TD 3 Les constructeurs, les variables statiques et l'interaction des variables

Le but de ce TD sera de simuler une pizzeria. Une pizza sera assemblée à partir d'une pâte, une sauce, et au plus trois ingrédients (toppings). La pâte pourrait être fine, croustillante ou épaisse. Nous allons avoir une sauce tomate, une sauce à la crème ou une sauce barbecue. Les ingrédients auront des noms (per exemple : œuf, champignons, jambon, etc.) et une taille de portion. Pour garder en tête toutes les méthodes de toutes les classes dans lesquelles on va travailler, je vous conseille de regarder les diagrammes de chaque classe qu'on va utiliser.

Exercice I : Classes et constructeurs

Prenons une classe Pate, qui inclue le code suivant :

```
public class Pate {
	private String type;
	private char taille;

	public String getType() {
		return this.type;
	}

	public char getTaille() {
		return this.taille;
	}

	public String toString() {
		return("Pate[" + this.taille + "; " + this.type + "]");
	}
}
```

- Dans une classe TD3, dans une méthode principale (main) on a ces deux lignes de code :

```
final Pate pateCrousti = new Pate();
System.out.println(pateCrousti);
```

Est-ce que ce code compile ? Qu'est-ce qui se passe si on veut l'exécuter ?

SOLUTION :

Le code compile. En Java, si on ne met pas de constructeur dans une classe, alors Java nous en fournit un. Celui-ci a obligatoirement la signature `<nom de classe>()` et crée un objet avec les attributs mis aux

valeurs par défaut. Donc pour la classe `Pate` l'attribut taille sera instancié à '' et l'attribut `type` sera instancié à `""`.

Ce qui donnera un affichage du type : `Pate[; ]`

- Nous allons ajouter dans la classe Pate le constructeur suivant :

```
public Pate(String type, char taille) {
	this.type = type;
	this.taille = taille;
}
```

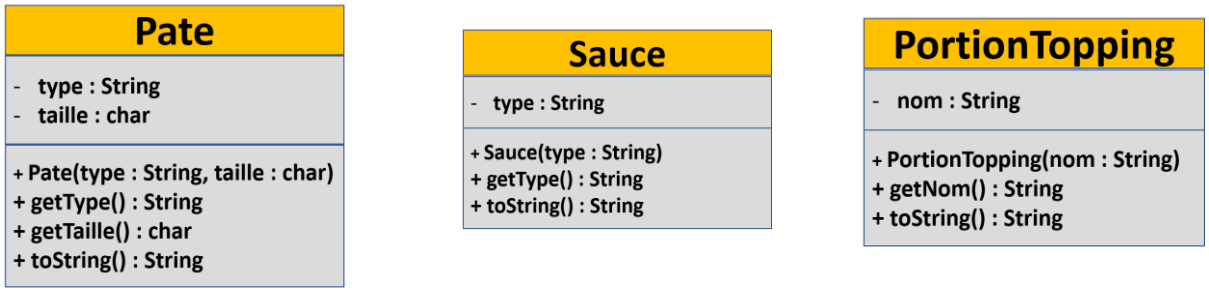
Est-ce que le code de l'exercice précédent compile ? Avec quel résultat ?

SOLUTION :

Le code ne compile plus. Du moment où on ajoute un constructeur à la classe Pate, le constructeur par défaut ne fonctionne plus. Donc Java cherche dans la classe Pate un constructeur avec la signature Pate(), ne le trouve pas et donc elle met une erreur à la compilation.

Exercice II : Construire des Pizzas

Nous allons partir sur les diagrammes de classe suivants :



Dans la classe `Pate` nous avons deux attributs : un attribut type de type `String`, qui peut prendre les valeurs `"croustillante"`, `"fine"`, `"epaisse"` ; ainsi qu'un attribut taille de type `char`, qui peut prendre les valeurs 'S', 'M', 'L'. Le constructeur met les valeurs des attributs aux valeurs mises en entrée. Les getters renvoient les attributs dont ils portent les noms. La méthode String toString() retourne le texte suivant :

`Pate[<taille> ; <type>]`

Dans la classe `Sauce` nous avons un seul attribut, notamment le type, qui peut prendre les valeurs : `"BBQ", "creme fraiche", "tomates".` Le constructeur de cette classe prend en entrée une valeur String qui sera assignée à l'attribut de la classe. Le getter retourne la valeur actuelle du type de la sauce. Finalement, la méthode `toString()` rend le texte suivant :

`Sauce[<type>]`

Dans la classe `PortionTopping` nous avons un attribut nom de type `String`. Le constructeur assigne à l'attribut la valeur en entrée. Le getter retourne la valeur actuelle de l'attribut nom. Finalement, la méthode `toString()` rend le texte suivant :

`Ingredient[<type>]`

Nous voulons construire une classe `Pizza` à partir de ces trois classes. On commence avec une classe `Pizza` qui aura les attributs suivants : un attribut pate de type `Pate`, un attribut sauce de type `Sauce`, un attribut taille de type `char`, un attribut `toppings` de type `PortionTopping[]` et un attribut `prix` de type `double`.

- On calcule le prix d'une pizza à partir d'un prix de base (selon la taille) et selon le nombre de portions de topping. Le prix de base est : 8 euros pour une pizza de taille S, 10 euros pour une pizza M et 12 euros pour une pizza L. Les premières trois toppings sont gratuites ; puis, pour chaque topping supplémentaire on ajoute 50 centimes au prix.

  Ecrivez une méthode privée avec la signature double `calculerPrix()`qui rend le prix de la pizza actuelle.

SOLUTION :

Premièrement, rappelez-vous que le type char est un type primitif. Pour ces types-là, on peut utiliser == pour faire une comparaison, plutôt que d'utiliser equals comme dans le cas des objets. Ceci donne le code :

```
	private double calculerPrix() {
		double prix = 0;
		if (this.taille == 'S') {
			prix += 8;
		}
		else {
			if (this.taille == 'M') {
				prix += 10;
			}
			else {
				prix += 12;
			}
```

```
            }
            for (int i = 3; i < this.toppings.length; i++) {
                        prix += 0.5;
            }
            return prix;
      }
```

- En utilisant votre méthode `calculerPrix()` écrivez un constructeur avec la signature `Pizza(Pate pate, Sauce sauce, PortionTopping[] toppings)` qui assigne les valeurs en entrée aux attributs : `pate`, `sauce`, `toppings` respectivement, qui met la taille de la pizza à la taille de la pâte et qui appelle la méthode `calculerPrix` pour assigner une valeur à l'attribut `prix`.

SOLUTION :

Pour trouver la taille de la pizza il va falloir utiliser la méthode getTaille() de la classe Pizza.

Comme nous sommes dans la classe Pizza nous sommes autorisés à utiliser la méthode calculerPrix(), qui est privée.

Ce qui donne le constructeur suivant :

```
      public Pizza(Pate pate, Sauce sauce, PortionTopping[] toppings) {
            nombrePizzas += 1;
            this.pate = pate;
            this.sauce = sauce;
            this.toppings = toppings;
            this.taille = pate.getTaille();
            this.prix = this.calculerPrix();
      }
```

- Ecrivez une méthode `String toString()` qui retourne le texte suivant pour chaque pizza :

```
Taille : <taille> ;
Pate[<taille> ; <type>] ;
Sauce[type] ;
Toppings[<noms des toppings separes pas un espace>] ;
Prix : <prix>
```

SOLUTION :

Pour la plupart, cette question ne devrait pas vous poser des soucis. La partie la plus difficile est la liste de toppings, qui doivent être retournés un par un, séparés par un espace. Attention : la consigne nous

demande d'envoyer seulement le nom de chaque topping ! Comme nous avons déjà à ce point instancié l'objet en question, nous pouvons partir sur une boucle for, qui part de 0 et s'arrête à this.toppings.length-1. Voici le code :

```
	public String toString() {
		String ingredients = "Toppings[";
		for (int i = 0; i <this.toppings.length; i++) {
			ingredients += this.toppings[i].getNom() + " ";
		}

		ingredients += "]";
		return ("Taille : " + this.taille + "; \n" + this.pate + "; \n" +
this.sauce + "; \n" + ingredients + "; \nPrix : " + this.prix);
	}
```

- Dans la classe TD3, dans une méthode principale, écrivez du code qui crée les prochaines pizzas :
  - Une pizza exotique de taille M, avec une pâte croustillante, une sauce à la crème fraîche et 3 toppings : jambon, ananas et du bleu.
  - Une pizza kebab de taille L, avec une pâte fine, une sauce BBQ et quatre toppings : kebab, emmental, paprika et olives
  - Une pizza Iberia de taille S, avec une pâte épaisse, une sauce tomates et quatre toppings : jambon sec, mozzarella, roquette et pesto vert.

SOLUTION :

On pourrait écrire du code très inefficace qui donne à chaque pate, à chaque sauce et à chaque topping un nom. Mais cela prendrait beaucoup de temps et c'est du code très moche. On peut plutôt utiliser la syntaxe prochaine :

```
		final Pizza pizzaExotique = new Pizza(new Pate("croustillante", 'M'),
new Sauce("creme"), new PortionTopping[] {new PortionTopping("jambon"), new
PortionTopping("ananas"), new PortionTopping("bleu")});
		final Pizza pizzaKebab = new Pizza(new Pate("fine", 'L'), new
Sauce("bbq"), new PortionTopping[] {new PortionTopping("kebab"), new
PortionTopping("emmental"), new PortionTopping("paprika"), new
PortionTopping("olives")});
		final Pizza pizzaIberia = new Pizza(new Pate("epaisse", 'S'), new
Sauce("tomates"), new PortionTopping[] {new PortionTopping("jambon sec"), new
PortionTopping("mozzarella"), new PortionTopping("roquette"), new
PortionTopping("pesto vert")});
```

Dans ce code, l'expression `new Pate("croustillante", 'M')` crée un nouvel objet de type Pate avec les attributs en entrée, mais sans lui donner un nom de variable. Cet objet existe désormais dans la mémoire, il peut être trouvé en écrivant `pizzaExotique.pate()` dans la classe `Pizza` ou en utilisant un getter ailleurs.

- Supposons maintenant que dans la classe `Pizza` a des getters pour chaque attribut. Ecrivez du code qui vous permet d'afficher le nombre de toppings de la pizza exotique, le nombre et tailles de pates utilisées et du code qui vérifie si la taille de la pizza Iberia coïncide avec la taille de la pate qu'on a utilisée pour cette pizza.

SOLUTION :

Pour trouver le nombre de toppings de la pizza exotique il va falloir retrouver la taille du tableau de toppings.

Pour trouver le nombre de toppings et de pâtes une solution serait de mettre les pizzas dans un tableau, et puis de compter, d'un côté les pâtes et d'un autre côté les toppings.

Finalement pour ce troisième point il faut trouer la taille de la pizza et puis la taille de la pâte de cette pizza.

Voici le code résultant :

```
	System.out.println("Toppings de la pizza exotique : "
+pizzaExotique.getToppings().length);
	Pizza[] mesPizzas = new Pizza[] {pizzaExotique, pizzaKebab, pizzaIberia};

	int nombrePates = 0;
	int nombreToppings = 0;
	for(int i = 0; i<mesPizzas.length; i++) {
		nombrePates++;
		nombreToppings += mesPizzas[i].getToppings().length;
	}
	System.out.println("Nombre de pates : " +nombrePates + "; Nombre de toppings :
" +nombreToppings);
	System.out.println("La taille de la pizza coincide avec la taille de la pate
:" + (pizzaIberia.getTaille() == pizzaIberia.getPate().getTaille()));
```

Exercice III : Les attributs statiques

Dans cet exercice nous allons travailler sur les attributs statiques. Rappel : on en a parlé des attributs statiques lors des CM.

- Qu'est-ce qu'un attribut statique ?

SOLUTION :

Un attribut statique est un attribut dont la valeur est égale pour n'importe quel membre de la classe.

- Une pizzeria veut estimer combien de portions de farine sont utilisées dans les pizzas qu'elle vend chaque jour. Pour faire cela il faut prendre en compte le nombre de pizzas vendues ainsi que leurs tailles. Nous allons ajouter un attribut statique `quantiteUtilisee` de type `int` aux attributs de la classe `Pate`. Cet attribut sera initialement mis à zéro.

  Nous allons modifier le constructeur tel que à chaque fois qu'on crée une `Pate`, on augmente la valeur de `quantiteUtilisee` de la façon suivante :
  - Si la pate est de taille `S`, on incrémente la valeur par 1
  - Si la pate est de taille `M`, on incrémente la valeur par 2
  - Si la pate est de taille `L`, on incrémente la valeur par 3

  De cette façon la pizzeria aura un nombre total de portions de farine utilisées.

  Ecrivez le nouveau constructeur de cette classe.

SOLUTION

On ajoute dans le constructeur le code demandé, obtenant :

```
	public Pate(String type, char taille) {
		this.type = type;
		this.taille = taille;
		if (taille == 'S') {
			this.quantiteUtilisee += 1;
		}
		else {
			if (taille == 'M') {
				this.quantiteUtilisee += 2;
			}
			else {
				this.quantiteUtilisee += 3;
			}
		}
	}
```

- Rappel : quelles sont les règles quant aux attributs statiques et les méthodes statiques ?

SOLUTION :

Un attribut statique peut être utilisé dans un contexte non-statique. Par contre, un attribut non-statique ne peut pas être appelé dans un contexte statique. Ceci veut dire que si on veut appeler une méthode statique dans un contexte statique, on n'a pas le droit de l'appeler pour une instance de classe.

- Ecrivez une méthode `static int getQuantiteUtilisee()` qui retourne la valeur de l'attribut statique `quantiteUtilisee`.

SOLUTION :

La tentation ici sera d'écrire cette méthode comme les autres getters (attention, ceci n'est pas un code correct !) :

```
public static int getQuantiteUtilisee() {
	return this.quantiteUtilisee;
}
```

Si on écrit ce code, Java rendra une erreur de compilation. La syntaxe `this.quantiteUtilisee` donne un contexte non-statique, car on ajoute l'objet `this`, pour lequel on appelle l'attribut (statique) quantiteUtilisee. Un code correct ne devrait jamais faire cela. On a deux choix :

- Soit on n'utilise aucun objet (nous sommes dans la classe Pate, donc nous pouvons juste appeler l'attribut sans ajouter le mot this)

```
public static int getQuantiteUtilisee() {
	return quantiteUtilisee;
}
```

- Soit (et cela serait mieux) on appelle l'attribut statique en remplaçant le nom de l'objet (dans ce cas, this) par le nom de la classe (Pate). Ceci n'est pas possible pour les attributs ou méthodes non-statiques ; pourtant, pour les attributs/méthodes non-statiques ceci est possible, car ils ne changent pas pour chaque instance de la classe.

```
public static int getQuantiteUtilisee() {
	return Pate.quantiteUtilisee;
}
```

- Dans la méthode main de la classe TD3 (qui aura les lignes de code dans l'exercice II.4) écrivez du code qui affiche la valeur de l'attribut `quantiteUtilisee`
  - En appelant une méthode de la classe `Pate` pour un objet du type `Pizza`
  - En appelant une méthode de la classe `Pate`, mais en n'utilisant aucun objet

SOLUTION :

La méthode dont on parle est la méthode `getQuantiteUtilisee()`. On pourrait envisager un code de ce type :

```
System.out.println(pizzaExotique.getPate().getQuantiteUtilisee());
```

Si on ne veut utiliser aucun objet, on se rappelle que pour des méthodes statiques on peut juste appeler le nom de la classe, plutôt qu'une instance de celle-ci. On peut donc écrire :

```
System.out.println(Pate.getQuantiteUtilisee());
```

- Dans la méthode main on a ces deux lignes de code. Est-ce qu'il compile ? Quel est le résultat de l'exécution ?

```
System.out.println(pizzaExotique.getPate().getQuantiteUtilisee());
System.out.println(pizzaKebab.getPate().getQuantiteUtilisee());
```

SOLUTION :

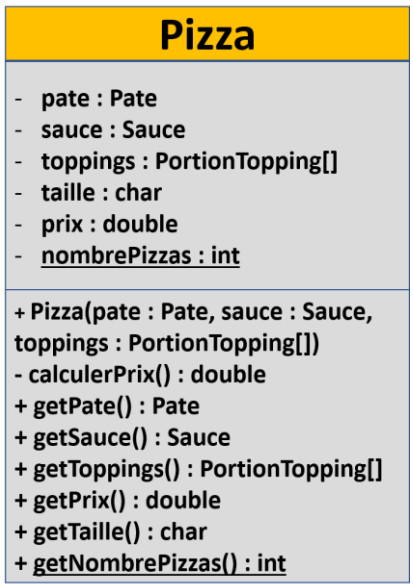
Oui, ce code compile. Toutes les deux lignes de code retourneront la même valeur (car un attribut statique a la même valeur pour chaque membre de la classe, y compris les deux pates de pizzaExotique et pizzaKebab). Si on suppose que les trois pizzas créées dans l'exercice II sont les seules alors on aura deux fois une sortie de 6.

Exercice IV : plus d'attributs statiques

Dans cet exercice nous allons essayer d'ajouter des rabats de prix pour les pizzas et de calculer quelques statistiques concernant leur production.

- Nous allons ajouter un attribut statique nombrePizzas de type int dans la classe Pizza. Cet attribut sera initialisé à zéro. A chaque fois qu'une pizza est produite, on augmente cet attribut par 1. Pour chaque 10eme pizza, on fait un rabat de prix de 10% sur le coût de la pizza.
    - Ecrivez à nouveau votre méthode calculerPrix() de la classe Pizza pour prendre cela en compte.
    - Ecrivez à nouveau votre constructeur de cette classe pour faire augmenter le nombre de pizzas à chaque instanciation

  Le diagramme de cette classe est maintenant :



Un attribut/méthode statique est toujours souligné dans un diagramme de classe !

SOLUTION :

Une façon de savoir s'il s'agit d'une 10ème, un 20ème, etc. pizza, c'est de savoir si le nombre total de pizzas au présent est divisible par 10. On peut tester ceci en vérifiant si la division du nombre de pizzas par 10 donne un reste de 0. Voici le code résultant :

```
	private double calculerPrix() {
		double prix = 0;
		if (this.taille == 'S') {
			prix += 8;
		}
		else {
			if (this.taille == 'M') {
				prix += 10;
			}
			else {
				prix += 12;
			}
		}

		for (int i = 3; i < this.toppings.length; i++) {
				prix += 0.5;
		}
		if (nombrePizzas % 10 == 0) {
			prix = 9/10 * prix;
			System.out.println("Discount! " + this.nombrePizzas + "eme pizza
!");
		}

		return prix;
	}
```

Pour la modification du constructeur ce qui est le plus important c'est de décider à quel point ajouter du code qui augmente la valeur de l'attribut statique. Il est essentiel de mettre à jour ce compteur avant le calcul du prix. (Pourquoi ?!)

Voici un exemple du code :

```
	public Pizza(Pate pate, Sauce sauce, PortionTopping[] toppings) {
		nombrePizzas += 1;
		this.pate = pate;
		this.sauce = sauce;
		this.toppings = toppings;
		this.taille = pate.getTaille();
		this.prix = this.calculerPrix();
	}
```

- Ajoutez un attribut statique nombreToppings de type int dans la classe PortionTopping, qui augmente par 1 à la création de chaque topping. Dans la méthode main de la classe TD3 ecrivez du code qui permet de savoir combien de toppings on a en moyenne pour chaque pizza.

SOLUTION :

Algorithmiquement le nombre moyen de toppings est trouvable en divisant le nombre de toppings par le nombre de pizzas. On a deux attributs statiques qui compte, l'un le nombre total de pizzas, l'autre le nombre de toppings. Le problème, par contre, est que ces deux attributs statiques sont des valeurs de type int. Si on divise deux entiers, Java arrondit le résultat. Pour savoir la vraie moyenne, non-arrondie, il faut dire à Java qu'elle donne un résultat de type double. On peut faire cela en forçant le type de la première variable à un double :

```
System.out.println(((double) PortionTopping.getNombrePortions()) / (Pizza.getNombrePizzas()));
```