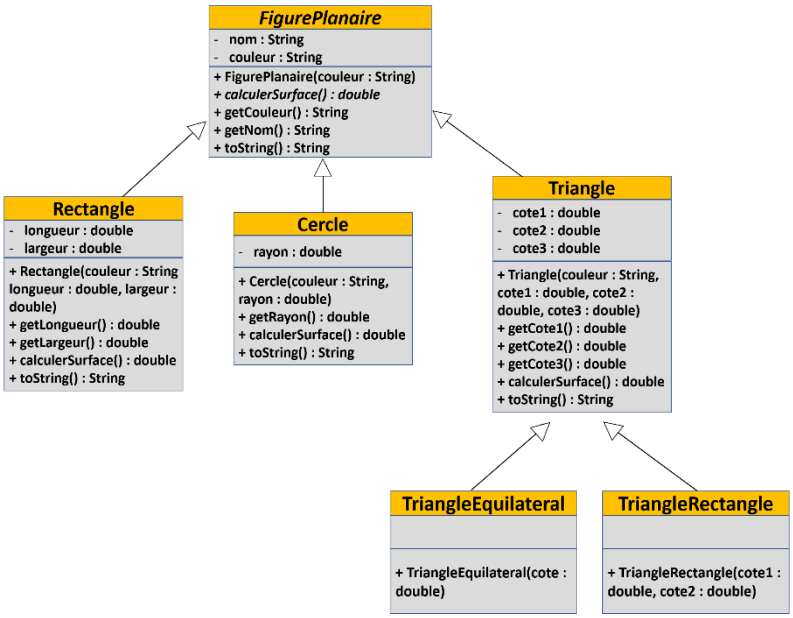
TD 5 Plus sur l'héritage

Dans ce TD nous allons approfondir un peu plus l'idée de l'héritage et nous allons utiliser la notion de classe abstraite. J'inclus à la fin de ce TD un rappel des éléments de géométrie nécessaires (les formules qui donnent les surfaces de différentes figures et la formule pour le volume d'un prisme).

Contexte

Nous allons considérer des diverses figures planaires (par exemple : un cercle, un rectangle, un carré, un triangle, etc.). Nous voulons premièrement calculer la surface de chaque figure, mais en fonction de figure, la surface est calculée selon une formule différente.

Nous partons sur le diagramme de classe suivant :



La classe *FigurePlanaire* sera une classe abstraite (le texte en italique dénote le caractère abstrait d'une méthode ou d'une classe sur les diagrammes de classe) qui aura deux attributs de type String, notamment `nom` et `couleur`. Cette classe aura quelques méthodes concrètes (le constructeur, une méthodes `get` pour les deux attributs et une méthode `toString()` ainsi qu'une méthode abstraite *`surface()`*.

A partir de cette classe nous aurons trois sousclasses concrètes représentant trois figures planaires différentes : un cercle, un rectangle et un triangle. La classe triangle aura elle-même deux sousclasses : `TriangleEquilateral` et `TriangleRectangle`.

ERRATA : dans le diagramme de classe il manque nom : String parmi les paramètres du constructeur de la classe FigurePlanaire. Dans les classe TriangleEquilateral et TriangleRectangle il manque un couleur :

String parmi les paramètres de chaque constructeur. Finalement, dans tout le TP il y a des incohérences entre la méthode surface() et calculerSurface(), qui doivent représenter la même méthode.

Exercice 1

Dans cet exercice le but sera de construire la classe abstraite *FigurePlanaire*.

- Rappel : c'est quoi une méthode abstraite ?

SOLUTION : une méthode abstraite est une méthode dont on déclare seulement la signature, sans la détailler dans la classe. D'habitude pour les méthodes concrètes, nous déclarons la signature, puis on ouvre des accolades et on écrit le code qui corresponde à cette méthode. Au contraire, dans une méthode abstraite, la seule chose qui est écrite est la signature, puis on met directement un ;

- Rappel : comment écrit-on une méthode abstraite ? (syntaxe)

SOLUTION : on déclare une méthode abstraite en utilisant la syntaxe :

```
<mentions> abstract <type de sortie ou void> <nom de la méthode>(<Paramètres>) ;
```

- Rappel : qu'est qu'une classe abstraite ?

SOLUTION : Une classe abstraite est une classe non-instanciable. Une classe abstraite fonctionne comme une classe quelconque : elle peut avoir des attributs, des méthodes concrètes et un ou plusieurs constructeurs. Elle peut également avoir des méthodes abstraites.

- Vrai ou faux : Une classe abstraite doit obligatoirement contenir au moins une méthode abstraite

SOLUTION : Faux. Une classe peut être abstraite même si elle ne contient aucune méthode abstraite. Le choix de faire de votre classe une classe abstraite (ou non) tient plus de votre réponse à la question : est-que j'aurai besoin d'instancier des objets de cette classe ?

- Vrai ou faux : Une classe qui a une méthode abstraite doit impérativement être définie en tant que classe abstraite

SOLUTION : Vrai. Dès qu'on a au moins une méthode abstraite dans une classe, elle doit impérativement devenir une classe abstraite.

- La méthode `toString()` de la classe *FigurePlanaire* retourne : `<nom de la figure>, <couleur>`). Sachant ceci écrivez la classe *FigurePlanaire*.

SOLUTION : Attention premièrement au diagramme de classe. La méthode calculerSurface est une méthode abstraite, ce qui veut dire que le code de la classe devient :

```
public abstract class FigurePlanaire {
	private String nom;
	private String couleur;
```

```
	public FigurePlanaire(String nom, String couleur) {
		this.nom = nom;
		this.couleur = couleur;
	}

	public String getNom() {
		return this.nom;
	}

	public String getCouleur() {
		return this.couleur;
	}

	public abstract double calculerSurface();

	public String toString() {
		return (this.nom + ", " + this.couleur);
	}

}
```

Exercice 2

Pour ce deuxième exercice, nous allons travailler sur les trois classes de *FigurePlanaire* : `Rectangle, Cercle, Triangle`. Ces trois classes sont concrètes.

- Vrai ou faux : les classes `Rectangle, Cercle` et `Triangle` peuvent, mais ne doivent pas impérativement implémenter la méthode surface.

SOLUTION : Faux. Ce sont des classes concrètes qui héritent une méthode abstraite (calculerSurface()) de leur superclasse. Pour rester concrètes, elles DOIVENT impérativement détailler toute méthode abstraite de leur superclasse.

- Rappel : qu'est-ce que les objets de la classe `Rectangle` héritent de la superclasse *FigurePlanaire* ?

SOLUTION : Ils héritent tout attribut et toute méthode de la superclasse ; toutefois les constructeurs sont hérités seulement de façon implicite.

- Vrai ou faux : est-ce qu'on peut avoir le code suivant dans une méthode principale (`main`) ?

```
FigurePlanaire figure = new FigurePlanaire("triangle", "bleu");
```

SOLUTION : Faux. Ceci est une instanciation de la classe FigurePlanaire, qui est abstraite et donc non-instanciable. Le code donnera une erreur de compilation.

- En plus des attributs hérités de la classe *FigurePlanaire*, les objets de type `Rectangle` ont deux attributs de type double, notamment largeur et longueur. Le constructeur de la classe

`Rectangle` mettra le nom de la figure à `"Rectangle"` et les autres attributs de l'objet aux valeurs mis en paramètre du constructeur.

Ecrivez ce constructeur.

SOLUTION : Faux. Ce sont des classes concrètes qui héritent une méthode abstraite (calculerSurface()) de leur superclasse. Pour rester concrètes, elles DOIVENT impérativement détailler toute méthode abstraite de leur superclasse.

- Les objets de type Cercle héritent de leur superclasse un attribut `nom` (mis par défaut à `"Cercle"`), une couleur de type `String` et ils ont également un attribut `rayon` de type `double`. La méthode toString() de la classe Cercle doit retourner le String suivant : `"Cercle, <couleur>, rayon = <rayon>"`. Ecrivez cette méthode.

SOLUTION : Nous pouvons bien sûr écrire cette méthode à partir de zéro, comme on l'a fait pour la classe FigurePlanaire. Mais lorsqu'on utilise l'héritage, l'idée est de se simplifier la vie en utilisant, au plus possible, les méthodes de la superclasse. Un premier instinct devrait être de se demander : qu'est-ce que fait la méthode toString de la classe FigurePlanaire ?

Nous allons donc observer que dans la classe Cercle on attend comme sortie «nom (=Cercle) », « couleur », rayon = « rayon ». La méthode toString() de la superclasse nous donne les premiers deux éléments. C'est pourquoi on utilise la méthode de la superclasse et le code devient alors :

```
@Override
public String toString() {
	return (super.toString() + ", rayon = " + this.rayon);
}
```

- Les objets de type Triangle héritent de leur superclasse un attribut `nom` (mis par défaut à `"Triangle"`), une couleur de type `String` et ils ont également trois attributs de type double, correspondant à la taille de leurs trois côtés. En utilisant les formules ci-dessous écrivez la méthode `surface` de la classe `Triangle`. Pour calculer la racine carrée d'un double `x`, on utilise la méthode statique `Math.sqrt(x)`.

SOLUTION : On utilise la formule donnée pour obtenir le code suivant :

```
@Override
public double calculerSurface() {
	double s = (this.cote1 + this.cote2 + this.cote3)/2;
	return (Math.sqrt(s*(s-this.cote1)*(s-this.cote2)*(s-this.cote3)));
}
```

- Ecrivez également les méthodes surface des classes Rectangle et Cercle, en sachant que la constante π est retrouvable en utilisant l'instruction Math.PI.

SOLUTION : Premièrement pour la classe Rectangle :

```
	@Override
	public double calculerSurface() {
		return (this.longueur * this.largeur);
	}
```

Puis pour la classe Cercle :

```
	@Override
	public double calculerSurface() {
		return (Math.PI * this.rayon * this.rayon);
	}
```

Exercice 3

Dans cet exercice nous allons créer des objets des types mentionnés ci-dessus et nous allons calculer leur surface. Le code dans cet exercice se trouve dans la méthode public static void main(String[] args) d'une classe TD5.

- Ecrivez du code pour créer les figures suivantes :
  - Un cercle rouge de rayon 10
  - Un rectangle bleu 5 par 10
  - Un carré noir de côté 10
  - Un triangle vert avec côtés 3, 5 et 7
  - Un triangle équilatéral marron de côté 3
  - Un triangle rectangle jaune de côtés 5 et 10.

SOLUTION : Ici l'astuce est de regarder le diagramme de classe sur la première page pour les derniers deux objets. Notamment, il va falloir utiliser les constructeurs des sousclasses TriangleEquilateral et TriangleRectangle. Voici le code qui permet de créer ces objets :

```
		final Cercle cercle = new Cercle("rouge", 10);
		final Rectangle rectangle = new Rectangle("bleu", 5, 10);
		final Rectangle carre = new Rectangle("noir", 10, 10);
		final Triangle triangle = new Triangle("vert", 3, 5, 7);
		final TriangleEquilateral triangleEquilateral = new
TriangleEquilateral("marron", 3);
		final TriangleRectangle triangleRectangle = new
TriangleRectangle("jaune", 5, 10);
```

- Nous voulons calculer la surface de chaque figure. À ce but, nous voulons mettre les figures créées dans un tableau qui s'appelle `figures`.
  - Quel sera le type du tableau ?
  - Ecrivez du code qui calcule la surface de chaque forme.

SOLUTION : On a besoin d'un tableau polymorphe, qui regroupe donc des objets de divers types, mais qui sont tous membres d'une sousclasse de la classe FigurePlanaire. Le type donc qui les réunit et qui est le type des éléments de ce tableau polymorphe sera FigurePlanaire[].

Voici le code :

```
final FigurePlanaire[] figures = {cercle, rectangle, carre, triangle, equi, rect};

for (int i = 0; i < figures.length; i++) {
	System.out.println(figures[i]);
	System.out.println("Surface " + figures[i].calculerSurface());
}
```

- Dans la méthode main on a cette ligne de code :

```
final FigurePlanaire figure = new FigurePlanaire("Triangle", "marron");
```

Est-ce que ce code compile ? Pourquoi (ou pourquoi pas) ?

SOLUTION : Toujours non, car encore une fois on essaie d'instancier une classe abstraite.

- Même question pour chacune de ces deux lignes de code :

```
System.out.println(cercle.getRayon());
System.out.println(figures[0].getRayon());
```

SOLUTION :La première ligne compile sans problèmes. Par contre la deuxième ne compile pas, puisque même si figures[0] est un cercle, le type des éléments du tableau figures[] est FigurePlanaire[]. Or, dans la classe FigurePlanaire il n'y a pas de méthode qui s'appelle getRayon().

Exercice 4

Nous allons maintenant étendre notre programme. Nous voulons calculer les modules des prismes avec la base ayant la forme d'une figure planaire donnée. La classe `Prisme` aura deux attributs :

Un attribut `base` de type *`FigurePlanaire`*
Un attribut `hauteur` de type `double`

Elle aura les méthodes suivantes :

Un constructeur avec la signature `Prisme(FigurePlanaire, double)`
Des méthodes `FigurePlanaire getBase()` et `double getHauteur()`
Une méthode `double calculerVolume()` qui calcule le volume d'une prisme

Une méthode `String toString()` qui retourne ”`Prisme, hauteur = <hauteur>, base : <base>`”.

- Ecrivez le constructeur et les méthodes `calculerVolume()` et `toString()` de la classe `Prisme`

SOLUTION : Voici le code :

```
	public double calculerVolume() {
		return (this.base.calculerSurface()*this.hauteur);
	}

	public String toString() {
		return ("Prisme, couleur " + this.base.getCouleur() + ", hauteur " + this.hauteur + ", base " + this.base);
	}
```

- Dans la classe TD5, affichez pour chacune des figures déjà créées, le volume d'un prisme avec ces figures en tant que base, et une hauteur de 10.

Solution : Si on veut le faire dans une seule ligne de code, il faut ajouter cette ligne dans la boucle for de l'exercice précédent (Exo 3.2) :

```
	System.out.println("Volume de la prisme avec cette base " + (new Prisme(figures[i], 10)).calculerVolume());
```

Rappel Math :

Cercle : surface $\pi \cdot r^2$

Rectangle : surface $longueur \cdot largeur$

Triangle : calculez $s = \frac{cote1 + cote2 + cote3}{2}$, et la surface $\sqrt{s(s-cote1)(s-cote2)(s-cote3)}$

Volume prisme : $surface_{base} \cdot hauteur$