EX1

1)Voir Projet

2)En écrivant sysout et ensuite en pressant Ctrl+Space cela nous emmène directement à System.out.println()

3)Permet de faire un raccourci pour la méthode ToString() qui hérite de la classe objet

4)Permet de créer une méthode main directement avec sa signature

5)Il nous est proposé d'écrire le constructeur ou la méthode set .

6)On peut renommer la classe même chose pour la variable.

7)A la maison.

EX2:

1)
```
public class Point {
	private int x;
	private int y;
	public static void main(String[] args) {
		Point p=new Point();
		System.out.println(p.x+" "+p.y);
}
```

La méthode main est dans la class Point, on peut donc créé une instance de point dans la class
Point car les entiers x et y sont privés .

2)x et y ne sont pas accessibles dans TestPoint avec un main car on fait appel à x et y sont dans la classe Point et ne sont pas accessibles dans la classe TestPoint car ils sont privés .
Un remède serait de mettre x et y en public.

3)Il faut toujours que les champs d'une classe soient privés pour qu'ils ne soient pas accessibles par tous dans tout le programme (dans toute les classes) et éviter que n'importe qui puisse modifier l'attribut à tout moment .

4)On utilise les accesseurs (getter) afin de pouvoir accéder a champ privé (c'est à dire non accessible dans toutes les classes) dans une autre classe.
Oui, si on veut récupérer les variables de point dans Textpoint .

```
public class TestPoint {
	public static void main(String[] args) {
		Point p=new Point();
		System.out.println(p.getX()+" "+p.getY());
		}
}

5)public class Point {
	private int x;
	private int y;
	public Point (int px,int py){
		this.x=px;
		this.y=py;
}
	public static void main(String[] args) {
		Point p=new Point();
		System.out.println(p.x+" "+p.y);


	}
}
```

Réponse :
Dans le main où on exécute la classe Point, cela ne fonctionne pas car le constructeur dépend de deux paramètres (px et py) , car on demande d'executer Point sans paramètre , donc ça ne marche pas .Il faut alors changer l'appel du constructeur dans le main.
Autrement dit, Point(): ne marche pas car le constructeur est mis à jour avec deux variables(px et py).

6)Ca ne change rien.

7)Pour connaitre le nombre de points qui ont été créés ,on pourra créer un variable static dans la classe point qui s'incrémente à chaque appel du constructeur de point .

8) Le nombre et le type des arguments du constructeur appelé va alors indiquer au compilateur java quel constructeur appelé .

9)On réécrit la méthode toString() qui hérite de la classe objet (va nous permettre de modifier l'adresse mémoire en chaîne de caractère lorqu'on fait un syso dans le main).

```
public String toString() {
        return("("+ this.x + "," + this.y+ ")");
    }
```

EX 3:

1)Il affiche True car p1 et p2 ont la même adresse mémoire et False car p1 et p3 n'ont pas la même adresse mémoire

2)Une méthode isSameAs(Point) renvoyant true si deux points ont les mêmes coordonnées.

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

    }
```

3)Le compilateur pour p2 va indiquer 0, c'est à dire que les coordonnées de p2 sont bien à l'indice 0 dans la liste .
Or,on a ajouté que p1 dans la liste .
p1=p2 (En java ça veut dire qu'ils ont la même adresse mémoire).
Ainsi list.indexof(point) va se référer à l'adresse mémoire et non au coordonnées car ops voit bien que pour p3 (ayant les mêmes coordonnées que p1 et P2), le compilateur va indiquer -1 .

Dans la doc de la fonction indexOf, la méthode appelé est equals de la classe Point.
On va donc redéfinir equals afin que list.indexOff puisse retourner 0 pour les points qui ont les mêmes coordonnées mais pas la même adresse mémoire (ici c'est le cas de p1 et p3 par exemple).

On propose le code suivant pour redéfinir la méthode :

```
@Override
    public boolean equals(Object ob) {
        Point p = (Point) ob;
        return p.x == this.x && p.y==this.y;
    }
```

EX 4

1)Voir code source

2)Si on dépasse la capacité du tableau , le compilateur affichera un message d'erreur .

3)Voir code

4)Voir code

5)On a NullPoniterException dans les deux cas car null n'est pas une adresse et ne peut être rajouté à une liste de Point.

6)Voir la classe ModernPolyline.

-pointCapacity: infiny, on peut toujours rajouter des points à la ligne.

-nbPoints : return LinkedList.size()

-contains : return LinkedList.contains(Point p)

Ex 5

1)Translate agit directement sur les coordonnées du point et ne va rien retourner .

2)Voir code

3)Voir code

4)Voir code

5)Problème: on peut modifier les coordonnée de c2 , il faut mettre l'attribut en final pour éviter cela.

6)Cela n'empêche pas de modifier la valeur du centre .
On rectifie cela de la manière suivante:

```
public Point getCentre() {
        return new Point(centre);
}
```

7)Voir code

8)Voir code

9)Voir code

La fonction est écrite dans Point et est appelée dans la méthode main de la même classe, on peut donc la rendre static.

EX6

1) Non ce n'est pas intéressant de faire l'héritage car la surface d'un anneau n'est pas défini de la même manière .

2)voir code

3)Voir code

4)On va croire que c'est un cercle donc il faut aussi redéfinir toString()

5)Voir code

6)Voir code