Université de Bourgogne
Année universitaire 2019-2020
Info 2a — programmation objet
TP 3-4 — *Délégation et sens des objets*

Objet : délégation et sens des objets.
Les méthodes `toString` et `equals` sont attachées à chaque instance. La méthode `toString` est faite pour restituer un résumé textuel du sens de l'objet (son aspect conceptuel : ce qu'il représente pour l'utilisateur). La méthode `equals` permet de comparer sémantiquement les objets, c'est-à-dire qu'elle restitue `true` si et seulement si leurs propriétés respectives les rendent équivalents pour l'utilisateur (conceptuellement équivalents). Ces deux méthodes doivent être définies par le programmeur dès lors que, sauf pour les objets particuliers prédéfinis comme les `String` et les tableaux de `char`, il n'est pas possible de présumer l'aspect conceptuel d'un objet.

I. Suite fractions (toString et equals)

Reprenez la classe `Fraction` que vous avez réalisée. Si ce n'est pas déjà fait, traitez les points suivants en faisant appel au prof. pour chaque question à laquelle vous avez du mal à répondre.

toString(…)

1. En supposant que `f` soit une variable qui référence la fraction $\frac{3}{2}$, essayez :

   ```
   System.out.println(f);
   ```

   Expliquez le résultat obtenu.

2. Écrivez la méthode de signature `String toString()`[1] qui permet de restituer — sans l'afficher — une forme textuelle de la fraction courante sous la forme :

   `"<numérateur>/<dénominateur>"`

   Testez l'affichage de `f.toString()` puis à nouveau l'affichage du 1. Notez les modifications. Essayez aussi la forme :

   ```
   System.out.println(new Fraction(3,1));
   ```

equals(…)

1. Testez les lignes suivantes :

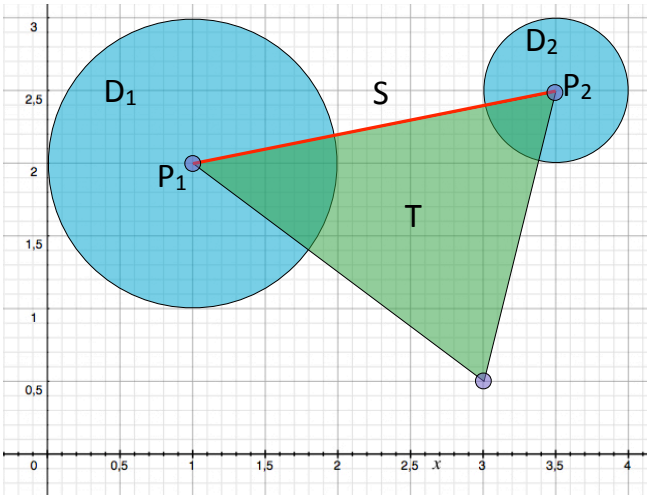
   ```
   Fraction f1=new Fraction(3,4),f2=new Fraction(3,4);
   if(f1==f2) System.out.println("Objets Fraction identiques");
   else System.out.println("Objets Fraction différents");
   ```

   Comment expliquez-vous le résultat ?

[1] Rappel : `toString` est une méthode écrite par le programmeur qui restitue une chaîne de caractères dont le contenu est le « résumé textuel » de l'objet (Ce qu'afficherait une méthode `affiche()`).

2. Remplacez `f1==f2` par `f1.equals(f2)`. Que pensez-vous de ces nouveaux résultats ?
3. Écrivez la méthode de signature `boolean equals(Fraction autre)` qui restitue la valeur `true` si les fractions sont mathématiquement équivalentes[2] (Par exemple, $\frac{6}{4}$ est mathématiquement équivalente à $\frac{9}{6}$.).
4. Testez à nouveau la méthode `equals` avec les exemples précédents et d'autres. Quelle différence constatez-vous avec les résultats précédents ? Si vous ne le connaissez pas déjà, déduisez-en le rôle de la méthode `equals`.

II. Objets géométriques



Il s'agit de créer les classes correspondant aux objets géométriques vus en TD. Écrivez les classes `Point`, `Disque` et `Segment`. La classe `Triangle` est aussi intéressante mais ne l'écrivez que si tout le reste est fini.
Comme toujours, à chaque ajout d'un élément, testez-le immédiatement pour éviter d'être « noyé/e/s » sous les erreurs. Pour ce faire, créez une classe principale avec une fonction `main` où vous placerez le code qui permet de créer des instances d'objets géométriques (par exemple, celles de la figure) et de visualiser leurs caractéristiques.

1. Pour chaque classe, écrivez les déclarations d'attributs et leurs accesseurs en lecture `getXX()` et en écriture `setXX(…)`. Pour ce faire, vous pouvez utiliser l'item de menu `Insert Code…` du menu `Source` de `Netbeans`. Ajoutez aux accesseurs les lignes de code qui permettent de contrôler les valeurs proposées, pour vérifier par exemple que le rayon d'un disque soit strictement positif et que son centre soit bien défini (différent de `null`)… Après avoir fini les questions suivantes, vous pourrez envisager d'autres vérifications. Par exemple que les points d'un segment ou d'un triangle soient distincts,…
   À l'issue de cette phase, les attributs ne doivent plus être accessibles à partir de l'extérieur des objets (modificateur d'accès `private`).

[2] C'est-à-dire de numérateurs et dénominateurs identiques après simplification.

2. Créez les constructeurs nécessaires[3], sachant qu'un point doit pouvoir être créé à partir de deux coordonnées ou d'un autre point par clonage, qu'un disque peut être créé à partir d'un point et d'un nombre décimal pour figurer respectivement son centre et son rayon, ou à partir d'un autre disque par clonage. Si vous en êtes au triangle, faites en sorte qu'il puisse être défini à partir de trois points (ses sommets `A`, `B` et `C`).

3. Créez les méthodes `toString` pour les différentes classes, de telle manière que, par exemple, on obtienne :

   Pour un point :
   ```
   Point (1.0,2.0)
   ```
   Pour un disque :
   ```
   Disque : {Point (3.5,2.5),0.5}
   ```

   Pour un segment :
   ```
   Segment : {Point (1.0,2.0),Point (3.5,2.5)}
   ```

   Pour un triangle :
   ```
   Triangle : {Point (1.0,2.0),Point (3.5,2.5),Point (3.0,0.5)}
   ```

4. Ajoutez des *accesseurs en lecture virtuels* `getXX()`[4], qui permettent d'accéder aux propriétés secondaires des objets. C'est le cas des périmètres ou des surfaces des objets qui en comportent, de la longueur d'un segment, de la distance d'un point à un autre, du point médian entre deux points, des côtés (`AB`, `AC`, `BC`) d'un triangle…

   Écrivez aussi des méthodes `isXX()` pour vérifier que certaines propriétés sont vérifiées[5]. Par exemple, qu'un point est sur la surface d'un disque…

5. Modifiez les `toString` pour faire apparaître les propriétés secondaires, par exemple :

   Pour un disque :
   ```
   Disque : {Point (3.5,2.5),0.5}
     Périmetre : 3.141592653589793
     Aire : 0.7853981633974483
   ```

   Pour un segment :
   ```
   Segment : {Point (1.0,2.0),Point (3.5,2.5)}
     Milieu : Point (2.25,2.25)
     Longueur : 2.5495097567963922
   ```

   Pour un triangle (éventuellement) :
   ```
   Triangle : {Point (1.0,2.0),Point (3.5,2.5),Point (3.0,0.5)}
     Périmetre : 7.111062569605222
     Aire : 2.374999999999999
   ```

6. Dans le programme de la fonction principale,

---

[3] Pour les constructeurs standard, vous pouvez aussi utiliser le menu Insert Code.

[4] Ils sont virtuels dès lors que les propriétés auxquelles elles donnent accès ne sont pas représentées par des attributs (elles n'existent pas dans la description) mais calculées — ce qui donne l'impression de leur existence.

[5] Ce sont des méthodes à valeur booléenne.

a. Écrivez les lignes qui permettent de créer les objets de l'exemple correspondant à la figure ci-dessus. Faites afficher les caractéristiques des objets ainsi créés. Testez votre programme.
b. Ajoutez à la suite ce qu'il faut pour que le point `P1` prenne les valeurs d'abscisse 2 et d'ordonnée 3, puis ajoutez à nouveau les lignes qui permettent d'afficher les caractéristiques des figures. Testez à nouveau votre programme. Que constatez-vous ?

7. Modifiez les lignes qui créent les objets de la figure pour rendre le disque `D1` indépendant du point `P1` (si vous ne comprenez pas cette instruction, demandez à l'enseignant responsable de votre groupe). Testez votre travail.
8. Ajoutez les méthodes `equals()` qui permettent de comparer sémantiquement les objets géométriques selon les spécificités du TD.

Si tout est fini...

1. Améliorez la méthode toString() pour qu'elle restitue un résumé textuel de la fraction sous sa forme la plus simple :
   a. `"n"` pour la fraction $\frac{n}{1}$.
   b. la forme simplifiée de la fraction (`"4/3"` pour la fraction $\frac{16}{12}$ ou `"3"` pour la fraction $\frac{12}{4}$).
2. Écrivez la classe `Triangle` selon les modalités employées ci-dessus pour les autres objets géométriques.
3. Créez des *accesseurs virtuels en écriture* qui permettent de modifier les valeurs des attributs en spécifiant des propriétés secondaires. Par exemple, pour la classe `Disque`, il est possible d'écrire les méthodes `setPerimetre(<double>)` et `setSurface(<double>)` qui modifieront implicitement le rayon, pour la classe `Triangle`, s'il y a lieu, il est possible de modifier ses côtés (`setAB(<Segment>)`,…) en modifiant en réalité ses sommets.