







# Exercise Class





# Exercice 0

Creez une classe `Player` qui contient les attributs suivants:

-  `name` de type `string`
-  `age` de type `int`
-  `lvl` de type `int`
-  `guilde` de type `string`

Toute seront public et non static.

Faire un `main()` qui instancie un joueur et ses attribut puis afficher les.



# Exercice 1

Creez une classe `Person` représentant une personne avec un nom et un âge.

Votre classe devra avoir les propriétés :



`Name` (le nom de la personne)



`Age` (l'âge de la personne)



# Exercice 1

Et devra avoir une méthode `GetIntroduction` retournant une chaîne de caractères de la forme :

"Bonjour, je m'appelle [Nom] et j'ai [Age] ans".

```
Person p = new Person();  
p.Name = "Albator";  
p.Age = 30;  
Console.WriteLine(p.GetIntroduction()); // Affiche "Bonjour, je m'appelle Albator et j'ai 30 ans"
```



## Exercice 2

Ecrire une classe `BankAccount` représentant un compte bancaire avec un numéro de compte, un solde et un taux d'intérêt.

Votre classe devra avoir les attributs public `balance` et `interestRate` et devra avoir les méthodes suivantes:



`Deposit`: cette méthode prend en paramètre un montant `amount` et ajoute ce montant au solde du compte.



**Withdraw** : cette méthode prend en paramètre un montant **amount** et retire ce montant du solde du compte (en vérifiant que le solde restant est suffisant).



**GetBalance** : cette méthode retourne le solde du compte.



**ApplyInterest** : cette méthode calcule et ajoute les intérêts au solde du compte en utilisant le taux d'intérêt.



## Exercice 2

```
BankAccount b = new BankAccount();  
b.balance = 1000;  
b.interestRate = 0.01;  
  
b.Deposit(500); // Ajoute 500 au solde  
b.Withdraw(200); // Retire 200 du solde  
double balance = b.GetBalance(); // Retourne le solde actuel (1300)  
b.ApplyInterest(); // Ajoute les intérêts au solde (1313)
```



## Exercice 3

Créer une classe `Rectangle` représentant un rectangle avec une largeur et une hauteur.

Votre classe devra avoir les attributs `width` et `height` et devra avoir les propriétés suivantes:

`Width` : cette propriété possède un getter permettant de lire la valeur de l'attribut `width` et un setter permettant de modifier cette valeur (en vérifiant que la valeur est positive).





## Exercice 3

**Height** : cette propriété possède un getter permettant de lire la valeur de l'attribut height et un setter permettant de modifier cette valeur (en vérifiant que la valeur est positive).

**Area** : cette propriété possède un getter permettant de calculer et de retourner l'aire du rectangle ( $\text{width} * \text{height}$ ).


```
Rectangle r = new Rectangle();  
r.Width = 10; // Modifie la largeur du rectangle  
r.Height = 5; // Modifie la hauteur du rectangle  
double area = r.Area; // Retourne l'aire du rectangle (50)
```




## Exercice 4

Ecrire une classe `Person` représentant une personne avec un nom et un âge.

Votre classe devra avoir les propriétés suivantes:

 `Name` (le nom de la personne) qui ne peut que être modifié lors de l'instanciation de l'objet, et pourra être lu à tout moment.

 `Age` (l'âge de la personne) qui pourra être modifié à tout moment.



## Exercice 4

Votre classe devra avoir un constructeur prenant en paramètre le nom de la personne.



## Exercice 5

Ecrire une classe `Circle` représentant un cercle avec un rayon. Votre classe devra avoir l'attribut privé `radius` et devra avoir un constructeur prenant en paramètre le rayon du cercle. Votre classe devra également avoir une méthode `GetArea` retournant l'aire du cercle.

Exemple :

```
Circle c = new Circle(10); // Créer un cercle de rayon 10  
double area = c.GetArea(); // Calcul l'aire du cercle (314,16)
```



## Exercice 6

Ecrire une classe `Employee` représentant un employé avec un nom, un salaire et un poste.

Votre classe devra avoir les attributs privés `name`, `salary` et `position` et devra avoir un constructeur prenant en paramètre le nom, le salaire et le poste de l'employé.

Votre classe devra également avoir une méthode `GetDescription` retournant une chaîne de caractères de la forme

```
"[Nom] travaille comme [Poste] et gagne [Salaire] par mois."
```



## Exercice 6

```
Employee e = new Employee("Albator", 350, "pirate de l'espace");  
// Crée un employé nommé Albator,  
// avec un salaire de 350 et un poste de pirate de l'espace  
string description = e.GetDescription();  
// Retourne "Albator travaille comme pirate de l'espace et gagne 350 par mois."
```



## Exercice 7

Ecrire une classe `Car` représentant une voiture avec un modèle, une marque et une année de modèle. Votre classe devra avoir les attributs privés `model`, `brand` et `year` et devra avoir deux constructeurs:

- 📌 Un constructeur prenant en paramètre le modèle et la marque de la voiture. L'année de modèle sera définie par défaut à 2020.
- 📌 Un constructeur prenant en paramètre le modèle, la marque et l'année de modèle de la voiture.



## Exercice 7

Ajouter une méthode `GetDescription` retournant une chaîne de caractères de la forme

`"[Modèle] de la marque [Marque] de l'année [Année]"`.

```
Car c1 = new Car("Model X", "Tesla");  
// Crée une voiture nommée Model X, de la marque Tesla et de l'année 2020  
Car c2 = new Car("Civic", "Honda", 2010);  
// Crée une voiture nommée Civic, de la marque Honda et de l'année 2010  
string description1 = c1.GetDescription();  
// Retourne "Model X de la marque Tesla de l'année 2020"  
string description2 = c2.GetDescription();  
// Retourne "Civic de la marque Honda de l'année 2010"
```





## Exercice 8

Ecrire une classe `Polygon` représentant un polygone avec un nombre arbitraire de côtés.

Votre classe devra avoir l'attribut privé `sides` et devra avoir trois constructeurs:

- 📌 Un constructeur ne prenant aucun paramètre. Cet constructeur devra définir le nombre de côtés du polygone à 3 par défaut.



## Exercice 8

- 📌 Un constructeur prenant en paramètre le nombre de côtés du polygone.
- 📌 Un constructeur prenant en paramètre un tableau de Point représentant les sommets du polygone. Vous devrez créer une classe Point avec les attributs x et y représentant les coordonnées du point.

Votre classe Polygon devra également avoir une méthode `GetDescription` retournant une chaîne de caractères de la forme "Polygone à [n] côtés."



# Exercice 8

```
Polygon p1 = new Polygon(); // Crée un polygone à 3 côtés
Polygon p2 = new Polygon(4); // Crée un polygone à 4 côtés
Point[] points = { new Point(0, 0), new Point(1, 0), new Point(1, 1), new Point(0, 1) };
Polygon p3 = new Polygon(points); // Crée un polygone à 4 côtés
string description1 = p1.GetDescription(); // Retourne "Polygone à 3 côtés."
string description2 = p2.GetDescription(); // Retourne "Polygone à 4 côtés."
string description3 = p3.GetDescription(); // Retourne "Polygone à 4 côtés."
```



## Exercice 9 Pizza

Ecrire une classe `Pizza` représentant une pizza avec un nom, une liste d'ingrédients et un prix.

Votre classe devra avoir les attributs privés

`name`, `ingredients` (qui sera un tableau) et `price` (double), ainsi que l'attribut public `Size`.

Votre classe devra avoir un constructeur prenant en paramètre le nom et la taille de la pizza.

(Le prix sera défini par défaut à 10.0 et la liste d'ingrédients sera vide.)



## Exercice 9 Pizza

Les tailles de pizza pourront etre :

"small", "medium", "large", "extra large" (si la taille ne correspond à aucune de ces valeurs, la taille sera "medium" par défaut)

Votre classe devra également avoir les méthodes suivantes:



# Exercice 9 Pizza



**AddIngredient** : qui prend en paramètre un ingrédient et qui l'ajoute à la liste des ingrédients de la pizza.



**RemoveIngredient** : qui prend en paramètre un ingrédient et qui le retire de la liste des ingrédients de la pizza (si cet ingrédient fait partie de la liste).



**GetPrice** : qui retourne le prix de la pizza en fonction de la taille et des ingrédients. (1€ par ingrédient et 2€ par taille de plus)



## Exercice 9 Pizza



**GetIngredients** : qui retourne la liste des ingrédients de la pizza.



**SetPrice** : qui prend en paramètre un prix et qui modifie le prix de la pizza.

Votre classe devra également avoir une surcharge de la méthode **AddIngredient** qui prend en paramètre un tableau d'ingrédients et qui ajoute tous ces ingrédients à la liste des ingrédients de la pizza. (vérifier a ne pas avoir de doublons dans la liste des ingrédients).





## Exercice 10

Ecrire une classe `Contact` représentant un contact avec un nom, un numéro de téléphone et une adresse email.

Il devra avoir les attributs private

```
name, phoneNumber (string contenant uniquement des chiffres) et  
email (contenant un @ obligatoirement)
```

et devra avoir un constructeur prenant en paramètre le nom, le numéro de téléphone et l'adresse email du contact.

Et un constructeur par default qui initialisera des chaines de caractères vides pour les attributs.





## Exercice 10

Votre classe devra également avoir les méthodes suivantes:



**GetName** : qui retourne le nom du contact.



**GetPhoneNumber** : qui retourne le numéro de téléphone du contact.



**Email** : qui retourne l'adresse email du contact.



**SetPhoneNumber** : qui prend en paramètre un numéro de téléphone et qui modifie le numéro de téléphone du contact.



## Exercice 10



`SetEmail` : qui prend en paramètre une adresse email et qui modifie l'adresse email du contact.

Ecrire une classe `ContactManager` représentant un gestionnaire de contacts.

Votre classe devra avoir l'attribut privé `contacts` qui sera un tableau de `Contact`.

Votre classe devra avoir un constructeur qui prendra un contact.



## Exercice 10

Elle devra également avoir les méthodes suivantes:



**AddContact** : qui prend en paramètre un contact et qui l'ajoute au tableau de contacts. (attention pas de doublons dans pour les numéros de téléphone et ou email)



**RemoveContact** : qui prend en paramètre un numero de telephone et qui le retire du tableau de contacts associer a celui-ci.



## Exercice 10



`ShowContacts` : qui affiche tous les contacts du tableau de contacts.

Votre classe ne pourra avoir que 10 contacts maximum. Dans le cas où le tableau de contacts est plein, la méthode `AddContact` devra retourner dans le terminal `Le gestionnaire de contacts est plein.`



## Exercice 11

Ecrire une classe `Event` représentant un événement avec un nom, une date et une description.

Il devra avoir les attributs private

`name`, `date` (string au format "dd/mm/yyyy") et `description` et devra avoir un constructeur prenant en paramètre le nom, la date.

La description sera vide par défaut et non obligatoire pour les events mais la date et le nom le seront.



# Exercice 11

Ecrire une classe `Calendar` représentant un calendrier avec un nom et une liste d'événements.



Il devra avoir un attribut privé `event` qui sera un tableau d'Event.

Votre classe devra avoir un constructeur par default qui initialisera un tableau vide pour les attributs.

Il devra également avoir les méthodes suivantes:



# Exercice 11

-  **addEvent** : qui prend en paramètre un événement et qui l'ajoute au tableau d'événements. (attention pas de doublons dans pour les noms d'événements)
-  **removeEvent** : qui prend en paramètre un nom d'événement et qui le retire du tableau d'événements associé à celui-ci.



# Exercice 11



`showEvents` : qui affiche tous les événements du tableau d'événements si aucun paramètre ne lui a été donné. Sinon elle affiche les événements qui ont lieu à la date donnée en paramètre.

Rajouter le nécessaire à votre classe `event` pour que la classe `calendar` puisse fonctionner correctement.





## Exercice 12

Libre de choix, écrire une classe qui vous plait et qui vous semble utile.

Elle devra avoir au moins 3 attributs et 3 méthodes.

Elle devra avoir un constructeur par default et un constructeur avec paramètres.

Un overload est requis en plus des méthodes demandées.

Essayez de faire quelque chose de complexe et qui vous plait.



## Exercice 13

Représenter le jeu du morpion en utilisant des classes.

Vous devrez avoir une classe `Board` qui représente le plateau de jeu.

Vous devrez avoir une classe `Player` qui représente un joueur.

Et une classe `Game` qui représente la partie.

La classe `Game` devra avoir un attribut `board` qui sera un objet de la classe `Board` et un attribut `players` qui sera un tableau de 2 objets de la classe `Player`.

Elle devra avoir une méthode `startGame` qui lancera la partie.