Langage et algorithmique

Rodéric Moitié

ENSTA Bretagne

Sommaire

Sommaire

Pourquoi une IHM ?

Intérêt :

- Présenter des informations de manière synthétique

Pourquoi une IHM ?

Intérêt :

- Présenter des informations de manière synthétique
- Permettre d'interagir avec le programme

Pourquoi une IHM ?

Intérêt :

- Présenter des informations de manière synthétique
- Permettre d'interagir avec le programme

Caractéristiques :

- Code sans difficulté technique

Pourquoi une IHM ?

Intérêt :

- Présenter des informations de manière synthétique
- Permettre d'interagir avec le programme

Caractéristiques :

- Code sans difficulté technique
- Code souvent long et peu lisible

Pourquoi une IHM ?

Intérêt :

- Présenter des informations de manière synthétique
- Permettre d'interagir avec le programme

Caractéristiques :

- Code sans difficulté technique
- Code souvent long et peu lisible
- Code pas intéressant à écrire

Pourquoi une IHM ?

Intérêt :

- Présenter des informations de manière synthétique
- Permettre d'interagir avec le programme

Caractéristiques :

- Code sans difficulté technique
- Code souvent long et peu lisible
- Code pas intéressant à écrire
- Consulter la documentation de l'API

Panorama

Différentes hiérarchies de classes

- **java.awt**
- **javax.swing**

Panorama

Différentes hiérarchies de classes

- **java.awt**
- **javax.swing**

Différents types de composants :

- Fenêtres

Panorama

Différentes hiérarchies de classes

- **java.awt**
- **javax.swing**

Différents types de composants :

- Fenêtres
  - **JFrame**
  - **JDialog**
  - **JApplet**

Panorama

Différentes hiérarchies de classes

- **java.awt**
- **javax.swing**

Différents types de composants :

- Fenêtres
- Conteneurs

Panorama

Différentes hiérarchies de classes

- **java.awt**
- **javax.swing**

Différents types de composants :

- Fenêtres
- Conteneurs
  - **JPanel**
  - **JScrollPane**

Panorama

Différentes hiérarchies de classes

- **java.awt**
- **javax.swing**

Différents types de composants :

- Fenêtres
- Conteneurs
- Objets graphiques

Panorama

Différentes hiérarchies de classes

- **java.awt**
- **javax.swing**

Différents types de composants :

- Fenêtres
- Conteneurs
- Objets graphiques
  - **JButton**
  - **JLabel**

Sommaire

Création d'une fenêtre

- Créer une fenêtre : instancier la classe **JFrame**.

```
JFrame fenetre = new JFrame("Titre de la fenetre");
```

Création d'une fenêtre

Rien n'apparaît...

Création d'une fenêtre

- Créer une fenêtre : instancier la classe **JFrame**.
- Rendre la fenêtre visible

```
JFrame fenetre = new JFrame("Titre de la fenetre");
fenetre.setVisible(true);
```

Création d'une fenêtre

FIGURE: Fenêtre version 1

Création d'une fenêtre

- Créer une fenêtre : instancier la classe **JFrame**.
- Rendre la fenêtre visible
- Redimensionner la fenêtre

```
JFrame fenetre = new JFrame("Titre de la fenetre");
fenetre.setVisible(true);
fenetre.setSize(320,200);
```

Création d'une fenêtre

FIGURE: Fenêtre version 2

Création d'une fenêtre

- Créer une fenêtre : instancier la classe **JFrame**.
- Rendre la fenêtre visible
- Redimensionner la fenêtre
- Changer le **look and feel**

```
JFrame.setDefaultLookAndFeelDecorated(true);
JFrame fenetre = new JFrame("Titre de la fenetre");
fenetre.setVisible(true);
fenetre.setSize(320,200);
```

Création d'une fenêtre

FIGURE: Fenêtre version 3

Composition d'une fenêtre

Objet de type **JFrame**

- Barre de titre
- Conteneur (**getContentPane()**)
- Propriétés (**get/set**)
  - taille
  - visibilité

Sommaire

Utilisation des gestionnaires d'agencement (Layout)

Différents types d'agencement :

- Par défaut dans un fenêtre : **BorderLayout**

Utilisation des gestionnaires d'agencement (Layout)

FIGURE: BorderLayout

Utilisation des gestionnaires d'agencement (Layout)

```
JFrame fenetre = new JFrame("Titre de la fenetre");
Container c = fenetre.getContentPane();
c.add(new JButton("Nord"), BorderLayout.NORTH);
```

Utilisation des gestionnaires d'agencement (Layout)

Différents types d'agencement :

- Par défaut dans un fenêtre : **BorderLayout**
- Possibilité de choisir un **GridLayout**

Utilisation des gestionnaires d'agencement (Layout)

FIGURE: GridLayout

Utilisation des gestionnaires d'agencement (Layout)

```
JFrame fenetre = new JFrame("Titre de la fenetre");
Container c = fenetre.getContentPane();
c.setLayout(new GridLayout(3,2));
c.add(new JButton("Un"));
```

Utilisation des gestionnaires d'agencement (Layout)

Différents types d'agencement :

- Par défaut dans un fenêtre : **BorderLayout**
- Possibilité de choisir un **GridLayout**
- ou un **FlowLayout**

Utilisation des gestionnaires d'agencement (Layout)

FIGURE: FlowLayout

Utilisation des gestionnaires d'agencement (Layout)

```
JFrame fenetre = new JFrame("Titre de la fenetre");
Container c = fenetre.getContentPane();
c.setLayout(new FlowLayout());
c.add(new JButton("Un"));
```

Utilisation des gestionnaires d'agencement (Layout)

Différents types d'agencement :

- Par défaut dans un fenêtre : **BorderLayout**
- Possibilité de choisir un **GridLayout**
- ou un **FlowLayout**
- ou encore un **BoxLayout**

Utilisation des gestionnaires d'agencement (Layout)

FIGURE: BoxLayout

Utilisation des gestionnaires d'agencement (Layout)

```
JFrame fenetre = new JFrame("Titre de la fenetre");
Container c = fenetre.getContentPane();
c.setLayout(new BoxLayout(c, BoxLayout.Y_AXIS));
c.add(new JButton("Un"));
```

Sommaire

Définition d'une classe

- Classe héritant de **JFrame**
- Variables d'instance : divers composants de l'interface
- Constructeur : construction de l'interface

Exemple :

```
public class Visu extends JFrame {
  private JButton boutonOK;
  public Visu() {
    boutonOK = new JButton();
    boutonOK.setText("OK");
    this.getContentPane().add(
          boutonOK,BorderLayout.SOUTH);
  }
```

Sommaire

Sommaire

Principe

Principe général :

- Ajouter un “écouteur” à l'objet
- Écouteur : classe implantant un **Listener** ou héritant d'un **Adapter**
- interface **ActionListener** : redéfinir **actionPerformed**
- interface **MouseListener** ou classe **MouseAdapter** : redéfinir **mouseClicked, mouseEntered, mouseExited, mousePressed** et **mouseReleased**
- interface **KeyListener** ou classe **KeyAdapter** : redéfinir **keyPressed**, **keyReleased** et **keyTyped**

Exemple

Écouteur :

```
class unEcouteur implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("clic");
  }
}
```

Utilisation :

```
JButton monBouton = new JButton();
monBouton.addActionListener(new unEcouteur());
```

Critique de la méthode

Avantage :

- Principe simple

Critique de la méthode

Avantage :

- Principe simple

Inconvénient :

- Obligation d'écrire une classe par écouteur !
- Problème de partage des données

Critique de la méthode

Avantage :

- Principe simple

Inconvénient :

- Obligation d'écrire une classe par écouteur !
- Problème de partage des données

Solution :

- Utiliser des classes internes

Sommaire

Principe des classes internes

- Principe : définir une classe à l'intérieur d'une autre classe

$\Rightarrow$ partage des données

- La classe interne peut utiliser les attributs privés de la classe externe.
- La classe interne utilise les données de la classe externe comme les siennes.
- Un objet interne est lié à un objet externe.

Exemple de classe interne

```
public class MaClasseExterne {
  private int x;
  private MaClasseInterne y =
              new MaClasseInterne();
  public void action() {
    y.go();
  }

  class MaClasseInterne {
    void go() {
      x=42;
    }
  }
}
```

Gestion d'évènements grâce à des classes internes

Application des classes internes :

- gestion des évènements
- une classe interne par écouteur

Avantage :

- plus de problème de partage de données

Inconvénient :

- toujours une classe par écouteur

Gestion d'évènement et classe interne

```
public class Visu extends JFrame {
  private JLabel monTexte;
  private JButton boutonGo;
  public Visu() {
    ...
    boutonGo.addActionListener(new EcouteurGo());
  }

  class EcouteurGo implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      monTexte.setText("Clic");
    }
  }
}
```

Autre possibilité

- possibilité : créer des objets dans créer de classe
- même principe qu'avec les classes internes
- possibilité d'utiliser les variables de la classe externe

```
boutonFin.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent e) {
    System.exit(0);
  }
});
```

Évènements liés à la fenêtre

Il existe des évènements liés à la fermeture d'une fenêtre :

- quitter l'application : **EXIT_ON_CLOSE**

```
this.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);
```

Évènements liés à la fenêtre

Il existe des évènements liés à la fermeture d'une fenêtre :

- quitter l'application : **EXIT_ON_CLOSE**
- ne rien faire : **DO_NOTHING_ON_CLOSE**

```
this.setDefaultCloseOperation(
            WindowConstants.DO_NOTHING_ON_CLOSE);
```

Évènements liés à la fenêtre

Il existe des évènements liés à la fermeture d'une fenêtre :

- quitter l'application : **EXIT_ON_CLOSE**
- ne rien faire : **DO_NOTHING_ON_CLOSE**
- cacher la fenêtre : **HIDE_ON_CLOSE**

```
this.setDefaultCloseOperation(
            WindowConstants.HIDE_ON_CLOSE);
```

Évènements liés à la fenêtre

Il existe des évènements liés à la fermeture d'une fenêtre :

- quitter l'application : **EXIT_ON_CLOSE**
- ne rien faire : **DO_NOTHING_ON_CLOSE**
- cacher la fenêtre : **HIDE_ON_CLOSE**
- cacher et libérer la fenêtre : **DISPOSE_ON_CLOSE**

```
this.setDefaultCloseOperation(
            WindowConstants.DISPOSE_ON_CLOSE);
```

Évènements liés à la fenêtre

Autre possibilité : ajouter un **WindowAdapter** ou un **WindowListener**

```
this.addWindowListener(new EcouteurWindow());
```

Évènements liés à la fenêtre

Autre possibilité : ajouter un **WindowAdapter** ou un **WindowListener**

```
this.addWindowListener(new EcouteurWindow());

class EcouteurWindow extends WindowAdapter {
    public void windowClosing (WindowEvent e) ...
    public void windowActivated (WindowEvent e) ...
    public void windowDeactivated (WindowEvent e) ...
    ...
}
```

Sommaire

Sommaire

Généralités

- API **awt** : méthodes de dessin en 2D
- Zone de dessin : conteneur. Par ex. **JFrame** ou **JPanel**
- Objet permettant de dessiner : **Graphics2D**
- Possibilité de définir une classe (interne ou non) héritant de **JPanel**

Graphics

Quelques méthodes de la classe **Graphics** :

clearRect : efface une partie de l'image

Graphics

Quelques méthodes de la classe **Graphics** :

clearRect : efface une partie de l'image

drawRect, drawOval, drawLine : affiche une figure

Graphics

Quelques méthodes de la classe **Graphics** :

clearRect	: efface une partie de l'image
drawRect, drawOval, drawLine	: affiche une figure
drawImage	: dessine une image

Graphics

Quelques méthodes de la classe **Graphics** :

clearRect	: efface une partie de l'image
drawRect, drawOval, drawLine	: affiche une figure
drawImage	: dessine une image
drawString	: affiche un texte

Graphics

Quelques méthodes de la classe **Graphics** :

clearRect	: efface une partie de l'image
drawRect, drawOval, drawLine	: affiche une figure
drawImage	: dessine une image
drawString	: affiche un texte
fillRect, fillOval, fillArc	: dessine et colorie une figure

Graphics

Quelques méthodes de la classe **Graphics** :

clearRect : efface une partie de l'image

drawRect, drawOval, drawLine : affiche une figure

drawImage : dessine une image

drawString : affiche un texte

fillRect, fillOval, fillArc : dessine et colorie une figure

setColor : modifie la couleur du pinceau

Graphics

Quelques méthodes de la classe **Graphics** :

clearRect : efface une partie de l'image

drawRect, drawOval, drawLine : affiche une figure

drawImage : dessine une image

drawString : affiche un texte

fillRect, fillOval, fillArc : dessine et colorie une figure

setColor : modifie la couleur du pinceau

Graphics

Quelques méthodes de la classe **Graphics** :

clearRect : efface une partie de l'image

drawRect, drawOval, drawLine : affiche une figure

drawImage : dessine une image

drawString : affiche un texte

fillRect, fillOval, fillArc : dessine et colorie une figure

setColor : modifie la couleur du pinceau

Voir également **Graphics2D**.

Sommaire

repaint

Rafraîchissement d'une fenêtre graphique :

- Appeler la méthode **paint(Graphics g)**
- Cette méthode appelle **paintComponent(Graphics g)**

Pour chaque composant (particulièrement les conteneurs)

- Possibilité de redéfinir **paint/paintComponent**

repaint

Rafraîchissement d'une fenêtre graphique :

- Appeler la méthode **paint(Graphics g)**
- Cette méthode appelle **paintComponent(Graphics g)**

Pour chaque composant (particulièrement les conteneurs)

- Possibilité de redéfinir **paint/paintComponent**

Remarque

Ne jamais appeler explicitement **paint/paintComponent**. Utiliser **repaint**.

Exemple d'animation

```
class PanneauDessin extends JPanel {
   private Color c1;
   private Color c2;
   public PanneauDessin() {
      c1 = new Color(alea.nextInt(256),
            alea.nextInt(256),alea.nextInt(256));
      c2 = new Color(alea.nextInt(256),
            alea.nextInt(256),alea.nextInt(256));
   }
}
```

Exemple d'animation

```
class PanneauDessin extends JPanel {
   public void paintComponent(Graphics g) {
      super.paintComponent(g);
```

Exemple d'animation

```
class PanneauDessin extends JPanel {
   public void paintComponent(Graphics g) {
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;
```

Exemple d'animation

```
class PanneauDessin extends JPanel {
   public void paintComponent(Graphics g) {
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;
      GradientPaint grad =
              new GradientPaint(x,y,c1,x+40,y+40,c2);
      g2.setPaint(grad);
```

Exemple d'animation

```
class PanneauDessin extends JPanel {
   public void paintComponent(Graphics g) {
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;
      GradientPaint grad =
             new GradientPaint(x,y,c1,x+40,y+40,c2);
      g2.setPaint(grad);
      g2.fillOval(x,y,40,40);
   }
}
```

Sommaire

JLabel

- **JLabel** : texte non interactif
- Possibilité d'ajouter des images

```
Container c = fenetre.getContentPane();
ImageIcon icon = new ImageIcon("icon.png");
c.setLayout(new GridLayout(3,1));
c.add(new JLabel("JLabel 1",JLabel.RIGHT));
c.add(new JLabel("JLabel 2",icon,JLabel.CENTER));
c.add(new JLabel("JLabel 3"));
```

JLabel

- **JLabel** : texte non interactif
- Possibilité d'ajouter des images

Champs de saisie

- **JTextField** : texte sur une ligne
- **JTextArea** : texte sur plusieurs lignes

```
Container c = fenetre.getContentPane();
c.add(new JTextField("JTF"),BorderLayout.NORTH);
c.add(new JTextArea("JTA"),BorderLayout.CENTER);
```

Champs de saisie

- **JTextField** : texte sur une ligne
- **JTextArea** : texte sur plusieurs lignes

Champs de saisie

- **JTextField** : texte sur une ligne
- **JTextArea** : texte sur plusieurs lignes
- **JScrollPane** : barre de défilement

```
Container c = fenetre.getContentPane();
c.add(new JTextField("JTF"),BorderLayout.NORTH);
JScrollPane sp=new JScrollPane(new JTextArea("JTA"));
c.add(sp,BorderLayout.CENTER);
```

Champs de saisie

- **JTextField** : texte sur une ligne
- **JTextArea** : texte sur plusieurs lignes
- **JScrollPane** : barre de défilement

Boutons

- **JButton**
- **JCheckBox**
- **JRadioButton**

```
c.add(new JButton("JButton"),BorderLayout.NORTH);
JPanel p1 = new JPanel();
p1.add(new JCheckBox("Selectionne 2",true));
JPanel p2 = new JPanel();
JRadioButton rb1=new JRadioButton("Select",true);
p2.add(rb1);
```

Boutons

- **JButton**
- **JCheckBox**
- **JRadioButton**

Boutons

- **JButton**
- **JCheckBox**
- **JRadioButton**
- **ButtonGroup** : regroupe les boutons

```
ButtonGroup gr = new ButtonGroup();
JRadioButton rb1=new JRadioButton("Selectionne",true);
gr.add(rb1);
```

Listes

- **JList**
- **JComboBox**

```
String entrees[]="Premiere", "Deuxieme", "Troisieme";
JComboBox comb = new JComboBox(entrees);
comb.setSelectedIndex(1);
c.add(comb,BorderLayout.NORTH);
comb.addItem("Derniere");
JList lst = new JList(entrees2);
c.add(lst,BorderLayout.WEST);
```

Listes

- **JList**
- **JComboBox**

Sommaire

Important

Séparer au maximum le programme de l'interface graphique.

Important

Séparer au maximum le programme de l'interface graphique. Séparer ce que fait le programme de la manière dont il le représente.

Important

Séparer au maximum le programme de l'interface graphique. Séparer ce que fait le programme de la manière dont il le représente.

- Exemple : projets d'informatique.

Important

Séparer au maximum le programme de l'interface graphique. Séparer ce que fait le programme de la manière dont il le représente.

- Exemple : projets d'informatique.

$\Rightarrow$ Question : comment lier le programme et l'interface **a posteriori** ?

Lien programme / interface graphique

Programme : **Simulation**

- Interface : variable de type **Simulation**

```
public class Visualisation extends JFrame {
    ...
    private Simulation leProgramme;
```

Lien programme / interface graphique

Programme : **Simulation**

- Interface : variable de type **Simulation**
- Méthodes des boutons : appeler des méthodes de **Simulation**

```
public class Visualisation extends JFrame {
     ...
     public void actionPerformed(ActionEvent e) {
          leProgramme.simule();
          repaint();
     }
```

Remarques sur la création d'IHM

- Travail fastidieux

Remarques sur la création d'IHM

- Travail fastidieux
- Beaucoup de traitements systématiques

Remarques sur la création d'IHM

- Travail fastidieux
- Beaucoup de traitements systématiques
- Éditeur **WYSIWYG** très utile

Remarques sur la création d'IHM

- Travail fastidieux
- Beaucoup de traitements systématiques
- Éditeur **WYSIWYG** très utile

$\Rightarrow$ solution : utiliser un IDE

Remarques sur la création d'IHM

- Travail fastidieux
- Beaucoup de traitements systématiques
- Éditeur **WYSIWYG** très utile

$\Rightarrow$ solution : utiliser un IDE

- Exemples :

Remarques sur la création d'IHM

- Travail fastidieux
- Beaucoup de traitements systématiques
- Éditeur **WYSIWYG** très utile

$\Rightarrow$ solution : utiliser un IDE

- Exemples :
  - **eclipse** avec le **plugin jigloo** (`http://www.cloudgarden.com/jigloo/`)

Remarques sur la création d'IHM

- Travail fastidieux
- Beaucoup de traitements systématiques
- Éditeur **WYSIWYG** très utile

$\Rightarrow$ solution : utiliser un IDE

- Exemples :
  - **eclipse** avec le **plugin jigloo** (`http://www.cloudgarden.com/jigloo/`)
  - **eclipse** avec le **plugin VE**

Remarques sur la création d'IHM

- Travail fastidieux
- Beaucoup de traitements systématiques
- Éditeur **WYSIWYG** très utile

$\Rightarrow$ solution : utiliser un IDE

- Exemples :
  - **eclipse** avec le **plugin jigloo** (`http://www.cloudgarden.com/jigloo/`)
  - **eclipse** avec le **plugin VE**
  - **NetBeans** (`http://netbeans.org/`)