Daouda KONE

Exercice 1

Question 1

La variable **s** est une chaine de caractère (**String**). Le compilateur cherche la valeur à droite de la variable pour calculter le type.

Question 2

La méthode **var s2 = s1;** fait une copie de l'adresse de s1 dans s2 donc **System.out.println(s1 == s2)** renvoi **true**

La méthide **var s3 = new String(s1);** créé une nouvelle variable avec une nouvelle adresse. Ainsi donc **System.out.println(s1 == s3);** renvoie false.

L'operateur == ne compare pas le contenue de la variable mais plutôt les adresses.

Question 3

Pour comparer le contenue de s4 et s5 il fait utiliser la méthode **System.out.println(s4.equals.s5)**.

Question 4

La code renvoi **true**. Nous nous attendons à **false** mais ici on voit **true**. Cela s'explique par le fait que la chaine de caractàre est stocké dans le cache à l'exécution.

Question 5

Les chaines de litterales ne sont pas mutables pour éviter de modifier toutes les variables qui contiennent la même chaine de caractère.

Question 6

La code renvoi **hello** en minuscule, cela signifie que la méthode **s8.toUpperCase();** ne s'est pas appliqué à la variable s8.

Pour que le code puisse affiche la chaine en majuscule, il fait stocker la methode toUpperCase() dans une variable et l'afficher et c'est cette varoiable qu'il va falloir afficher.

public class Calcul{

```
public static void main(String[] args){

    var s8 = "hello";

    var s9=s8.toUpperCase();
```

```
    System.out.println(s9);
}
```

}

Exercice 2

Question 1

public class Morse{

```
public static void main(String[] args) {

    if (args.length == 0) {

        System.out.println("Any arguments.");
        return;
    }

    var rt = "";
    for (var arg : args) {
        rt += arg + " Stop. ";
    }

    System.out.println(rt);
}
```

}

J'ai fait la condition **if** au debut du code pour m'assurer que j'ai saisi au moins un élément.

Question 2

L'objet **java.lang.StringBuilder** permet de manipuler des chaînes de caractères de manière plus efficace que l'opérateur +, car en Java, les String sont immutables. Chaque concaténation avec + crée une nouvelle chaîne en mémoire, ce qui est peu performant lorsque beaucoup de chaînes sont ajoutées.

Question 3

public class Morse{

```
public static void main(String[] args) {

    if (args.length == 0) {
```

```
        System.out.println("Any arguments.");
        return;
    }

    var rt = new StringBuilder();

    for (var arg : args) {

        rt.append(arg);
        rt.append(" Stop. ");
    }

    System.out.println(rt);
}
```

}

Question 4

On peut utiliser ' ' au à place de " " car l'opérateur + va force ' ' à se convertir *String*.

Question 5

On utise **StringBuilder** lorsqu'on doit faire plusieurs concaténations en boucle.

On évitera de faire + dans la methode **append** éviter des allocations inutiles.

Exercice 3

Question 1

public record Point(double x, double y) {

```
// Ici je fais  la méthode statique
public static Point PointMilieu1(Point p1, Point p2){

    var mid= new Point((p1.x + p2.x) / 2, (p1.y + p2.y) / 2);

    return mid ;
}

// La méthode d'instance

public Point PointMilieu2(Point other){

    var mid = new Point((this.x + other.x) / 2, (this.y + other.y) / 2);
```

```
    return mid;
}
```

}

J'utilise "other" pour prendre l'autre point et "this pour le champs du point courant

public class Application {

public static void main(String[] args) {

```
var p1 = new Point(0.0, 0.0);
var p2 = new Point(1.0, 1.0);

var p3 =Point.PointMilieu1(p1, p2); // appel de la méthode statique
System.out.println("Milieu avec la méthode statique: " + p3);
var p4 = p1.PointMilieu2(p2);// appel de la méthode d'instance
System.out.println("Milieu avec la méthode d'instance: " + p4);
```

} }

Question 2

Prémière vesion accède directement aux champs x et y comme s'il étaient des varibales d'instance, sans encapsulation classique.

Alors que la deuxième version accède aux valeurs en faisant appel aux méthodes d'accès **x()** et **y()**, qui sont de base automatiquement générées par le *record*. Elle retourn simplement les valeurs des chaomps correspondants.

Le version la plus performante est la **prémière version**. Car elle accède directement au champs sans passer par une méthode d'accès.

Question 3

Dans **Point.java** J'ai fait:

public double radiusOfCircle() {

```
var radious = Math.sqrt(x * x + y * y);
return radious;
```

}

public double thetaOfCircle(){

```
var theta = Math.atan2(y, x);
return theta;
```

}

Dans **PolarConverter.java** j'ai fait:

public static void main(String[] args) {

```
var point = new Point(3,4);

var theta = point.thetaOfCircle();
var r = point.thetaOfCircle()
System.out.println("Les coordonnées polaires de " + point + " sont: "+"("+ r + ", " + th
```

}

Question 4

J'ai créer un nouveau fichier **PolarCoordinates.java** :

public record PolarCoordinates(double r, double theta) {

```
public PolarCoordinates {
    if (r < 0) {
        throw new IllegalArgumentException("Le rayon r doit être positif.");
    }
    if (theta < -Math.PI || theta > Math.PI) {
        throw new IllegalArgumentException("theta doit être compris entre -  et  .");
    }
}

@Override
public String toString() {
    return "(" + r + ", " + theta + ")";
}
```

}

Dans le fichier **Point.java** j'ai mis:

public PolarCoordinates polarCoordinate() {

```
    return new PolarCoordinates(
        Math.sqrt(x * x + y * y),
        Math.atan2(y, x)
    );
}
```

Et dans le fichier **PolarConverter.java** j'ai mis :

var polar = point.polarCoordinate(); System.out.println("Les cordonnées polaire de" + point + " sont: " + polar);

pour faire l'affichage.