Travaux Dirigés de Java n°3

—Maîtrise d'informatique—

▸ **Exercice 1. (Typage et sous-typage)** *Définir une classe* `Disque` *contenant trois champs de type* `double`*:* `x` *et* `y` *représentent le centre du disque et* `rayon` *le rayon de ce disque.*

Définir un constructeur pour cette classe acceptant trois paramètres pour initialiser chacun des champs.

Définir, dans la classe `Disque`*, une méthode* `translate(double dx, double dy)` *qui déplace le disque et une méthode* `toString()` *retournant une représentation de ce disque sous la forme d'une chaîne de caractères de type* `String` *(par exemple* `Disque - centre: (0,0) - rayon: 1`*).*

Créer quelques objets de la classe `Disque` *dans la méthode* `main()` *de la classe et tester ces méthodes sur ces objets. Qu'affiche le code suivant ? Expliquer.*

```
Disque d1 = new Disque(0,0,1);
System.out.println(d1.toString());
Object o1 = d1;
System.out.println(d1==o1);
System.out.println(o1.toString());
System.out.println(o1);
```

La méthode statique `f` *suivante accepte un* `Object` *en paramètre et est ensuite appelée avec une référence à un objet de type* `Disque` *en argument. Que rajouter au code ci-dessous pour qu'il compile correctement ?*

```
public static void f(Object o) {
  o.translate(1.0,1.0);
}
...
Disque d1 = new Disque(0,0,1);
f(d1);
```

Que se passe t-il lors de l'appel `f("toto");`*? Que faut-il ajouter à* `f` *pour être sûr qu'aucune exception ne sera levée lors de l'exécution ?*

▸ **Exercice 2. (Redéfinition et surcharge)** *Définir une méthode* `equals` *testant l'égalité de deux disques. Qu'affiche le code suivant ? Expliquer et modifier si nécessaire pour obtenir le comportement attendu.*

```
Disque d1 = new Disque(0,0,1);
Disque d2 = new Disque(0,0,1);
System.out.println(d1.equals(d2));
```

```
Object o1 = d1;
Object o2 = d2;
System.out.println(o1.equals(o2));
System.out.println(o1.equals(d2));
System.out.println(d1.equals(o2));
```

▸ **Exercice 3. (Héritage)** *Définir une classe* `Anneau` *comme héritant de la classe* `Disque`*. On considère pour cela qu'un anneau est défini comme un disque avec un rayon supplémentaire, "interne", qui définit le sous-disque intérieur manquant à un disque pour définir un anneau. Ainsi, le seul champ à définir explicitement dans* `Anneau` *est le champ* `rayonInterne` *de type* `double`*.*

Que se passe t-il lorsqu'on crée une instance d'anneau utilisant le constructeur par défaut (`new Anneau();`*) ? Que faut t-il ajouter ?*

Définir un constructeur pour `Anneau` *acceptant quatre paramètres (les deux coordonnées, le rayon et le rayon interne) ainsi qu'un constructeur surchargé sans paramètre qui crée un anneau ayant les caractéristiques d'un disque par défaut et un rayon interne nul.*

Définir une méthode `largeur()` *dans la classe* `Anneau` *qui retourne la largeur de l'anneau.*

Créer un objet de chaque classe et tester cette méthode.

▸ **Exercice 4. (Redéfinition et masquage)** *Dans la classe* `Disque`*, définir une méthode* `surface()` *donnant la surface du disque.*

Que dire de la méthode `surface()` *héritée de* `Disque` *dans* `Anneau` *? Il en va de même pour la méthode* `toString()`*.*

Redéfinir ces deux méthodes dans la classe `Anneau`*.*

Définir un champ `couleur` *dans la classe* `Disque` *comme un entier représentant un code de couleur.*

Redéfinir un champ `couleur` *dans la classe* `Anneau` *comme étant de type* `String`*.*

Que dire des comportements des membres redéfinis. Quelle différence entre un champ redéfini et une méthode redéfinie ?

Comment accéder au champ représentant le code couleur d'un objet de classe `Anneau` *?*

Comment accéder à la méthode `surface()` *définie dans la classe* `Disque` *à partir d'un objet de la classe* `Anneau` *?*

▸ **Exercice 5. (Classe abstraite)** *Écrire une classe* `Carre` *équipée d'une méthode* `surface()`*. Écrire une méthode, en dehors des classes* `Disque` *et* `Anneau`*, qui prend en argument un tableau pouvant contenir des* `Disque`*, des* `Anneau` *et des* `Carre` *qui retourne la somme des surfaces des figures.*

Écrire une méthode `min()` *acceptant indifférement n'importe quelle figure et retournant celle dont la surface est la plus petite. Exemple:*

```
Disque d = new Disque(0,0,1);
System.out.println(d==d.min(new Carre(0,0,2))); // Affiche true
```