TD1 – Java - JAMEL ALLAL

Exercice 1

2) ça complète automatiquement en inscrivant system.out.println

3) ça propose la méthode to String

4) ça propose la methode main

5) ça propose d'implémenter le constructeur de la classe. Ensuite si on tap set et ctrl + space ça nous propose d'implémenter un setter pour le champ foo créer précédemment.

6) ça permet de refractor le nom de la classe (renommer le nom de classe partout où elle est utilisée) et idem sur le champs foo également (renommer le nom du champs partout où il est utilisé)

7) Ctrl + clic sur une classe permet de se rendre à la déclaration de classe, donc ici en le faisant sur String on se rend à la déclaration de la classe String avec les méthodes qu'elle implémente.

Exercice 2

1)

1 : car on utilise le constructeur par défaut

2 : car la méthode appelée contenant le code ci-dessus est définie dans la classe Point où on a les variable p et y. On peut donc y accéder directement sans faire de getter

2) il y a une erreur, on nous dit que les champs Point.x et et Point.y ne sont pas visibles, et pour cause, ils sont en private. Pour corriger ça soit on peut mette les champs x et y en public (pas super) ou sinon on créer des getters pour récupérer leur valeurs et il faudra ecrire p.get(x) et p.get(y) dans le sysout (recommandé).

3) car un attribut peut être utilisé dans le fonctionnement interne d'une classe et peut être amené à fournir sa valeur à d'autres classes sans pour autant permettre à ces dernières de la modifier. Il est possible que certaine valeur dans un programme ne doive surtout pas être modifier. Les mettre en private permet donc de se protéger contre d'éventuelle erreur de nous-même (par oublis) ou encore par les autres développeurs voir contre des potentiels utilisateurs dans le cas d'une api distribuée.

4) C'est un getter qui permet d 'accéder à une variable en private. On doit l'utiliser ici pour régler le problème de la question 2).

5) Comme on a initialisé un constructeur avec des paramètres, le constructeur par défaut n'est plus disponible.

6) Il y a une erreur, eclipse ne comprend pas si c'est l'attribut x et y de la classe Point à qui on souhaite affecté la valeur de x et y passé en paramètres du constructeur ou bien l'inverse. C'est pour ça qu'il faut préciser this.x et this.y pour bien faire références aux attributs de la classe.

7) Il faudrait créer un champs statique dans la classe Point qu'on incrémenterais dans le constructeur (à chaque fois qu'on nouveau point est créer).

```java
public class Point {
    private double x;
    private double y;
    private static int nbPointsCreated = 0;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
        nbPointsCreated++;
    }
```

8) Le compilateur sait quel constructeur appelé en fonction des paramètres utilisés. Dans notre exemple actuel , si on fait appel au constructeur en prenant en paramètres les coordonnées du point le compilateur va chercher le constructeur qui prend en parramétre 2 double , si on fais appel au constructeur en prenant en paramètre un point il va chercher le constructeur qui attend un objet de type Point en paramètres.

9) Il faut redéfinir (@Override) la méthode toString pour la classe Point : on écris return "(" + this.x + "," + this.y + ")";

```java
@Override
public String toString() {

    return "(" + this.x + "," + this.y + ")";
}
```

Exercice 3

1) Le code affiche true puis false. Car la comparaison entre les objets se fais par rapport à l'adresse mémoire et non les valeurs des champs. En effet lorqu'on fais p2 = p1 ; p2 et p1 pointe désormais vers la même adresse mémoire. Tandis que p1 et p3 malgrés qu'ils aient les mêmes valeurs de champs ont une adresse différente.

2)

```
public boolean isSameAs(Point p) {
    return (this.x == p.getX() && this.y == p.getY());

}
```

3) Le problème est que le point p3 n'est pas repéré dans la liste alors qu'il a les mêmes coordonnées que le point p2 qui lui a les mêmes coordonnées que le point p1. En partant du principe que 2 points qui ont les mêmes coordonnées doivent être considérés comme identiques. On voudrait voir le même index pour p3 dans la liste que pour p2 (car mêmes coordonnées). Dans index of la méthode qui nous intéressent qui est appelée est ".equals". Il nous suffit donc de modifier cette méthode dans la classe Point afin que lorsque qu'il s'agit d'un object de type Point on return true a Point1.equals(Point2) si les 2 points (Point1 & Point2) ont les mêmes coordonnées. De ce fait la méthode indexOf fera appel à la méthode nouvellement défini dans la classe point.

```
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;

    return this.isSameAs((Point) obj);
}
```

Exercice 4

1)

```java
public class Polyline {

    private Point[] line;
    private int indexInsert;

    public Polyline(int maxNumerOfPoints) {
        this.line = new Point[maxNumerOfPoints];
        this.indexInsert = 0;

    }
```

2) On a une erreur de type java.lang.ArrayIndexOutOfBoundsException. Pour éviter ce problème il faudrait ajouter une condition pour veriffier si la liste est pleine avant d'inserer un nouveau Point avec la méthode add (nombre de points dans le tableau < à la taille du tableau). Si elle est pleine on pourra renvoyer une IllegalStateException avec un msg d'erreur approprié ; exemple "liste pleine".

3)

```java
  public int pointCapacity() {
      return line.length;
  }

  public int nbPoints() {
      return indexInsert;
  }
```

4)

```java
public boolean contains(Point p1) {
    for (Point p2 : line) {
        if (p2.equals(p1)) {
            return true;
        }
    }
    return false;
}
```

5) Cela dépend de la condition du if dans la boucle for each de notre méthode contains; si on verifie que un point p1 passé en parrametre est contenu dans notre liste en faisant p1 .equals p2 (p2représentant tous les points de notre liste) alors on aura une erreur de type "java.lang.NullException" car on appelle une méthode sur un objet null. Si on inverse le « .equals » en faisant p2.equals(p1) comme nous l'avons fait dans notre méthode alors cela renverra false.

Si on fait add(null) avant ça va bien ajouter le point dans la liste mais si on essaye de parcourir notre liste on aura forcement une erreur de type "java.lang.NullException" car la méthode .equals ne peut pas être appelée sur un objet null. Object.requireNonNull(o) lève une exception

6) La méthode pointCapacity n'a plus d'utilité car une Linkedlist n'a pas de capacité maximale par définition et on a décidé de supprimer la capacité maximum.

```java
public class Polyline {
    private LinkedList<Point> line;

    public Polyline() {
        this.line = new LinkedList<>();
    }
```

$

```java
    public int nbPoints() {
        return line.size();
    }

    public boolean contains(Point p) {
        return line.contains(p);
    }
```

Exercice 5

1) Pour écrire cette fonction on a deux possibilités, de manière mutable ou non mutable.
mutable : on modifie directement les valeurs des variables de notre point.

```
public void translate(double tx, double ty) {
    x += tx;
    y += ty;

}
```

Non mutable -> on retourne un nouvel objet Point avec les nouvelles coordonnées (en prenant en compte la translation)

```
public Point translate(double tx, double ty) {
    return new Point(x + tx, y + ty);

}
```

2) CODE

3)CODE

4) CODE

5) Le problème est que c et c2 ont le même point comme centre. De ce fait lorsque l'on translate un Circle, vu que c'est le point utilisé comme son centre que l'on déplace, cela a pour conséquence de déplacer les deux Circle c et c2. Pour corriger ce problème il suffit de faire une « copie défensive » lorsqu'on créer un Circle :

```
public Circle(Point center, double radius) {
    this.center = new Point(center);
    this.radius = radius;
}
```

6) Le problème serait que en déplaçant le point qui est au centre du cercle on déplacerait le cercle également. Pour corriger ce problème il faut une nouvelle fois utiliser la copie défensive pour le getter :

```
public Point getCenter() {
    return new Point(center);
}
```

7)

```
public double area() {
    return Math.PI * (radius * radius);
}
```

```
@Override
public String toString() {
    return "Center of the circle : " + center + " and radius  : " + radius + " and area : " + area();
}
```

8) On calcule la distance entre le point p et le point center. Si le rayon est supérieur ou égale à cette distance , on peut considérer que le point p appartient au cercle.

```
public boolean contains(Point p) {
    double distance = Math.sqrt(Math.pow(center.getY() - p.getY(), 2) + Math.pow(center.getX() - p.getX(), 2));
    return radius >= distance;
}
```

9)

```
public static boolean contains(Point p, Circle... circles) {
    for (Circle c : circles) {
        if (c.contains(p)) {
            return true;
        }
    }
    return false;
}
```

On doit la déclare en static car elle de dépend pas d'une instance de Circle en question.

Exercice 6

1) Oui on devrait utiliser de l'héritage car un anneau à des points communs avec un cercle. En effet, il a un centre et un rayon (comme le cercle) + un 2ème rayon. On aura ainsi la classe Ring qui héritera de la classe Circle avec un attribut supplémentaire qui est son 2ème rayon .

```java
public class Ring extends Circle {
    private double innerRadius;

    public Ring(Point center, double outerRadius, double innerRadius) {
        super(center, outerRadius);
        if (innerRadius >= outerRadius) {
            throw new IllegalArgumentException("innerRadius have to be smaller than outerRadius");
        }

        this.innerRadius = innerRadius;
    }

}
```

2) On écrit equals pour la classe mère Circle :

```java
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Circle other = (Circle) obj;

    return Objects.equals(center, other.center)
            && Double.doubleToLongBits(radius) == Double.doubleToLongBits(other.radius);
}
```

3)Et pour la classe fille Ring :

```java
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (!super.equals(obj))
        return false;
    if (getClass() != obj.getClass())
        return false;
    Ring other = (Ring) obj;
    return Double.doubleToLongBits(innerRadius) == Double.doubleToLongBits(other.innerRadius);
}
```

4)Sans code supplémentaire on utilisera la méthode toString de la classe mère donc sans les informations propres à l'anneau (il manque le 2ème rayon). Après correction on obtient :

```java
@Override
public String toString() {
    return "The center of the Ring is : " + super.getCenter() + " and inner radius : " + innerRadius
            + " and outer radius : " + super.getRadius();
}
```

5/6)

```java
@Override
public boolean contains(Point p) {
    double distance = Math.sqrt(
            Math.pow(super.getCenter().getX() - p.getX(), 2) + Math.pow(super.getCenter().getY() - p.getY(), 2));
    return innerRadius >= distance;
}

public static boolean contains(Point p, Ring... rings) {
    for (Ring r : rings) {
        if (r.contains(p)) {
            return true;
        }
    }
    return false;
}
```