
    private boolean existe;
```

Utilisation des délégués – une petite dernière

```
public class Delegue2 implements ActionListener {
   private final Ardoise ardoise;
   private boolean existe;

   public Delegue2(Ardoise ard, boolean ex) {
      ardoise = ard;
      existe = ex;
   }
```

Utilisation des délégués – une petite dernière

```
public class Delegue2 implements ActionListener {
   private final Ardoise ardoise;
   private boolean existe;

   public Delegue2(Ardoise ard, boolean ex) {
      ardoise = ard;
      existe = ex;
   }

   @Override
   public void actionPerformed(ActionEvent e) {
      ardoise.setPossedeDisque(existe);
      ardoise.repaint();
   }
}
```

Gestion par le modèle MVC

Modèle MVC

Gestion par le modèle MVC

Modèle MVC

- Patron de conception (Design Pattern)

Gestion par le modèle MVC

Modèle MVC

- Patron de conception (Design Pattern)
- Permet de séparer les applications en 3 parties :

Gestion par le modèle MVC

Modèle MVC

- Patron de conception (Design Pattern)
- Permet de séparer les applications en 3 parties :
  - Modèle : modèle de données

Gestion par le modèle MVC

Modèle MVC

- Patron de conception (Design Pattern)
- Permet de séparer les applications en 3 parties :
  - Modèle : modèle de données
  - Vue : Présentation à l'utilisateur

Gestion par le modèle MVC

Modèle MVC

- Patron de conception (Design Pattern)
- Permet de séparer les applications en 3 parties :

  Modèle : modèle de données
  Vue : Présentation à l'utilisateur
  Contrôleur : Gestion des événements

Gestion par le modèle MVC

Modèle MVC

- Patron de conception (Design Pattern)
- Permet de séparer les applications en 3 parties :
  - Modèle : modèle de données
  - Vue : Présentation à l'utilisateur
  - Contrôleur : Gestion des événements
- Mis en œuvre de manière native par Swing

Mise en place dans notre application

Relations

Mise en place dans notre application

Relations

- La **Vue** connaît et observe le **Modèle** ;

Mise en place dans notre application

Relations

- La **Vue** connaît et observe le **Modèle** ;
- Le **Contrôleur** connaît la **Vue** et le **Modèle** ;

Mise en place dans notre application

Relations

- La **Vue** connaît et observe le **Modèle** ;
- Le **Contrôleur** connaît la **Vue** et le **Modèle** ;
- Le **Contrôleur** est le *listener* de la **Vue**.

Application à notre problème

Notre modèle

Application à notre problème

Notre modèle

- Réduit à l'information :
  Est-ce que le rond rouge est dessiné ?

Application à notre problème

Notre modèle

- Réduit à l'information :
  Est-ce que le rond rouge est dessiné ?
- Propriété : ***boolean*** *existe*

Application à notre problème

Notre modèle

- Réduit à l'information :
  Est-ce que le rond rouge est dessiné ?
- Propriété : ***boolean*** *existe*
- Hérite de la classe *Observable*

Application à notre problème

Notre modèle

- Réduit à l'information :
  Est-ce que le rond rouge est dessiné ?
- Propriété : ***boolean*** *existe*
- Hérite de la classe *Observable*

Notre Vue

Application à notre problème

Notre modèle

- Réduit à l'information :
  Est-ce que le rond rouge est dessiné ?
- Propriété : ***boolean*** *existe*
- Hérite de la classe *Observable*

Notre Vue

- L'interface graphique

Application à notre problème

Notre modèle

- Réduit à l'information :
  Est-ce que le rond rouge est dessiné ?
- Propriété : ***boolean*** *existe*
- Hérite de la classe *Observable*

Notre Vue

- L'interface graphique
- Implémente l'interface (java) *Observer*

Application à notre problème

Notre modèle

- Réduit à l'information :
  Est-ce que le rond rouge est dessiné ?
- Propriété : ***boolean*** *existe*
- Hérite de la classe *Observable*

Notre Vue

- L'interface graphique
- Implémente l'interface (java) *Observer*

Notre contrôleur

Application à notre problème

Notre modèle

- Réduit à l'information :
  Est-ce que le rond rouge est dessiné ?
- Propriété : ***boolean*** *existe*
- Hérite de la classe *Observable*

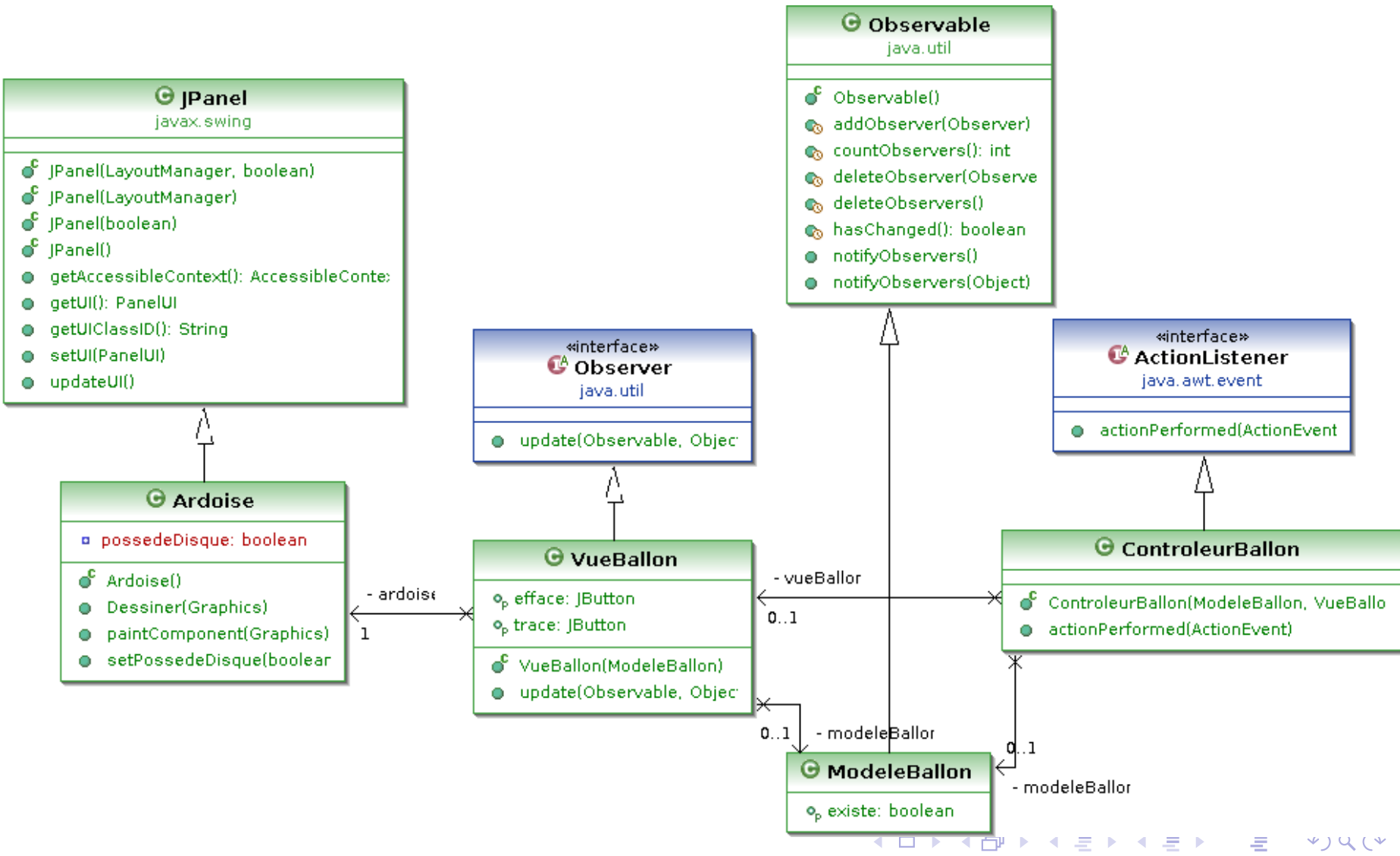
Notre Vue

- L'interface graphique
- Implémente l'interface (java) *Observer*

Notre contrôleur

- Mise en place du *Listener*

Relations entre les classes



ModeleBallon

```
public class ModeleBallon extends Observable {
   private boolean existe;
   public boolean isExiste() {
      return existe; }
   /**
   * Setter of the property <tt>existe</tt>
   * @param existe The existe to set.
   */
   public void setExiste(boolean existe) {
      this.existe = existe;
```

ModeleBallon

```
public class ModeleBallon extends Observable {
   private boolean existe;
   public boolean isExiste() {
      return existe; }
   /**
   * Setter of the property <tt>existe</tt>
   * @param existe The existe to set.
   */
   public void setExiste(boolean existe) {
      this.existe = existe;

      setChanged();
      notifyObservers();
   }
}
```

ControleurBallon

```
public class ControleurBallon implements ActionListener {
  private VueBallon vueBallon;
  private ModeleBallon modeleBallon;
```

ControleurBallon

```
public class ControleurBallon implements ActionListener {
  private VueBallon vueBallon;
  private ModeleBallon modeleBallon;

  public ControleurBallon(ModeleBallon leModele, VueBallon laVue)
  { modeleBallon = leModele; vueBallon = laVue; }
```

ControleurBallon

```
public class ControleurBallon implements ActionListener {
   private VueBallon vueBallon;
   private ModeleBallon modeleBallon;

   public ControleurBallon(ModeleBallon leModele, VueBallon laVue)
   { modeleBallon = leModele; vueBallon = laVue; }

   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == vueBallon.getTrace())
          modeleBallon.setExiste(true);
      else if (e.getSource() == vueBallon.getEfface())
          modeleBallon.setExiste(false); }
}
```

VueBallon

```
public class VueBallon extends JFrame implements Observer {
```

VueBallon

```
public class VueBallon extends JFrame implements Observer {

  public void update(Observable arg0, Object arg1) {
    ardoise.setPossedeDisque(modeleBallon.isExiste());
    ardoise.repaint();}
```

VueBallon

```
public class VueBallon extends JFrame implements Observer {

  public void update(Observable arg0, Object arg1) {
    ardoise.setPossedeDisque(modeleBallon.isExiste());
    ardoise.repaint();}

  public VueBallon(ModeleBallon unModele){
    modeleBallon = unModele;
    modeleBallon.addObserver(this);
    // Description de l'interface }
```

VueBallon

```
public class VueBallon extends JFrame implements Observer {

  public void update(Observable arg0, Object arg1) {
    ardoise.setPossedeDisque(modeleBallon.isExiste());
    ardoise.repaint();}

  public VueBallon(ModeleBallon unModele){
    modeleBallon = unModele;
    modeleBallon.addObserver(this);
    // Description de l'interface }

private JButton efface; public JButton getEfface() {return efface; }
private JButton trace; public JButton getTrace() { return trace; }
private Ardoise ardoise = new Ardoise(); public Ardoise
     getArdoise()...
private ModeleBallon modeleBallon; }
```

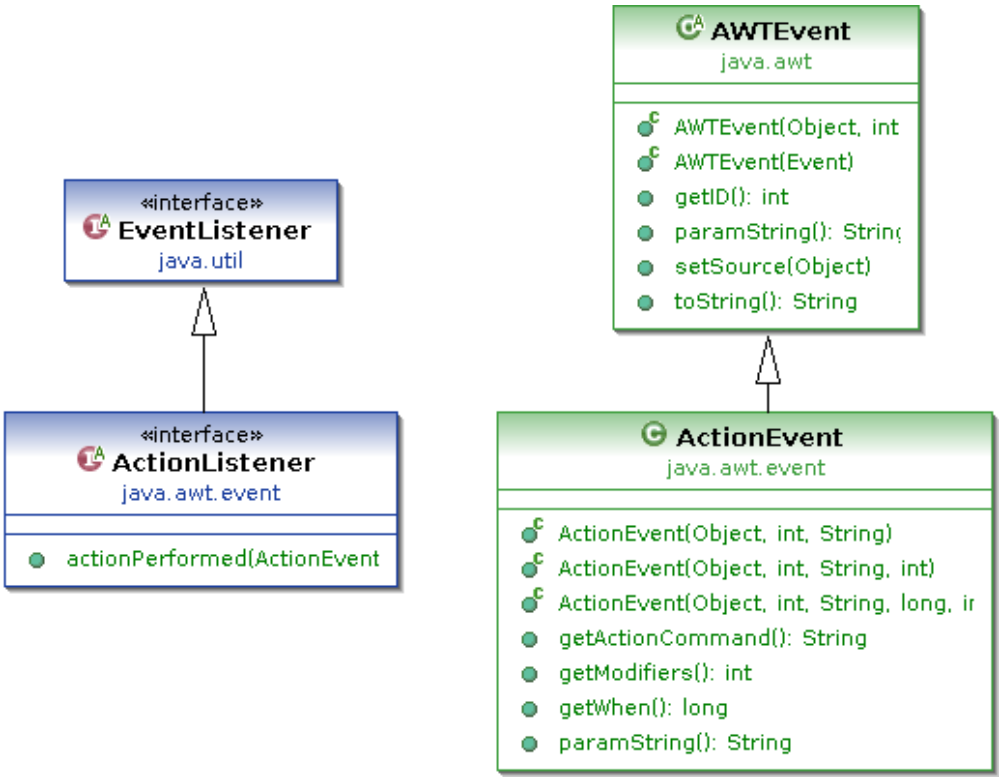
Main

```
public class Main {
  public static void main(String[] args) {
    ModeleBallon mb= new ModeleBallon();
    VueBallon vb = new VueBallon(mb);
    ControleurBallon cb = new ControleurBallon(mb, vb );
```

Main

```
public class Main {
   public static void main(String[] args) {
      ModeleBallon mb= new ModeleBallon();
      VueBallon vb = new VueBallon(mb);
      ControleurBallon cb = new ControleurBallon(mb, vb );

      vb.getTrace().addActionListener(cb);
      vb.getEfface().addActionListener(cb);
   }
}
```

ActionListener



Autres Listener

`docs.oracle.com/javase/tutorial/uiswing/events`

Quelques interfaces au nom évocateur

- FocusListener
- WindowListener
- ItemListener
- MouseListener
- MouseMotionListener
- MouseWheelListener
- ComponentListener
- KeyListener
- DocumentListener
- UndoableEditListener
- CaretListener
- ...

Choix

Cases à cocher

- *JCheckBox jcb = **new** JCheckBox("Fromage", **false**);*

Choix

Cases à cocher

- *JCheckBox jcb = **new** JCheckBox("Fromage", **false**);*

Boutons radio

- *JRadioButton fr = **new** JRadioButton("Fromage", **false**);*
- Un seul élément sélectionné à la fois
- Regroupés dans un conteneur spécifique
  *ButtonGroup groupe = **new** ButtonGroup();*
- *groupe.add(fr); groupe.add(dessert);*

Choix

Cases à cocher

- *JCheckBox jcb = **new** JCheckBox("Fromage", **false**);*

Boutons radio

- *JRadioButton fr = **new** JRadioButton("Fromage", **false**);*
- Un seul élément sélectionné à la fois
- Regroupés dans un conteneur spécifique
  *ButtonGroup groupe = **new** ButtonGroup();*
- *groupe.add(fr); groupe.add(dessert);*

Événement

- *ItemListener*
- Méthode ***void** itemStateChanged(ItemEvent e)*

Liste de Choix

```
public class FenetreListeNom extends JFrame implements
    ListSelectionListener {
  JList liste;
  JLabel etiquette = new JLabel(" ");
```

Liste de Choix

```
public class FenetreListeNom extends JFrame implements
      ListSelectionListener {
   JList liste;
   JLabel etiquette = new JLabel(" ");

   public void valueChanged(ListSelectionEvent e) {
     etiquette.setText((String)liste.getSelectedValue());}
```

Liste de Choix

```
public class FenetreListeNom extends JFrame implements
     ListSelectionListener {
  JList liste;
  JLabel etiquette = new JLabel(" ");

  public void valueChanged(ListSelectionEvent e) {
    etiquette.setText((String)liste.getSelectedValue());}

  public FenetreListeNom(){
    String choix[] = {"Pierre", "Paul", "Jacques", "Lou", "Marie"};
    liste = new JList(choix);
    liste.addListSelectionListener(this);
    // Dessin de l'interface + packing ..;
    ... }

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

Adapter

Objet implémentant des interfaces

Adapter

Objet implémentant des interfaces

- Objets implémentant les comportements par défaut

Adapter

Objet implémentant des interfaces

- Objets implémentant les comportements par défaut
- Souvent vides

Adapter

Objet implémentant des interfaces

- Objets implémentant les comportements par défaut
- Souvent vides
- Doivent être dérivées pour être utiles

Adapter

Objet implémentant des interfaces

- Objets implémentant les comportements par défaut
- Souvent vides
- Doivent être dérivées pour être utiles
- Permet de ne redéfinir que les méthodes intéressantes

Adapter

Objet implémentant des interfaces

- Objets implémentant les comportements par défaut
- Souvent vides
- Doivent être dérivées pour être utiles
- Permet de ne redéfinir que les méthodes intéressantes

Exemples

Adapter

Objet implémentant des interfaces

- Objets implémentant les comportements par défaut
- Souvent vides
- Doivent être dérivées pour être utiles
- Permet de ne redéfinir que les méthodes intéressantes

Exemples

- MouseAdapter, ComponentAdapter, ...

Adapter

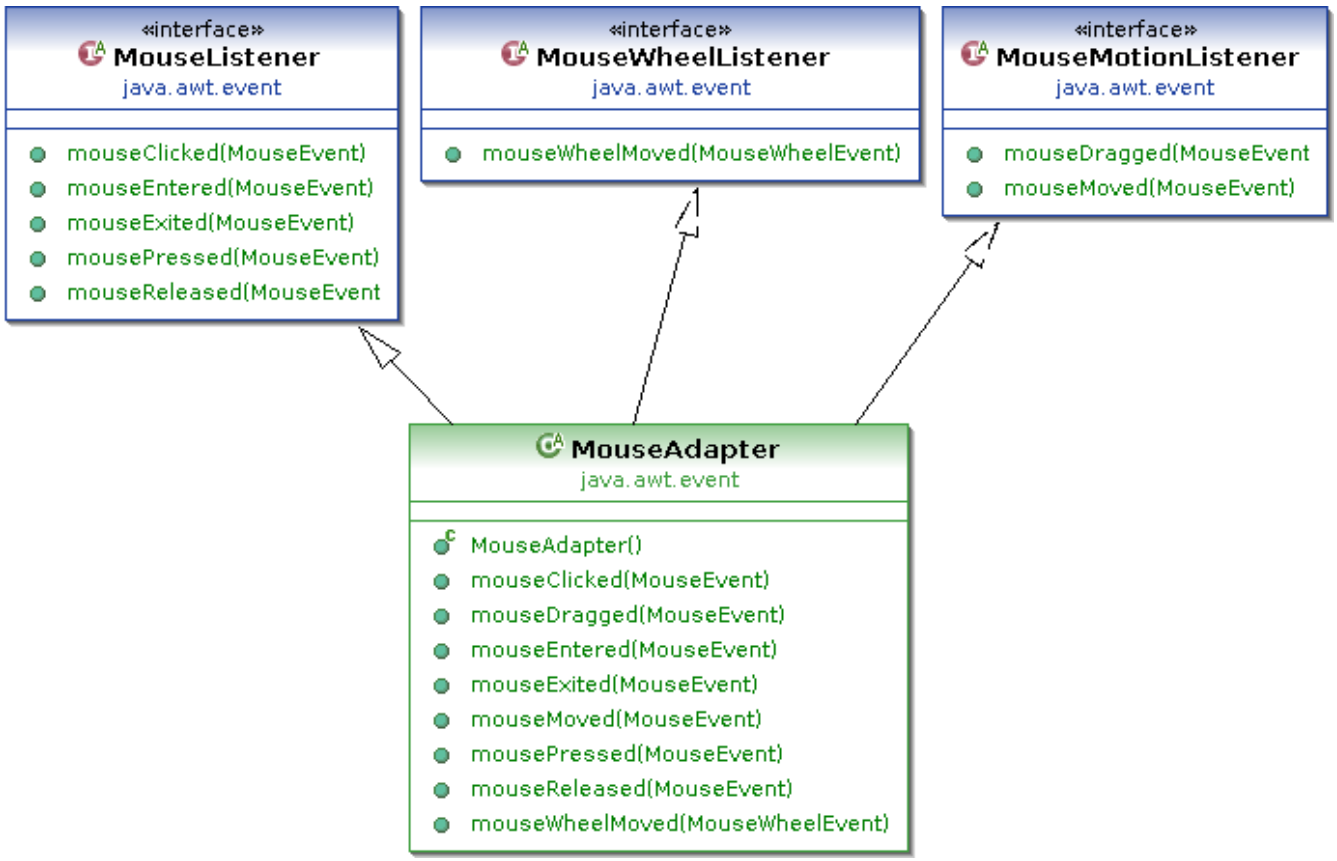
Objet implémentant des interfaces

- Objets implémentant les comportements par défaut
- Souvent vides
- Doivent être dérivées pour être utiles
- Permet de ne redéfinir que les méthodes intéressantes

Exemples

- MouseAdapter, ComponentAdapter, ...
- Gestion XML : ParserAdapter

MouseAdpater



avec MouseListener

```
public class Entendu extends JPanel implements MouseListener{
    public Entendu(){
      setPreferredSize(new Dimension(200, 200));
      addMouseListener(this);
      setBackground(Color.RED);}
```

avec MouseListener

```
public class Entendu extends JPanel implements MouseListener{
    public Entendu(){
      setPreferredSize(new Dimension(200, 200));
      addMouseListener(this);
      setBackground(Color.RED);}

    public void mouseEntered(MouseEvent arg0) {
       setBackground(Color.GREEN);}
```

avec MouseListener

```
public class Entendu extends JPanel implements MouseListener{
    public Entendu(){
      setPreferredSize(new Dimension(200, 200));
      addMouseListener(this);
      setBackground(Color.RED);}

    public void mouseEntered(MouseEvent arg0) {
       setBackground(Color.GREEN);}

    public void mouseExited(MouseEvent arg0) {
      setBackground(Color.RED); }
```

avec MouseListener

```
public class Entendu extends JPanel implements MouseListener{
    public Entendu(){
      setPreferredSize(new Dimension(200, 200));
      addMouseListener(this);
      setBackground(Color.RED);}

    public void mouseEntered(MouseEvent arg0) {
       setBackground(Color.GREEN);}

    public void mouseExited(MouseEvent arg0) {
      setBackground(Color.RED); }

    public void mousePressed(MouseEvent arg0) {}
    public void mouseClicked(MouseEvent arg0) {}
    public void mouseReleased(MouseEvent arg0) {} }
```

avec MouseAdapter

```
public class MouseChangeCouleur extends MouseAdapter {
   private JPanel jp;
   public MouseChangeCouleur(JPanel panneau) {
       jp = panneau;}
```

avec MouseAdapter

```
public class MouseChangeCouleur extends MouseAdapter {
  private JPanel jp;
  public MouseChangeCouleur(JPanel panneau) {
      jp = panneau;}

  public void mouseEntered(MouseEvent e){
    jp.setBackground(Color.YELLOW);}
```

avec MouseAdapter

```
public class MouseChangeCouleur extends MouseAdapter {
  private JPanel jp;
  public MouseChangeCouleur(JPanel panneau) {
      jp = panneau;}

  public void mouseEntered(MouseEvent e){
    jp.setBackground(Color.YELLOW);}
  public void mouseExited(MouseEvent e){
    jp.setBackground(Color.RED);}}
```

```
public class EntenduBis extends JPanel {
  public EntenduBis(){
    setPreferredSize(new Dimension(200, 200));
    addMouseListener(new MouseChangeCouleur(this));
    setBackground(Color.RED);}}
```

Le tout ensemble

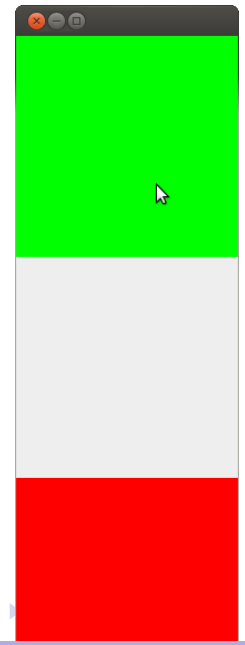
```
public class UsingEntendu extends JFrame
      {
  private Entendu ent;
  private EntenduBis ent2;
  private JPanel panneau;
  public UsingEntendu() {
    ent = new Entendu();
    ent2 = new EntenduBis();
```

Le tout ensemble

```
public class UsingEntendu extends JFrame
      {
  private Entendu ent;
  private EntenduBis ent2;
  private JPanel panneau;
  public UsingEntendu() {
    ent = new Entendu();
    ent2 = new EntenduBis();

    panneau = new JPanel();
    panneau.setPreferredSize(
        new Dimension(200, 600));
```

Le tout ensemble

```
public class UsingEntendu extends JFrame
      {
  private Entendu ent;
  private EntenduBis ent2;
  private JPanel panneau;
  public UsingEntendu() {
    ent = new Entendu();
    ent2 = new EntenduBis();

    panneau = new JPanel();
    panneau.setPreferredSize(
        new Dimension(200, 600));

    // Composition + packing
    ... }
  public static void main(String[] args)
  { new UsingEntendu();}}
```



Quelques ressources utiles

Cours

- Cours Irène Charon
- Cours Eric Lecolinet

Tutoriels

- Documentation Java : `http://docs.oracle.com/javase/tutorial/uiswing/TOC.html`
- Beaucoup d'exemples : `http://www.java2s.com/Code/Java/Swing-JFC/CatalogSwing-JFC.htm`