

R2.02 - Développement d'applications avec IHM

Abdelbadie Belmouhcine, Mohammed Yasser Khayata

Institut Universitaire de Technologie de Vannes - Université Bretagne Sud
abdelbadie.belmouhcine@univ-ubs.fr

27 mai 2024



- 1 Introduction
- 2 Interfaces graphiques
- 3 Programmation événementielle
- 4 Les bases de JavaFX
- 5 L'architecture MVC (Model-View-Controller)
- 6 Lecture et écriture dans des fichiers
- 7 Accès aux bases de données
- 8 Aller plus loin en JavaFX

Introduction

Rappels du PN

❖ Objectifs :

- ☞ Initier au développement d'une application avec une IHM
- ☞ Capacité à produire des applications avec des interfaces utilisateurs

❖ Compétences :

- ☞ C1 AC 2 : Élaborer des conceptions simples (P3, P4, SAÉ)
- ☞ C1 AC 4 : Développer des interfaces utilisateurs (P4, SAÉ)
- ☞ C5 AC 1 : Appréhender les besoins du client et de l'utilisateur (P3, SAÉ)
- ☞ C6 AC 1 : Appréhender l'écosystème numérique (P3, SAÉ)

❖ Savoirs de référence étudiés :

- ☞ Programmation événementielle
- ☞ Programmation d'interfaces utilisateurs, utilisation de composants graphiques
- ☞ Compréhension et mise en place de la séparation entre la vue et le modèle
- ☞ Liaison de données entre propriétés (databinding, master/detail)
- ☞ Sensibilisation à l'ergonomie
- ☞ Assurance de la persistance des données

R2.02 en 2023-2024

Développement d'applications avec IHM :

- ❖ **Ergonomie** : 1 séance par semaine (x 5 semaines) en P3
 - ❖ Enseignant : Thierry Morineau (pour tous les groupes)
- ❖ **Programmation** : 2 TD/TP par semaine (x 6 semaines) en P4
 - ❖ Enseignants :
 - ❖ Groupe A : Abdelbadie Belmouhcine
 - ❖ Groupes B, C, D : Mohammed Yasser Khayata
- ❖ **Inscription sur Moodle OBLIGATOIRE**
 - ❖ Supports
 - ❖ Contrôles
 - ❖ Rendus

Organisation en P4

❖ TP programmation :

❖ Durée : 2 x 1h30

❖ **Rendu du TP à la fin des 3 heures (sujet calibré + bonus)**

❖ SAÉ S2 - **Attractivité des communes bretonnes** :

❖ Les 2 dernières séances seront consacrées à la SAÉ S2 - **Attractivité des communes bretonnes**

❖ Pour toute **question**, veuillez la poser en TD/TP ou par mail/Teams/Moodle

Évaluation

❖ CC + CT

❖ CC

- = programmation + ergonomie
- = qualité + régularité des rendus
- = Moodle!

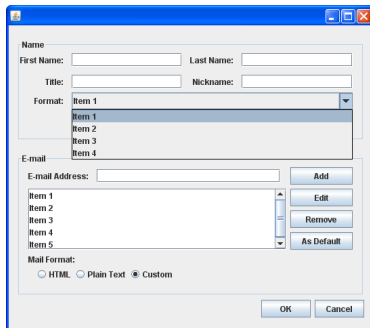
❖ Importance de l'IHM :

- 👉 Un module important pour la SAÉ du semestre 2
- 👉 Important pour les futurs projets, les stages, etc

Interfaces graphiques

Interface graphique

- ❖ Système de fenêtrage
- ❖ Interaction plus riche avec l'utilisateur
- ❖ GUI = Graphical User Interface



- ES Taille et position des fenêtres
- ES Afficher, saisir des données textuelles
- ES Présenter les informations dans des listes, des menus
- ES Afficher des images, les modifier
- ES Interaction avec la souris, le clavier, une surface tactile
- ES Présentation riche, agréable et ergonomique de l'information (texte, image, son, vidéo)

Un premier exemple (TP1)



```
import javax.swing.*;

public class HelloWorldSwing extends JFrame {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                HelloWorldSwing frame = new HelloWorldSwing();
                frame.pack();
                frame.setVisible(true);
            }
        });
    }

    public HelloWorldSwing() {
        setTitle("HelloWorldSwing");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Hello World");
        add(label);
    }
}
```

Bonnes et moins bonnes pratiques

❖ Exemple précédent :

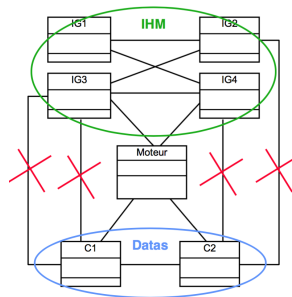
👉 Avantages ?

👉 Inconvénients ?

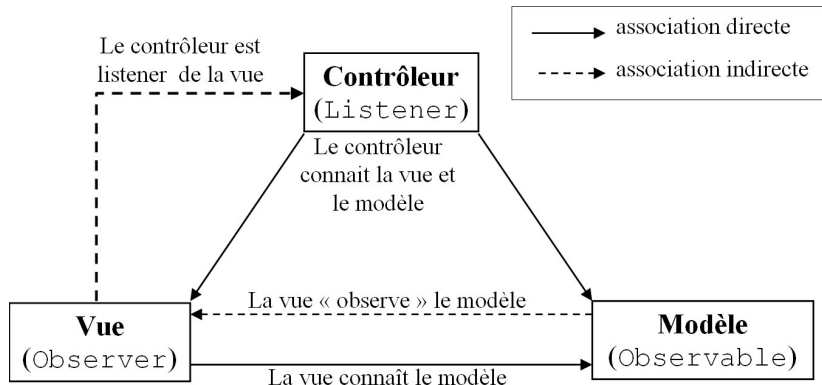
Patron de conception MVC = Model View Controller

- ❖ Les applications peuvent être complexes, ainsi que l'interface correspondante. Il peut alors être souhaitable de séparer :

- ☞ La **classe de l'interface graphique** : elle contient la description de l'interface graphique avec ses différents composants ; c'est **la vue**
- ☞ La **classe de l'application elle-même** : elle contient tous les attributs et méthodes qui doivent participer à la tâche principale à accomplir : c'est **le modèle**
- ☞ Une **classe de contrôle** qui contient l'ensemble des **listeners** et qui, lorsque des événements parviennent via **la vue**, prévient **le modèle** en conséquence : c'est le **contrôleur**



MVC



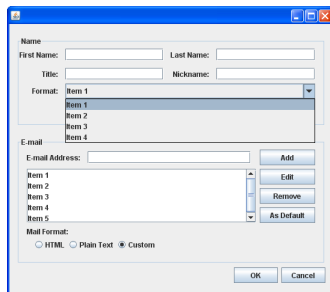
API Java pour les IHM

- ❖ Java et les IHM = une longue histoire
- ❖ Au début, Java 1.0 (1995) : AWT (Abstract Windows Toolkit, `java.awt.*`)
 - ☞ Composants dits lourds (= s'appuient sur du code natif, dessinés par l'OS)
- ❖ Ensuite, Java 2 = 1.2 (1997) : Swing, `javax.swing`
 - ☞ Composants légers (= 100% Java, dessinés par Java)
 - ☞ **Ne pas mixer les 2 ! Préférer Swing !**
- ❖ Un peu plus récemment, JavaFX (2008), inclus dans le JDK... jusqu'en 2018 !
 - ☞ **Il reste une bonne alternative qui sera étudiée plus tard en P4**

Les composants graphiques

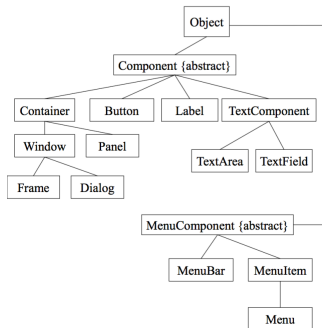
- ❖ Les composants graphiques, aussi appelés widgets (Window Gadgets), sont des éléments essentiels des interfaces utilisateur. Ils incluent :

- 👉 Boutons
- 👉 Cases à cocher
- 👉 Liste de sélection
- 👉 Menu déroulant
- 👉 Barre de défilement
- 👉 Zone de saisie de texte
- 👉 etc.



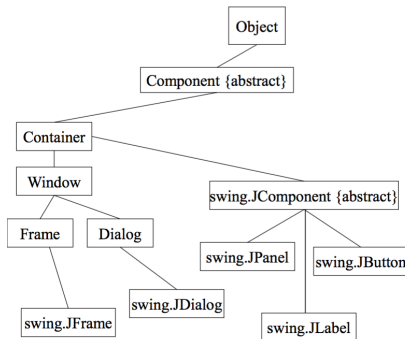
AWT

- ☞ **Conteneurs** : ils contiennent d'autres éléments graphiques, y compris des conteneurs
- ☞ **Composants élémentaires**
- ☞ **Composants de haut-niveau** (Frame, Dialog, Applet)



Swing

- 👉 Les classes Swing héritent des classes AWT
- 👉 Pour les reconnaître, elles suivent généralement cette convention de nommage : **JNomDeLaClasse**



Swing

❖ Avantages de Swing sur AWT :

- 👉 Bibliothèque plus fournie
- 👉 Widgets plus design
- 👉 Meilleure portabilité graphique

❖ Liens avec AWT :

- 👉 Swing est une spécialisation d'AWT (les classes Swing héritent des classes AWT)
- 👉 Le mécanisme de réaction aux événements reste le même

Composants Swing

❖ JLabel

👉 `getText()`

👉 `setText()`

👉 Alignement

❖ JButton

❖ JComboBox

JFrame



Utilisation du JFrame

```
import javax.swing.*;
public class Cadre1 {
    // Attribut
    private JFrame demo;
    // Constructeur
    public Cadre1(String titre) {
        demo = new JFrame(titre);
        demo.setSize(100, 100);
        demo.setVisible(true);
    }
}
```



Héritage du JFrame

```
import javax.swing.*;
public class Cadre2 extends JFrame
{
    // Constructeur
    public Cadre2(String titre) {
        super(titre);
        this.setSize(100, 100);
        this.setVisible(true);
    }
}
```

Ajouts de composants

- ❖ Pour ajouter un widget à une fenêtre, on utilise la méthode `add(Component comp)` de la classe `Container`



```
import javax.swing.*;
public class Cadre extends JFrame {
    // Constructeur
    public Cadre(String titre) {
        super(titre);
        // Ajout de composants graphiques
        // Un simple JLabel
        JLabel label1 = new JLabel("Un JLabel");
        // Un simple JButton
        JButton myBut = new JButton("bouton");
        // Ajout DANS la fenêtre
        this.add(label1);
        this.add(myBut);
        this.setSize(100, 100);
        this.setVisible(true);
    }
}
```

Ajouts de composants

❖ Où placer les composants ?

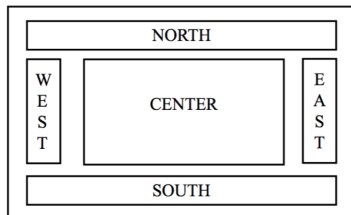
- 👉 En précisant les coordonnées et la taille de chaque objet graphique à l'intérieur du Container :
- 👉 Utilisation de `setBounds(int x, int y, int width, int height)` de la classe `Component`

➡ Inconvénients ?

Gestionnaire de placement

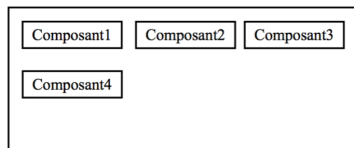
- ❖ 30 gestionnaires qui implémentent `java.awt.LayoutManager`
 - 👉 Permet le placement automatique dans des zones prédéfinies et s'adapte automatiquement en cas de redimensionnement
- ❖ Utilisation :
 - 👉 Associer un gestionnaire de placement à l'objet graphique de type **Container**
 - `maFrame.setLayout(new TrucLayout());`
 - 👉 Ajouter les composants graphiques dans le **Container** avec ou sans utilisation d'une contrainte de placement
 - `add(Component comp, Object constraints)`
 - `maFrame.add(myButton, "enHautAGauche");`
 - `maFrame.add(myButton);`

BorderLayout



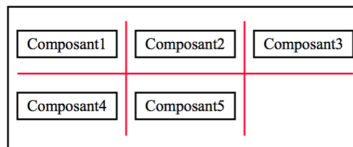
```
// Définition du gestionnaire de placement BorderLayout
// pour l'objet « monCont » de type Container.
monCont.setLayout ( new BorderLayout() );
monCont.add ( unComposant, BorderLayout.NORTH );
monCont.add ( unComposant, BorderLayout.SOUTH );
monCont.add ( unComposant, BorderLayout.EAST );
monCont.add ( unComposant, BorderLayout.WEST );
monCont.add ( unComposant, BorderLayout.CENTER );
```


FlowLayout



```
// Définition du gestionnaire de placement FlowLayout
// pour l'objet « monCont » de type Container.
// Disposition des composants de gauche à droite.
monCont.setLayout ( new FlowLayout() );
monCont.add ( composant1 );
monCont.add ( composant2 );
monCont.add ( composant3 );
monCont.add ( composant4 );
```

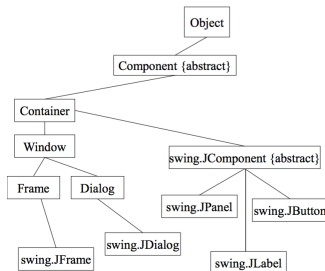
GridLayout



```
// Définition du gestionnaire de placement GridLayout
// pour l'objet « monCont » de type Container.
// Disposition des composants de gauche à droite et de haut en bas.
monCont.setLayout ( new GridLayout (2, 3) );
monCont.add ( composant1 );
monCont.add ( composant2 );
monCont.add ( composant3 );
monCont.add ( composant4 );
monCont.add ( composant5 );
```

Les classes Container

- ❖ Les seuls objets qui peuvent contenir d'autres objets graphiques
- ❖ Construction de l'interface par emboîtement de poupées russes



Exemple MVC I



Modele

```
import java.util.Observable;
@SuppressWarnings("deprecation")
public class Modele extends Observable {
    private double tempCelsius;
    public double getTempCelsius() {
        return tempCelsius;
    }
    public double getTempFahrenheit() {
        return tempCelsius * 9 / 5 + 32;
    }
    public void setTempFahrenheit(double tempF) {
        this.tempCelsius = (tempF - 32) * 5 / 9;
        setChanged();
        notifyObservers();
    }
    public void setTempCelsius(double tempC) {
        this.tempCelsius = tempC;
        setChanged();
        notifyObservers();
    }
}
```

Exemple MVC II



Vue 1

```
import java.awt.GridLayout;
import java.util.*;
import javax.swing.*;
@SuppressWarnings("deprecation")
public class Vue1 extends JFrame implements Observer {
    private JLabel label = new JLabel("Temperature en C:");
    private JTextField text = new JTextField();
    public JTextField getText() { return text; }
    private Modele modele;
    public Vue1(Modele m) {
        super("Temperature celsius");
        this.modele = m;
        this.modele.addObserver(this);
        this.text.setText(m.getTempCelsius() + "");
        this.getContentPane().setLayout(new GridLayout(0, 2));
        this.add(label);
        this.add(text);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.pack();
        this.setVisible(true);
    }
    public void update(Observable o, Object arg) {
        this.text.setText(this.modele.getTempCelsius() + "");
    }
}
```

Exemple MVC III



Vue 2

```
import java.awt.GridLayout;
import java.util.*;
import javax.swing.*;
@SuppressWarnings("deprecation")
public class Vue2 extends JFrame implements Observer {
    private JLabel label = new JLabel("Temperature en F:");
    private JTextField text = new JTextField();
    public JTextField getText() { return text; }
    private Modele modele;
    public Vue2(Modele m) {
        super("Temperature Farenheit");
        this.modele = m;
        this.modele.addObserver(this);
        this.text.setText(m.getTempFarenheit() + "");
        this.getContentPane().setLayout(new GridLayout(0, 2));
        this.add(label);
        this.add(text);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.pack();
        this.setVisible(true);
    }
    public void update(Observable o, Object arg) {
        this.text.setText(this.modele.getTempFarenheit() + "");
    }
}
```

Exemple MVC IV



Contrôleur

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class Controleur implements ActionListener {
    Modele modele;
    Vue1 vue1;
    Vue2 vue2;
    public Controleur(Modele modele, Vue1 vue1, Vue2 vue2) {
        this.modele = modele;
        this.vue1 = vue1;
        this.vue2 = vue2;
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == vue1.getText())
            modele.setTempCelsius(Double.parseDouble(vue1.getText().getText()));
        else if (e.getSource() == vue2.getText())
            modele.setTempFahrenheit(Double.parseDouble(vue2.getText().getText()));
    }
}
```

Exemple MVC V



Application

```
import javax.swing.SwingUtilities;
public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                Modele modele = new Modele();
                Vue1 frame1 = new Vue1(modele);
                Vue2 frame2 = new Vue2(modele);
                Controleur controleur = new Controleur(modele, frame1, frame2);
                frame1.getText().addActionListener(controleur);
                frame2.getText().addActionListener(controleur);
                frame1.setLocationRelativeTo(null); // Center the frame
                int xOffset = frame1.getX() + frame1.getWidth() / 2;
                int yOffset = frame1.getY() + frame1.getHeight() / 2;
                frame2.setLocation(xOffset, yOffset);
            }
        });
    }
}
```


Introduction

Interfaces graphiques

Programmation événementielle

Les bases de JavaFX

L'architecture MVC (Model-View-Controller)

Lecture et écriture dans des fichiers

Accès aux bases de données

Aller plus loin en JavaFX

Programmation événementielle

Les types d'événement

Les écouteurs d'événement

Les réactions

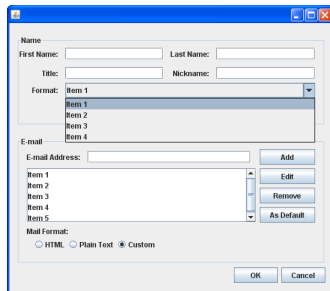
Les différentes formes de classes d'écouteurs

Encore un peu de Swing ?

Programmation événementielle

Un programme graphique

- ❗ Un programme = une suite d'instructions...
- ❗ Le comportement d'une IHM dépend de ce que veut l'utilisateur...
- ⚠ Mais on ne peut pas anticiper les actions de l'utilisateur!



?

Un programme graphique

?

- ❖ Une IHM s'appuie sur le paradigme de la programmation événementielle :
 - ☞ On anticipe toutes les actions possibles de l'utilisateur
 - ☞ On décrit (par du code) les réactions que le programme doit avoir à ces différentes actions

Programmation événementielle

- ❶ L'action de l'utilisateur sur l'interface graphique provoque un événement (**event**)
 - ❷ Si l'interface graphique (**listener**) est à l'écoute de cet événement
 - ❸ Alors le programme peut réagir par l'intermédiaire d'un module de réaction
- ➡ **3 éléments nécessaires :**
- 👉 Un événement **XXXEvent**
 - 👉 Un widget à l'écoute de **XXXEvent**
 - 👉 Une méthode de réaction à **XXXEvent**

Exemple



```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

public class Beeper extends JPanel implements ActionListener {
    JButton button;

    public Beeper() {
        super(new BorderLayout());
        button = new JButton("Click Me");
        button.setPreferredSize(new Dimension(200, 80));
        add(button, BorderLayout.CENTER);
        button.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
    }

    private static void createAndShowGUI() {
        // Create and set up the window.
        JFrame frame = new JFrame("Beeper");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JComponent newContentPane = new Beeper();
        newContentPane.setOpaque(true); // content panes must be opaque
        frame.setContentPane(newContentPane);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

java.awt.event

❖ Tous les événements qu'une IHM Java peut détecter se trouvent dans `java.awt.event`

- ☞ `KeyEvent` : événements du clavier
- ☞ `MouseEvent` : événements de la souris
- ☞ `WindowEvent` : événements de fenêtre
- ☞ `ActionEvent` : événements d'action
- ☞ etc.

Événements Java

- ❖ Tous les événements précédents sont **des classes Java** qui **héritent de `java.util.EventObject`**
- ❖ La classe **`EventObject`** fournit la méthode **`Object getSource()`** qui permet de savoir sur quel composant l'utilisateur a agi
- 🔊 Cela permet une seule méthode de réaction pour un même type d'événement en provenance de plusieurs widgets différents

Question 1

- ❖ Est-ce que tous les événements peuvent être capturés par tous les widgets ?

Question 1

- ❖ Est-ce que tous les événements peuvent être capturés par tous les widgets ?

Événements	Widgets
ActionEvent	JButton JTextField JMenu JMenuItem List
MouseEvent	(J)Component JList
KeyEvent	(J)Component
FocusEvent	(J)Component
InputMethodEvent	JTextComponent JTextField JTextArea
ItemEvent	JCheckBox Choice List
WindowEvent	JWindow

Question 2

- ❖ Que se passe-t'il lorsqu'un widget subit une action de l'utilisateur ?

Question 2

- ❖ Que se passe-t'il lorsqu'un widget subit une action de l'utilisateur ?
- 👉 Réponse : Un objet **xxxEvent** est automatiquement instancié et envoyé à tous les objets (de réaction) qui sont à l'écoute de **xxxEvent**

Comment mettre un widget à l'écoute ?

- ❖ Ce processus est automatique
- ❖ Si on veut qu'un widget réagisse à un événement autorisé, il faut lui attacher explicitement un objet écouteur (`listener`) de ce type d'événement (`XXXListener`)... sinon rien ne se passe
- ❖ Pour chaque widget concerné par `XXXEvent`, il existe une méthode `addXXXListener(XXXListener o)`
- ❖ Le `XXXListener` va se mettre à l'écoute de l'événement et y réagir grâce à ses méthodes de réaction

Exemple

 Beeper1.java

```
public class Beeper1 extends JFrame {
    JButton button;
    public Beeper1() {
        super("Beeper");
        button = new JButton("Click Me");
        button.setPreferredSize(new Dimension(200, 80));
        button.addActionListener(new Ecouteur());
        add(button);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Beeper1();
            }
        });
    }
}
```

 Ecouteur.java

```
public class Ecouteur implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
    }
}
```

Interfaces pour les réactions

- ❖ Pour attacher un écouteur à un widget, on utilise la méthode `addXXXListener(objetXXXListener)`
- ❖ Le type `XXXListener` est une interface
 - ➡ Il faut donc créer une nouvelle classe qui implémente l'interface `XXXListener`
- ⚠ `ActionListener` : 1 méthode
- ⚠ `MouseListener` : 5 méthodes

? Comment faire ?

MouseListener

❖ Il faut implémenter les 5 méthodes :

☞ mouseClicked

☞ mouseEntered

☞ mouseExited

☞ mousePressed

☞ mouseReleased

❖ Même si on n'a pas besoin de gérer tous ces événements !

Exercice

❖ Comment faire pour que l'écouteur puisse modifier l'interface ?

👉 Par exemple : le clic sur le bouton change l'apparence du bouton (+1)

Solution

- ❖ Pour permettre à l'écouteur de modifier l'interface, on peut inclure une référence vers le composant graphique concerné. Cette référence peut être initialisée lors de la création de l'écouteur
- ❖ Par exemple, pour le bouton :
 - ☞ L'écouteur possède un attribut qui référence le bouton
 - ☞ Cet attribut est initialisé lors de l'appel au constructeur de l'écouteur



```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class ButtonClickCounter extends JFrame {
    private JButton button;
    private int clickCount;
    public ButtonClickCounter() {
        clickCount = 0;
        setTitle("Button Click Counter");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        button = new JButton("Click Me (0)");
        button.setPreferredSize(new Dimension(200, 80));
        button.addActionListener(new ButtonClickListener());
        add(button);
        setVisible(true);
        pack();
    }
    private class ButtonClickListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            clickCount++;
            button.setText("Click Me (" + clickCount + ")");
        }
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new ButtonClickCounter();
            }
        });
    }
}
```

Comment éviter les multiples implémentations vides ?

- ❖ Si on n'a besoin de spécifier le comportement que pour certains des événements définis dans une interface `XXXListener`, on peut passer par des classes abstraites `XXXAdapter` qui implémentent `XXXListener`
 - 👉 Le `XXXAdapter` offre une implémentation de toutes les méthodes définies par l'interface
 - 👉 On se limite à une redéfinition des méthodes intéressantes
 - ⚠ Adapter est une classe
 - 🔗 Ecouteur extends `XXXAdapter` au lieu de implements `XXXListener`

Conclusion

- ❖ Pour créer la réaction à un événement agissant sur un widget :
 - ❶ Identifier les événements que peut capturer le widget en se référant au tableau ou à la documentation (Javadoc)
 - ❷ Créer un objet écouteur (`objetEcouteur`), c'est-à-dire une instance de la classe qui implémente toutes les méthodes de l'interface `XXXListener` ou redéfinit des méthodes de la classe abstraite `XXXAdapter`
 - ❸ Attacher l'objet écouteur (`objetEcouteur`) au widget à l'aide de la méthode `addXXXListener(objetEcouteur)`
 - ❹ Écrire le code qui réalise la réaction à l'événement `XXXEvent` à l'intérieur d'une ou plusieurs méthodes de la classe qui implémente `XXXListener` ou hérite de `XXXAdapter`

Événements	Ecouteurs	Méthodes	Widgets
ActionEvent	ActionListener	actionPerformed	JButton JTextField JMenu JMenuItem List
MouseEvent	MouseListener	mouseClicked mouseEntered mouseExited mousePressed mouseReleased	(J)Component JList
KeyEvent	KeyListener	keyPressed keyTyped keyReleased	(J)Component
FocusEvent	FocusListener	focusGained focusLost	(J)Component
InputMethodEvent	InputMethodListener	inputMethodTextChanged	JTextComponent JTextField JTextArea
ItemEvent	ItemListener	itemStateChanged	JCheckBox Choice List
WindowEvent	WindowListener	windowActivated windowClosing	JWindow

Solution 1

Gestion des événements dans des classes d'écouteur dédiées



Bon découpage du code (modèle MVC)



Nécessite de passer une référence sur la classe IHM au constructeur de la classe externe



Demande l'écriture de nombreux accesseurs dans la classe IHM pour permettre l'accès à ses composants



Chaque écouteur d'événements sur un widget nécessite la création d'une nouvelle classe (augmentation du nombre de classes et de fichiers)



Cela reste la meilleure approche en terme de conception !



Néanmoins, il est important de limiter le nombre de classes de réaction



ButtonClickCounter1.java

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class ButtonClickCounter1 extends JFrame {
    private JButton button;
    public JButton getButton() {
        return button;
    }
    public ButtonClickCounter1() {
        setTitle("Button Click Counter");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        button = new JButton();
        button.setPreferredSize(new Dimension(200, 80));
        button.addActionListener(new ButtonClickListener(this));
        add(button);
        setVisible(true);
        pack();
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new ButtonClickCounter1();
            }
        });
    }
}
```



ButtonClickListener1.java

```
import javax.swing.JButton;  
import java.awt.event.*;  
public class ButtonClickListener1 implements ActionListener {  
    private ButtonClickCounter1 vue;  
    private int clickCount;  
    public ButtonClickListener1(ButtonClickCounter1 vue) {  
        this.vue = vue;  
        clickCount = 0;  
        vue.getButton().setText("Click Me (" + clickCount + ")");  
    }  
    public void actionPerformed(ActionEvent e) {  
        clickCount++;  
        ((JButton)e.getSource()).setText("Click Me (" + clickCount + ")");  
    }  
}
```


Solution 2

La classe d'interface graphique est également un écouteur



Pas besoin d'écrire des accesseurs



La classe d'interface graphique contient beaucoup de code



Toutes les réactions sur tous les widgets sont dans la classe d'interface graphique



La classe d'interface graphique doit implémenter toutes les interfaces de réaction



L'utilisation de multiples Adapter n'est pas possible (pas d'héritage multiple en Java)



Mélange des aspects graphiques (vue) et réaction (contrôleur)



À ÉVITER !



```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class ButtonClickCounter2 extends JFrame implements ActionListener {
    private JButton button;
    private int clickCount;

    public ButtonClickCounter2() {
        clickCount = 0;
        setTitle("Button Click Counter");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        button = new JButton("Click Me (0)");
        button.setPreferredSize(new Dimension(200, 80));
        button.addActionListener(this);
        add(button);
        setVisible(true);
        pack();
    }

    public void actionPerformed(ActionEvent e) {
        clickCount++;
        button.setText("Click Me (" + clickCount + ")");
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new ButtonClickCounter();
            }
        });
    }
}
```

Solution 3

La classe d'écouteur est définie comme une classe interne de l'IG



Pas besoin d'écrire des accesseurs (accessibilité de droit entre les 2 classes)



Les classes internes peuvent hériter des Adapter



Toutes les réactions sur tous les widgets sont dans le fichier définissant la classe d'IG



Mélange des aspects graphiques (vue) et réaction (contrôleur), dans le même fichier



À ÉVITER !



```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class ButtonClickCounter extends JFrame {
    private JButton button;
    private int clickCount;
    public ButtonClickCounter() {
        clickCount = 0;
        setTitle("Button Click Counter");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        button = new JButton("Click Me (0)");
        button.setPreferredSize(new Dimension(200, 80));
        button.addActionListener(new ButtonClickListener());
        add(button);
        setVisible(true);
        pack();
    }
    private class ButtonClickListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            clickCount++;
            button.setText("Click Me (" + clickCount + ")");
        }
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new ButtonClickCounter();
            }
        });
    }
}
```

Solution 4

La classe d'écouteur est définie comme une classe interne anonyme de l'IG



Classe anonyme = classe interne (donc mêmes avantages que solution 3)



Mélange des aspects graphiques (vue) et réaction (contrôleur), dans le même fichier !



Code illisible !!!



À ÉVITER !



Alors pourquoi ?

Solution très souvent utilisée dans les IDE pour faire de la génération automatique de code



```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class ButtonClickCounter3 extends JFrame {
    private JButton button;
    private int clickCount;
    public ButtonClickCounter3() {
        clickCount = 0;
        setTitle("Button Click Counter");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        button = new JButton("Click Me (0)");
        button.setPreferredSize(new Dimension(200, 80));
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                clickCount++;
                button.setText("Click Me (" + clickCount + ")");
            }
        });
        add(button);
        setVisible(true);
        pack();
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new ButtonClickCounter3();
            }
        });
    }
}
```

Conclusion



Privilégier la solution 1

Il faut néanmoins essayer de limiter le nombre de classes de réaction



Comment faire ?

➡ Mutualiser !

- 1 Utiliser une même classe pour écouter les événements sur plusieurs widgets
- 2 Utiliser un même objet pour gérer les réactions
- 3 Accéder au widget à l'origine de l'événement par la méthode `Object getSource()` de la classe d'événement

CounterApp.java

```
import javax.swing.*;
import java.awt.*;

public class CounterApp extends JFrame {
    private int counter;
    private JLabel counterLabel;
    public CounterApp() {
        setTitle("Counter App");
        setSize(300, 200);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        counterLabel = new JLabel("Counter: " + counter);
        panel.add(counterLabel, BorderLayout.CENTER);
        JButton incrementButton = new JButton("+");
        JButton resetButton = new JButton("RAZ");
        ButtonListener listener = new ButtonListener(this);
        incrementButton.addActionListener(listener);
        resetButton.addActionListener(listener);
        panel.add(incrementButton, BorderLayout.NORTH);
        panel.add(resetButton, BorderLayout.SOUTH);
        add(panel);
        setVisible(true);
    }
    public void updateCounterLabel(int counter) {
        counterLabel.setText("Counter: " + counter);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new CounterApp();
            }
        });
    }
}
```




ButtonListener.java

```
import java.awt.event.*;

public class ButtonListener implements ActionListener {
    private int counter;
    private CounterApp view;

    public ButtonListener(CounterApp view) {
        counter = 0;
        this.view = view;
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("+")) {
            counter++;
        } else if (e.getActionCommand().equals("RAZ")) {
            counter = 0;
        }
        view.updateCounterLabel(counter);
    }
}
```

Lancer une application graphique



Dans le thread EDT

```
public static void main(String[] args) {  
    javax.swing.SwingUtilities.invokeLater(  
        new Runnable() {  
            public void run() {  
                new InterfaceGraphique();  
            }  
        });  
}
```



Dans le thread principal

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

Autres gestionnaires de placement

- ☞ **CardLayout** : Permet de superposer plusieurs composants dans le même espace, où un seul est visible à la fois
- ☞ **BoxLayout** : Organise les composants en ligne ou en colonne, avec une flexibilité dans la taille et l'alignement
- ☞ **GridBagLayout** : Offre un contrôle précis sur la disposition des composants en utilisant des contraintes de grille
- ☞ ...

Menus

- ❖ La barre de menu (`JMenuBar`) peut être ajoutée à une fenêtre à l'aide de la méthode `setJMenuBar()`.
- ❖ Menus :
 - ☞ `JMenu` : Un menu à ajouter à la barre de menu
 - ☞ `JMenuItem` : Un élément à ajouter à un menu
- ❖ Les réactions sont déclenchées par les `ActionEvent`
 - ☞ `getSource()` : Obtient le composant à l'origine de l'événement
 - ☞ `getActionCommand()` : Obtient la commande d'action associée à l'événement

Fenêtres de dialogue prédéfinies

❖ JOptionPane

- 👉 showMessageDialog
- 👉 showConfirmDialog
- 👉 showInputDialog
- 👉 showOptionDialog

❖ JFileChooser

Introduction

Interfaces graphiques

Programmation événementielle

Les bases de JavaFX

L'architecture MVC (Model-View-Controller)

Lecture et écriture dans des fichiers

Accès aux bases de données

Aller plus loin en JavaFX

Introduction à JavaFX

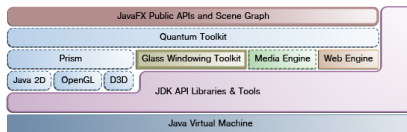
De Swing à JavaFx

Expressions Lambda

Les bases de JavaFX

JavaFX

- ❖ Introduit en 2008 pour remplacer AWT et Swing
- ❖ JavaFX 2.0 depuis 2011
- ❖ Devenu open source depuis JDK 11 (à télécharger depuis openjfx.io)
- ❖ Intégré à Java SE 8 (inclus dans JDK/JRE)
- ❖ Composé de plusieurs éléments :
 - 👉 Prism : moteur graphique
 - 👉 Glass : système de fenêtrage
 - 👉 Moteur web



Compilation/Exécution

- ❶ Télécharger le SDK `openjfx-22_linux-x64`
(ou autre selon votre OS) à partir de `openjfx.io`
- ❷ Déziper `openjfx-22_linux-x64`
- ❸ Extraire le dossier `lib` et le renommer en `javafx` dans
votre répertoire personnel
- ❹ `javac --module-path ~/javafx --add-modules
javafx.controls <fichiers_sources>`
- ❺ `java --module-path ~/javafx --add-modules
javafx.controls <Classe_Principale>`

JavaFX vs AWT et Swing

❖ Swing and AWT étaient d'anciens frameworks Java remplacés par la plateforme JavaFX pour le développement d'applications Internet riches dans JDK 8

❶ Lors de l'introduction de Java, les classes IHM étaient regroupées dans une bibliothèque appelée Abstract Windows Toolkit (AWT)

☞ AWT est sujet à des bugs spécifiques à la plateforme

☞ AWT est adapté pour le développement d'interfaces graphiques simples, mais pas pour les projets IHM complets

❷ Les composants de l'IHM AWT ont été remplacés par une bibliothèque plus robuste, polyvalente et flexible appelée composants Swing

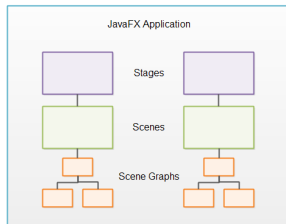
☞ Les composants Swing sont dessinés directement sur des canevas en utilisant du code Java

☞ Les composants Swing dépendent moins de la plateforme cible et utilisaient moins de ressources IHM natives

❸ Avec la sortie de Java 8, Swing a été remplacé par une toute nouvelle plateforme IHM : JavaFX

Structure de base d'une IHM JavaFX

- ❖ `javafx.application.Application` est le point d'entrée des applications JavaFX
 - ☞ JavaFX crée un thread d'application pour exécuter la méthode **start** de l'application, traiter les événements d'entrée et exécuter les chronologies d'animation
 - ☞ On redéfinit la méthode **start(Stage)**
- ❖ `javafx.stage.Stage` est le conteneur JavaFX de plus haut niveau (c'est-à-dire, la fenêtre)
 - ☞ Le principal **Stage** est construit par la plateforme
- ❖ `javafx.scene.Scene` est le conteneur pour tout le contenu dans un graphe de scène dans le stage
- ❖ `javafx.scene.Node` est la classe de base pour les noeuds du graphe de scène (c'est-à-dire, les composants)



Points à retenir sur JavaFX

- ☞ La classe principale d'une application JavaFX hérite de `javafx.application.Application`
- ☞ La méthode `start()` est le point d'entrée principal de l'application JavaFX
- ☞ L'interface utilisateur est définie par deux niveaux : le **Stage** et la **Scene**
- ☞ La classe **Stage** est le conteneur de plus haut niveau principal
- ☞ La classe **Scene** est le conteneur pour tout le contenu de l'interface utilisateur
- ☞ Le contenu d'une scène est représenté par un graphe hiérarchique de noeuds, où chaque noeud est un composant graphique

Exemple



```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
public class PPJavaFX extends Application {
    @Override // Redéfinition de start
    public void start(Stage primaryStage) {
        //Créer un bouton et le placer sur la scene
        Button btOK = new Button("OK");
        Scene scene = new Scene(btOK, 200, 250);
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.setTitle("Premier pas en JavaFX"); // Set the stage title
        primaryStage.show(); // Display the stage
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Pas trop de changement...

- ❖ Gestionnaires de placement
- ❖ Composants : Text, Label, TextField, PasswordField, Button, ...
- ❖ Réaction aux événements :
 - 👉 implements ActionListener → implements EventHandler<ActionEvent>
 - 👉 addActionListener → setOnAction
 - 👉 ...

Gestionnaires de placement (Layout Panes)

Classe	Description	Équivalent Swing
Pane	Classe de base pour les layout panes. Contient <code>getChildren()</code> qui retourne une liste de noeuds dans le pane.	-
StackPane	Place les noeuds les uns sur les autres dans le centre du pane.	-
FlowPane	Place les noeuds ligne par ligne horizontalement ou colonne par colonne verticalement.	JPanel+FlowLayout
GridPane	Place les noeuds en cellules dans une grille à deux dimensions.	JPanel+GridLayout
BorderPane	Place les noeuds en haut, à droite, en bas, à gauche et au centre.	JPanel+BorderLayout
HBox	Place les noeuds en une seule ligne.	JPanel+BoxLayout (X_AXIS)
VBox	Place les noeuds en une seule colonne.	JPanel+BoxLayout (Y_AXIS)

Exemple I



FlowPane

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.geometry.Insets;

public class DemoFlowPane extends Application {
    @Override
    public void start(Stage primaryStage) {
        FlowPane pane = new FlowPane();
        pane.setPadding(new Insets(11, 12, 13, 14));
        pane.setHgap(5);
        pane.setVgap(5);
        // Placer les noeuds dans le panneau
        pane.getChildren().addAll(new Label("Prénom:"), new TextField());
        pane.getChildren().addAll(new Label("Nom:"), new TextField());
        // Créer une Scene et la placer dans le stage
        Scene scene = new Scene(pane, 210, 120);
        primaryStage.setTitle("DémO FlowPane");
        primaryStage.setScene(scene); // Place la scene dans le stage
        primaryStage.show(); // Afficher le stage
    }

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

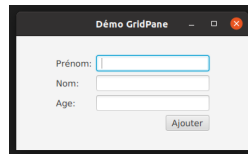
Exemple II



GridPane

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.GridPane;
import javafx.scene.control.*;
import javafx.geometry.*;

public class DemoGridPane extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Créer le panneau et ses propriétés
        GridPane pane = new GridPane();
        pane.setAlignment(Pos.CENTER);
        pane.setHgap(5.5);
        pane.setVgap(5.5);
        // Placer les nœuds à leurs positions (colonne, ligne)
        pane.add(new Label("Prénom:"), 0, 0);
        pane.add(new TextField(), 1, 0);
        pane.add(new Label("Nom:"), 0, 1);
        pane.add(new TextField(), 1, 1);
        pane.add(new Label("Age:"), 0, 2);
        pane.add(new TextField(), 1, 2);
        Button btaj = new Button("Ajouter");
        pane.add(btaj, 1, 3);
        GridPane.setHalignment(btaj, HPos.RIGHT);
        // Créer une Scene et la placer dans le stage
        Scene scene = new Scene(pane);
        primaryStage.setTitle("Démo GridPane");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

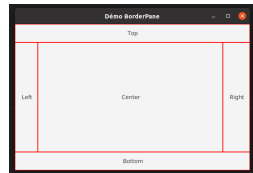


Exemple III

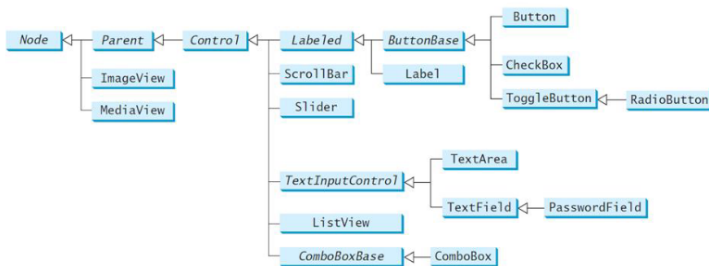


BorderPane

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Label;
import javafx.geometry.Insets;
public class DemoBorderPane extends Application {
    @Override
    public void start(Stage primaryStage) {
        BorderPane pane = new BorderPane();
        pane.setTop(new CustomPane("Top"));
        pane.setRight(new CustomPane("Right"));
        pane.setBottom(new CustomPane("Bottom"));
        pane.setLeft(new CustomPane("Left"));
        pane.setCenter(new CustomPane("Center"));
        primaryStage.setTitle("D mo BorderPane");
        Scene scene = new Scene(pane);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
class CustomPane extends StackPane {
    public CustomPane(String title) {
        getChildren().add(new Label(title));
        setStyle("--fx-border-color: red");
        setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
    }
}
```

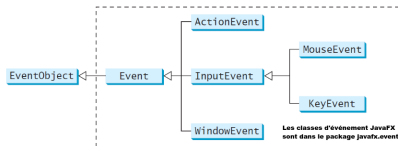


Les composants JavaFX (control)



https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm

Actions utilisateur, événements et gestionnaires



Action de l'utilisateur	Objet source	Type d'événement déclenché	Méthode d'inscription à l'événement
Cliquer sur un bouton	Button	ActionEvent	<code>setOnAction (EventHandler<ActionEvent>)</code>
Appuyer sur Entrée dans un champ de texte	TextField	ActionEvent	<code>setOnAction (EventHandler<ActionEvent>)</code>
Cocher ou décocher	RadioButton	ActionEvent	<code>setOnAction (EventHandler<ActionEvent>)</code>
Cocher ou décocher	CheckBox	ActionEvent	<code>setOnAction (EventHandler<ActionEvent>)</code>
Sélectionner un nouvel élément	ComboBox	ActionEvent	<code>setOnAction (EventHandler<ActionEvent>)</code>
Appuyer sur la souris	Noeud, Scene	MouseEvent	<code>setOnMousePressed (EventHandler<MouseEvent>)</code>
Relâcher la souris	Noeud, Scene	MouseEvent	<code>setOnMouseReleased (EventHandler<MouseEvent>)</code>
Cliquer sur la souris	Noeud, Scene	MouseEvent	<code>setOnMouseClicked (EventHandler<MouseEvent>)</code>
La souris entre dans le noeud ou la scène	Noeud, Scene	MouseEvent	<code>setOnMouseEntered (EventHandler<MouseEvent>)</code>
La souris sort du noeud ou de la scène	Noeud, Scene	MouseEvent	<code>setOnMouseExited (EventHandler<MouseEvent>)</code>
La souris bouge sur le noeud ou la scène	Noeud, Scene	MouseEvent	<code>setOnMouseMoved (EventHandler<MouseEvent>)</code>
La souris est traînée sur le noeud ou la scène	Noeud, Scene	MouseEvent	<code>setOnMouseDragged (EventHandler<MouseEvent>)</code>
Enfoncer une touche	Noeud, Scene	KeyEvent	<code>setOnKeyPressed (EventHandler<KeyEvent>)</code>
Libérer une touche	Noeud, Scene	KeyEvent	<code>setOnKeyReleased (EventHandler<KeyEvent>)</code>
Enfoncer une touche de saisie	Noeud, Scene	KeyEvent	<code>setOnKeyTyped (EventHandler<KeyEvent>)</code>

Expressions Lambda

- ☞ Les expressions lambda sont une nouvelle fonctionnalité introduite dans Java 8
- ☞ Elles permettent de définir des fonctions prédéfinies pour le type d'entrée
- ☞ Les expressions lambda peuvent être considérées comme une méthode anonyme avec une syntaxe concise
- ☞ On peut utiliser les expressions lambda pour tous listener ayant une seule méthode



Classe anonyme

```
bouton.setOnAction(  
    new EventHandler<ActionEvent>() {  
        @Override  
        public void handle(ActionEvent e) {  
            // Code pour traiter l'événement e  
        }  
    });
```



Expression lambda

```
bouton.setOnAction(e -> {  
    // Code pour traiter l'événement e  
});
```

Syntaxe de base d'une expression lambda

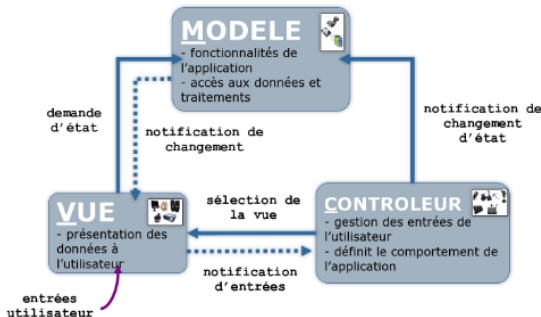
- ❖ La syntaxe de base pour une expression lambda est :
 - ☞ `(type1 param1, type2 param2, ...) -> expression`
 - ☞ `(type1 param1, type2 param2, ...) -> instructions;`
- ❖ Le type de données d'un paramètre peut être déclaré explicitement ou implicitement inféré par le compilateur
- ❖ Les parenthèses peuvent être omises s'il n'y a qu'un seul paramètre sans type de données explicite

L'architecture MVC (Model-View-Controller)

MVC : Historique

- ❖ L'architecture MVC (Modèle-Vue-Contrôleur) a été conçue pour organiser les applications interactives en séparant clairement :
 - 👉 Un module chargé exclusivement de la gestion des données (Modèle)
 - 👉 Un module chargé exclusivement de la représentation des données (l'interface graphique, Vue)
 - 👉 Un module chargé exclusivement de traiter les actions/réactions avec l'utilisateur (Contrôleur)
- ❖ Cette idée, développée par la société Rank Xerox dans les années 1970, a été appliquée aux interfaces graphiques (GUI) dès les années 1980

MVC (version avec Observateur/Observé)



MVC : Avantages

- 😊 Amène un bon découpage du code en structure orientée objet
- 😊 Sépare le code en trois modules (presque) indépendants :
Modèle, Vue et Contrôleur
- 😊 Facilite la maintenance : la modification d'un module n'influence pas les autres
- 😊 Le module Vue, qui demande le plus de travail, peut avoir plusieurs versions sans perturber les autres modules
- 😊 Facilite le développement en équipe en permettant une meilleure répartition des tâches
- 😊 Favorise une concentration accrue sur l'ergonomie en permettant une séparation claire entre la logique métier et la présentation graphique

MVC : Organisation en Java

❖ 1 module = 1 paquetage

👉 Package control

👉 Package view

👉 Package data (ou model)

View

- ❖ Contient TOUTES les classes Java qui mettent en place le décor (GUI)
- ❖ Mais ne contient PAS :
 - ☞ Les classes de réaction, car elles sont à l'écoute des événements utilisateur (partie contrôle)
 - ☞ Les classes qui mémorisent et agissent sur les données (partie modèle)

Model

- ❖ Contient TOUTES les classes Java qui agissent sur les données :
 - ☞ Gère les opérations sur les données telles que l'ajout, la suppression, la modification et la recherche
 - ☞ Gère l'écriture et la lecture des données pour la persistance
- ❖ Ne contient AUCUNE :
 - ☞ Déclaration ou importation de classes du paquetage view

Control

- ❖ Contient TOUTES les classes Java qui sont à l'écoute des événements utilisateur (= les classes de réaction aux événements)
- ❖ En réaction aux événements :
 - 👉 Elles agissent sur les données
 - 👉 Elles modifient l'IG (interaction utilisateur)

Exemple MVC

(`java.beans.PropertyChangeListener`) I



Modele

```
package modele;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
public class Modele {
    private double tempCelsius;
    private PropertyChangeSupport changeSupport = new PropertyChangeSupport(this);
    public double getTempCelsius() {
        return this.tempCelsius;
    }
    public double getTempFahrenheit() {
        return this.tempCelsius * 9 / 5 + 32;
    }
    public void setTempFahrenheit(double tempF) {
        double oldVal = this.tempCelsius;
        this.tempCelsius = (tempF - 32) * 5 / 9;
        this.changeSupport.firePropertyChange("Fahrenheit", oldVal, this.tempCelsius);
    }
    public void setTempCelsius(double tempC) {
        double oldVal = this.tempCelsius;
        this.tempCelsius = tempC;
        this.changeSupport.firePropertyChange("Celsius", oldVal, this.tempCelsius);
    }
    public void addPropertyChangeListener(PropertyChangeListener listener) {
        this.changeSupport.addPropertyChangeListener(listener);
    }
    public void removePropertyChangeListener(PropertyChangeListener listener) {
        this.changeSupport.removePropertyChangeListener(listener);
    }
}
```

Exemple MVC

(`java.beans.PropertyChangeListener`) II



Vue 1

```
package vue;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.geometry.*;
import javafx.stage.Stage;
import modele.*;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;

public class Vue1 extends Stage {
    private Label label = new Label("Température en C°");
    private TextField text = new TextField();
    public TextField getText() {
        return this.text;
    }
    private Modele modele;
    public Vue1(Modele m) {
        this.setTitle("Température celsius");
        GridPane pane = new GridPane();
        pane.setAlignment(Pos.CENTER);
        pane.setHgap(5.5);
        pane.setVgap(5.5);
        this.modele = m;
        this.text.setText(m.getTempCelsius() + "");
        pane.add(this.label, 0, 0);
        pane.add(this.text, 1, 0);
        setScene(new Scene(pane, 400, 50));
    }
}
```

Exemple MVC

(`java.beans.PropertyChangeListener`) III



Vue 2

```
package vue;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.geometry.*;
import javafx.stage.Stage;
import modele.Modele;

public class Vue2 extends Stage {
    private Label label = new Label("Temperature en F:");
    private TextField text = new TextField();
    public TextField getText() {
        return this.text;
    }

    private Modele modele;
    public Vue2(Modele m) {
        this.setTitle("Temperature Fahrenheit");
        GridPane pane = new GridPane();
        pane.setAlignment(Pos.CENTER);
        pane.setHgap(5.5);
        pane.setVgap(5.5);
        this.modele = m;
        this.text.setText(m.getTempFahrenheit() + "");
        pane.add(this.label, 0, 0);
        pane.add(this.text, 1, 0);
        setScene(new Scene(pane, 400, 50));
    }
}
```


Exemple MVC

(`java.beans.PropertyChangeListener`) IV



Contrôleur

```
package control;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import javafx.event.*;
import modele.*;
import vue.Vue1;
import vue.Vue2;

public class Controleur implements EventHandler<ActionEvent>, PropertyChangeListener {
    private Modele modele;
    private Vue1 vue1;
    private Vue2 vue2;
    public Controleur(Modele modele, Vue1 vue1, Vue2 vue2) {
        this.modele = modele;
        this.vue1 = vue1;
        this.vue2 = vue2;
    }
    public void handle(ActionEvent e) {
        if (e.getSource() == this.vue1.getText())
            this.modele.setTempCelsius(Double.parseDouble(this.vue1.getText().getText()));
        else if (e.getSource() == this.vue2.getText())
            this.modele.setTempFahrenheit(Double.parseDouble(this.vue2.getText().getText()));
    }
    public void propertyChange(PropertyChangeEvent event) {
        if(event.getPropertyName().equals("Celsius")){
            this.vue2.getText().setText(modele.getTempFahrenheit() + "");
        }
        else{
            this.vue1.getText().setText(modele.getTempCelsius() + "");
        }
    }
}
```

Exemple MVC

(`java.beans.PropertyChangeListener`) V



Application

```
import control.Controleur;
import javafx.application.Application;
import javafx.stage.Stage;
import modele.Modele;
import vue.Vue1;
import vue.Vue2;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.geometry.Insets;
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        Modele modele = new Modele();
        Vue1 vue1 = new Vue1(modele); // Place la scene dans le stage
        primaryStage = vue1;
        primaryStage.show(); // Afficher le stage
        Vue2 vue2 = new Vue2(modele);
        double xOffset = primaryStage.getX() + primaryStage.getWidth() + 10;
        vue2.setX(xOffset);
        vue2.setY(primaryStage.getY());
        vue2.show();
        Controleur controleur = new Controleur(modele, vue1, vue2);
        vue1.getText().setOnAction(controleur);
        vue2.getText().setOnAction(controleur);
        modele.addPropertyChangeListener(controleur);
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Exemple MVC

(`javafx.beans.InvalidationListener`) I



Modele

```
package modele;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
public class Modele {
    private DoubleProperty tempCelsius = new SimpleDoubleProperty();
    public double getTempCelsius() {
        return tempCelsius.get();
    }
    public DoubleProperty getTempCelsiusProperty() {
        return tempCelsius;
    }
    public double getTempFahrenheit() {
        return tempCelsius.get() * 9 / 5 + 32;
    }
    public void setTempFahrenheit(double tempF) {
        this.tempCelsius.set((tempF - 32) * 5 / 9);
    }
    public void setTempCelsius(double tempC) {
        this.tempCelsius.set(tempC);
    }
}
```

Exemple MVC

(`javafx.beans.InvalidationListener`) II



Vue 1

```
package vue;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.geometry.*;
import javafx.stage.Stage;
import modele.Modele;

public class Vue1 extends Stage {
    private Label label = new Label("Température en C:");
    private TextField text = new TextField();
    public TextField getText() {
        return this.text;
    }
    private Modele modele;
    public Vue1(Modele m) {
        this.setTitle("Température celsius");
        GridPane pane = new GridPane();
        pane.setAlignment(Pos.CENTER);
        pane.setHgap(5.5);
        pane.setVgap(5.5);
        this.modele = m;
        this.text.setText(m.getTempCelsius() + "");
        pane.add(this.label, 0, 0);
        pane.add(this.text, 1, 0);
        setScene(new Scene(pane, 400, 50));
    }
}
```

Exemple MVC

(`javafx.beans.InvalidationListener`) III



Vue 2

```
package vue;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.geometry.*;
import javafx.stage.Stage;
import modele.Modele;
import javafx.beans.InvalidationListener;
import javafx.beans.Observable;

public class Vue2 extends Stage {
    private Label label = new Label("Température en F:");
    private TextField text = new TextField();
    public TextField getText() {
        return this.text;
    }
    private Modele modele;
    public Vue2(Modele m) {
        this.setTitle("Température Farenheit");
        GridPane pane = new GridPane();
        pane.setAlignment(Pos.CENTER);
        pane.setHgap(5.5);
        pane.setVgap(5.5);
        this.modele = m;
        this.text.setText(m.getTempFahrenheit() + "");
        pane.add(this.label, 0, 0);
        pane.add(this.text, 1, 0);
        setScene(new Scene(pane, 400, 50));
    }
}
```

Exemple MVC

(`javafx.beans.InvalidationListener`) IV



Contrôleur

```
package control;
import javafx.event.*;
import modele.Modele;
import vue.Vue1;
import vue.Vue2;

import java.beans.PropertyChangeEvent;
import javafx.beans.InvalidationListener;
import javafx.beans.Observable;

public class Contrôleur implements EventHandler<ActionEvent>, InvalidationListener {
    private Modele modele;
    private Vue1 vue1;
    private Vue2 vue2;
    public Contrôleur(Modele modele, Vue1 vue1, Vue2 vue2) {
        this.modele = modele;
        this.vue1 = vue1;
        this.vue2 = vue2;
    }
    public void handle(ActionEvent e) {
        if (e.getSource() == this.vue1.getText())
            this.modele.setTempCelsius(Double.parseDouble(this.vue1.getText().getText()));
        else if (e.getSource() == this.vue2.getText())
            this.modele.setTempFahrenheit(Double.parseDouble(this.vue2.getText().getText()));
    }
    public void invalidated(Observable ov) {
        this.vue1.getText().setText(modele.getTempCelsius() + "");
        this.vue2.getText().setText(modele.getTempFahrenheit() + "");
    }
}
```

Exemple MVC

(`javafx.beans.InvalidationListener`) V



Application

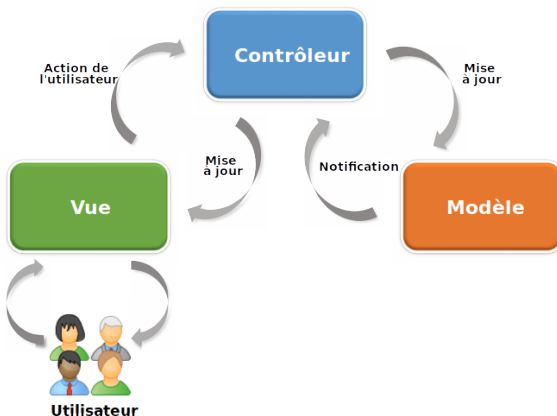
```
import java.util.Observable;

import control.Cycle;
import javafx.application.Application;
import javafx.stage.Stage;
import modele.Model;
import vue.Vue1;
import vue.Vue2;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.geometry.Insets;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        Model modele = new Model();
        Vue1 vue1 = new Vue1(modele); // Place la scene dans le stage
        primaryStage = vue1;
        primaryStage.show(); // Afficher le stage
        Vue2 vue2 = new Vue2(modele);
        double xOffset = primaryStage.getWidth() + 10;
        vue2.setX(xOffset);
        vue2.setY(primaryStage.getY());
        vue2.show();
        Cycle cycle = new Cycle(modele, vue1, vue2);
        vue1.getText().setOnAction(cycle);
        vue2.getText().setOnAction(cycle);
        modele.getTempCelsiusProperty().addListener(cycle);
    }

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

MVC (version sans Observateur/Observé)



Compilation avec plusieurs packages

❖ Dans un dossier de projet :


- ☞ Si le code source est dans un dossier **src**
- ☞ Si l'on veut placer les fichiers compilés dans un dossier **build**
- ☞ Si vous vous trouvez dans le **dossier du projet**
- ☞ `javac -d build -sourcepath src `find . -name '*.java'``
- ☞ `java -cp build <Classe_Principale>`

Lecture et écriture dans des fichiers

Écrire dans une console



```
System.out.println("mon premier test");
```

 On appelle la méthode `println` de l'attribut `out` de la classe `System`



JavaDoc

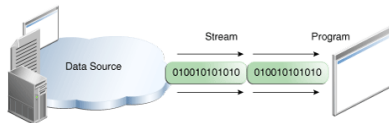
out est un `PrintStream`

- ❖ `Stream` = flot/flux de données
 - 👉 avec un début et une fin
 - 👉 en lecture (`in`) ou en écriture (`out`)
- ❖ De nombreuses classes dans `java.io`

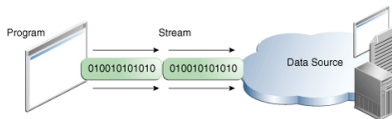


- ❖ 4 classes mères abstraites :
 - 👉 `InputStream` et `OutputStream` (bits)
 - 👉 `Reader` et `Writer` (caractères)

Les flux de données



IN=Lecture



OUT=Écriture

Écrire dans un fichier

- ❖ Idem mais on remplace la console par un fichier



```
out.println("mon premier test");
```

- 👉 On appelle la méthode `println` de l'objet `out`

? Comment spécifier que `out` pointe vers le bon fichier ?



JavaDoc

Flux en écriture vers un fichier



```
PrintStream out = new PrintStream ("test.txt");  
out.println("mon premier test");
```

ou



```
PrintWriter out = new PrintWriter ("test.txt");  
out.println("mon premier test");
```



Une solution simple... mais incomplète !

Flux en écriture vers un fichier



```
try
{
    PrintWriter out = new PrintWriter ("test.txt");
    out.println("mon premier test");
}
catch (FileNotFoundException ex)
{
    System.out.println(ex.getMessage());
    // ou ex.printStackTrace();
}
```



Une solution simple... mais incomplète !

Flux en écriture vers un fichier



```
try
{
    PrintWriter out = new PrintWriter ("test.txt");
    out.println("mon premier test");
    out.close();
}
catch (IOException ex)
{
    System.out.println(ex.getMessage());
}
```



Une solution simple... mais limitée !

Flux en écriture vers un fichier



```
PrintWriter out = null;
try
{
    out = new PrintWriter ("test.txt");
    out.println("mon premier test");
}
catch (FileNotFoundException ex1)
{
    System.out.println(ex1.getMessage());
}
finally{
    try
    {
        if (out != null)
            out.close();
    }
    catch(IOException ex2)
    {
        System.out.println(ex2.getMessage());
    }
}
```



Toujours solution simple... mais toujours limitée !

Flux en écriture vers un fichier

- ❖ Le `try-with-resource` permet de déclarer la ressource à clore dans le `try`, ce qui en simplifie grandement la structure

☞ Valable pour n'importe quelle classe qui implémente l'interface `java.lang.AutoCloseable`



```
try(PrintWriter out = new PrintWriter ("test.txt"))
{
    out.println("mon premier test");
}
catch (FileNotFoundException ex1)
{
    System.out.println(ex1.getMessage());
}
```



Une solution plus simple... mais toujours limitée !

Un meilleur contrôle sur le flux de sortie vers un fichier

❖ Fichier = File... ➡ FileOutputStream, FileWriter, etc.

? Combiner FileWriter et PrintWriter ?



```
FileWriter f = new FileWriter ("test.txt");
PrintWriter p = new PrintWriter (f);

PrintWriter out =
new PrintWriter (new FileWriter ("test.txt"));

PrintWriter out =
new PrintWriter (new FileWriter ("test.txt", true));
```

Encore plus de contrôle



```
try(FileWriter file = new FileWriter("test.txt") ;  
    BufferedWriter buf = new BufferedWriter(file);  
    PrintWriter out = new PrintWriter (buf))  
{  
    out.println("une ligne à écrire");  
}  
catch(IOException ex)  
{  
    ex.printStackTrace();  
}
```

- ☞ La mise en tampon permet de gagner en efficacité!
- ☞ Toutes ces classes héritent de `Writer`
- ☞ C'est un patron de conception bien connu (**décorateur**)...
...enseigné en 2ème année!

Lire depuis la console



```
Scanner sc = new Scanner(System.in);  
String s = sc.nextLine();
```



`System.in` est un objet de type `InputStream`



Peut-on lire directement dans un fichier ?



JavaDoc

Lire depuis un fichier texte

- ❖ Comme pour l'écriture, plusieurs classes aux fonctionnalités complémentaires sont utilisées pour la lecture :

- 👉 `FileReader`

- 👉 `BufferedReader`



```
try(FileReader file = new FileReader ("test.txt");
    BufferedReader in = new BufferedReader(file))
{
    String s=in.readLine();
    while(s!=null)
    {
        System.out.println(s);
        s=in.readLine();
    }
}
catch(IOException e)
{
    e.printStackTrace();
}
```

Utilisation de `BufferedReader.lines`

- ❖ Retourne le flux des lignes du lecteur auquel on l'applique
- ❖ Sert de pont entre le monde des lecteurs et celui des flux



Afficher toutes les lignes non vides du fichier `in.txt`

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class TestForEach {
    public static void main(String[] args) {
        try (BufferedReader r = new BufferedReader(new FileReader("in.txt"))) {
            r.lines()
                .filter(l -> !l.isEmpty())
                .forEach(l -> { System.out.println(l); });
        }
        catch(IOException ex){
            ex.printStackTrace();
        }
    }
}
```


Transformation °F en °C



```
import java.io.*;

public class TestForEachF {
    public static void main(String[] args) {
        try (BufferedReader r = new BufferedReader(new FileReader("f.txt"));
            PrintWriter w = new PrintWriter(new FileWriter("c.txt"))) {
            r.lines()
                .filter(l -> !l.isEmpty())
                .map(Double::parseDouble)
                .map(f -> (f - 32d) * (5d / 9d))
                .forEach(w::println);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```



Remplacer la classe Scanner

? Peut-on écrire une classe Scanner ?



```
try(BufferedReader br = new BufferedReader(new InputStreamReader(System.in))){  
    String s = br.readLine();  
}  
catch(IOException ex)  
{  
    ex.printStackTrace();  
}
```

Décomposition des lignes du fichier

❖ La classe `java.util.StringTokenizer`

- 👉 Utilisée pour décomposer une chaîne de caractères en tokens (mots)
- 👉 Méthodes : `hasMoreTokens()` et `nextToken()`
- 👉 Alternative à `Scanner`

❖ La classe `String` et la méthode `split`

❖ Conversion en types primitifs avec les classes `Wrappers`

- 👉 `Integer`, `Double`, `Boolean`, etc.
- 👉 Exemple : `int num = Integer.parseInt("123");`

Les fichiers textes, des cas particuliers

- ❖ L'ordinateur ne distingue pas un fichier texte d'un fichier binaire : c'est l'humain qui le fait !
- ❖ Un fichier texte est un fichier binaire où :
 - ☞ Chaque octet correspond au code ASCII du caractère
 - ☞ Chaque ligne se termine par un caractère de fin de ligne (**LF** sous **linux** et **CR + LF** sous **windows**)
 - ☞ Le fichier se termine par le caractère de fin de fichier (**EOF**)
- ❖ Les éditeurs de texte savent lire les fichiers texte

Les fichiers binaires

❖ Composé de 0 et de 1

? **Comment le lire ?**

- 😊 rapidité de lecture/écriture
- 😊 compacité
- 😊 protection (nécessite le bon décodeur)
- 😞 pas lisible facilement
- 😞 nécessite de connaître le format (dans l'en-tête ou ailleurs)

Lecture et écriture de données binaires

DataInputStream



DataOutputStream



Exemple I



```
import java.io.*;
public class DemoDataIO {
    public static void main(String[] args) {
        // écrire les données
        try (DataOutputStream out = new DataOutputStream(new FileOutputStream("facture.txt")))
        {
            double[] prix_unitaires = { 19.99, 9.99, 3.99, 4.99 };
            int[] unites = { 12, 8, 29, 50 };
            String[] descs = { "T-shirt Java",
                               "Tasse Java",
                               "Épingle Java",
                               "Porte-clés Java" };
            for (int i = 0; i < prix_unitaires.length; i++) {
                out.writeDouble(prix_unitaires[i]);
                out.writeInt(unites[i]);
                out.writeUTF(descs[i]);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Exemple II



```
// lire encore le fichier
try (DataInputStream in = new DataInputStream(new FileInputStream("facture.txt"))) {
    double prix_unitaire;
    int unite;
    String desc;
    double total = 0.0;
    while (in.available() != 0) {
        prix_unitaire = in.readDouble();
        unite = in.readInt();
        desc = in.readUTF();
        System.out.println("Vous avez commandé " +
            unite + " unités de " +
            desc + " à " + prix_unitaire + " $");
        total = total + unite * prix_unitaire;
    }
    System.out.println("Pour un total de " + total + " $");
} catch (IOException ex) {
    ex.printStackTrace();
}
}
```


Aller plus loin

❖ Java est un langage objet

? peut-on lire / écrire des objets ?

? pourquoi le faire ?

Lire / écrire des objets

- ❖ `Personne = nom, prénom, naissance`
- ❖ `Date = jour, mois, année`
- ❖ Écrire un programme qui lit ou écrit une personne dans un fichier
 - 👉 en texte
 - 👉 en binaire

Exemple I



```
public class Date {
    private int jour;
    private int mois;
    private int annee;
    public Date(int jour, int mois, int annee) {
        this.jour = jour;
        this.mois = mois;
        this.annee = annee;
    }
    public int diff(Date autre) {
        int a = (annee - autre.annee);
        int m = (mois - autre.mois);
        int d = jour - autre.jour;
        if (a == 0)
            return 0;
        else {
            if (a > 0) {
                if (m < 0 || (m == 0 && d < 0))
                    a = a - 1;
                return a;
            }
        }
        return autre.diff(this);
    }
}
```



```
public String toString() {
    return jour + "/" + mois + "/" + annee;
}
public int getJour() {
    return jour;
}
public void setJour(int jour) {
    this.jour = jour;
}
public int getMois() {
    return mois;
}
public void setMois(int mois) {
    this.mois = mois;
}
public int getAnnee() {
    return annee;
}
public void setAnnee(int annee) {
    this.annee = annee;
}
}
```

Exemple II



```
import java.util.Calendar;
public class Personne{
    private String nom, prenom;
    private Date naissance;
    private int age;
    public Personne(String nom, String prenom, Date naissance)
    {
        this.nom = nom;
        this.prenom = prenom;
        this.naissance = naissance;
        Calendar mtn = Calendar.getInstance();
        this.age = naissance.diff(
            new Date(mtn.get(java.util.Calendar.
                DAY_OF_MONTH),
                    mtn.get(java.util.Calendar.MONTH) + 1,
                    mtn.get(java.util.Calendar.YEAR)));
    }
    public String toString() {
        return nom + "," + prenom + "," + naissance + "," + age
        ;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
}
```



```
public String getPrenom() {
    return prenom;
}
public void setPrenom(String prenom)
{
    this.prenom = prenom;
}
public Date getNaissance() {
    return naissance;
}
public void setNaissance(Date
naissance) {
    this.naissance = naissance;
}
public int getAge() {
    return age;
}
}
```

Exemple III



```
import java.io.*;
import java.util.*;
public class DemoPersonneText {
    public static void main(String[] args) {
        Personne p = new Personne("Belmouhcine", "Abdelbadie", new Date(27, 10, 1949));
        System.out.println(p);
        try (FileWriter file = new FileWriter("personne.txt");
            BufferedWriter buf = new BufferedWriter(file);
            PrintWriter out = new PrintWriter(buf)) {
            out.println(p.getNom() + "," + p.getPrenom() + "," + p.getNaissance().getJour() + ","
                + p.getNaissance().getMois() + "," + p.getNaissance().getAnnee());
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        List<Personne> personnes = new LinkedList<>();
        try (FileReader file = new FileReader("personne.txt");
            BufferedReader buf = new BufferedReader(file)) {
            buf.lines().forEach(s -> {
                String[] ligne = s.split(",");
                personnes.add(new Personne(ligne[0], ligne[1],
                    new Date(Integer.parseInt(ligne[2]), Integer.parseInt(ligne[3]), Integer.parseInt(ligne[4])))
            );
            });
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        System.out.println(personnes.get(0));
    }
}
```

Exemple IV



```
import java.io.*;
import java.util.*;
public class DemoPersonneBin {
    public static void main(String[] args) {
        Personne p = new Personne("Belmouhcine", "Abdelbadie", new Date(27, 10, 1949));
        System.out.println(p);
        try (DataOutputStream out = new DataOutputStream(new FileOutputStream("personne.bin", true))) {
            out.writeUTF(p.getNom());
            out.writeUTF(p.getPrenom());
            out.writeInt(p.getNaissance().getJour());
            out.writeInt(p.getNaissance().getMois());
            out.writeInt(p.getNaissance().getAnnee());
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        List<Personne> personnes = new LinkedList<>();
        try (DataInputStream in = new DataInputStream(new FileInputStream("personne.bin"))) {
            while (in.available() != 0)
                personnes.add(
                    new Personne(in.readUTF(), in.readUTF(), new Date(in.readInt(), in.readInt(), in.readInt()))
                );
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        System.out.println(personnes.get(0));
    }
}
```

Le mécanisme de sérialisation

- ❖ Java offre un mécanisme qui permet d'écrire / lire facilement des objets (et tous leurs constituants de façon récursive)
 - ❖ Interface `Serializable`
 - ❖ Lecture : `ObjectInputStream`
 - ❖ Ecriture : `ObjectOutputStream`

Exemple I



```
import java.io.Serializable;

public class Date implements Serializable{
    private static final long serialVersionUID = 1L;
    private int jour;
    private int mois;
    private int annee;
    public Date(int jour, int mois, int annee) {
        this.jour = jour;
        this.mois = mois;
        this.annee = annee;
    }
    public int diff(Date autre) {
        int a = (annee - autre.annee);
        int m = (mois - autre.mois);
        int d = jour - autre.jour;
        if (a == 0)
            return 0;
        else {
            if (a > 0) {
                if (m < 0 || (m == 0 && d < 0))
                    a = a - 1;
                return a;
            }
        }
        return autre.diff(this);
    }
}
```



```
public String toString() {
    return jour + "/" + mois + "/" + annee;
}
public int getJour() {
    return jour;
}
public void setJour(int jour) {
    this.jour = jour;
}
public int getMois() {
    return mois;
}
public void setMois(int mois) {
    this.mois = mois;
}
public int getAnnee() {
    return annee;
}
public void setAnnee(int annee) {
    this.annee = annee;
}
}
```


Exemple II



```
import java.io.*;
import java.util.Calendar;
public class Personne implements Serializable {
    private static final long serialVersionUID = 1L;
    private String nom, prenom;
    private Date naissance;
    private transient int age;
    public Personne(String nom, String prenom, Date naissance)
    {
        this.nom = nom;
        this.prenom = prenom;
        this.naissance = naissance;
        Calendar mtn = Calendar.getInstance();
        this.age = naissance.diff(
            new Date(mtn.get(java.util.Calendar.
                DAY_OF_MONTH),
                    mtn.get(java.util.Calendar.MONTH) + 1,
                    mtn.get(java.util.Calendar.YEAR)));
    }
    public String toString() {
        return nom + "," + prenom + "," + naissance + "," + age
        ;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
}
```



```
public String getPrenom() {
    return prenom;
}
public void setPrenom(String prenom)
{
    this.prenom = prenom;
}
public Date getNaissance() {
    return naissance;
}
public void setNaissance(Date
naissance) {
    this.naissance = naissance;
}
public int getAge() {
    return age;
}
}
```

Exemple III



```
import java.io.*;
import java.util.*;
public class DemoPersonneObj {
    public static void main(String[] args) {
        Personne p = new Personne("Belmouhcine", "Abdelbadie", new Date(27, 10, 1959));
        System.out.println(p);
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("personne.o"))) {
            out.writeObject(p);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("personne.o"))) {
            p = (Personne) in.readObject();
        } catch (IOException ex1) {
            ex1.printStackTrace();
        } catch (ClassNotFoundException ex2) {
            ex2.printStackTrace();
        }
        System.out.println(p);
    }
}
```

Aller plus loin

- ➡ Spécifier le mécanisme de lecture / écriture en surchargeant les méthodes `readObject` / `writeObject`
- ➡ Utiliser `transient`

Exemple



```
import java.io.*;
import java.util.Calendar;
public class Personne implements Serializable {
    private static final long serialVersionUID = 1L;
    private String nom, prenom;
    private Date naissance;
    private transient int age;
```



```
private void readObject(ObjectInputStream obj) throws ClassNotFoundException, IOException
{
    obj.defaultReadObject();
    Calendar mtn = Calendar.getInstance();
    this.age = naissance.diff(
        new Date(mtn.get(java.util.Calendar.DAY_OF_MONTH),
            mtn.get(java.util.Calendar.MONTH) + 1,
            mtn.get(java.util.Calendar.YEAR)));
}
```

Introduction

Interfaces graphiques

Programmation événementielle

Les bases de JavaFX

L'architecture MVC (Model-View-Controller)

Lecture et écriture dans des fichiers

Accès aux bases de données

Aller plus loin en JavaFX

Java Database Connectivity

Data Access Object

Accès aux bases de données

BD & Java : JDBC

❖ JDBC = Java Database Connectivity

👉 API Java qui permet la connexion à une base de données (BD) et l'exécution de requêtes

❖ Pour qu'un programme Java puisse accéder à une BD, **c'est simple !**

👉 Il faut :

- ① Un SGBD (ex : Apache Derby 100% Java, MySQL, etc.)
- ② Un pilote (driver) JDBC pour ce SGBD particulier (inclus avec Derby, Connector/J for MySQL, etc.)
- ③ Du code Java qui utilise le package `java.sql`

Pilotes JDBC

❖ 4 types de pilotes JDBC :

- ☞ Type 1 : implémentent JDBC sous la forme d'un mapping vers une autre API d'accès aux données (par exemple ODBC) ; dépendent d'une bibliothèque native ; portabilité limitée ; ex. : JDBC-ODBC Bridge
- ☞ Type 2 : écrits partiellement en Java et en code natif ; utilisent une bibliothèque native spécifique à la BD ; portabilité limitée ; ex : Oracle OCI
- ☞ Type 3 : pur Java et communiquent avec un serveur via un protocole indépendant de la BD ; le serveur fait suivre la requête du client vers la source de données
- ☞ Type 4 : pur Java et implémentent le protocole réseau pour une source de données spécifiques ; le client se connecte directement à la source de données ; ex : MySQL Connector/J, JavaDB, etc.

❖ Installer un pilote JDBC = télécharger le pilote + l'ajouter au CLASSPATH

❖ Si le pilote n'est pas de type 4 ➡ souvent besoin d'installer une API côté client

Pilotes JDBC pour MySQL

- 📄 Télécharger le pilote depuis
<https://dev.mysql.com/downloads/connector/j/>
- 📄 Choisir le connecteur indépendant de la plateforme
(**platform independent**)

Connexion à une Base de Données

❖ Utilisation de DriverManager ou DataSource



```
Connection c = DriverManager.getConnection(String url);
```

ou



```
Connection c = DriverManager.getConnection(String url, String user, String  
password);
```

avec url qui précise le SGBD utilisé et son emplacement, par exemple :

🔗 jdbc:mysql://localhost:3306/myDB

🔗 jdbc:derby:testdb;create=true

la forme exacte de l'URL dépend du SGBD / pilote JDBC !

Exécution d'une requête SQL

ES Création d'un Statement à partir d'un objet Connection



Création d'un Statement simple

```
Statement stmt = connection.createStatement();
```



Création d'un PreparedStatement

```
PreparedStatement stmt = connection.prepareStatement("SELECT NOM,PRENOM FROM TABLE WHERE ANNEE=?");  
stmt.setInt(1,2022);
```

ES Exécution d'une requête



Pour des requêtes de type SELECT

```
ResultSet rs = stmt.executeQuery(query);
```



Pour des requêtes de type INSERT, DELETE, UPDATE

```
stmt.executeUpdate(query) ;
```

Accéder aux résultats d'une requête

- ❖ Le `ResultSet` contient un ensemble de résultats qu'on peut parcourir via un itérateur :



```
while (rs.next()) {  
    String nom=rs.getString("NOM");  
    int age=rs.getString("AGE");  
    System.out.println(nom+" "+age);  
}
```

Fermer la connexion

- ❖ Il faut veiller à appeler la méthode `close` quand on a fini d'utiliser les objets `Connection`, `Statement`, ou `ResultSet`
- ❖ On peut aussi utiliser un bloc `try-with-resources` :



```
try(Statement stmt = connection.createStatement()) {  
    ...  
}  
catch(SQLException e) { ... }
```

- ❖ qui déclare et fermera automatiquement la ressource `stmt` lorsque le bloc `try` se terminera (avec ou sans exception)

Modèle MVC (Model-View-Controller) – Rappel

- ❖ **Modèle** : Le modèle ne contient que les données de l'application, il ne contient aucune logique décrivant comment présenter ces données à un utilisateur
- ❖ **Vue** : La vue présente les données du modèle à l'utilisateur. La vue sait comment accéder aux données du modèle, mais elle ne sait pas ce que ces données signifient ni ce que l'utilisateur peut faire pour les manipuler
- ❖ **Contrôleur** : Le contrôleur se situe entre la vue et le modèle. Il écoute les événements déclenchés par la vue et exécute les réactions appropriées à ces événements. Dans la plupart des cas, la réaction consiste à appeler une méthode sur le modèle. Comme la vue et le modèle sont souvent reliés par un mécanisme de notification, le résultat de cette action est alors automatiquement reflété dans la vue

DAO (Data Access Object)

- ❖ Objets d'accès aux données
- ❖ Permettent la réalisation des opérations **CRUD** sur les objets
- ❖ Encapsule la logique de récupération, d'enregistrement et de mise à jour des données dans votre stockage de données (une base de données, un système de fichiers, etc.)
- ❖ Les objets d'accès aux données permettent de réaliser le mapping entre les objets du modèle et le stockage de données

Exemple – Classe POJO User



```
package model.data;
public class User {

    private String login;
    private String pwd;

    public User(String login, String pwd) {
        this.login = login;
        this.pwd = pwd;
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getPwd() {
        return pwd;
    }

    public void setMdp(String pwd) {
        this.pwd = pwd;
    }
}
```

Exemple – Classe DAO de base (abstraite)



```
package model.dao;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.List;
public abstract class DAO<T> {
    private static String driverClassName = "com.mysql.cj.jdbc.Driver";
    private static String url = "jdbc:mysql:///ihm_demo";
    private static String username = "abdelbadie";
    private static String password = "ihm$BUT1@2024";
    protected Connection getConnection() throws SQLException {
        // Charger la classe du pilote
        try {
            Class.forName(driverClassName);
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
            return null;
        }
        // Obtenir la connection
        return DriverManager.getConnection(url, username, password);
    }
    public abstract List<T> findAll();
    public abstract T findById(Long id);
    public abstract int update(T element);
    public abstract int delete(T element);
    public abstract int create(T element);
}
```


Exemple – Classe DAO correspondant à la classe User



```
package model.dao;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.*;  
import model.data.User;  
public class UserDao extends DAO<User> {
```

Exemple – create (C)



```
public int create(User user) {  
    String query = "INSERT INTO USER(LOGIN, PWD) VALUES ('" + user.getLogin() + "','" + user.getPwd() + "')";  
    try (Connection con = getConnection(); Statement st = con.createStatement()) {  
        return st.executeUpdate(query);  
    } catch (SQLException ex) {  
        ex.printStackTrace();  
        return -1;  
    }  
}
```

Exemple – update (U)



```
public int update(User user) {  
    String query = "UPDATE USER SET LOGIN='" + user.getLogin() + "', PWD='" + user.getPwd() + "' WHERE LOGIN='"  
        + user.getLogin() + "'";  
    try (Connection con = getConnection(); Statement st = con.createStatement()) {  
        return st.executeUpdate(query);  
    } catch (SQLException ex) {  
        ex.printStackTrace();  
        return -1;  
    }  
}
```

Exemple – delete (D)



```
public int delete(User user) {  
    String query = "DELETE FROM USER WHERE LOGIN='" + user.getLogin() + "'";  
    try (Connection con = getConnection(); Statement st = con.createStatement()) {  
        return st.executeUpdate(query);  
    } catch (SQLException ex) {  
        ex.printStackTrace();  
        return -1;  
    }  
}
```

Exemple – find all (R)



```
public List<User> findAll() {  
    List<User> users = new LinkedList<>();  
    try (Connection con = getConnection(); Statement st = con.createStatement()) {  
        ResultSet rs = st.executeQuery("SELECT * FROM USER");  
        while (rs.next()) {  
            String nom = rs.getString("LOGIN");  
            String pwd = rs.getString("PWD");  
            users.add(new User(nom, pwd));  
        }  
    } catch (SQLException ex) {  
        ex.printStackTrace();  
    }  
    return users;  
}
```

Exemple – find (R)



Problème d'injection

```
public User findByLoginPwd(String login, String pwd) {  
    try (Connection con = getConnection(); Statement st = con.createStatement()) {  
        ResultSet rs = st.executeQuery("SELECT * FROM USER WHERE LOGIN='" + login + "' AND PWD='" + pwd + "'");  
        while (rs.next()) {  
            String l = rs.getString("LOGIN");  
            String p = rs.getString("PWD");  
            return new User(l, p);  
        }  
    } catch (SQLException ex) { ex.printStackTrace(); }  
    return null;  
}
```



Solution

```
public User findByLoginPwd2(String login, String pwd) {  
    try (Connection con = getConnection();  
        PreparedStatement st = con.prepareStatement("SELECT * FROM USER WHERE LOGIN= ? AND PWD= ?")) {  
        st.setString(1, login); st.setString(2, pwd);  
        ResultSet rs = st.executeQuery();  
        while (rs.next()) {  
            String l = rs.getString("LOGIN");  
            String p = rs.getString("PWD");  
            return new User(l, p);  
        }  
    } catch (SQLException ex) { ex.printStackTrace(); }  
    return null;  
}
```

Compilation et exécution

❖ Dans un dossier de projet :

- ☞ Si le code source est dans un dossier **src**
- ☞ Si l'on veut placer les fichiers compilés dans un dossier **build**
- ☞ Si le pilote JDBC et le SDK Javafx (le dossier lib du SDK renommé en **javafx**) sont dans un dossier **lib**
- ☞ Si vous vous trouvez dans le **dossier du projet**
- ☞

```
javac -d build -sourcepath src --module-path lib/javafx --add-modules javafx.controls `find . -name '*.java'`
```
- ☞

```
java -cp build/:lib/mysql-connector-j-8.4.0.jar --module-path lib/javafx --add-modules javafx.controls <Classe_Principale>
```

Compilation et exécution avec alias (simplification)

❖ Ouvrir le fichier `~/.bashrc` (`nano ~/.bashrc`)

❖ Ajouter les deux lignes suivantes :

```
alias myjavac="javac -d build --module-path lib/javafx --add-modules  
javafx.controls -sourcepath src"  
alias myjava="java -cp build/:lib/mysql-connector-j-8.4.0.jar --module-path  
lib/javafx --add-modules javafx.controls"
```

❖ Après avoir quitter le fichier `.bashrc`, exécuter la commande : `source ~/.bashrc`

❖ Dans un dossier de projet :

```
Si le code source est dans un dossier src  
Si l'on veut placer les fichiers compilés dans un dossier build  
Si le pilote JDBC et le SDK Javafx (le dossier lib du SDK renommé en javafx)  
sont dans un dossier lib  
Si vous vous trouvez dans le dossier du projet  
myjavac `find . -name '*.java'`  
myjava <Classe_Principale>
```


Aller plus loin en JavaFX

Pas trop de changement...

- ❖ Gestionnaires de placement
- ❖ Composants : Text, Label, TextField, PasswordField, Button, ...
- ❖ Réaction aux événements :
 - 👉 implements ActionListener ➡ implements EventHandler<ActionEvent>
 - 👉 addActionListener ➡ setOnAction
 - 👉 ...

Mais si quand même !

❖ Utilisation d'un CSS dans le code JavaFX



```
scene.getStylesheets().add(getClass().getResource("/login.css").toExternalForm());
```

❖ Identification de composants



```
scenetitle.setId("welcome-text");  
actiontarget.setId("actiontarget");
```

❖ Tout le reste se fait dans le CSS...



login.css

```
.root {  
    -fx-background-image: url("background.jpg");  
    -fx-background-size: cover;  
    -fx-background-repeat: no-repeat;  
}  
.label {  
    -fx-font-size: 12px;  
    -fx-font-weight: bold;  
    -fx-text-fill: #333333;  
    -fx-effect: dropshadow(gaussian, rgba(255, 255, 255, 0.5), 0, 0, 0, 1);  
}  
#welcome-text {  
    -fx-font-size: 32px;  
    -fx-font-family: "Arial Black";  
    -fx-fill: #818181;  
    -fx-effect: innershadow(three-pass-box, rgba(0, 0, 0, 0.7), 6, 0.0, 0, 2);  
}  
#actiontarget {  
    -fx-fill: FIREBRICK;  
    -fx-font-weight: bold;  
    -fx-effect: dropshadow(gaussian, rgba(255, 255, 255, 0.5), 0, 0, 0, 1);  
}  
.button {  
    -fx-text-fill: white;  
    -fx-font-family: "Arial Narrow";  
    -fx-font-weight: bold;  
    -fx-background-color: linear-gradient(#61a2b1, #2A5058);  
    -fx-effect: dropshadow(three-pass-box, rgba(0, 0, 0, 0.6), 5, 0.0, 0, 1);  
}  
.button:hover {  
    -fx-background-color: linear-gradient(#2A5058, #61a2b1);  
}
```



Login.java


```
public class Login extends Application {
    public void start(Stage primaryStage) {
        primaryStage.setTitle("JavaFX Welcome");
        GridPane grid = new GridPane();
        grid.setAlignment(Pos.CENTER);
        grid.setHgap(10);
        grid.setVgap(10);
        grid.setPadding(new Insets(25, 25, 25, 25));
        Text scenetitle = new Text("Welcome");
        scenetitle.setId("welcome-text");
        grid.add(scenetitle, 0, 0, 2, 1);
        grid.add(new Label("User Name:"), 0, 1);
        grid.add(new TextField(), 1, 1);
        grid.add(new Label("Password:"), 0, 2);
        grid.add(new PasswordField(), 1, 2);
        Button btn = new Button("Sign in");
        HBox hbBtn = new HBox(10);
        hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
        hbBtn.getChildren().add(btn);
        grid.add(hbBtn, 1, 4);
        final Text actiontarget = new Text();
        grid.add(actiontarget, 0, 6);
        GridPane.setColumnSpan(actiontarget, 2);
        GridPane.setHalignment(actiontarget, RIGHT);
        actiontarget.setId("actiontarget");
        btn.setOnAction(e -> actiontarget.setText("Sign in button pressed"));
        Scene scene = new Scene(grid, 300, 275);
        scene.getStylesheets().add(getClass().getResource("/login.css").toExternalForm());
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



Isoler davantage la vue...

- ❖ La définition du contenu de l'IHM peut être faite en mode déclaratif dans un fichier XML

 https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/doc-files/introduction_to_fxml.html

```
BorderPane border = new BorderPane();  
Label toppanetext = new Label("Page Title");  
border.setTop(toppanetext);  
Label centerpanetext = new Label("Some data here");  
border.setCenter(centerpanetext);
```



```
<BorderPane>  
  <top>  
    <Label text="Page Title"/>  
  </top>  
  <center>  
    <Label text="Some data here"/>  
  </center>  
</BorderPane>
```

- ❖ Le rendu visuel est fait dans le CSS
- ❖ On conserve en Java uniquement le code de réaction aux évènements et la logique applicative



fxml_example.fxml

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
<GridPane fx:controller="FXMLExampleController" xmlns:fx="http://javafx.com/fxml"
alignment="center" hgap="10" vgap="10" styleClass="root">
  <padding>
    <Insets top="25" right="25" bottom="10" left="25"/>
  </padding>
  <Text id="welcome-text" text="Welcome"
    GridPane.columnIndex="0" GridPane.rowIndex="0"
    GridPane.columnSpan="2"/>
  <Label text="User Name:"
    GridPane.columnIndex="0" GridPane.rowIndex="1" />
    <TextField
      GridPane.columnIndex="1" GridPane.rowIndex="1" />
  <Label text="Password:"
    GridPane.columnIndex="0" GridPane.rowIndex="2" />
  <PasswordField fx:id="passwordField"
    GridPane.columnIndex="1" GridPane.rowIndex="2" />
  <HBox spacing="10" alignment="bottom_right"
    GridPane.columnIndex="1" GridPane.rowIndex="4">
    <Button text="Sign In"
      onAction="#handleSubmitButtonAction" />
  </HBox>
  <Text fx:id="actiontarget" GridPane.columnIndex="0"
    GridPane.columnSpan="2" GridPane.halignment="RIGHT"
    GridPane.rowIndex="6" />
  <stylesheets>
    <URL value="@login.css" />
  </stylesheets>
</GridPane>
```




login.css

```
.root {
    -fx-background-image: url("background.jpg");
    -fx-background-size: cover;
    -fx-background-repeat: no-repeat;
}

.label {
    -fx-font-size: 12px;
    -fx-font-weight: bold;
    -fx-text-fill: #333333;
    -fx-effect: dropshadow(gaussian, rgba(255, 255, 255, 0.5), 0, 0, 0, 1);
}

#welcome-text {
    -fx-font-size: 32px;
    -fx-font-family: "Arial Black";
    -fx-fill: #818181;
    -fx-effect: innershadow(three-pass-box, rgba(0, 0, 0, 0.7), 6, 0.0, 0, 2);
}

#actiontarget {
    -fx-fill: FIREBRICK;
    -fx-font-weight: bold;
    -fx-effect: dropshadow(gaussian, rgba(255, 255, 255, 0.5), 0, 0, 0, 1);
}

.button {
    -fx-text-fill: white;
    -fx-font-family: "Arial Narrow";
    -fx-font-weight: bold;
    -fx-background-color: linear-gradient(#61a2b1, #2A5058);
    -fx-effect: dropshadow(three-pass-box, rgba(0, 0, 0, 0.6), 5, 0.0, 0, 1);
}

.button:hover {
    -fx-background-color: linear-gradient(#2A5058, #61a2b1);
}
```



FXMLExample.java

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;


public class FXMLExample extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("/fxml_example.fxml"));
        stage.setTitle("FXML Welcome");
        stage.setScene(new Scene(root, 300, 275));
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(FXMLExample.class, args);
    }
}
```



FXMLExampleController.java

```
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.text.Text;
public class FXMLExampleController {
    @FXML
    private Text actiontarget;
    @FXML
    protected void handleSubmitButtonAction(ActionEvent event) {
        actiontarget.setText("Sign in button pressed");
    }
}
```

 The `@FXML` annotation is used to tag nonpublic controller member fields and handler methods for use by FXML markup

Pourquoi JavaFX ?

- ❖ Swing ne permet pas de développer des IHM modernes
- ❖ FXML permet de séparer la création de l'IHM d'une application JavaFX en la séparant complètement de la partie métier, et définir une IHM avec une approche déclarative en XML est plus simple
- ❖ **JavaFX Scene Builder** est un outil graphique qui simplifie la conception d'une IHM (génération automatique de code FXML)
- ❖ Support des CSS pour séparer l'apparence visuelle de la logique applicative
- ❖ Ajout simplifié de média (audio, vidéo, ...)
- ❖ Animations de meilleure qualité avec un double buffering
- ❖ Rendu visuel de contenu HTML
- ❖ et JavaFX/Gluon fonctionne aussi sur mobile (iOS/Android/Raspberry Pi)

Compilation et exécution

❖ Dans un dossier de projet :

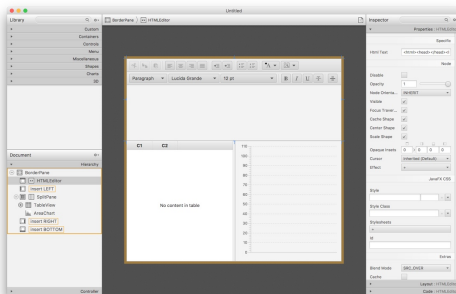
- ☞ Si le code source est dans un dossier `src`
- ☞ Si les fichiers `.fxml` sont dans un sous-dossier `resources` de `src`
- ☞ Si l'on veut placer les fichiers compilés dans un dossier `build`
- ☞ Si le SDK Javafx (le dossier lib du SDK renommé en `javafx`) sont dans un dossier `lib`
- ☞ Si vous vous trouvez dans le `dossier du projet`
- ☞

```
javac -d build -sourcepath src --module-path lib/javafx --add-modules  
javafx.controls,javafx.fxml `find . -name '*.java'`
```
- ☞

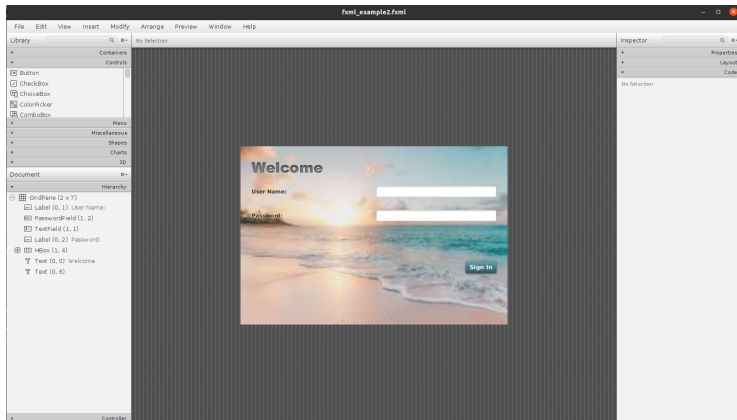
```
java -cp build:src/resources --module-path lib/javafx --add-modules  
javafx.controls,javafx.fxml <Classe_Principale>
```

JavaFX Scene Builder

<https://www.oracle.com/java/technologies/javafxscenebuilder-1x-archive-downloads.html>



- ☞ Outil graphique (WYSIWYG)
- ☞ Génération automatique du code FXML
- ☞ Support du CSS
- ☞ Mode prévisualisation
- ☞ Intégration dans l'IDE NetBeans



Introduction

Interfaces graphiques

Programmation événementielle

Les bases de JavaFX

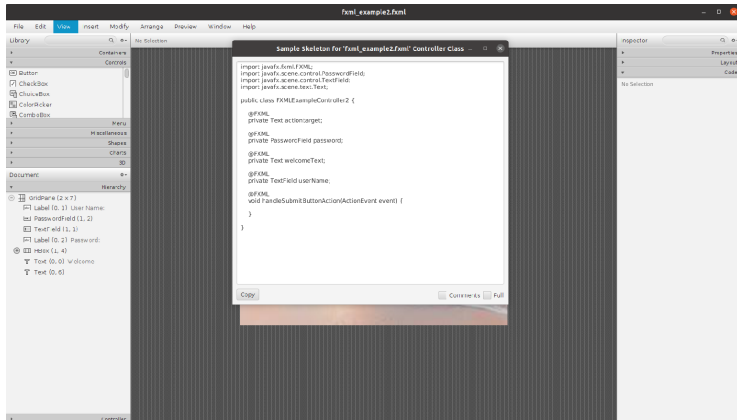
L'architecture MVC (Model-View-Controller)

Lecture et écriture dans des fichiers

Accès aux bases de données

Aller plus loin en JavaFX

De Swing à JavaFx
JavaFX Scene Builder
Résumé





fxml_example2.fxml

```
<GridPane alignment="CENTER" hgap="10.0" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" styleClass="root" styleSheets="@login_.css" vgap="10.0"
  xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8" fx:controller="FXMLExampleController2"
>
  <rowConstraints>
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
  </rowConstraints>
  <padding>
    <Insets bottom="10.0" left="25.0" right="25.0" top="25.0" />
  </padding>
  <children>
    <Label text="User Name:" GridPane.rowIndex="1" />
    <PasswordField fx:id="password" GridPane.columnIndex="1" GridPane.rowIndex="2" />
    <TextField fx:id="userName" GridPane.columnIndex="1" GridPane.rowIndex="1" />
    <Label text="Password:" GridPane.rowIndex="2" />
    <HBox alignment="BOTTOM_RIGHT" prefHeight="100.0" prefWidth="200.0" spacing="10.0" GridPane.columnIndex="1"
      GridPane.rowIndex="4">
      <children>
        <Button mnemonicParsing="false" onAction="#handleSubmitButtonAction" text="Sign In" />
      </children>
    </HBox>
    <Text fx:id="welcomeText" strokeType="OUTSIDE" strokeWidth="0.0" text="Welcome" GridPane.columnSpan="2" />
    <Text fx:id="actiontarget" strokeType="OUTSIDE" strokeWidth="0.0" GridPane.columnSpan="2" GridPane.halignment="RIGHT" GridPane.rowIndex="6" />
  </children>
</GridPane>
```



FXMLExampleController2.java

```
public class FXMLExampleController2 {
    @FXML
    private Text actiontarget;
    @FXML
    private Text welcomeText;
    @FXML
    private PasswordField password;
    @FXML
    private TextField userName;
    @FXML
    void handleSubmitButtonAction(ActionEvent event) {
        if(!userName.getText().equals("abdelbadie") || !password.getText().equals("123456")){
            actiontarget.setText("Wrong user name and/or password");
        }
        else{
            Scene scene=null;
            try {
                Parent root = FXMLLoader.load(getClass().getResource("/view/table.fxml"));
                scene= new Scene(root, 600, 500);
            } catch (IOException e) {
                e.printStackTrace();
            }
            Stage stage= (Stage) ((Node)event.getSource()).getScene().getWindow();
            stage.setScene(scene);
        }
    }
}
```



table.fxml

```

<GridPane alignment="TOP_RIGHT" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
  prefHeight="480.0" prefWidth="640.0" styleClass="root" styleSheets="@../style/login.css" xmlns="http://
  javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="tableviewexample.controller">
  <columnConstraints>
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
  </columnConstraints>
  <rowConstraints>
    <RowConstraints maxHeight="219.0" minHeight="10.0" prefHeight="32.0" valignment="CENTER" vgrow="ALWAYS" />
    <RowConstraints maxHeight="436.0" minHeight="10.0" prefHeight="418.0" valignment="CENTER" vgrow="ALWAYS" />
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
  </rowConstraints>
  <children>
    <HBox alignment="CENTER_LEFT" prefHeight="100.0" prefWidth="200.0" spacing="10.0" GridPane.rowIndex="2">
      <children>
        <TextField fx:id="firstTF" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
          Infinity" prefHeight="25.0" prefWidth="140.0" />
        <TextField fx:id="lastTF" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
          Infinity" prefHeight="25.0" prefWidth="140.0" />
        <TextField fx:id="mailTF" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
          Infinity" prefWidth="140.0" />
        <Button fx:id="add" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
          mnemonicParsing="false" onAction="#addPerson" prefWidth="140.0" text="Add" />
      </children>
    </HBox>
    <TableView fx:id="table" editable="true" prefHeight="200.0" prefWidth="200.0" GridPane.rowIndex="1">
      <columns>
        <TableColumn fx:id="firstcol" onEditCommit="#editFirst" prefWidth="150.0" text="First name" />
        <TableColumn fx:id="lastcol" minWidth="0.0" onEditCommit="#editLast" prefWidth="150.0" text="Name" />
        <TableColumn fx:id="mailcol" onEditCommit="#editMail" prefWidth="150.0" text="Mail" />
      </columns>
      <GridPane.margin>
        <Insets />
      </GridPane.margin>
    </TableView>
    <Button fx:id="signout" alignment="BOTTOM_LEFT" mnemonicParsing="false" onAction="#sign_out" text="Sign out"
      GridPane.halignment="RIGHT" />
  </children>
</GridPane>

```



TableController.java

```
public class TableController {
    @FXML
    private TextField lastTF;
    @FXML
    private Button add;
    @FXML
    private TextField mailTF;
    @FXML
    private TextField firstTF;
    @FXML
    private TableColumn<Person, String> firstcol;
    @FXML
    private TableColumn<Person, String> lastcol;
    @FXML
    private TableView<Person> table;
    @FXML
    private TableColumn<Person, String> mailcol;
    @FXML
    void addPerson(ActionEvent event) {
        if (!lastTF.getText().equals("") && !firstTF.getText().equals("") && !mailTF.getText().equals(""))
            table.getItems().add(new Person(lastTF.getText(), firstTF.getText(), mailTF.getText()));
        lastTF.setText("");
        firstTF.setText("");
        mailTF.setText("");
    }
}
```



TableController.java (suite)

```

@FXML
void sign_out(ActionEvent event) {
    Scene scene=null;
    try {
        Parent root = FXMLLoader.load(getClass().getResource("/view/fxml_example2.fxml"));
        scene= new Scene(root, 300, 275);
    } catch (IOException e) {
        e.printStackTrace();
    }
    Stage stage= (Stage) ((Node)event.getSource()).getScene().getWindow();
    stage.setScene(scene);
}
@FXML
void editFirst(CellEditEvent<Person, String> event) {
    table.getSelectionModel().getSelectedItem().setFirstname(event.getNewValue().toString());
}
@FXML
void editLast(CellEditEvent<Person, String> event) {
    table.getSelectionModel().getSelectedItem().setLastname(event.getNewValue().toString());
}
@FXML
void editMail(CellEditEvent<Person, String> event) {
    table.getSelectionModel().getSelectedItem().setMail(event.getNewValue().toString());
}
@FXML
void initialize() {
    firstcol.setCellValueFactory(new PropertyValueFactory<>("firstname"));
    lastcol.setCellValueFactory(new PropertyValueFactory<>("lastname"));
    mailcol.setCellValueFactory(new PropertyValueFactory<>("mail"));
    firstcol.setCellFactory(TextFieldTableCell.forTableColumn());
    lastcol.setCellFactory(TextFieldTableCell.forTableColumn());
    mailcol.setCellFactory(TextFieldTableCell.forTableColumn());
}
}

```

Résumé

- ❖ JavaFX plus riche que Swing : web, animation/3D, media, multi-touch, etc.
- ❖ JavaFX conserve le principe de prog événementielle et d'IHM basée sur des composants...
- ❖ mais la description de l'IHM peut être déportée dans un fichier FXML, son visuel dans un fichier CSS (et la logique reste en Java)
- ❖ Beaucoup de Component Swing ont un équivalent Control JavaFX (enlever le J, Dialog/Alert pour JOptionPane, ... attention : List/TableView fonctionne différemment de JList/JTable)
- ❖ En JavaFX, le gestionnaire de placement est directement intégré dans le conteneur (XXXLayout → XXXPane)
- ❖ JavaFX est maintenant open source (openjfx), de nombreux composants sont disponibles...

Introduction

Interfaces graphiques

Programmation événementielle

Les bases de JavaFX

L'architecture MVC (Model-View-Controller)

Lecture et écriture dans des fichiers

Accès aux bases de données

Aller plus loin en JavaFX

