Exercice 1 – Syntaxe des objets

Q 1.1 Classe `Complexe`
La classe `Complexe` possède :
- deux attributs double `reelle` et `imag`,
- un constructeur à 2 arguments initialisant les deux attributs
  NB : signature obligatoire : `public Complexe(double reelle, double imag)`,
- un constructeur sans argument, qui initialise les arguments aléatoirement entre -2 et 2.
  NB : pas besoin de réfléchir à l'inclusion ou pas des extrêmes (faire simple)
  NB2 : utiliser obligatoirement la commande `this()` dans ce second constructeur.

Q 1.1.1 Donner le code de la classe `Complexe`

```
1 public class Complexe{ // 1pt
2  private double reelle, imag; // 1pt
3
4  public Complexe(double reelle, double imag){ // 1pt
5    this.reelle = reelle;
6    this.imag = imag;
7  }
8  public Complexe(){// 2pts
9    this(Math.random()*4-2,Math.random()*4-2);
10 }
11
12 public String toString(){// 1pt
13   return "("+reelle+","+imag+"i)";
14 }
15 public Complexe addition(Complexe c){ // 3pts
16   return new Complexe(reelle+c.reelle, imag+c.imag);
17 }
18
19 public boolean estReel(){// 1pt
20   return imag == 0;
21 }
22 // ou bien
23 public boolean estReel(){// 1pt
24   if( imag == 0) return true;
25   return false;
26 }
27
28 }
```

Q 1.1.2 Ajouter les méthodes suivantes (à vous de déterminer les signatures) :
- `toString` qui génère une chaine de caractère de la forme : (*reelle* `+` *imag* `i`)
- `addition` de deux complexes,
- `multiplication` de deux complexes,
- `estReel` qui teste si le complexe est en fait réel (dans le cas où la partie imaginaire est nulle).

Q 1.1.3 Donner le code de la classe `TestComplexe` qui, dans un `main`, effectue les opérations suivantes :
- créer 3 complexes, les afficher,
- tester s'ils sont réels ou pas,
- les additionner, multiplier et afficher les résultats
  NB : vous ajouterez en commentaire les résultats attendus

```
1 public class TestComplexe{
2  public static void main(String[] args){ // 1 pt
```

```
3
4     // 2pt pour le code
5     Complexe c1 = new Complexe();
6     Complexe c2 = new Complexe(1,0);
7     Complexe c3 = new Complexe(1,1);
8
9     System.out.println(c1);
10    System.out.println(c2);
11    System.out.println(c3);
12
13    System.out.println(c2.estReel()); // true
14    System.out.println(c3.estReel()); // false
15
16    System.out.println(c2.addition(c3)); // (2,1i)
17
18  }
19 }
```

Q 1.1.4 Donner les instructions pour compiler ces deux classes et exécuter le test.

1pt
javac Complexe.java
javac TestComplexe.java
java TestComplexe

Q 1.2 Segment Complexe
Donner le code de la classe `SegmentComplexe` qui contient 2 complexes en attributs. Ajouter un constructeur à deux arguments et des accesseurs.

```
1 public class SegmentComplexe{
2   private Complexe c1, c2;
3
4   public SegmentComplexe(Complexe c1, Complexe c2){
5     this.c1 = c1; this.c2 = c2;
6   }
7
8   // + accesseur
9
10 }
```

Exercice 2 – Code mystère + question

On imagine que toutes les fonctions précédentes sont opérationnelles et qu'une méthode `toString` a été redéfinie dans `SegmentComplexe` qui rend une chaine contenant la description des deux points composant le segment. On fait également l'hypothèse que la méthode suivante est ajoutée dans la classe `Complexe` :

```
1 // Dans la classe Complexe
2 public void translateDeUn(){ reelle+= 1; imag += 1; }
```

Dans une classe `Test`, à l'intérieur d'un `main`, on tape le code suivant :

```
1 Complexe c1 = new Complexe(1,0);
2 Complexe c2 = new Complexe(2,0);
3 Complexe c3;
4 SegmentComplexe s1 =
5     new SegmentComplexe(c1, c2);
6
7 c3 = c1.addition(c2);
8 System.out.println(c3);
9 if(c3.estReel())
10   System.out.println("c3 est reel");
11
12 System.out.println(s1);
13 c1.translateDeUn();
14 System.out.println(s1);
15 c1 = c3;
16 System.out.println(s1);
17
18 Complexe c4;
19 if(c4.estReel())
20   System.out.println("c4 est reel");
```

Q 2.1 Ce code contient une erreur (qui empêche la compilation et l'exécution) : trouver là (après la ligne 10).

// 2pts
Le problème est l19 :
– l18 : c4 non instancié
– l19 : on tente d'invoquer une méthode

Q 2.2 Donner les affichages produits lors de l'exécution (en imaginant que le problème précédent a été réglé).

(3,0i) // 0.5pt
c3 est reel // 0.5pt
(1,0i) (2,0i) // 0.5pt
(2,1i) (2,0i) // 1pt
(2,1i) (2,0i) // 0.5pt