HUANG ROGER M1 MIAGE

Exercice 1 : Eclipse

2. Le raccourci *sysout + space* dans la fonction main permet d'écrire la fonction *system.out.println()* plus rapidement.

3. *toStr + space* ouvre un menu déroulant qui propose de d'autocompléter avec des propositions qui contiennent « *toStr* », notamment en premier choix, la méthode *toString().*

4. Menu déroulant qui propose les méthodes les plus courantes qui sont implémentées dans une classe, d'abord le constructeur, puis toutes les redéfinitions courantes telles que les méthodes *equals, hashcode*…

5. En mettant comme attribut un entier appelé « foo », *ctrl+space* propose désormais les fonctions *getFoo()* et *setFoo()*. Si on écrit set, on nous propose l'auto-complétion pour le *setFoo().*

6. Alt + shift + R sur le nom de la classe, permet de modifier le nom de la classe, partout dans le projet. Pareil pour *foo*, permet de modifier le nom de l'attribut dans tout le projet.

Exercice 2 : Point

2. On ne peut pas accéder aux attributs x et y de Point car ce sont des attributs privés. Pour cela, on doit implémenter les méthodes *getX()* et *getY()* pour les lire.

3. On veut que tous les attributs soient privés pour faire en sorte de respecter le principe d'encapsulation des données. On veut contrôler ce qu'on est autorisé à faire avec la classe.

4. Un accesseur est une fonction qui permet d'accéder aux attributs de la classe. *GetAttribut* permet de lire l'attribut, *setAttribut* permet de le modifier. On peut le faire dans notre exemple, au lieu d'écrire *p.x* on pourra faire *p.getX().*

6. Les paramètres d'une fonction ne doivent pas entrer en conflit avec le nom d'un attribut de la classe. Ainsi pour le paramètre x, on peut par exemple écrire à la place _x.

7. On peut créer une variable globale qui va enregistrer à chaque appel de constructeur le nombre d'objets crée (en l'incrémentant).

8. Le compilateur sait quel constructeur appeler en fonction de ses arguments.

Exercice 3 : Equality

1. Le code affiche : true false. En effet le == compare les adresses des objets. Sachant qu'on a affecté p1 à p2, on aura p1 == p2. Par contre p3 est un nouveau Point crée en mémoire, donc on aura !(p1 == p3)

3. *IndexOf* dans les *arrayList* retourne l'indice dans le tableau de l'élément cherché. Il utilise la fonction *equals* pour la comparaison.

Exercice 4 : Polyline

2. Si on dépasse la taille du tableau, une exception est levée. On peut régler ce problème en utilisant une *arrayList*. La taille d'une *arrayList* est dynamique. Par défaut, elle est de 10 puis elle se redimensionne lorsqu'on dépasse. On peut aussi redimensionner manuellement le tableau en vérifiant à chaque fois le dépassement en utilisant la méthode *length*.

5. Si on ajoute *null* au tableau, cela n'est pas considéré comme un point.

6. Une *LinkedList* n'a pas de capacité dans la mesure où sa taille est dynamique.

Exercice 5 : Mutability and circle

5. Quand on translate le point d'un cercle, cela translate le point de tous les cercles faisant référence à ce point. Pour corriger ce problème, on peut rendre la classe Point non mutable.

6. On doit rendre la classe Circle non mutable

Exercice 6 : Anneaux

1. On utilise l'héritage car un anneau est un cercle avec un deuxième rayon.

4. Si on ne redéfinit pas *toString*, cela affiche le *toString* de la classe mère, c'est-à-dire du cercle.