Chapitre XI

Java: La classe !

XI.1 Nombres complexes

Lorsque les mathématiciens ont souhaité, au XVIe siècle, jouer avec les racines carrées de nombres négatifs, ils ont inventé les nombres *imaginaires*...

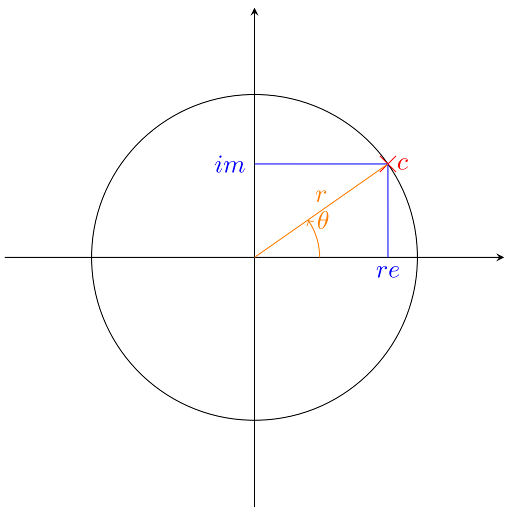
Définition

On définit le *nombre imaginaire* $i = \sqrt{-1}$.
C'est-à-dire que i est défini par l'équation $i^2 + 1 = 0$.

On peut définir un *nombre complexe* de deux manières:

- comme la somme d'un nombre réel et d'un nombre imaginaire: $c = re + i * im$
- comme une distance à l'origine et un angle: $c = re^{i\theta}$

La figure ci-dessous illustre les 2 façon de représenter un même point.



On peut donc calculer, pour un même nombre complexe c ses coordonnées réelles et imaginaires (re et im) en fonction de ses coordonnées polaires (r et θ):

- $re = r\cos(\theta)$ et $im = r\sin(\theta)$
- $r = \sqrt{re^2 + im^2}$ et $\theta = \arctan(\frac{re}{im})$

On peut noter que:

- r est toujours positif
- θ est compris entre 0 et 2π.

On considère une classe `Complexe` qui a les attributs suivants:

- `- re: double`
- `- im: double`
- `- r: double`
- `- theta: double`

et les méthodes suivantes:

- `+ Complexe()`
- `+ Complexe(re: double, im: double)`
- `+ getReel(): double`
- `+ getImaginaire(): double`
- `+ getModule(): double`
- `+ getAngle(): double`
- `+ setReel(re: double): void`
- `+ setImaginaire(im: double): void`
- `+ setModule(r: double): void`
- `+ setAngle(theta: double): void`
- `+ ajouteComplexe(c: Complexe): void`
- `+ multiplieReel(nb: double): void`

Question 1.1
Ecrire la classe `Complexe` en veillant à garantir à tout instant la cohérence de la classe.

On pourra ajouter des méthodes `private` qui calculent r en fonction de re et im, θ en fonction de re et im, re en fonction de r et θ et im en fonction de r et θ...

Question 1.2
Pourquoi ne peut on pas ajouter de constructeur `+ Complexe(r: double, theta: double)` ?

On considère la méthode `main` suivante:

```java
public static void main(String[] args) {
    Complexe c = new Complexe();

    c.setImaginaire(5.0);
    c.getAngle();
    c.setAngle(-1.57...); // on ajoute -pi/2
    c.multiplieReel(2.0);
}
```

Question 1.3
Dessinez le diagramme APO du programme ci-dessus et vérifiez à chaque instant la cohérence de la classe. Corrigez éventuellement votre code à la lumière de ce test...

Question 1.4
Comment rendre cette classe immuable ?

XI.2 Trinôme du second degré : il revient et il a des complexes...

Dans l'exercice IV.1, on recherchait les solutions de l'équation dans l'ensemble des réels. On souhaite à présent calculer les solutions *complexes* d'une équation du second degré aux coefficients réels. Dans le cas où l'on cherche des solutions dans le domaine des complexes, on obtient les mêmes résultats que pour les réels dans les cas où $\Delta = 0$ et où $\Delta > 0$, mais dans le cas où $\Delta < 0$, on a également 2 solutions complexes :

$$x = \frac{-b \pm i\sqrt{-\Delta}}{2a}$$

On rappelle aussi qu'un nombre réel peut être vu comme un nombre complexe dont la partie imaginaire est nulle.

On va maintenant utiliser la classe `Complexe` créée lors des exercices précedents. La nouvelle classe trinome `TrinomeComplexe` aura :

- Toujours 3 attributs réels a, b et c de type `double`
- Toujours un quatrième attribut delta de type double
- Une méthode `nbRacines():int` qui renvoie 0, 1 ou 2 (c'est-à-dire le nombre de racines**réelles**)
- Deux méthodes : `+getRacine1():Complexe` et `+getRacine2():Complexe` qui renvoie deux nouveaux complexes correspondant au racines du trinome

Question 2.1
Écrire et tester la classe `TrinomeComplexe`