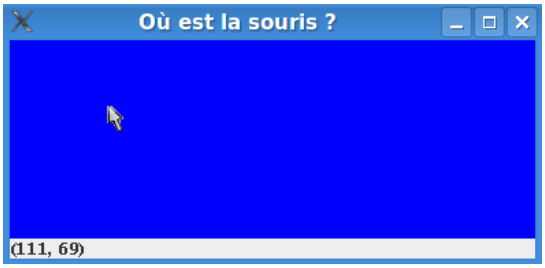
Exercice 1: « Mais où est la souris ? »

On se propose de réaliser l'interface graphique suivante, où un déplacement de la souris dans le "centre" de la fenêtre affiche dans le "sud" de la fenêtre les coordonnées de la souris.



Pour cela :

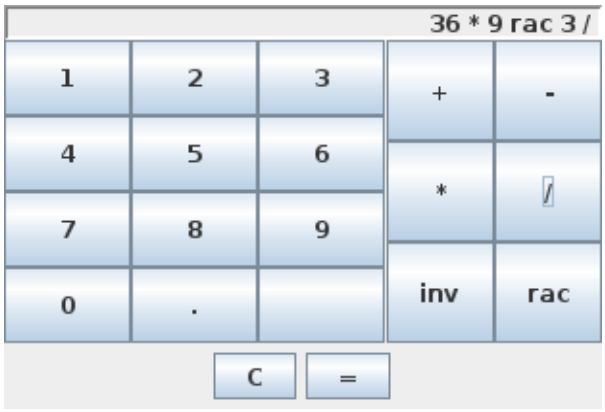
- Créez une classe **`PositionSouris`** qui hérite de la classe `JLabel` et qui implémente l'interface `MouseMotionListener`. L'action associée pour un déplacement de souris (méthode `void mouseMoved(MouseEvent)`) aura pour effet de modifier le texte (méthode `setText(String)`) de **`PositionSouris`**.
- Créez ensuite la classe **`FenetrePositionSouris`** qui hérite de `JFrame`. Cette classe disposera ses composants, dans le « contentPane » (`getContentPane()`), selon un « layout » (méthode `setLayout(Layout)`) de type `BorderLayout`.
  - Le bas de la fenêtre accueillera un objet de type `PositionSouris`. Utilisez la methode `add` qui prend en paramètres cet objet et la constante `BorderLayout.SOUTH` qui définie son positionnement.
  - Au centre (`BorderLayout.CENTER`) sera disposé un `JPanel` avec pour couleur d'arrière plan `Color.BLUE`. Ce `JPanel` sera à l'écoute des événements de déplacements de souris (méthode `addMouseMotionListener(...)`).
- Créez enfin un programme de test pour valider cet exercice.

On cherche maintenant à ajouter des formes en cliquant sur la zone bleue.

- Créez une classe **`MiniCanvas`** qui étend `JPanel`. La coloration en bleu de ce composant se retrouvera dans le constructeur de cette classe.
- Cette classe gère une liste de formes à laquelle on peut ajouter des formes grâce à une méthode `ajouterForme(Forme f)`.
- Créez une classe forme qui représente une forme. Cette classe comporte 3 champs : une abscisse x, une ordonnée y et un troisième champ qui indique le type de forme (on se contentera ici d'un booléen : vrai si c'est un rectangle, faux si c'est une ellipse).
- Ajoutez un `MouseListener` au **`MiniCanvas`** de manière à pouvoir ajouter un rectangle vert à l'endroit où se trouve la souris grâce à un clic gauche et une ellipse rouge grâce à un clic droit.
- Réimplémentez la méthode `paintComponent(Graphics g)` pour que les formes se dessinent dans le panel.
- Utilisez la méthode `repaint()` à chaque ajout de forme de manière à ce que l'image soit mise à jour en direct.

Exercice 2 : Calculatrice

On se propose de réaliser l'interface graphique de la calculatrice implémentée dans les TP4 et 5 comme dans la capture suivante :



Pour cela :

- Créez une classe **`Calculatrice`** qui hérite de la classe `JFrame`.
  - Cette classe disposera ses composants, dans un « contentPane », selon un « layout » (méthode `setLayout(Layout)`) de type `BorderLayout`.
  - Créez un composant de type `JLabel` et ajoutez-le (méthode `add`) dans la zone en haut (`BorderLayout.NORTH`) à notre **`Calculatrice`.**
  - Créez un composant `JPanel` qui représentera les « opérateurs » et placez-le à l'est de la calculatrice (`BorderLayout.EAST`).
  - Créez un composant `JPanel` qui représentera les « chiffres » et placez-le au centre de la calculatrice (`BorderLayout.CENTER`).
    - Ajoutez les `JButton` dont les labels sont « 1 », ..., « 9 », « 0 », « . » au `JPanel` correspondant.
  - Créez un composant `JPanel` qui représentera les « exécutions » et placez-le au sud de la calculatrice (`BorderLayout.SOUTH`).
- Faites en sorte que la classe **`Calculatrice`** implémente l'interface `ActionListener`. Dans la méthode `public void actionPerformed(ActionEvent)` que vous devez implémenter dans la classe **`Calculatrice`:**
  - En fonction du type du bouton, changez le contenu du `JLabel` placé dans la zone en haut :
    - si c'est un nombre ou un espace, ajouter `b.getText()` dans le texte du `JLabel` ;
    - si c'est un opérateur, ajouter l'opérateur au moyen de `b.getText()` entouré de deux espaces au texte du `JLabel` ;
    - si c'est le bouton C, le texte du `JLabel` est réinitialisé ;
    - si c'est le bouton =, le texte du `JLabel` est envoyé à un objet de la classe `Expression` qui lui appliquera la méthode `eval()` et le résultat de cette méthode sera affiché dans le `JLabel`. Si une erreur se produit, il faudra afficher le message de l'erreur dans une nouvelle fenêtre grâce à `JOptionPane.showMessageDialog(...)`.
  - convertissez le contenu de `e.getSource()` en `JButton` grâce à :
    - `JButton b = (JButton)e.getSource().`
- Créez une classe de test.

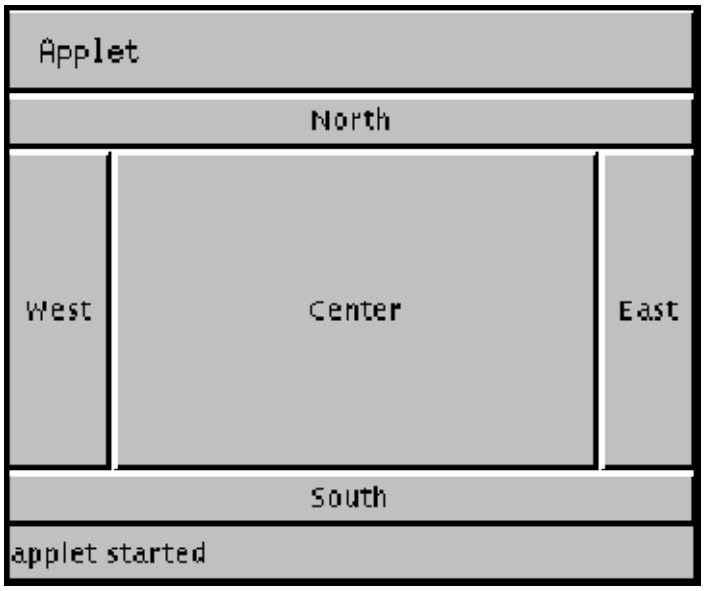
Rappels :

JFrame : fenêtre destiné à être la fenêtre principal de votre 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()`. Pour fermer la fenêtre et terminer le programme, on utilise lui donne la valeur `JFrame.DISPOSE_ON_CLOSE`.

JPanel : est un container destiné à contenir d'autre composants. Il est menu de gestionnaire de placement (Layout) qui gère la stratégie de placement des différents composants.

ContentPane : container d'une `JFrame`, qui contient la partie utile de la fenêtre dans laquelle on va afficher nos composants.

Layout : gestionnaire de placement, qui s'occupe de placer correctement nos composants dans la fenêtre en fonction des paramètres qu'on leur a donné.
Il existe de nombreux gestionnaires de Layout dans Swing, en voici quelques uns:
`BorderLayout` : ce Layout place les composants dans 5 zones du container du : la zone du haut, la zone du bas, la zone de gauche, celle de droite et la zone du centre.



CardLayout, *FlowLayout*, *GridLayout*... pour plus d'informations consultez la documentation java dans l'api java.awt.

JLabel : composant qui écrit un texte. Il possède les méthodes `getText()` et `setText()` qui permettent de gérer le texte affiché.

`JButton` : composant qui représente un bouton cliquable. On spécifie une action lors d'un clic sur ce bouton grâce à la méthode `addActionListener()`.

paintComponent(Graphics g) : méthode protégée héritée de `java.awt.Component` et qui peint un composant graphique. Cette méthode est appelée chaque fois que le composant a besoin d'être peint (lors d'un redimensionnement, d'une iconisation...). L'objet de type Graphics 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 : représente une couleur en mode RGBA. Certaines couleurs sont déjà instanciées en tant que constantes de classes.

ActionListener : interface d'écouteur d'événement d'actions diverses et variées telles qu'un clic sur un bouton. Elle ne possède qu'une méthode prenant un `ActionEvent` en paramètre : `actionPerformed`.

ActionEvent : événement représentant une action sur un composant. La méthode `getSource()` permet de savoir quel composant a déclenché cet événement.

MouseMotionListener : interface d'écouteur d'événement de mouvement de souris qui possède deux méthodes prenant un `MouseEvent()` en paramètre : `mouseMoved()` et `mouseDragged()`. Elles seront respectivement appelées lorsque la souris a bougé et quand la souris a bougé lorsqu'un bouton est enfoncé.

MouseListener : interface d'écouteur d'événement de souris qui possède 5 méthodes prenant chacune un `MouseEvent` en paramètre :

- `mouseClicked()` : appelée quand un bouton est appuyé puis relâché ;
- `mousePressed()` : appelée quand un bouton est appuyé ;
- `mouseReleased()` : appelée quand un bouton est relâché ;
- `mouseEntered()` : appelée quand la souris est entrée dans le composant ;
- `mouseExited()` : appelée quand la souris est sortie du composant.

MouseAdapter : classe implémentant les interfaces `MouseListener`, `MouseMotionListener` et `MouseWheelListener` qui donne à corps vide à toutes leurs méthodes.

MouseEvent : événement qui contient diverses méthodes donnant des informations sur la position de la souris, comme sa position (`getX()` et `getY()`) ou le bouton responsable de l'événement (`getButton()`). Cette méthode peut renvoyer les valeurs `MouseEvent.BUTTON1` pour le bouton gauche ou `MouseEvent.BUTTON3` pour le bouton droit.

Pour afficher une fenêtre de dialogue contenant un message informatif pour l'utilisateur, on utilise la méthode :
`JoptionPane.`*`showMessageDialog`*`(Component parent, String message, String title,` **`int`** `messageType)`.
Le paramètre `messageType` peut prendre les valeurs `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `WARNING_MESSAGE`, `QUESTION_MESSAGE`, ou `PLAIN_MESSAGE`.

Pour utiliser les composants graphiques vous utiliserez les bibliothèques « awt » et « swing ». Utilisez au mieux l'auto-complétion d'Eclipse pour importer vos classes dans vos fichiers sources, car le package des classes est parfois difficile à trouver. N'hésitez pas à consulter la documentation Java : il est impossible de connaître l'ensemble des classes Swing et leurs méthodes. En voici quelques unes :

- `JTextField` représente un champ de texte ;
- `JTextArea` représente un champ de texte multi-lignes ;
- `JSpinner` représente un champ de texte spécialisé pour les entiers avec des boutons + et – et incrémentent/décrémentent la valeur ;
- `JCheckBox` représente une case à cocher ;
- `JComboxBox` représente une liste déroulante ;
- `JColorChooser` représente un ensemble de composants permettant de choisir une couleur ;
- `JFileChooser` représente une fenêtre de dialogue permettant de choisir un fichier.