1. Constructeur :

Le constructeur sert à initialiser les attributs de la classe.

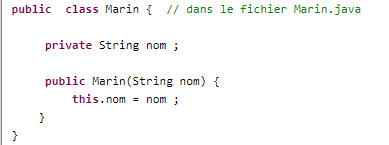
Il est unique en PHP, il peut y en avoir plusieurs en Java.

**Cas 1** : pas de classe parente.

En général, on passe en paramètre l’ensemble des attributs de la classe qui ont besoin d’être initialisés.

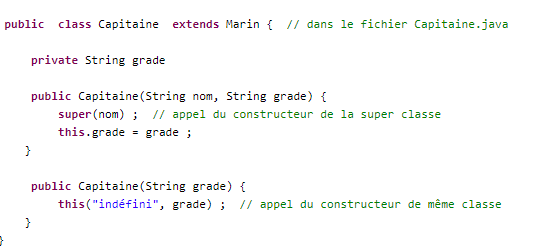
On initialise alors chaque attribut avec sa valeur correspondante passée en paramètre.

Exemple en Java: <http://blog.paumard.org/cours/java/chap04-structure-classe-instanciation.html>



**Cas 2** : une classe parente.

Dans ce cas, on appelle le constructeur de la classe parente pour initialiser les attributs hérités de la classe mère puis on initialise les attributs spécifiques à la classe fille.



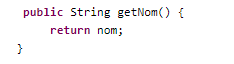
Ici, on est en Java et la classe fille Capitaine dispose de 2 constructeurs, selon que le nom du Capitaine est connu ou pas.

1. Accesseurs :

La bonne pratique : attributs privés. On y accède donc en lecture et écriture par des accesseurs.

**Accès lecture : getter.**

Même exemple classe Marin :



Le getter ne prend aucun paramètre et renvoie l’attribut demandé auquel il correspond.

**Accès écriture : setter.**

Même exemple classe Marin :



Le setter ressemble à un constructeur : il prend en paramètre une valeur de même type que l’attribut à écrire (modifier), l’affecte, mais ne renvoie rien.

1. Comparaison d’objet et affichage d’objet en contexte String:

Parfois en EDC on peut trouver des questions sur la comparaison d’objet (égalité sémantique) et la redéfinition de la méthode equals ou l’affichage de la « valeur » d’un objet en contexte String.

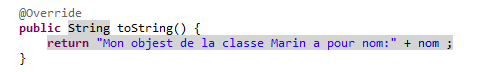
Par exemple, dans une classe utilisant Marin :



Donne un hashcode qui ne veut rien dire : 

On peut définir ce qu’on veut en redéfinissant la méthode toString() dans la classe Marin.

**Exemple** :



Cette fois ci le code la classe utilisatrice :



Va donner le résultat :



1. Test unitaire :

On a :

1. Une classe que l’on teste
2. Une méthode de la classe que l’on souhaite tester
3. Généralement dans le setup() on initialise une ou plusieurs instances d’objet de la classe que l’on teste
4. Un nom de méthode de test

* Une annotation @Test avant la méthode
* Le nom de la méthode qui commence par test

Dans cette méthode de test, généralement :

1. On utilise une librairie de tests unitaires fournissant un ensemble de méthodes de test, par exemple assertEquals, assertTrue …
2. On compare le résultat de la méthode à un résultat ***attendu*** , calculé à la main

* Pour la syntaxe exacte, référez vous à la documentation présente dans l’EDC. Par exemple pour JunitV4, on a quelque chose du genre :

assertEquals(expected, actual);

* Alors que pour une librairie plus moderne comme hamcrest (cf TP) on a :

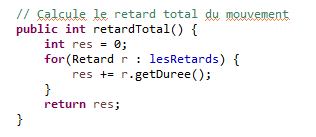
assertThat(actual, is(equalTo(expected)));

Vous remarquerez donc que l’ordre de ce qui est attendu et exécuté est inversé.

Exemple : adapté de l’EDC Aeroplan.

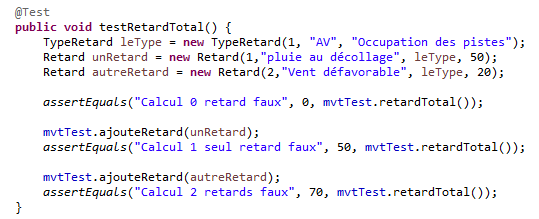
Classe testée : Mouvement

Méthode testée : retardTotal()



lesRetards est une collection (ArrayList) de Retard

Code de la méthode de test :



On effectue 3 tests :

* Premier test : aucun retard, le résultat doit être 0
* Second test : un retard de 50 ajoutée, le résultat doit être 50
* Troisième test : un retard de 20 ajouté, le résultat doit être 70 = 50 + 20.

1. Les collections

On en trouve régulièrement en EDC épreuve 5 du SIO. En général, il y a une annexe qui présente le fonctionnement des collections : initialisation, parcours et quelques méthodes (accès, tri ..) et i y a des questions qui obligent à utiliser cette documentation.

On trouve régulièrement dans les classes un attribut de type collection : rappel : cela marche à l’inverse de la BDD : dans une BDD, la clef étrangère va du côté du multiple (\*) et chaque attribut est **monovalué**.

En programmation, et en particulier en EDC, on trouve des attributs multivalués dans certaines classes, typiquement une collection, pour évaluer la capacité algorithmique des candidats, lecture de document etc

Il y a plusieurs choix de collection en Java suivant l’usage. Cf par exemple :

<https://jmdoudoux.developpez.com/cours/developpons/java/chap-collections.php#collections>

Quelques meilleures pratiques :

**Concernant l’EDC**

* Parcours d’une collection : ***for***

**private** List<**Marin**> equipage = **new** ArrayList<>();

for (Marin unMarin : equipage) {

    // faire ce que vous avez à faire avec un Marin

}

Normalement vous ne devriez plus rencontrer d’Iterator mais …

En PHP : foreach

**Hors périmètre de l’EDC**

* choisir le bon type de collection : cf lien précédent. Type imposé en EDC
* déclarer comme type l’interface plutôt que la classe concrète

Exemple :

List<String> listNomsPirates = new ArrayList<String>();

Plutôt que :

ArrayList<String> listNomsPirates = new ArrayList<String>();

Cela permet de changer la classe concrète facilement selon le besoin.