FICHE 3 : VARIABLES

Objectifs

- Comprendre l'assignation d'une variable et le passage des paramètres.
- Représenter les objets en mémoire.
- Travailler en utilisant `this`
- Savoir ce qu'est l'en-tête d'une méthode ainsi que sa signature.
- Comprendre ce qu'est la surcharge de méthodes et être capable de surcharger une méthode ou un constructeur.

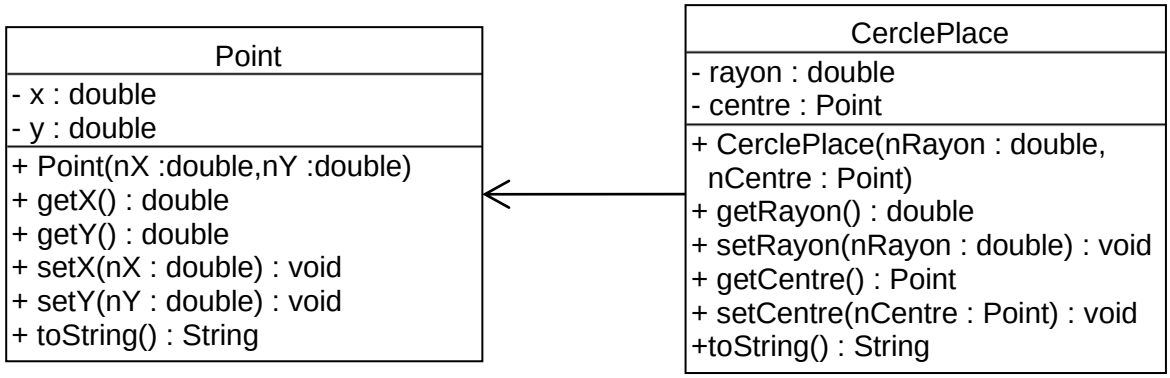
Vocabulaire

this	Surcharge (overloading)	signature	en-tête

Exercices

1. Des cercles dans le plan

Considérons les classes `Point` et `CerclePlace` que vous avez dû implémenter à la séance précédente et dont l'UML est repris ci-dessous (ces classes sont fournies sur moodle si besoin).



1.1. Pour chaque programme ci-dessous, faites la représentation mémoire et dites ce qu'il va afficher :

```
public class TestCercle1 {
	public static void main(String[] args) {
		Point centre = new Point(5.0,2.0);
		CerclePlace c1 = new CerclePlace(3.0,centre);
		double rayon = c1.getRayon();
		rayon = 6.0;
		System.out.println("Rayon de c1 : " + c1.getRayon()) ;
		centre = c1.getCentre();
		centre = new Point(4.0,-2.0);
		System.out.println("Centre de c1 : " + c1.getCentre()) ;
	}
}
```

```
public class TestCercle2 {
	public static void main(String[] args) {
		Point centre = new Point(5.0,2.0);
		CerclePlace c1 = new CerclePlace(3.0,centre);
		CerclePlace c2 = c1;
		c1.setRayon(10.0);
		System.out.println("Rayon de c2 : " + c2.getRayon()) ;
	}
}

public class TestCercle3 {
	public static void main(String[] args) {
		Point centre = new Point(5.0,2.0);
		CerclePlace c1 = new CerclePlace(3.0,centre);
		centre.setX(-2.0);
		centre.setY(-3.0);
		System.out.println("Centre de c1 : " + c1.getCentre()) ;
		CerclePlace c2 = new CerclePlace(2.0,centre);
		Point centreC1 = c1.getCentre();
		centreC1.setX(-4.0);
		System.out.println("Centre de c2 : " + c2.getCentre()) ;
	}
}

public class TestCercle4 {
	public static void main(String[] args) {
		Point centre = new Point(5.0,2.0);
		CerclePlace c = new CerclePlace(1.0,centre);
		centre = c.getCentre();
		deplacer(centre.getX(),centre.getY(),2.0,4.0);
		System.out.println("Centre de c : " + c.getCentre());
	}
	public static void deplacer (double coordonneeX, double coordonneeY,
				double decalageX, double decalageY){
		coordonneeX = coordonneeX + decalageX;
		coordonneeY = coordonneeY + decalageY;
	}
}

public class TestCercle5 {
	public static void main(String[] args) {
		Point point1 = new Point(4.0,7.0);
		Point point2 = new Point(-3.0,5.0);
		double rayon = 4.0;
		CerclePlace c1 = new CerclePlace(rayon, point1);
		rayon = 7.0;
		CerclePlace c2 = new CerclePlace(rayon, point2);
		deplacer(c1, point2);
		deplacer(c2, point1);
		System.out.println("Coordonnées de point1 : " + point1);
		System.out.println("Coordonnées de point2 : " + point2);
		System.out.println("Centre de c1 : " + c1.getCentre());
		System.out.println("Centre de c2 : " + c2.getCentre());
	}

	public static void deplacer(CerclePlace cercle, Point nCentre){
		Point copie = new Point(nCentre.getX(),nCentre.getY());
		cercle.setCentre(copie);
	}
}
```

1.2. En java, modifiez les paramètres des constructeurs et des méthodes des classes `Point` et `CerclePlace` afin qu'ils portent les mêmes noms que ceux des attributs et distinguez-les en utilisant `this`.

1.3. Ajoutez dans la classe `CerclePlace` deux constructeurs. Le premier prend en paramètre uniquement le centre et crée un cercle de rayon 1 ayant ce centre. Le deuxième ne prend rien en paramètre et crée un cercle de rayon 1 centré au point (0,0). Il faut que ces constructeurs invoquent un autre constructeur.

2. Article

Un article possède les informations suivantes :

- une référence ;
- un nom ;
- une description ;
- un prix hors TVA ;
- un taux de TVA (de type réel).

On doit pouvoir créer un article en fournissant toutes ses informations dans l'ordre ci-dessus. Il doit aussi être possible de créer un article en fournissant toutes ses informations excepté le taux de TVA qui prendra alors la valeur par défaut de 21%.

Toutes les informations doivent être consultables. Il doit aussi être possible de modifier la description, le prix hors TVA et le taux de TVA.

Il faut aussi une méthode **`toString`**`()` qui tient compte uniquement du nom et de la référence de l'article.

De plus, il faudra qu'on puisse faire les opérations suivantes (faites de la surcharge !) :

- calculer le prix de l'article TVA comprise. Par exemple, sachant qu'un article a un prix de 20€ hors TVA et que la TVA est de 21%, cette méthode doit renvoyer : 20 * 1.21, c.-à-d. 24.2.
- calculer le prix de l'article TVA comprise si on accorde une réduction (la réduction sera exprimée en pourcent – entier entre 0 et 100). Par exemple, pour un article qui a un prix de 20€ hors TVA, si la TVA est de 21% et que la réduction est de 10%, cette méthode doit renvoyer : 20 * 1.21 * 0.9, c.-à-d. 21.78.

On vous demande de réaliser le **diagramme de classes en UML** correspondant à un `Article`.

Lorsque votre **diagramme est valide** (demandez-le au professeur), écrivez la classe Java correspondante.

Enfin, testez votre classe avec le programme `TestArticle` disponible sur moodle. Vérifiez que la sortie est identique au contenu du fichier affichage_TestArticle.txt.

3. Club du livre

Un club du livre désire informatiser quelques informations concernant ses membres. Récupérez l'ébauche de la classe `Membre` qui se trouve sur moodle et complétez en tenant compte des commentaires. Ensuite, écrivez une classe `TestMembre` en respectant les étapes précisées ci-dessous (inventez les numéros de téléphone) :

1. Créez un membre dans une variable `membre1` s'appelant Emmeline Leconte.
2. Essayez d'initialiser le parrain de `membre1` à `membre1` et vérifiez que la méthode a bien renvoyé `false`.
3. Affichez `membre1` et vérifiez qu'il n'a pas de parrain.
4. Créez un nouveau membre dans une variable `membre2` s'appelant Isabelle Cambron.

5. Initialisez le parrain de `membre1` à `membre2`. Vérifiez que la méthode renvoie bien `true`.
6. Affichez `membre1` et vérifiez que le parrain s'appelle Isabelle Cambron.
7. Créez un nouveau membre dans une variable `membre3` s'appelant Raphaël Baroni.
8. Essayez d'initialiser le parrain de `membre1` à `membre3` et vérifiez que la méthode a bien renvoyé `false`.
9. Affichez `membre1` et vérifiez que le parrain s'appelle toujours Isabelle Cambron.

4. Des séries d'étudiants

À l'IPL, un étudiant est caractérisé par son nom, son prénom et son numéro de matricule. Pour les séances d'exercices, nous avons des séries d'étudiants. Tout étudiant est assigné à une série.
Une série est caractérisée par un nom de type caractère (1, 2, … ou encore a, b, …). Après quelques semaines de cours, chaque série élit un délégué qui la représente au conseil de département.

Dessinons les classes `Etudiant` et `Serie`.

Un étudiant possède comme attribut un numéro de matricule, un nom, un prénom et une série (de type `Serie`). Dans la classe `Etudiant`, on définit également un constructeur qui initialise toutes les variables de l'état. On implémente des getters et pas de setters. On définit la méthode `toString`. De plus, un étudiant doit pouvoir changer de série. Dans la classe `Etudiant`, on ajoute donc une méthode `changerSerie` qui permet à un étudiant de changer de série sous certaines conditions. Cette méthode prend en paramètre une série et renvoie un `boolean`. Elle renvoie `false` si l'étudiant est délégué de la série (dans ce cas, il ne peut pas en changer) ou si la série passée en paramètre est la même que celle en attribut (il est déjà dans cette série). Sinon la méthode enregistre la série en paramètre dans l'attribut et renvoie `true`.

Une série possède comme attribut un nom (constitué d'un caractère) et un délégué de type `Etudiant`. On y définit un constructeur qui prend en paramètre une valeur pour le nom de la série ; il ne prend donc pas de délégué en paramètre (le délégué sera initialisé à `null` lors de la création de la série). On implémente des getters et pas de setters. On y définit aussi la méthode `toString`. Enfin, dans la classe `Serie`, on définit une méthode `elireDelegue` qui permet d'indiquer le délégué d'une série. Cette méthode prend en paramètre un étudiant et renvoie un `boolean`. Elle renvoie **`false`** si l'étudiant passé en paramètre n'est pas dans la série courante ou s'il y a déjà un délégué élu. Sinon la méthode enregistre l'étudiant en paramètre comme délégué (dans l'attribut) et renvoie `true`.

a) Réalisez le diagramme de classes de Serie et Etudiant.
b) Implémentez `Etudiant` et `Serie` exécutez le programme `TestEtudiantSerie` (à télécharger depuis moodle) et vérifiez que la sortie est identique au contenu du fichier affichage_TestEtudiantSerie.txt.
c) Représentez les objets en mémoire au terme de l'exécution du programme de test.