I5

TP 1 : La définition et l'utilisation des classes

1 Une Première Définition de Classe

L'objectif de la première classe à écrire est de modéliser une horloge classique qui indique une heure et une minute (pas de seconde). Nous allons appeler cette classe `Horloge`. Un objet de classe `Horloge` est initialisé par un constructeur avec deux arguments : l'heure et la minute. Comme pour toute classe, la méthode `to_s()` renvoie une chaîne de caractères qui caractérise l'objet (par exemple, `21h54`). La méthode `avancer()` permet de avancer le temps d'une minute. **Attention :** Avancer l'heure `23h59` d'une minute → `0h0`.

Une définition correcte de la classe devrait faire fonctionner le programme ci-dessous et afficher la sortie indiquée à droite.

```
1 monHorloge = Horloge.new(2,59)
2 puts "Il est:"
3 puts monHorloge
4 for i in 1..60 do
5     monHorloge.avancer()
6 end
7 puts "60 minutes sont passees. Il est:"
8 puts monHorloge
9 for i in 1..(60*24) do
10    monHorloge.avancer()
11 end
12 puts "Une journee est passee. Il est:"
13 puts monHorloge
```

```
Il est:
2h59
60 minutes sont passees. Il est:
3h59
Une journee est passee. Il est:
3h59
```

2 Opérations avec des numéros complexes

Continuer la classe `Complexe` présentée en cours afin de supporter les opération arithmétique ci-dessous.

* La multiplication avec un complexe a été présenté en cours. Continuer l'implémentation afin de pouvoir multiplier le complexe avec un réel. Pour cela, il faut décider si l'argument de l'opérateur `*` (appelé `autreCompl`) dans le cours est un complexe ou pas. Vous pouvez utiliser la méthode `classe()` comme dans l'exemple :

```
if(autreFacteur.class()==Complexe)
    calculer le produit avec un complexe
else
    calculer le produit avec un réel
end
```

\+ L'addition avec un complexe *ou bien* avec un réel. Tester les opérateurs que vous écrivez au fur et à mesure.

\- La soustraction : l'autre terme pourrait être complexe ou réel.

** Pour calculer c^i, il suffit d'exécuter $(i-1)$ multiplications répétées. Chaque multiplication devrait avoir un facteur égal à l'objet de base (l'objet qui appelle l'opérateur **). Pour passer l'objet de base comme facteur, il est possible d'utiliser le mot clé `self`. Par exemple, `puissance = puissance*self`.

Une approche alternative est décrite ci-dessous :

si $i == 1$ il faut renvoyer c

si $i > 1$ calculer le produit entre c et c^{i-1}. Il faut appeler l'opérateur `**` à l'intérieur → utiliser `self**(i-1)`.

`/` Division uniquement avec des réels.

Une définition correcte de la classe devrait faire fonctionner le programme ci-dessous et afficher la sortie indiquée à droite.

```
1 c1 = Complexe.new(2,3);
2 c2 = Complexe.new(4,2);
3 c3 = Complexe.new(4,5);
4 puts (c1 + c2 )
5 puts (c1 + c2 - 1)
6 puts (c1 + c2 - 1) * (c3+1)
7 puts (c1-c2+3)
8 puts ((c1-c2+3)**4)
9 puts ((c1-c2+3)**4)/2.0
```

```
6 + i5
5 + i5
0 + i50
1 + i1
-4 + i0
-2.0 + i0.0
```

Si votre classe `Complexe` fonctionne correctement, faites valider le programme par l'enseignant.

3 Utiliser la classe Java `StringTokenizer`

Utiliser la classe Java `StringTokenizer` afin de définir une classe `Proposition` avec :

- un constructeur qui permet d'initialiser la proposition avec une chaîne de caractères. La classe `Proposition` devrait avoir un attribut privé pour sauvegarder cette chaîne de caractères.
- une méthode `nbMots` qui permet de renvoyer le nombre de mots dans la proposition.
- une méthode `plusLongMot` qui renvoie le mot le plus long dans la proposition.

Une définition correcte de la classe devrait faire fonctionner le programme ci-dessous et afficher la sortie indiquée à droite.

```
1 p = Proposition.new("Je␣suis␣ici!!");
2 puts "#Mots=" + p.nbMots().to_s()
3 puts "Le␣plus␣long␣mot:" + p.plusLongMot()
```

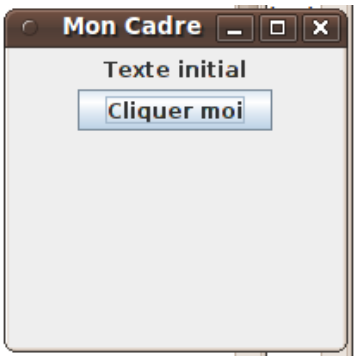
```
#Mots=3
Le plus long mot:suis
```

Le programme doit être lancé avec `jruby`

4 Une Interface Graphique de Base

Exercice 4.1 Construire une fenêtre (cadre) Swing classique de taille 200 × 200. Le panel de cette fenêtre contient juste un bouton ("Cliquer moi") et un label ("Texte initial"). Vous pouvez commencer avec le code ci-dessous, ou bien celui présenté en TD.

```
 1 require _________
 2 import javax.swing.JButton
 3 import javax.swing.JLabel
 4 import javax.swing._______
 5 import javax.swing._______
 6
 7 monPanel = JPanel.new
 8 _________ = JLabel.new("Texte initial")
 9 monBouton = JButton.new(____________)
10 fenetre = JFrame.new("Mon Cadre")
11
12 monPanel.add(_________)
13 monPanel.___(monBouton)
14
15 fenetre.setContentPane(________)
16 fenetre.setVisible(true)
17 fenetre.setSize(200,_______)
18 fenetre.setDefaultCloseOperation(JFrame::EXIT_ON_CLOSE)
```



Exercice 4.2 Modifier le programme pour :

1. Changer la taille de la fenêtre à 500×500
2. Ajouter encore 10 boutons (objets `JButton`) étiquetés *abc1*, *abc2*, *abc3*, ... *abc10*.
3. Ajouter trois zones de textes (objet `JTextField`).

Exercice 4.3 Placer la fenêtre à la position 100,100. Ensuite, déplacer *progressivement* la fenêtre vers la position 300,500.