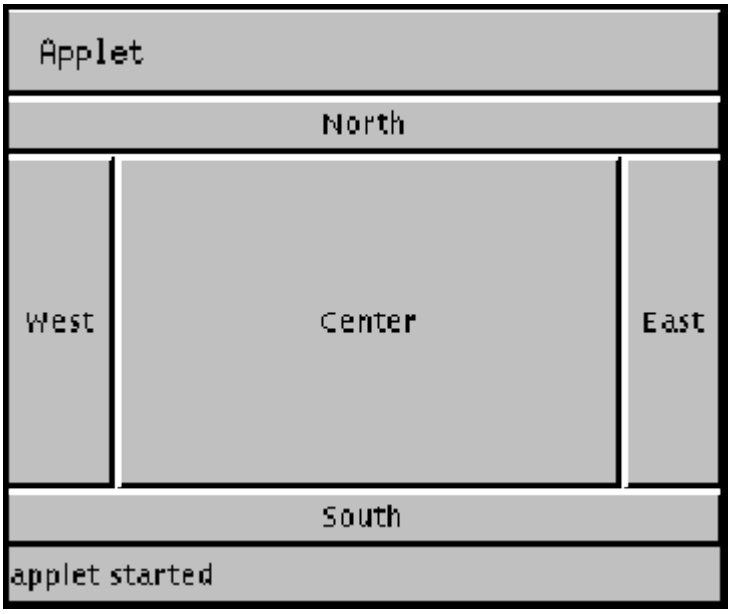
Swing est une bibliothèque incluse dans l'API Java (packages `javax.swing…`) qui permet de développer des interfaces graphiques pour des applications. Elle dispose de différentes classes qui représentent les différents composants d'une interface classique :

- **`JFrame`** : Fenêtre principale de l'application. Elle contient une barre de titre et peut accueillir une barre de menu, un bouton de fermeture, un bouton de redimensionnement et un bouton pour minimiser la fenêtre. Pour spécifier l'action à réaliser lorsqu'on ferme la fenêtre, on utilise la méthode `setDefaultCloseOperation(`**`int`**`)` qui prend en paramètre des constantes de classe. Pour fermer la fenêtre et terminer le programme, on utilise ainsi **`Jframe.`***`EXIT_ON_CLOSE`*.
- **`JPanel`** : Container destiné à contenir d'autre composants. Il dispose d'un gestionnaire de placement (*layout*) qui gère la stratégie de placement des différents composants.
- **`Layout`** : Gestionnaire de placement qui place des composants dans un autre composant en fonction des paramètres qu'on leur a donné. Il existe de nombreux gestionnaires de *layout* différents dans l'API Swing et il est toujours possible de créer ses propres layouts.
  - **`BorderLayout`** : Place les composants dans 5 zones différentes selon le schéma suivant. Seule la zone située au centre est extensible horizontalement et verticalement.



  - **`GridLayout`** : Place les composants dans une grille en spécifiant le nombre de lignes et le nombre de colonnes à l'avance.
  - **`FlowLayout`** : Place les composants les uns à la suite des autres, comme du texte.

Ajouter un composant

Pour ajouter des composants à un autre composant, on utilise la méthode `add(`**`Component`**`)`. Cette méthode existe aussi sous la forme `add(`**`Component,int`**`)` (ainsi que `add(`**`Component,Object,int`**`)`). Ces paramètres supplémentaires représentent des contraintes de placement utilisés par les différents layouts. Pour savoir comment doivent être définies ces contraintes, il faut consulter la documentation du layout utilisé. Les 5 constantes définies par **`BorderLayout`** sont ainsi des **`int`**. Notez que n'importe quel composant de Swing peut en contenir d'autres.

- **`JLabel`** : Composant qui écrit un texte. Ce texte est accessible au moyen des méthodes `getText()` et `setText()`.
- **`JButton`** : Composant qui représente un bouton cliquable. On spécifie une action lors d'un clic sur ce bouton grâce à la méthode `addActionListener(`**`ActionListener`**`)`.
- **`JTextField`** : Composant qui représente un champ de texte.
- **`JTextArea`** : Composant qui représente un champ de texte multi-lignes.

- **`JSpinner`** : Champ qui représente un champ de texte spécialisé pour représenter un certain type de valeur (tel qu'un entier) avec des boutons + et − qui incrémentent / décrémentent la valeur.
- **`JCheckBox`** : Composant qui représente une case à cocher.
- **`JComboxBox`** : Composant qui représente une liste déroulante.
- **`JColorChooser`** : Composant qui représente un ensemble de composants permettant de choisir une couleur.
- **`JFileChooser`** : Composant qui représente une fenêtre de dialogue permettant de choisir un fichier.

Événements

On peut associer à certains composants une action particulière à effectuer lorsqu'un *événement* a lieu. Par exemple, on peut associer une action lorsqu'on clique sur un bouton, lorsqu'on déplace la souris dans un composant, lorsqu'on modifie la valeur d'un champ… Pour associer une action à un événement, il est nécessaire d'implémenter une interface particulière (appelée *écouteur* ou *listener*) ainsi qu'une ou plusieurs méthodes. Chacune de ces méthodes prend généralement un objet en paramètre qui contient diverses informations relatives à l'événement telles que la source (méthode `getSource()`), c'est-à-dire le composant dans lequel l'événement a eu lieu ou encore quand l'événement s'est déclenché (méthode `getWhen()`).

Pour savoir quels interfaces implémenter en fonction des différents événements, il est nécessaire de consulter la documentation de chacun des composants.

- **`ActionListener`** : Interface qui décrit le comportement d'un composant lorsqu'une action a été déclenchée sur celui-ci, typiquement un bouton lorsqu'on clique dessus. Cette interface contient la méthode `actionPerformed(`**`ActionEvent`**`)` qui prend en paramètre un **`ActionEvent`**, un objet qui contient diverses informations relatives à l'événement. Ici, la source de l'événement serait le bouton.
- **`MouseMotionListener`** : Interface qui décrit le comportement d'une souris lorsqu'elle se déplace dans un composant. Cette interface contient deux méthodes :
  - `mouseMoved(`**`MouseEvent`**`)` : appelée lorsque la souris a bougé ;
  - `mouseDragged(`**`MouseEvent`**`)` : appelée lorsqu'un bouton est enfoncé.
- **`MouseListener`** : Interface qui décrit divers comportement d'une souris. Cet interface contient cinq méthodes :
  - `mouseClicked(`**`MouseEvent`**`)` : appelée lorsqu'un bouton est appuyé puis relâché ;
  - `mousePressed(`**`MouseEvent`**`)` : appelée lorsqu'un bouton est appuyé ;
  - `mouseReleased(`**`MouseEvent`**`)` : appelée lorsqu'un bouton est relâché ;
  - `mouseEntered(`**`MouseEvent`**`)` : appelée lorsque la souris est entrée dans le composant ;
  - `mouseExited(`**`MouseEvent`**`)` : appelée lors que la souris est sortie du composant.
- **`MouseAdapter`** : Classe implémentant les interfaces **`MouseListener`**, **`MouseMotionListener`** et **`MouseWheelListener`** (gestion de la molette) en donnant un corps vide à toutes leurs méthodes.
- **`MouseEvent`** : Objet qui contient diverses informations concernant les différents événements liés à la souris. On peut ainsi récupérer la position de la souris (méthodes `getX()` et `getY()`) ou le bouton responsable de l'événement (méthode `getButton()`) : cette méthode peut renvoyer les valeurs **`MouseEvent.`***`BUTTON1`* pour le bouton gauche ou **`MouseEvent.`***`BUTTON3`* pour le bouton droit.

paintComponent(Graphics g)

Cette méthode protégée (**`protected`**) est héritée de `JComponent` (donc par tous les composants Swing). Elle permet de peindre un composant graphique. Cette méthode est appelée chaque fois que le composant a besoin d'être peint, par exemple lors d'un redimensionnement. L'objet `g` a déjà été instancié avant l'appel de cette méthode. On peut donc utiliser ses méthodes pour peindre à l'intérieur du composant. On peut dessiner ou remplir des rectangles, des ellipses, écrire du texte, sélectionner une couleur…

Color

Objet représentant une couleur en mode RGBA (rouge, vert, bleu, couche alpha pour la transparence). Certaines couleurs sont déjà instanciées en tant que constantes de classes (**`Color.`***`RED`*, **`Color`***`.BLUE`*…).

N'hésitez pas à consulter la documentation Java car celle de Swing est particulièrement complexe. Il est impossible de connaître l'ensemble des classes Swing et leurs méthodes.