

Swing

Interfaces utilisateur graphiques





Swing

Course objectives

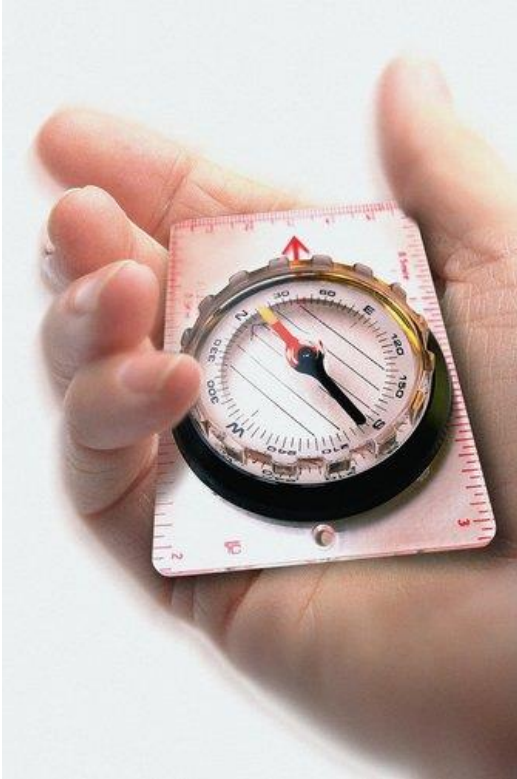
En complétant ce cours, vous serez en mesure de :

- Construire vos propres interfaces graphiques
- Savoir manipuler les principaux composants comme TextField, Buttons...
- Gérer les événements
- Peindre des surfaces personnalisées
- Utiliser l'aspect et la convivialité



Swing

Course plan



- Introduction
- Conteneurs
- Composants communs
- Gestionnaire de mise en page
- Événements
- Menus
- Référencement
- Peinture
- Internationalisation

Swing

INTRODUCTION



Histoire de Swing

- Swing fait partie de la classe Java Foundations :
 - Principales bibliothèques produisant des affichages et des interfaces graphiques utilisateur (GUI)
 - Drag'n'Drop, I18N, Java 2D, Accessibility, ...
- Fonctionne au-dessus de l'Abstract Window Toolkit (AWT) :
 - Gère les composants natifs, les problèmes de portabilité,
 - Toujours utilisé aux niveaux inférieurs de Swing



WYSIWYG Tools

- **What You See Is What You Get**
- Permet de construire des interfaces graphiques en faisant glisser des composants.
- Les environnements de développement intégré (IDE) fournissent leurs propres plugins :
 - Celui d'Eclipse est WindowBuilder :
 - <http://www.eclipse.org/windowbuilder/>
 - Celui de NetBeans est Matisse : Intégré dans l'IDE
 - Celui d'IntelliJ est Swing GUI Designer : Intégré dans l'IDE



Architecture

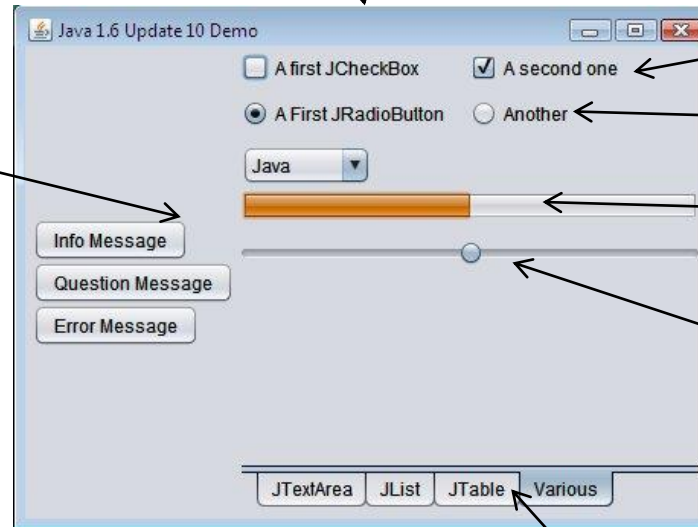
- Introduction

**Window, main container
(JFrame)**

JComboBox

Buttons
(JButton)

**Window's container
(JPanel)**



JCheckBox

JRadioButton

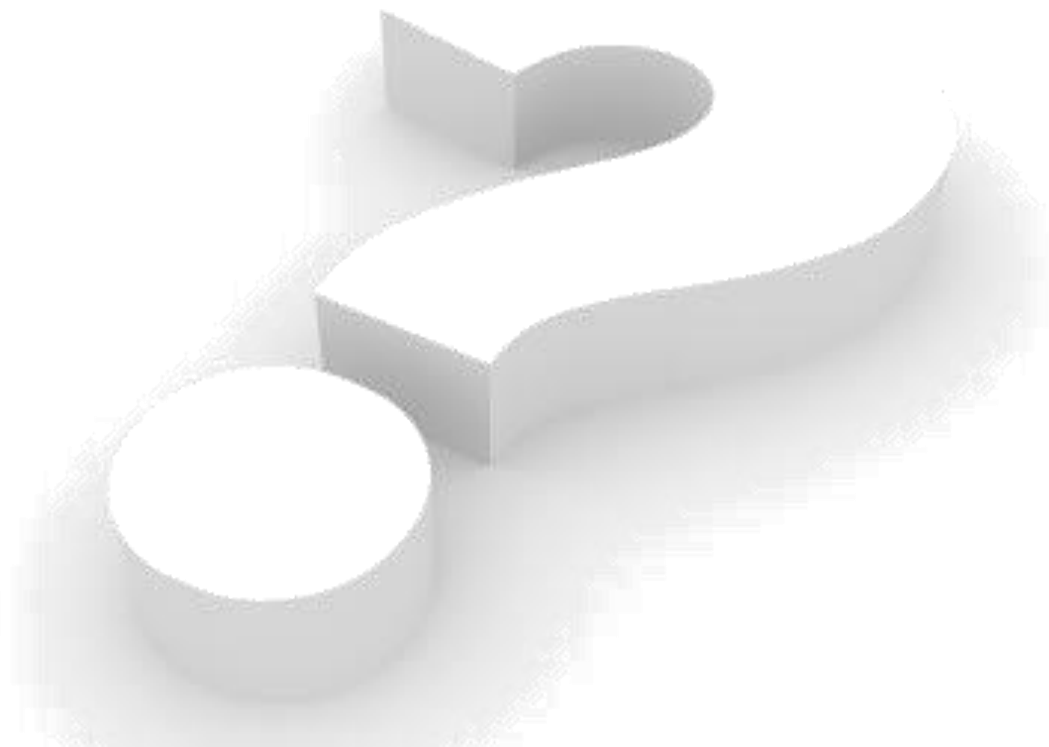
JProgressBar

JSlider

JTabbedPane



Questions ?



Swing

CONTAINERS



JFrame: description

- Un conteneur de haut niveau.
- Représente une fenêtre pour votre logiciel
- Peut avoir :
 - Une taille.
 - Un titre.
 - Une opération de fermeture.
 - Des composants (comme des boutons, des champs de formulaire, ...).
 - ...



JFrame: methods

- `setSize(int largeur, int hauteur)` ou `setSize(Dimension dim)` :
 - Définit la taille de la fenêtre.
- `setTitle(String titre)` :
 - Définit le titre de la fenêtre.
- `setVisible(boolean visible)` :
 - Détermine si la fenêtre est affichée ou cachée.
- `setDefaultCloseOperation(int opération)` :
 - Définit l'action à effectuer lorsque la fenêtre est fermée. Par défaut, aucune action n'est effectuée.
- `setResizable(boolean redimensionnable)` :
 - Permet de rendre la fenêtre redimensionnable ou non.



JFrame: example

```
public class MyFrame extends JFrame {  
    ...  
    public MyFrame() {  
        this.setSize(300, 200);  
  
        // define the default operation when  
        // the frame is closed  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        // define the title of the frame  
        this.setTitle("My wonderful frame");  
    }  
    ...  
}
```



Containers

JPanel: description

- Permet d'agréger des composants.
- Peut être utilisé comme conteneur principal d'une fenêtre.
- Possède une disposition.
 - Nous verrons cela plus tard...

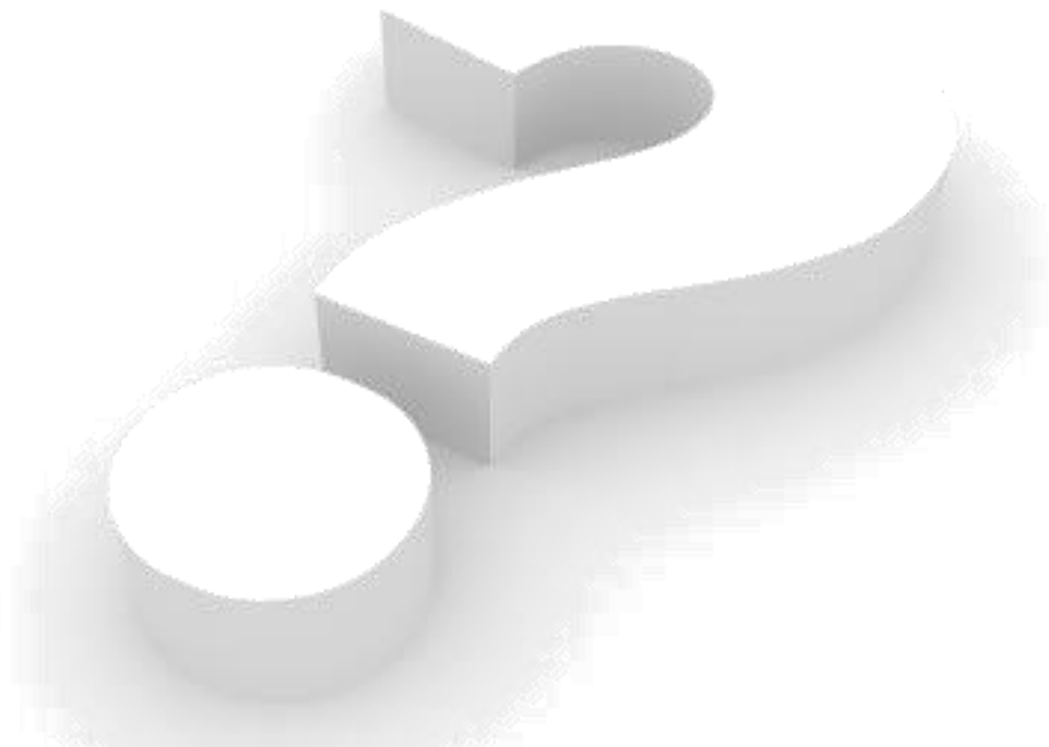


JPanel: example

```
public class MyFrame extends JFrame {  
    ...  
    public MyFrame() {  
        ...  
        JPanel panel = new JPanel();  
        panel.add(new JButton("My button"));  
  
        // add more elements if you like  
  
        this.setContentPane(panel);  
    }  
    ...  
}
```



Questions ?



Swing

COMPOSANTS COURANTS



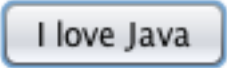
JLabel

- Affiche un texte, une image ou les deux.
 - Pour afficher un texte :
 - `setText(String texte)`
 - Pour afficher une image :
 - `setIcon(Icon icône)`
- Constructeurs courants :
 - `JLabel()`
 - `JLabel(String texte)`
 - `JLabel(Icon icône)`



JButton

- Un bouton classique.
- Vous devez définir l'action lorsque vous cliquez dessus :
 - Méthode `addActionListener(ActionListener événement)`
 - (Soyez patient, nous le verrons plus tard 😊).
- Constructeurs courants :
 - `JButton()`
 - `JButton(String texte)`
 - `JButton(Icon image)`

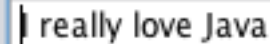


I love Java



JTextField

- Un champ de texte pour saisir des valeurs.
- Possède un nombre de colonnes (= une sorte de largeur).
- Constructeurs courants :
 - `JTextField(int colonnes)`
 - `JTextField(String texteParDéfaut)`
 - `JTextField(String texteParDéfaut, int colonnes)`
- Méthodes utiles :
 - `void setText(String texte)`
 - Définit le texte dans le champ de texte.
 - `String getText()`
 - Obtient le texte saisi dans le champ de texte.

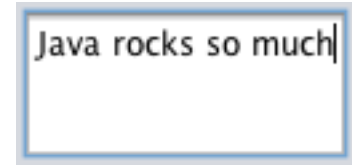


| really love Java



JTextArea

- Une zone de texte avec un nombre de colonnes et de lignes.
- Constructeurs courants :
 - `JTextArea(int lignes, int colonnes)`
 - `JTextArea(String texteParDéfaut)`
 - `JTextArea(String texteParDéfaut, int lignes, int colonnes)`
- Méthodes utiles :
 - `String getText()` / `void setText(String texte)`
 - Obtient ou définit le texte dans la zone de texte.
 - `int getRows()` / `void setRows(int nbDeLignes)`
 - `int getColumns()` / `void setColumns(int nbDeCol)`





JCheckBox

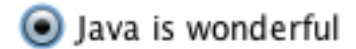
- Représente une case à cocher.
- Constructeurs courants :
 - JCheckBox(String texte)
 - JCheckBox(String texte, boolean sélectionné)
- Méthodes utiles :
 - boolean isSelected() :
 - Renvoie true si la case à cocher est cochée.
 - void setSelected(boolean sélectionné) :
 - Définit si la case à cocher doit être sélectionnée.
- String getText() / void setText(String libellé) :
 - Obtient ou définit le libellé de la case à cocher.

☒ Java is soooo cool



JRadioButton

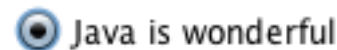
- Un bouton radio.
- Constructeurs courants :
 - JRadioButton(String texte)
 - JRadioButton(String texte, boolean sélectionné)
- Méthodes utiles :
 - boolean isSelected() / void setSelected(boolean) :
 - Obtient ou définit si le bouton radio est sélectionné.





ButtonGroup

- Permet de "regrouper" des boutons radio :
 - Afin de permettre une sélection exclusive.
- Méthodes utiles :
 - void add(AbstractButton btn) :
 - Ajoute un bouton (également un bouton radio) au groupe.
 - void remove(AbstractButton btn) :
 - Supprime un bouton du groupe.





JProgressBar

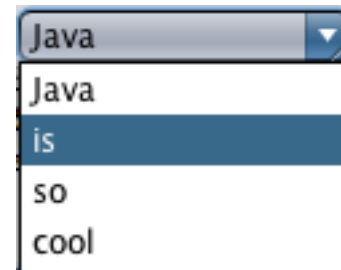
- Une barre de progression.
- Constructeur courant :
 - JProgressBar(int valeurMinimale, int valeurMaximale).
- Méthodes utiles :
 - int getValue() / void setValue() :
 - Obtient ou définit la valeur de progression.
 - void setString(String texte) :
 - Un texte écrit sur la barre.
 - void setStringPainted(boolean peint) :
 - Indique si le texte est écrit sur la barre. false par défaut.





JComboBox

- Une liste déroulante.
- Constructeurs :
 - JComboBox(Object[] éléments)
 - JComboBox(Vector<?> éléments)
- Méthodes utiles :
 - int getSelectedIndex()
 - Object getSelectedItem()
- **Remarque** : le texte affiché dans le **JComboBox** est le résultat de toString() des éléments.



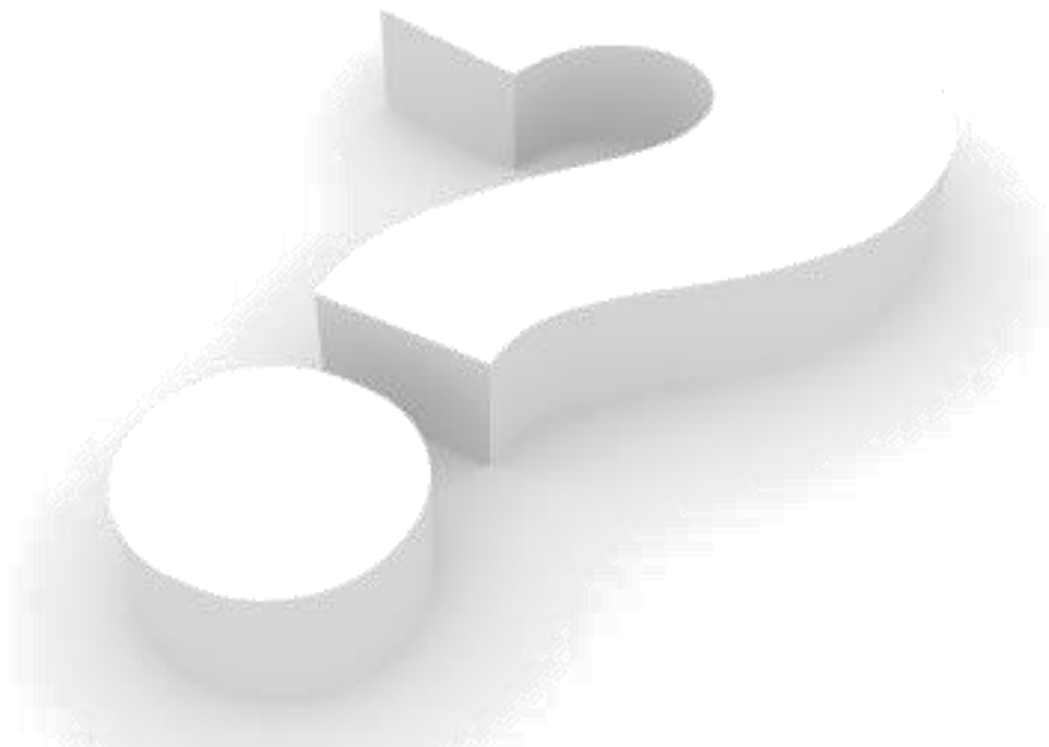


L'ActionCommand

- Disponible sur de nombreux composants.
- Est une chaîne de caractères.
- N'a rien à voir avec l'action que vous effectuez sur un composant :
 - Clic, focus, sélection...
- Peut être utilisé comme identifiant des composants.
- Définissez-le avec :
 - `void setActionCommand(String commande).`
- Obtenez-le avec :
 - `String getActionCommand().`



Questions ?

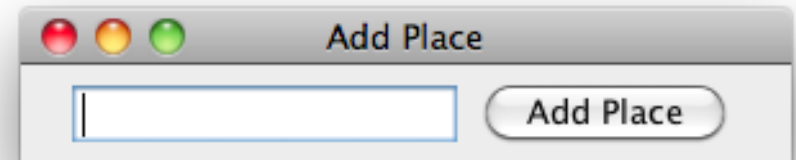




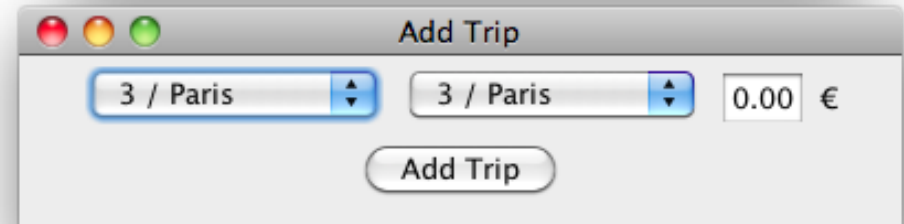
Exercice (1/2)

- Mettre à jour le projet JavaParadise.
- Créez une nouvelle classe nommée GuiLauncher :
 - Nous l'utiliserons pour lancer la version graphique de notre application.
- Créez un package `com.supinfo.javaparadise.gui`.
 - Créez deux classes étendant `JFrame` :

- `AddPlaceFrame` :



- `AddTripFrame` :





Exercice (2/2)

- Vous rencontrez un problème ? Vous avez un bouton très grand à l'intérieur de vos fenêtres ?
 - Deux solutions :
 - Essayez de changer le contenu de votre fenêtre par une nouvelle instance de JPanel.
 - Essayez de changer la disposition de votre fenêtre en utilisant un FlowLayout.
- Pourquoi ce problème se produit-il ? Et qu'est-ce qu'une disposition ?
 - Soyez patient, c'est le sujet du prochain chapitre ! 😊

Swing

LAYOUT MANAGER



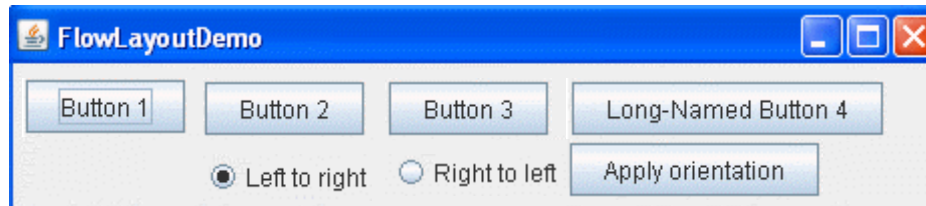
Definition

- Gérez le placement des composants.
 - Dans les conteneurs :
 - Classes étendant la classe Container telles que JPanel.
- Il en existe plusieurs :
 - Choisissez celle qui convient le mieux à vos besoins !
- Configurez-la :
 - Pour tous les conteneurs :
 - `setLayout(LayoutManager layout).`
 - Pour JPanel uniquement :
 - `JPanel(LayoutManager layout).`



FlowLayout

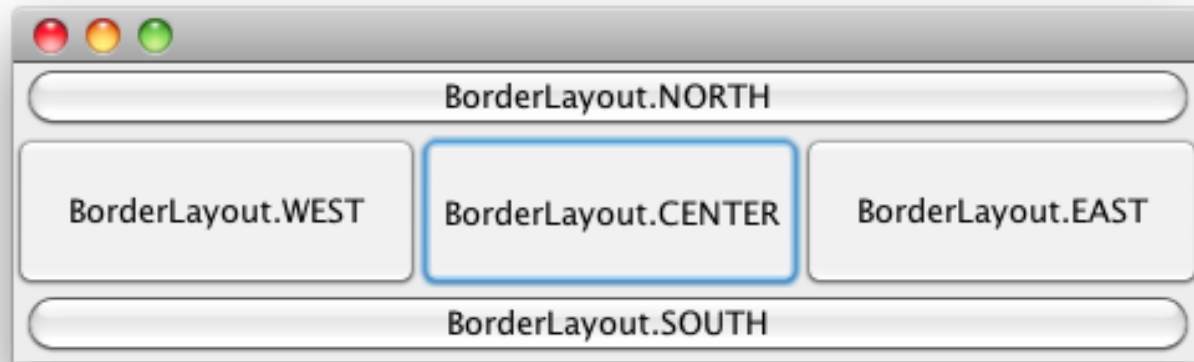
- Le layout par défaut d'un JPanel est le FlowLayout. Voici comment il dispose les composants :
 - Les composants sont placés de gauche à droite sur la même "line".
 - S'il n'y a pas suffisamment d'espace sur la ligne actuelle pour ajouter un composant, un nouveau "line" est créé automatiquement pour placer le composant suivant.
 - Chaque ligne est centrée horizontalement par rapport au conteneur.
 - Cela signifie que si les lignes ne sont pas pleines, les composants seront centrés horizontalement dans le JPanel.





BorderLayout

- Séparé en 5 régions.
- La région par défaut est BorderLayout.CENTER.
- C'est la disposition par défaut du contenu de la fenêtre JFrame.



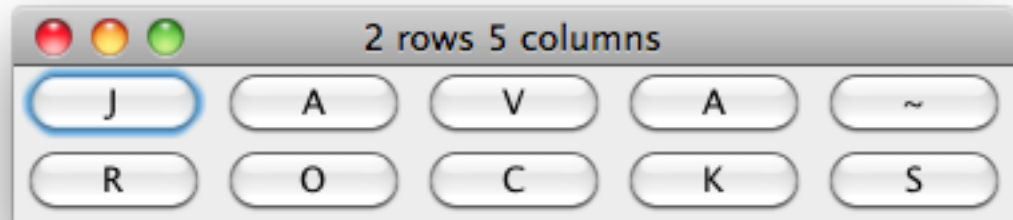
- Exemple :

```
JPanel panel = new JPanel(new BorderLayout());  
panel.add(new JButton("My Button"), BorderLayout.EAST);
```



GridLayout

- Afficher les composants dans un "tableau".
- Toutes les cellules ont exactement la même taille.
- Le composant occupe tout l'espace disponible dans la cellule.



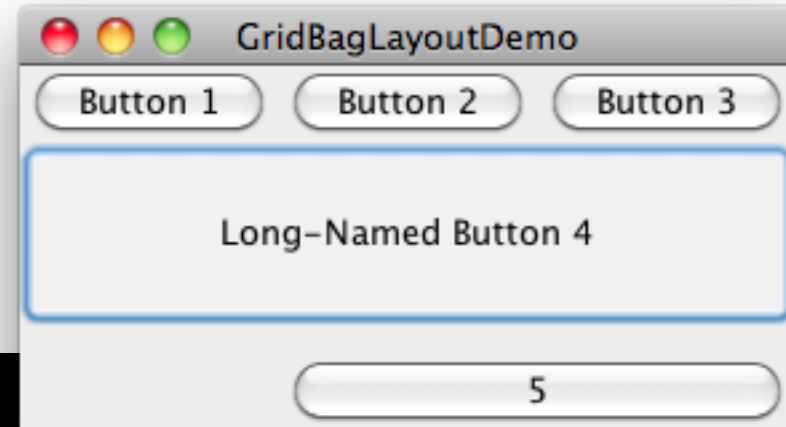
- Exemple :

```
JPanel panel = new JPanel(new GridLayout(2, 5));  
panel.add(new JButton("J"));  
panel.add(new JButton("A"));  
// ...
```



GridBagLayout

- Un des gestionnaires de disposition les plus flexibles et complexes.
- Place les composants dans une grille de lignes et de colonnes comme **GridLayout**, mais :
 - Il permet à des composants spécifiés de couvrir plusieurs lignes ou colonnes.
 - Toutes les lignes et colonnes n'ont pas nécessairement la même hauteur et largeur.





GridBagLayout

- Pour spécifier les caractéristiques d'un composant, vous devez utiliser un objet **GridBagConstraints**.
- Les variables d'instance courantes que vous pouvez définir sont :
 - int **gridx**, **gridy** : Spécifie la position du composant dans la grille.
 - int **gridwidth**, **gridheight** : Spécifie le nombre de cellules utilisées par le composant.
 - int **fill** : Spécifie comment redimensionner le composant lorsque la zone d'affichage du composant est plus grande que sa taille. Les valeurs valides sont définies comme des constantes **GridBagConstraints**.
 - int **anchor** : Détermine où placer le composant à l'intérieur de la zone d'affichage du composant. Les valeurs valides sont définies comme des constantes **GridBagConstraints**.



GridBagLayout

- Exemple :

```
pane.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
c.fill = GridBagConstraints.HORIZONTAL;
JButton button1 = new JButton("Button 1");
pane.add(button1, c);
JButton button2 = new JButton("Button 2");
c.weightx = 0.5;
pane.add(button2, c);
JButton button3 = new JButton("Button 3");
pane.add(button3, c);
JButton button4 = new JButton("Long-Named Button 4");
c.ipady = 40; c.weightx = 0.0; c.gridwidth = 3; c.gridy = 1;
pane.add(button4, c);
// ...
```



Positionnement absolu

- Vous pouvez placer vos composants où vous le souhaitez.
- Voici comment le configurer :
 - Définissez le conteneur de mise en page sur null.
 - Ajoutez vos composants au conteneur.
 - Spécifiez les limites pour chaque composant ajouté.





Positionnement absolu

- Mais quels sont les limites ? Elles sont composées de :
 - Une coordonnée X.
 - Une coordonnée Y.
 - La largeur du composant.
 - La hauteur du composant.
- Vous pouvez les définir avec la méthode :
 - `setBounds(int x, int y, int width, int height)`
- Exemple :

```
Dimension dim = myLabel.getPreferredSize();  
myLabel.setBounds(10, 10, dim.width, dim.height);
```



Positionnement absolu

- Exemple :

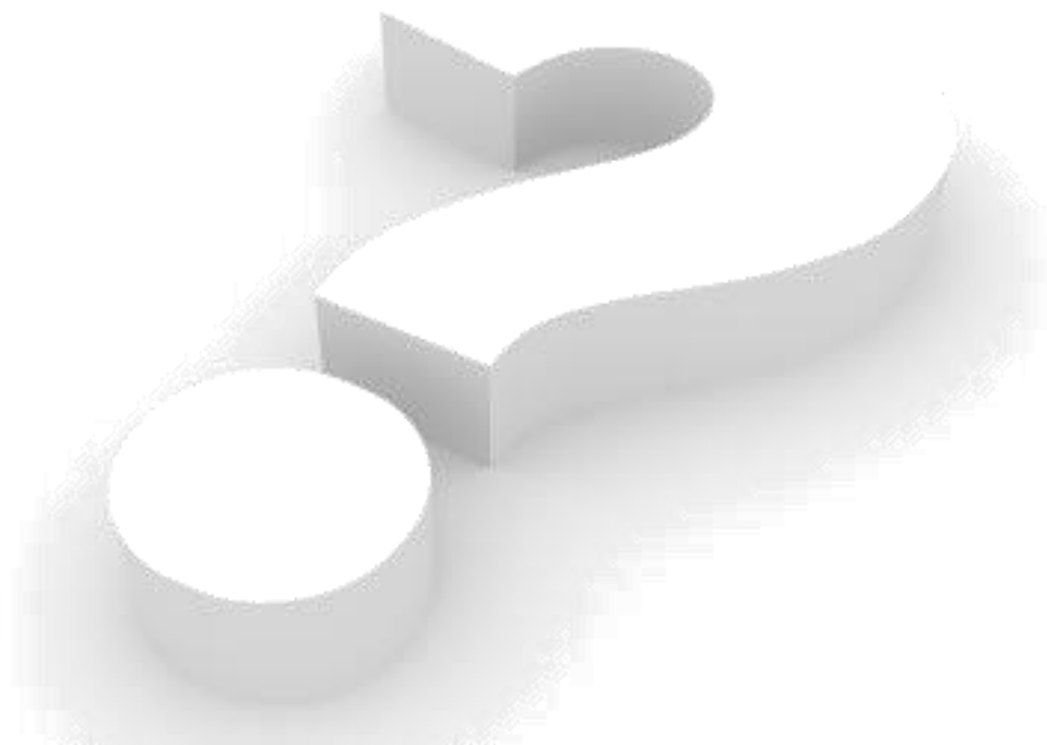
```
// ...
JPanel panel = new JPanel();
panel.setLayout(null);

JButton playButton = new JButton("Play");
JButton exitButton = new JButton("Exit");
Dimension playSize = playButton.getPreferredSize();
Dimension exitSize = exitButton.getPreferredSize();
playButton.setBounds(10, 10, playSize.width, playSize.height);
exitButton.setBounds(10, 50, exitSize.width, exitSize.height);

panel.add(playButton);
panel.add(exitButton);
// ...
```




Questions ?





Exercices (1/2)

- Mettez à jour votre classe **AddTripFrame** :
 - Le contenu de la fenêtre doit utiliser un **BorderLayout** comme gestionnaire de mise en page.
 - Ajoutez un **JPanel** avec un **GridBagLayout** et ajoutez-y vos champs de formulaire sans le bouton de soumission.
 - Les étiquettes doivent être alignées à droite.
 - Ajoutez un autre **JPanel** avec un **FlowLayout** et placez-y le bouton de soumission avec un bouton d'annulation.
 - Ensuite, placez vos deux **JPanel** dans votre frame.
 - Consultez l'illustration sur la diapositive suivante.



Exercices (2/2)

BorderLayout

BagGridLayout

Add Trip

From : 3 / Paris

To : 6 / Tokyo

Price : 990.20

Cancel Add Trip

FlowLayout

Swing

EVENTS



La règle d'abonnement

- Les éléments graphiques...
 - Les boutons, les fenêtres, ...
- ... doivent souscrire à des **écouteurs (Listeners)**!
- Les écouteurs sont :
 - Des interfaces qui gèrent un événement :
 - un clic sur un bouton, maximiser une fenêtre, ...
 - Le développeur choisit d'exécuter une action lorsque qu'un événement se produit.
 - Appelés automatiquement lorsqu'un événement se produit.



Events

La règle d'abonnement

- C'est très flexible :
 - Vous pouvez faire ce que vous voulez.
- C'est facile à configurer.
- C'est encore plus facile à utiliser.



ActionListener

- **ActionListener** est une interface.
- Elle capture l'événement de clic sur un bouton.
- Vous devez définir la méthode
actionPerformed(ActionEvent e) :
 - Cette méthode est automatiquement appelée lorsque vous cliquez sur un bouton.
- La classe **JButton** contient la méthode
addActionListener(ActionListener listener).
 - Lorsque vous appelez cette méthode sur un JButton, on dit que le **listener** souscrit à l'événement.



Définir un écouteur

- Vous avez trois possibilités :
 - Faire que votre classe actuelle l'implémente.
 - Créer une nouvelle classe qui l'implémente.
 - Créer une classe anonyme.
- Chacune de ces possibilités a ses avantages et ses inconvénients.
 - Nous allons les voir chacune.



Définir un écouteur

- Faire que votre classe actuelle l'implémente :

```
public class MyFrame extends JFrame implements ActionListener {  
  
    private JButton button;  
  
    public MyFrame() {  
        button = new JButton("My button");  
        button.addActionListener(this);  
    }  
    @Override  
    public void actionPerformed(ActionEvent ae) {  
        System.out.println("I performed a click ! It works !!");  
    }  
}
```



Définir un écouteur

- Créer une nouvelle classe qui l'implémente :

```
public class MyFrame extends JFrame {  
    private JButton button;  
    public MyFrame() {  
        button = new JButton("My button");  
        button.addActionListener(new MyListener());  
    }  
}
```

```
public class MyListener implements ActionListener {  
  
    @Override  
    public void actionPerformed(ActionEvent ae) {  
        System.out.println("I performed a click ! It works !!");  
    }  
}
```



Définir un écouteur

- Créer une classe anonyme :

```
public class MyFrame extends JFrame {  
  
    private JButton button;  
  
    public MyFrame() {  
        button = new JButton("My button");  
        button.addActionListener(new ActionListener() {  
            @Override  
            public void actionPerformed(ActionEvent e) {  
                System.out.println("I performed a click ! It works !!");  
            }  
        });  
        add(button);  
    }  
}
```



Autres écouteurs

- **WindowListener :**
 - Lorsque la fenêtre est activée, réduite en icône, ...
- **MouseListener :**
 - Lorsque la souris est pressée, cliquée, ...
- **KeyListener :**
 - Lorsqu'une touche est pressée, relâchée, ...
- **FocusListener :**
 - Lorsque le focus est gagné ou perdu.
- ...

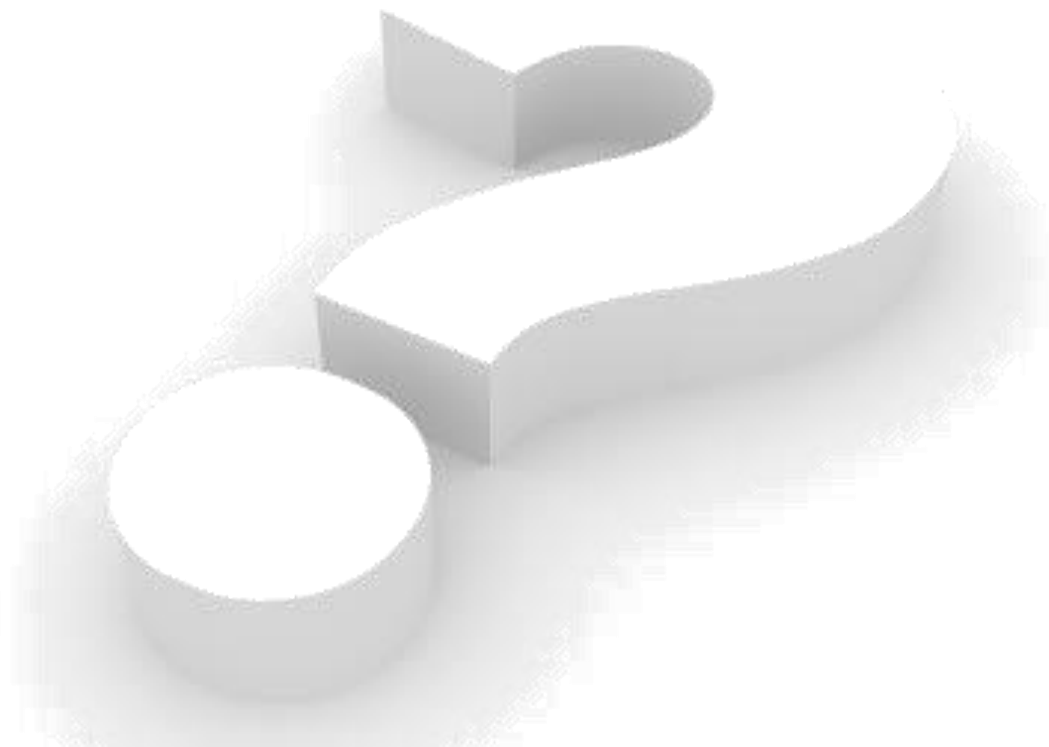


ActionEvent

- Dans le package `java.awt.event`.
- Contient les informations sur l'événement.
- Méthodes utiles :
 - `String getActionCommand()` :
 - Obtient la commande d'action définie sur le bouton.
 - `Object getSource()` :
 - Obtient le composant qui lance l'événement (le bouton).



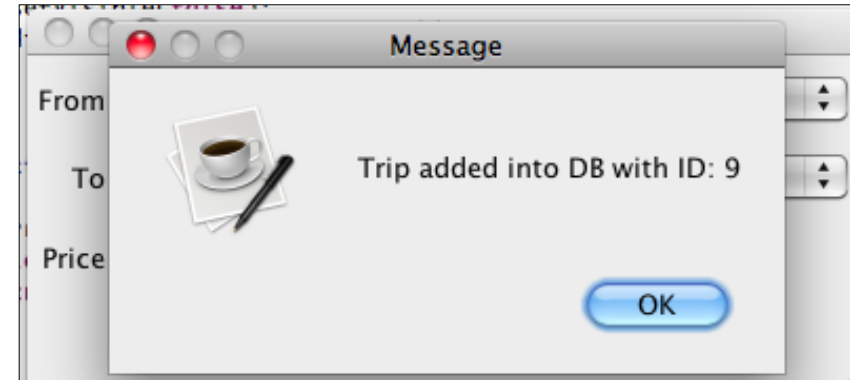
Questions ?





Exercice

- Mettez à jour les frames de votre JavaParadise :
 - Ajoutez des écouteurs à vos boutons d'annulation et de soumission.
 - Gérez les valeurs incorrectes pour les champs.
 - Affichez une fenêtre contextuelle pour informer l'utilisateur lorsque l'objet est correctement persisté avec l'ID généré.



Swing

MENUS



JMenuBar

- Une barre contenant tous les ... JMenu.
- Peut être définie pour un JFrame :
 - `JFrame.setJMenuBar(JMenuBar menuBar)`
- Ajoutez des menus à cela :
 - `JMenuBar.add(JMenu menu)`



JMenu

- Contient tous les ... JMenuItem.
- Est contenu dans un JMenuBar.
- Ajoute un élément à un menu :
 - JMenuItem.add(JMenuItem item)
- Peut avoir un raccourci :
 - void setMnemonic(int mnemonic)
- Exemple :

```
JMenu menu = new JMenu("File");  
// keyboard shortcut: ALT+F  
menu.setMnemonic(KeyEvent.VK_F);
```



JMenuItem

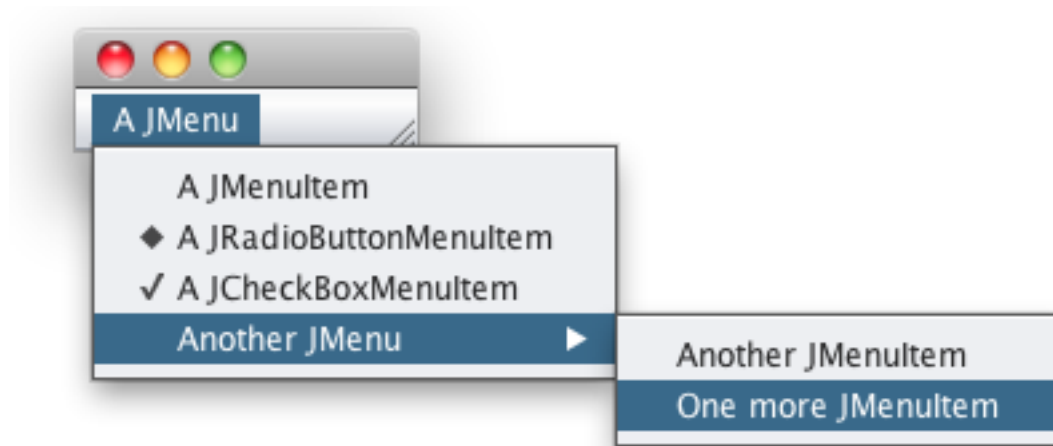
- Un élément pour un menu.
- Fonctionne de la même manière qu'un JButton :
 - Peut avoir une ActionCommand.
 - Utilise un ActionListener pour quand l'utilisateur clique dessus.
- D'autres types existent :
 - JRadioButtonMenuItem :
 - Fonctionne comme un JRadioButton.
 - JCheckBoxMenuItem :
 - Fonctionne comme une JCheckBox.



Menus

Example

- Un JMenuBar contenant un JMenu...





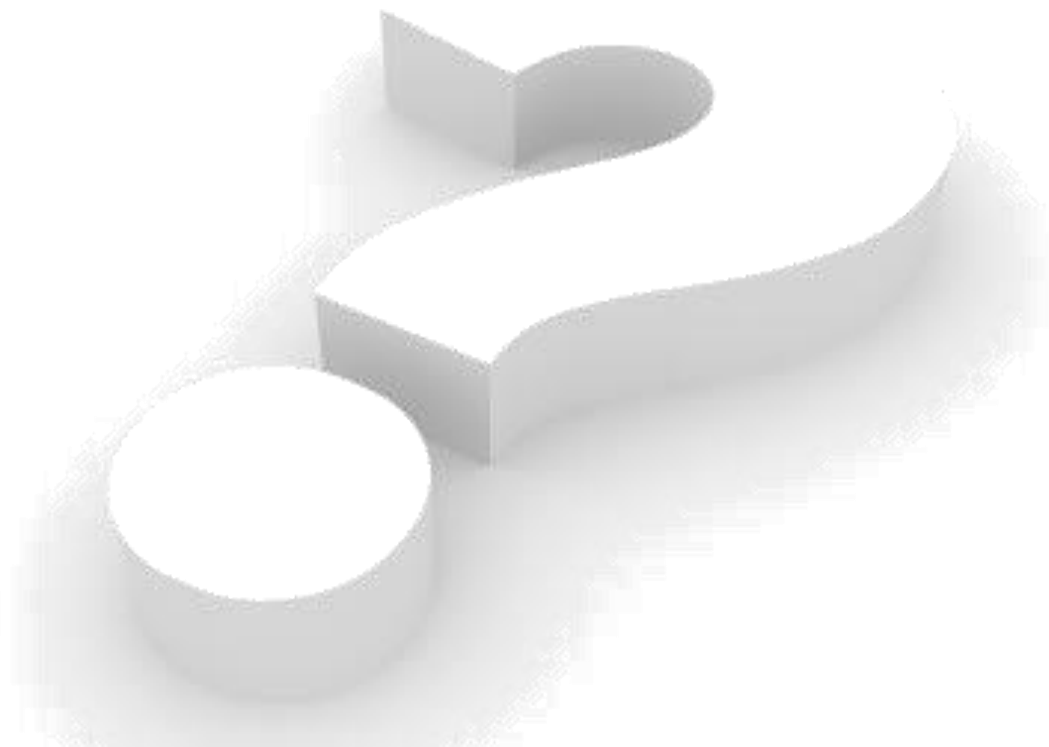
Menus

JPopupMenu

- Un menu contextuel.
- Généralement lorsque l'utilisateur effectue un clic droit quelque part.
- Peut être rempli avec des JMenuItem.
 - Similaire à un JMenu.



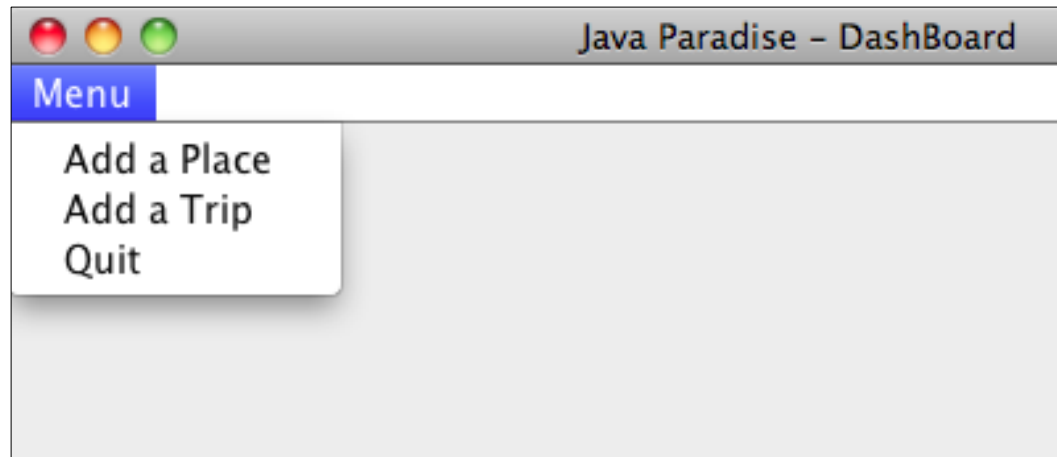
Questions ?





Exercice (1/2)

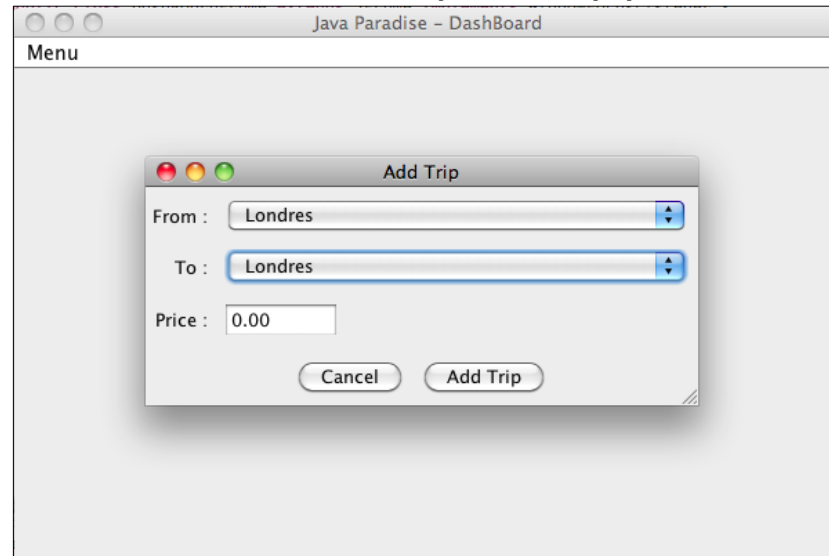
- Créez une classe **DashBoardFrame** étendant JFrame avec :
 - Une taille de 600 x 400.
 - Un titre "Java Paradise - Tableau de bord".
 - EXIT_ON_CLOSE comme opération de fermeture par défaut.
 - Une JMenuBar contenant des JMenu et des JMenuItem comme suit :





Exercice (2/2)

- Implémentez les **écouteurs** d'action des **JMenuItem**s pour afficher le cadre correspondant.
- Mettez à jour la classe **GuiLauncher** et affichez uniquement le cadre du tableau de bord lorsque l'application démarre.



Swing

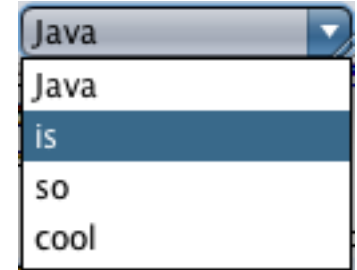
LISTING



Listing

JComboBox

- Une liste déroulante.
- Constructeurs :
 - `JComboBox(Object[] items)`
 - `JComboBox(Vector<?> items)`
- Méthodes utiles :
 - `int getSelectedIndex()`
 - `Object getSelectedItem()`
- Remarque : le texte affiché dans le JComboBox est le résultat de `toString()` des éléments.

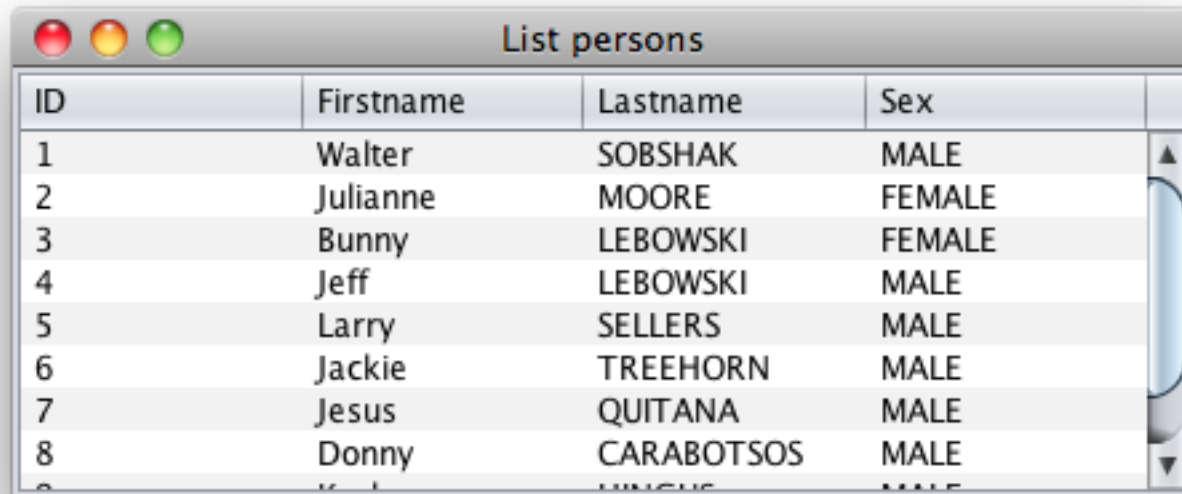




Listing

JTable

- Utilisé pour afficher des données dans une feuille de calcul.
- Utilise un modèle pour organiser la présentation des données.
- Généralement placé dans un **JScrollPane**.



| ID | Firstname | Lastname | Sex |
|----|-----------|------------|--------|
| 1 | Walter | SOBSHAK | MALE |
| 2 | Julianne | MOORE | FEMALE |
| 3 | Bunny | LEBOWSKI | FEMALE |
| 4 | Jeff | LEBOWSKI | MALE |
| 5 | Larry | SELLERS | MALE |
| 6 | Jackie | TREEHORN | MALE |
| 7 | Jesus | QUITANA | MALE |
| 8 | Donny | CARABOTSOS | MALE |



Listing

Table model

- Interface principale : **TableModel**.
- Classe abstraite principale : **AbstractTableModel**.
- Classe concrète principale : **DefaultTableModel**.
- Elle gère chaque cellule.
 - Éditable ou non.
- La vue est automatiquement gérée !
- Généralement, vous définissez votre propre modèle en étendant la classe abstraite ou en implémentant l'interface :
 - Combien de colonnes sont affichées ?
 - Combien de lignes ?
 - Quels sont les en-têtes de texte du tableau ?
 - ...



Table model

- Vous pouvez substituer certaines méthodes :
 - `int getColumnCount()`
 - Renvoie le nombre de colonnes que vous décidez d'afficher.
 - `int getRowCount()`
 - Renvoie le nombre de lignes affichées.
 - `String getColumnName(int col)`
 - Renvoie l'en-tête de la colonne col.
 - La première colonne a l'index 0.
 - `Object getValueAt(int row, int col)`
 - Renvoie la valeur dans la cellule (ligne ; colonne).



Listing

Table model

- Vous pouvez substituer certaines méthodes :
 - boolean `isCellEditable(int row, int col)`
 - Spécifie si la cellule est modifiable par l'utilisateur.
 - void `setValueAt(Object value, int row, int col)`
 - Définit la nouvelle valeur dans la cellule (ligne ; colonne).
 - Doit être redéfini si la cellule est modifiable !



Listing

Table model

- Example:

```
public class MyModel extends AbstractTableModel {  
    private List<Person> persons;  
    // ...  
    @Override  
    public int getColumnCount() {  
        return 4;  
    }  
    @Override  
    public int getRowCount() {  
        return getPersons().size();  
    }  
    @Override  
    public boolean isCellEditable(int row, int col) {  
        return false;  
    } // ...  
}
```



Listing

Table model

- Exemple:

```
// ...
@Override
public String getColumnName(int col) {
    if(col == 0) return "ID";
    else if(col == 1) return "Firstname";
    // ...
}
@Override
public Object getValueAt(int row, int col) {
    Person p = getPersons(row);
    if(col == 0) return p.getId();
    else if(col == 1) return p.getFirstname();
    // ...
}
}
```




Listing

Table model

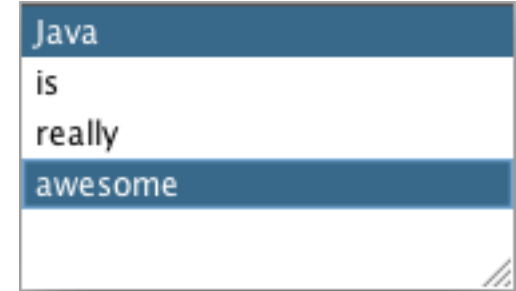
- Si vous souhaitez mettre à jour la vue...
 - Lorsqu'une ligne est ajoutée/supprimée.
 - Lorsque les données sont mises à jour.
 - ...
- ... appelez la méthode `fireTableDataChanged()`
- La vue se rafraîchira automatiquement 😊



Listing

JList

- Une liste simple.
- Peut avoir une sélection multiple.
- Peut également avoir un modèle.
- Constructeurs :
 - `JList(Object[] items)`
 - `JList(Vector<?> items)`
- Méthodes utiles :
 - `int getSelectedIndex()`
 - `int[] getSelectedIndices()`
 - `Object getSelectedValue()`
 - `Object[] getSelectedValues()`

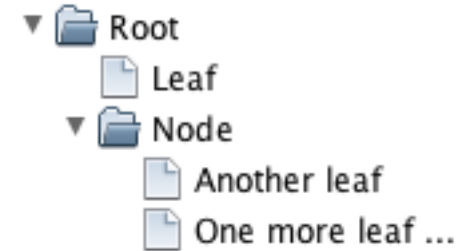




Listing

JTree

- Comme un explorateur.
- Composé de nœuds et de feuilles.
- Contient MutableTreeNode :
 - Pour les nœuds.
 - Pour les feuilles.



```
DefaultMutableTreeNode root = new DefaultMutableTreeNode("Root");  
root.add(new DefaultMutableTreeNode("Leaf"));
```

```
DefaultMutableTreeNode node = new DefaultMutableTreeNode("Node");  
node.add(new DefaultMutableTreeNode("Another leaf"));  
node.add(new DefaultMutableTreeNode("One more leaf..."));
```

```
root.add(node);
```



JTree

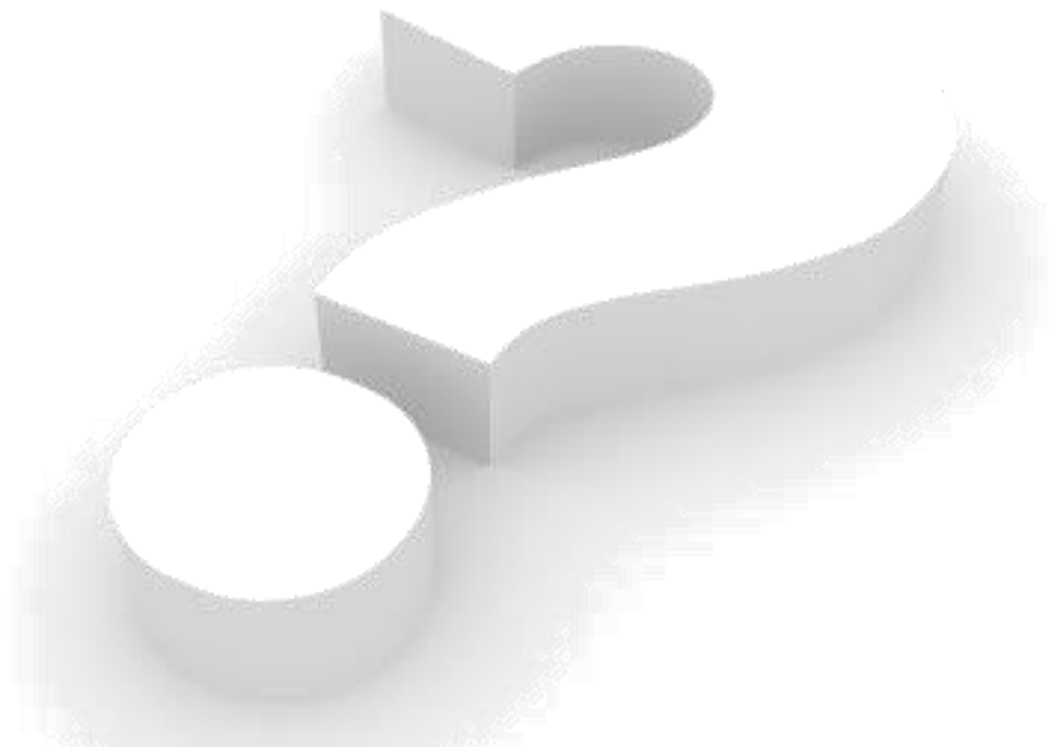
- Récupérer la sélection :

```
DefaultMutableTreeNode node =  
    (DefaultMutableTreeNode) getLastSelectedPathComponent();
```

- Méthodes utiles sur un DefaultMutableTreeNode :
 - boolean isLeaf()
 - Indique si le nœud est une feuille ou non.
 - boolean isRoot()
 - Indique si le nœud est le nœud racine de l'arbre.
 - TreeNode[] getPath()
 - Obtient le chemin de la racine au nœud spécifié.



Questions ?





Listing

Exercices (1/4)

- Mise à jour de DashBoardFrame :
 - Ajoutez un composant **JTable** à votre frame pour afficher la liste de tous les voyages dans la base de données.
 - Utilisez un **TableModel** personnalisé...
 - Permettez à l'utilisateur de trier le tableau par une colonne.



Exercices (2/4)

| Java Paradise - DashBoard | | | |
|---------------------------|---------------|---------------|---------|
| Menu | | | |
| ID | Departure | Destination ▲ | Price |
| 19 | Paris | Candillargues | 0.00 |
| 20 | Candillargues | Candillargues | 0.00 |
| 7 | Paris | Lyon | 999.99 |
| 10 | Paris | Montréal | 0.00 |
| 11 | Paris | Montréal | 0.00 |
| 21 | Montréal | Montréal | 1000.00 |
| 8 | Paris | New York | 999.99 |
| 12 | Tokyo | Paris | 0.00 |
| 13 | Paris | Paris | 0.00 |
| 14 | Paris | Paris | 0.00 |
| 15 | Paris | Paris | 0.00 |
| 16 | Paris | Paris | 0.00 |
| 17 | Paris | Paris | 0.00 |
| 18 | Paris | Paris | 0.00 |
| 1 | Paris | Tokyo | 1000.00 |
| 2 | Paris | Tokyo | 1000.00 |
| 3 | Paris | Tokyo | 999.00 |
| 4 | Paris | Tokyo | 1000.00 |
| 5 | Paris | Tokyo | 1000.00 |
| 6 | Paris | Tokyo | 999.99 |
| 9 | Paris | Tokyo | 999.99 |



Listing

Exercices (3/4)

- Si vous avez le temps :
 - Ajoutez deux **ComboBox** au **NORD** de la fenêtre pour filtrer le **JTable** par les lieux de départ et de destination.
 - Pour ce faire :
 - Ajoutez des écouteurs d'éléments aux **ComboBox**.
 - Utilisez la classe **RowFilter** sur votre **JTable**.



Exercices (4/4)

Java Paradise - DashBoard

Menu

Departure : Destination :

| ID | Departure | Destination | Price |
|----|-----------|-------------|---------|
| 1 | Paris | Tokyo | 1000.00 |
| 2 | Paris | Tokyo | 1000.00 |
| 3 | Paris | Tokyo | 999.00 |
| 4 | Paris | Tokyo | 1000.00 |
| 5 | Paris | Tokyo | 1000.00 |
| 6 | Paris | Tokyo | 999.99 |
| 9 | Paris | Tokyo | 999.99 |

Swing

PEINTURE



Presentation

- Nous avons vu comment créer des cadres simples avec des composants Swing.
- Mais comment développer vos propres composants ?
Comment développer des formes personnalisées à l'intérieur d'un cadre Swing ?
- Grâce à une méthode de **JComponent** que vous pouvez remplacer :
 - `void paintComponent(Graphics g)`
- Généralement, nous dessinons à l'intérieur d'un panneau tel qu'un **JPanel**.
 - Et nous remplaçons la méthode pour définir les formes personnalisées



Graphics

- Un objet Graphics encapsule les informations d'état nécessaires pour les opérations de rendu de base prises en charge par Java.
- Comment en obtenir un ?
 - Implicitement en tant que paramètre de la méthode :
 - `void paintComponent(Graphics)`
 - Explicitement grâce à la méthode de **JComponent** :
 - `Graphics getGraphics()`
- La première solution est le meilleur choix.
 - Mais la deuxième peut parfois être utile.



Graphics

- Les méthodes utiles sont :
 - `getColor()` et `setColor(Color c)` :
 - Obtient ou définit la couleur actuelle de ce contexte graphique.
 - `getFont()` et `setFont(Font f)` :
 - Obtient ou définit la police actuelle.
 - `drawLine(int x1, int y1, int x2, int y2)` :
 - Dessine une ligne, en utilisant la couleur actuelle, entre les points (x1, y1) et (x2, y2).
 - `drawRect(int x, int y, int width, int height)` :
 - Dessine le contour du rectangle spécifié.
 - `fillRect(int x, int y, int width, int height)` :
 - Remplit le rectangle spécifié.



Graphics

- Les méthodes utiles sont :
 - `drawOval(int x, int y, int width, int height)` :
 - Dessine le contour d'un ovale.
 - `fillOval(int x, int y, int width, int height)` :
 - Remplit un ovale délimité par le rectangle spécifié avec la couleur actuelle.
 - `drawImage(Image img, int x, int y, ImageObserver observer)` :
 - Dessine l'image spécifiée aux coordonnées spécifiées.
 - `drawString(String str, int x, int y)` :
 - Dessine le texte donné par la chaîne spécifiée, en utilisant la police et la couleur actuelles de ce contexte graphique.



Repaint

- **paintComponent(Graphics)** est uniquement invoqué par Swing pour dessiner les composants.
 - Vous ne pouvez pas l'appeler manuellement pour redessiner / rafraîchir votre composant.
- Pour cela, utilisez l'une des méthodes suivantes :
 - **repaint()** :
 - Demande de repeindre le composant.
 - **repaint(Rectangle r)** :
 - Demande de repeindre la région spécifiée du composant.



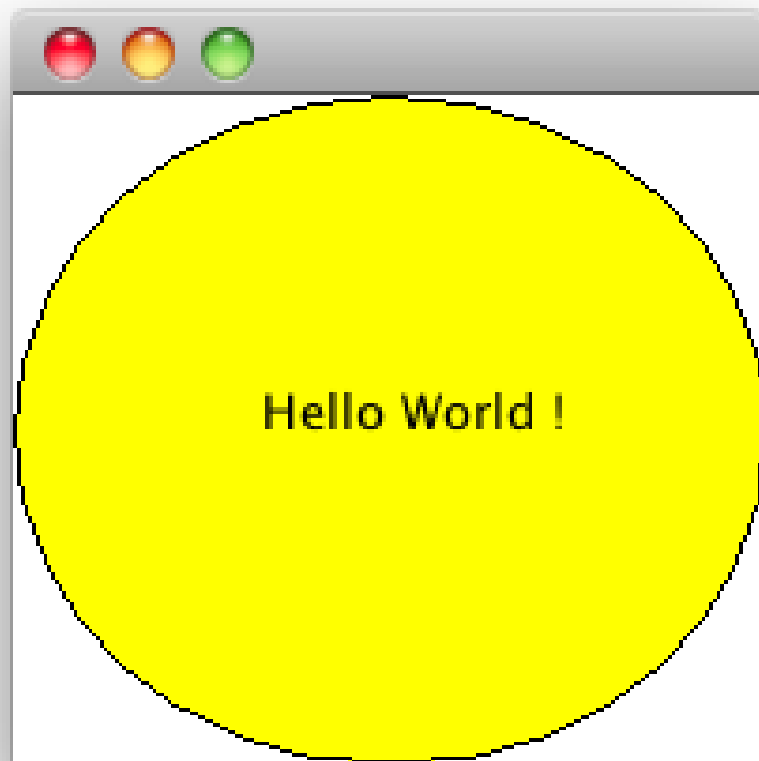
Example

```
public class MyPanel extends JPanel {  
  
    @Override  
    protected void paintComponent(Graphics g) {  
        super.paintComponent(g);  
  
        g.setColor(Color.YELLOW);  
        g.fillOval(0, 0, getWidth(), getHeight());  
  
        g.setColor(Color.BLACK);  
        g.drawOval(0, 0, getWidth(), getHeight());  
  
        g.drawString("Hello World !", getWidth() / 3, getHeight() / 2);  
    }  
}
```



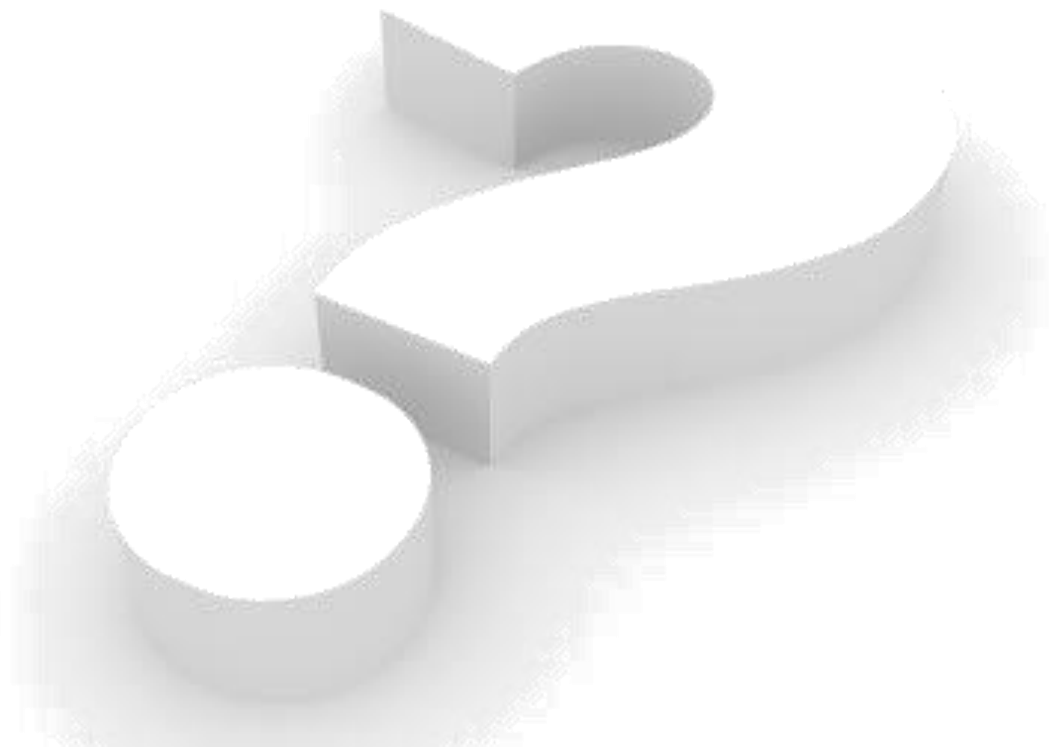

Example

- Peinture





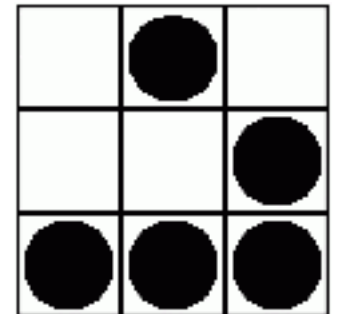
Questions ?





Exercice

- Maintenant que vous savez comment dessiner des formes avec Swing :
 - Essayez de créer une version graphique du Jeu de la Vie !
 - Représentez-le comme vous le souhaitez, en fonction de vos goûts.
 - Soyez imagitatif 😊



Swing

INTERNATIONALIZATION



Introduction

- Java est basé sur :
 - ResourceBundle.
 - Fichier de propriétés.
- Java prend en charge l'encodage UTF-8 :
 - La majorité des langues sont prises en charge.



Fichier de propriétés

- Ils contiennent la traduction de chaque langue que vous fournissez :
 - La syntaxe est : **clé=valeur**
- Ils doivent respecter certaines règles sur leur nom pour être valides :
 - Le suffixe des fichiers doit être :
 - <lang><country>.properties
 - Français du Canada : lang_fr_CA.properties
 - Anglais américain : lang_en_US.properties
 - <lang>.properties
 - Français : lang_fr.properties
- L'absence de suffixe indique que le fichier est le fichier de langue par défaut.



Fichier de propriétés

- Exemples :

- myLang_fr.properties

```
cancel=Annuler  
validate=Valider  
personList=Liste des personnes
```

- myLang_en.properties

```
cancel=Cancel  
validate=Validate  
personList=List of persons
```



ResourceBundle

- Une classe manipulant les fichiers de propriétés.
- Pour en obtenir un :
 - **ResourceBundle.getBundle(String basename)**
- Exemple :

```
ResourceBundle.getBundle("com.cci.sun.myapp.lang.myLang");
```

- Pour récupérer la valeur des clés :
 - **String getString(String key);**

```
myBundle.getString("cancel");
```

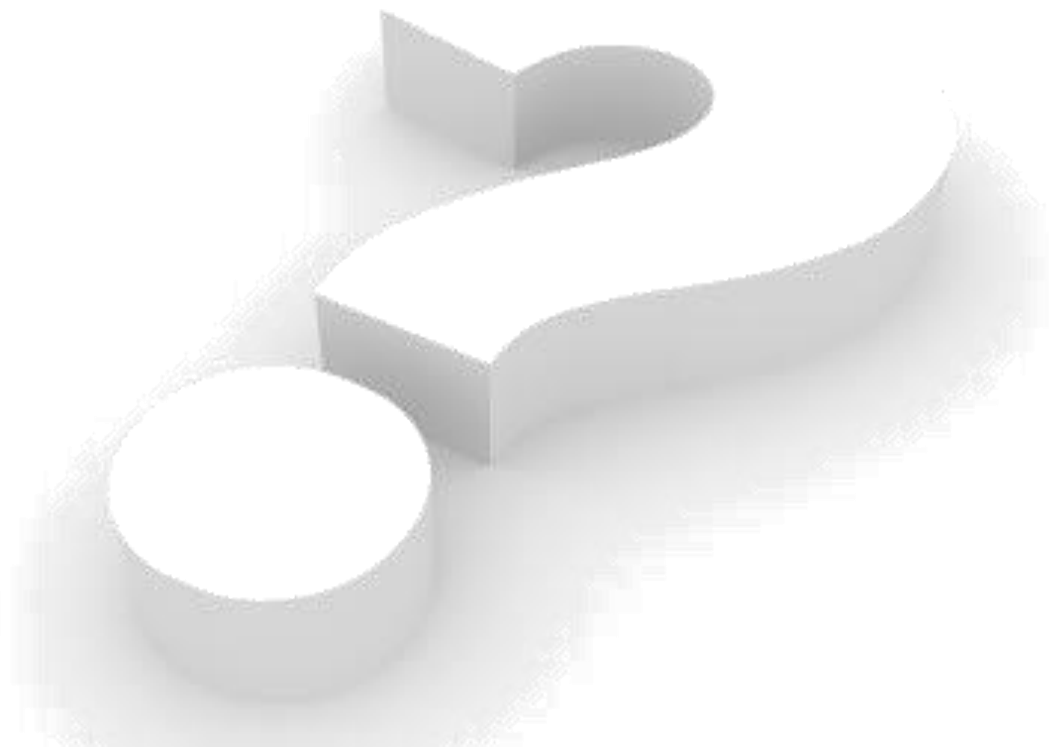



La classe Locale

- Représente une région géographique spécifique.
- Méthodes utiles :
 - **static Locale getDefault()**
 - Obtient la locale par défaut utilisée par la JVM.
 - **static void setDefault(Locale l)**
 - Définit la locale par défaut utilisée.
 - **static Locale[] getAvailableLocales()**
 - Obtient toutes les locales prises en charge par la JVM.
 - **String getCountry()**
 - Renvoie le pays de la locale.
 - **String getLanguage()**
 - Renvoie la langue de la locale.



Questions ?





Exercice

- Créez un package `com.cci.javaparadise.lang`.
- Créez un fichier de langue par défaut nommé `javaparadise.properties` qui contient les traductions en anglais.
- Créez un fichier de langue pour la langue de votre choix.
- Internationalisez JavaParadise en utilisant l'I18N



Swing

Fin

Merci pour votre attention.