

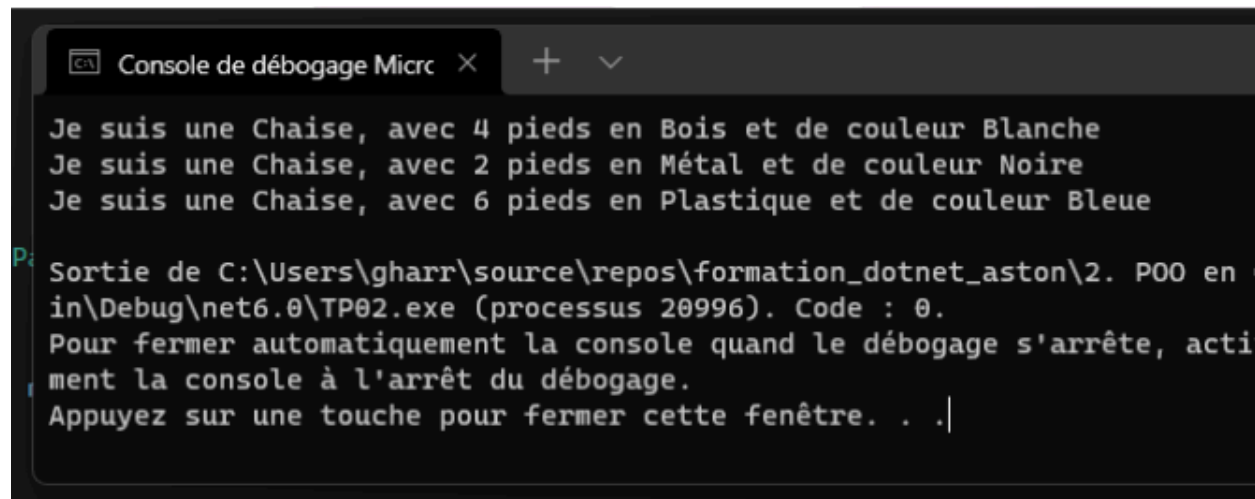
# Exercices sur la POO

# Chaise

## Sujet

1. Créer une classe **Chaise** possédant comme variables d'instance le nombre de pieds, le matériaux et la couleur de l'objet
2. Afficher ses informations en surchargeant une méthode de la classe Object
3. La classe Chaise pourra être instanciée avec ou sans paramètres (Constructeur par défaut)
4. Afficher toutes les chaises (Possibilité de simplifier avec une méthode ToString)

## Exemple



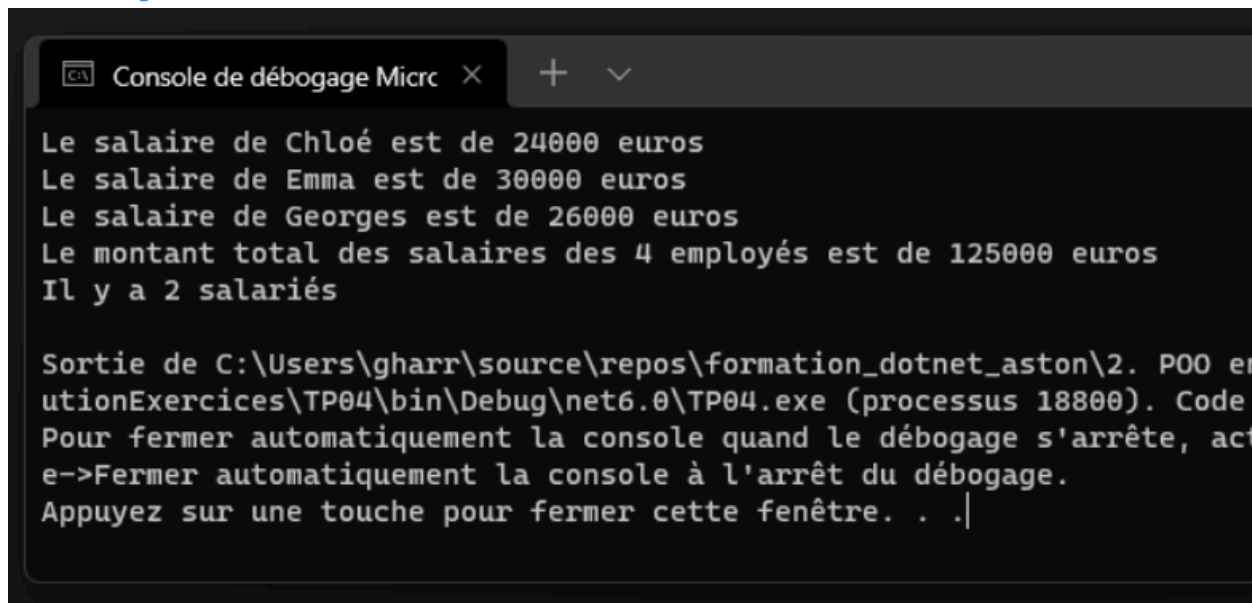
```
Console de débogage Micrc x + v
Je suis une Chaise, avec 4 pieds en Bois et de couleur Blanche
Je suis une Chaise, avec 2 pieds en Métal et de couleur Noire
Je suis une Chaise, avec 6 pieds en Plastique et de couleur Bleue
Sortie de C:\Users\gharr\source\repos\formation_dotnet_aston\2. P00 en in\Debug\net6.0\TP02.exe (processus 20996). Code : 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre. . . |
```

# Salarié

## Sujet

1. Créer une classe **Salarié** ayant pour attributs : le matricule, le service, la catégorie, le nom et le salaire de l'employé
2. Cette classe aura également comme méthode **AfficherSalaire()**
3. Cette classe pourra, via deux champs et une méthode, permettre de savoir le nombre total d'employés, le salaire total et remettre à zéro la valeur du nombre d'employés dans l'entreprise
4. Créer une IHM pour tester le fonctionnement de l'application

## Exemple



```
Console de débogage Micr... x + v

Le salaire de Chloé est de 24000 euros
Le salaire de Emma est de 30000 euros
Le salaire de Georges est de 26000 euros
Le montant total des salaires des 4 employés est de 125000 euros
Il y a 2 salariés

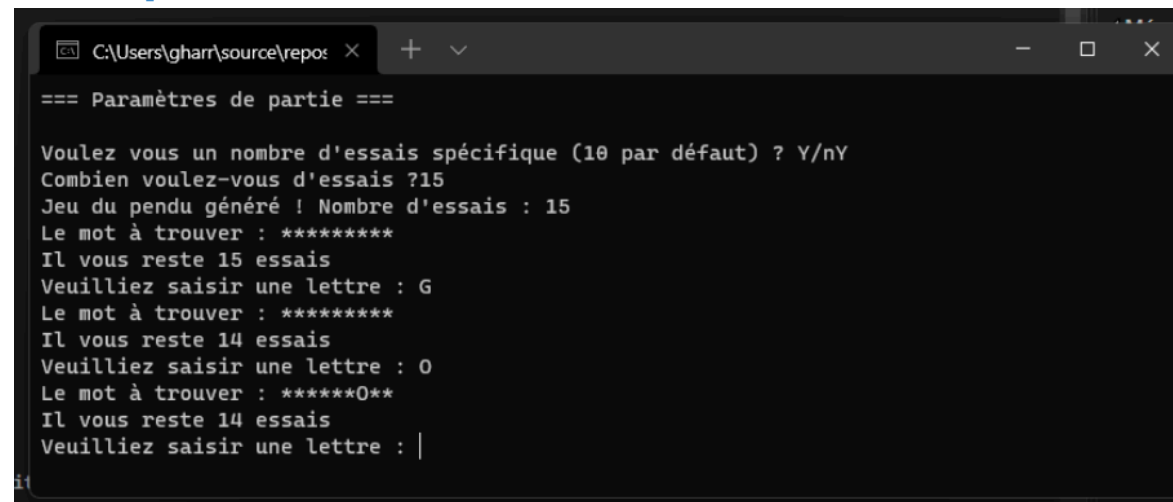
Sortie de C:\Users\gharr\source\repos\formation_dotnet_aston\2. P00 e...
utionExercices\TP04\bin\Debug\net6.0\TP04.exe (processus 18800). Code
Pour fermer automatiquement la console quand le débogage s'arrête, act
e->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre. . .|
```

# Pendu

## Sujet

1. Réaliser un jeu du pendu en créant une classe **Pendu** qui possédera au minimum comme attributs : le **masque**, le **nombre d'essais** ainsi que le **mot à trouver**. Cette classe aura comme méthodes : **TestChar()**, **TestWin()** et **GenererMasque()**.
2. Le joueur aura **par défaut 10 chances** pour gagner.
3. Utiliser **une autre classe** servant à **générer les mots** pour le jeu, à partir d'un tableau d'entrées potentielles
4. Optionnellement, le joueur pourra **choisir un nombre de coups** pour sa partie
5. Créer une **IHM** pour tester l'application

## Exemple



```
C:\Users\gharr\source\repos >
=== Paramètres de partie ===
Voulez vous un nombre d'essais spécifique (10 par défaut) ? Y/nY
Combien voulez-vous d'essais ?15
Jeu du pendu généré ! Nombre d'essais : 15
Le mot à trouver : *****
Il vous reste 15 essais
Veuillez saisir une lettre : G
Le mot à trouver : *****
Il vous reste 14 essais
Veuillez saisir une lettre : O
Le mot à trouver : *****O**
Il vous reste 14 essais
Veuillez saisir une lettre : |
```

## Citernes

Le but est de créer une classe qui décrit une citerne d'eau **WaterTank**.

- Elle aura un **poids à vide**, une **capacité totale** et un **niveau de remplissage**.
- La classe proposera également les méthodes suivantes :
  - Une méthode indiquant **le poids total** de la citerne.
  - Une méthode pour **remplir** la citerne avec un **nombre de litre** d'eau.
  - Une méthode pour **vider** la citerne d'eau d'un **nombre de litre** d'eau.
- La classe possédera, également, un attribut pour la **totalité des volumes** des citernes d'eau.
- Créez un programme pour tester votre classe (**IHM**)

Une fois fini ajouter la gestion des cas où la citerne est vide ou déborde :

- la méthode pour remplir renverra **l'excès** d'eau
- la méthode pour vider ne renverra que **l'eau disponible**

```
Poids total de la citerne 1 : 20  
Poids total de la citerne 2 : 15
```

```
-----  
Quantité d'eau dans la citerne 1 : 10  
Quantité d'eau dans la citerne 2 : 10  
Quantité d'eau dans toutes les citernes : 20
```

```
-----  
Quantité d'eau dans la citerne 1 après ajout de 11 litres: 20/20  
Excès d'eau récupéré : 1  
Quantité d'eau dans la citerne 2 après tentative de retrait de 11 litres: 0/10  
Quantité d'eau récupérée : 10
```

```
-----  
Quantité d'eau dans la citerne 1 : 20  
Quantité d'eau dans la citerne 2 : 0  
Quantité d'eau dans toutes les citernes : 20
```

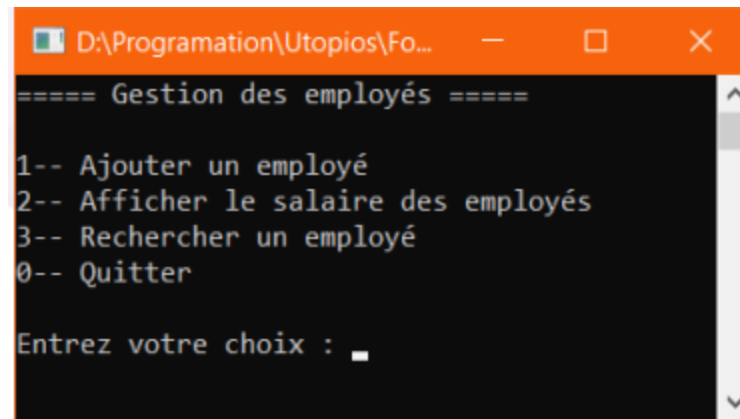
# Salariés avec Heritage

## Sujet

Créer une classe **Commercial** en dérivant la classe **Salarie** de l'exercice Salarié. Cette classe aura 2 propriétés supplémentaires pour calculer un montant de commission : **chiffre d'affaire** et **commission en %**.

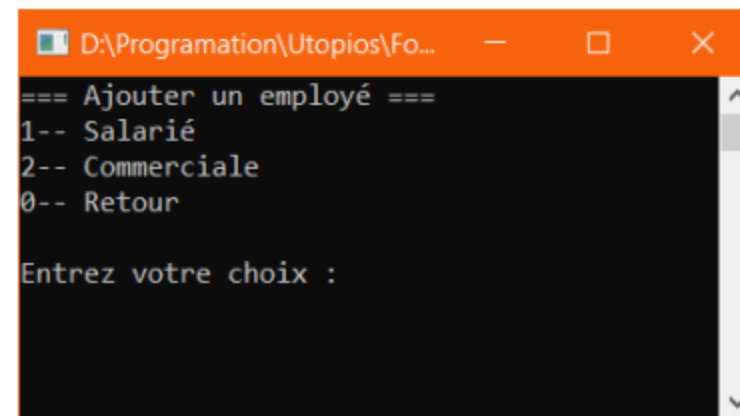
- Créer les **deux constructeurs** de la classe Commercial. Ne pas oublier d'appeler les constructeurs équivalents de **la classe de base** (mère).
- Surcharger la méthode **AfficherSalaire()** pour calculer le salaire réel (**fixe + commission**).
- Ajoutez des méthodes **ToString** pour l'affichage des salariés et commerciaux.
- Écrire un programme (**IHM**) qui permet à une entreprise de 20 Employés (Salariés et commerciaux) :
  - D'**ajouter** des employés
  - D'**afficher les salaires** de chaque employé
  - De **rechercher** un employé par le début de son nom et afficher son **salaire**

## Exemple



```
===== Gestion des employés =====
1-- Ajouter un employé
2-- Afficher le salaire des employés
3-- Rechercher un employé
0-- Quitter

Entrez votre choix : _
```



```
=== Ajouter un employé ===
1-- Salarié
2-- Commerciale
0-- Retour

Entrez votre choix :
```

## Salariés avec Heritage (suite exemples)

```
D:\Programation\Utopios\FormationC#...
=== Ajouter un employé ===
1-- Salarié
2-- Commerciale
0-- Retour

Entrez votre choix : 1
Merci de saisir le nom : Apeupré Jean-Michel
Merci de saisir le matricule : M001
Merci de saisir la categorie: C001
Merci de saisir le service : S001
Merci de saisir le salaire: 1800
```

```
D:\Programation\Utopios\FormationC#-NET\Codes\Découv...
=== Ajouter un employé ===
1-- Salarié
2-- Commerciale
0-- Retour

Entrez votre choix : 2
Merci de saisir le nom : Duss Jean-Claude
Merci de saisir le matricule : M002
Merci de saisir la categorie: C002
Merci de saisir le service : S002
Merci de saisir le salaire: 1300
Merci de saisir le chiffre d'affaire du commerciale : 13000
Merci de saisir la commission : 3
```

```
D:\Programation\Utopios\FormationC#-NET\Codes\Découv...
===== Salaire des employés =====
-----
TpClasseSalarieHeritage.Classes.Salarie
Le salaire fixe de Apeupré Jean-Michel est de 1800 euros
-----
TpClasseSalarieHeritage.Classes.Commerciale
Je suis un commerciale
Le salaire fixe de Duss Jean-Claude est de 1300 euros
Le salaire avec commission de Duss Jean-Claude est 1690 euros
-----
===== Gestion des employés =====
1-- Ajouter un employé
2-- Afficher le salaire des employés
3-- Rechercher un employé
0-- Quitter

Entrez votre choix :
```

```
D:\Programation\Utopios\FormationC#-NET\Codes\Découv...
===== Recherche employé par nom =====
Merci de saisir le nom : Duss Jean-Claude
Le salaire fixe de Duss Jean-Claude est de 1300 euros
Le salaire avec commission de Duss Jean-Claude est 1690 euros
===== Gestion des employés =====
1-- Ajouter un employé
2-- Afficher le salaire des employés
3-- Rechercher un employé
0-- Quitter

Entrez votre choix :
```

# Compte bancaire

## Sujet

1. Créer une classe **abstraite** **CompteBancaire**  
Cette classe aura : un **solde**, un **client** et une **liste d'opérations**(dépôt ou retrait).
2. Créer les classes : **ComptePayant**, **CompteEpargne**, **CompteCourant** qui héritent de **CompteBancaire**.
3. Créer une classe **Client** avec les attributs suivants : **nom**, **prénom**, **identifiant**, **liste des comptes** et numéro de téléphone. On créera le client au début de l'application
4. Créer une classe **Opération** avec les attributs suivants : **numéro**, **montant** et **statut** (depot/retrait en **enum**)
5. Créer une **IHM** pour tester l'application. Pour un compte au choix de l'utilisateur on pourra effectuer un **dépôt**, un **retrait** ou **afficher le solde et les opérations**

## Exemple

```
C:\Users\gharr\source\repo: x + v
=== Menu Principal ===
1. Lister les comptes bancaires
2. Créer un compte bancaire
3. Effectuer un dépôt
4. Effectuer un retrait
5. Afficher les opérations et le solde
0. Quitter le programme
Votre choix : 2
=== Création de Compte ===
1. Créer un compte courant
2. Créer un compte épargne
3. Créer un compte payant
0. Annuler la création de compte
Votre choix : |
```



# Figure

## Sujet

1. Créer une classe **Point** possédant comme attributs **posX: double** et **posY: double** ainsi qu'une méthode **ToString()**
2. Créer une classe **abstraite Figure** possédant un attribut **origine** de type **Point**  
Les classes créées ensuite **hériteront** de **Figure**
3. Créer une interface **IDeplacable** contenant la méthode **Deplacement(double, double)** permettant de déplacer l'**origine** de la figure  
La classe **Figure** l'implémentera
4. Créer une classe **Carré** ayant comme nouvel attribut son **côté**
5. Créer une classe **Rectangle** ayant comme nouveaux attributs sa **longueur** et sa **largeur**
6. Créer une classe **Triangle** ayant comme attributs sa **base** et sa **hauteur** (ce triangle sera **isocèle**)
7. Toutes les Figures auront une méthode **ToString** et la méthode **Deplacement** implémentée
8. Réaliser une **IHM** pour tester l'application

## Exemple

```
Coordonnées du carré ABCD (Coté = 2) :
A = 2;4
B = 4;4
C = 4;2
D = 2;2
-----
Coordonnées du rectangle ABCD (Longueur = 3, Largeur = 5) :
A = 2;4
B = 5;4
C = 5;-1
D = 2;-1
-----
Coordonnées du triangle ABCD (Base = 4, Hauteur = 5) :
A = 2;4
B = 4;-1
C = 0;-1
-----
Déplacement du carré par (1,3) :
Coordonnées du carré ABCD (Coté = 2) :
A = 3;7
B = 5;7
C = 5;5
D = 3;5
```

# La pile

## Sujet

1. Créer une classe **Pile<T>** contenant un attribut `T[] elements`
2. Ajouter une méthode permettant d'**empiler** un nouvel élément
3. Ajouter une méthode permettant de **dépiler** le dernier élément empilé
4. Ajouter une méthode permettant de **recupérer** un élément par son index et ainsi de le **retirer** de la pile
5. Créer une IHM séparée en 3 parties : une pour une pile de **string**, une pour une pile de **decimal** et une pour une pile d'objet **Personne**(nom, prenom, age) (on pourra utiliser des méthodes génériques pour faciliter la saisie)

*LIFO : Last In First Out => Pile/Stack*

*FIFO : First In First Out => File/Queue*

## Exemple

```

C:\Users\gharr\source\repos x + v

=== Menu Principal ===

1. Empiler
2. Dépiler
3. Récupérer à X
0. Quitter
Votre choix : 1

Valeur à empiler : Test
Test a été ajoutée à la pile !

=== Menu Principal ===

1. Empiler
2. Dépiler
3. Récupérer à X
0. Quitter
Votre choix : 3

Veuillez donner un indice :1
La valeur trouvée à l'indice 1 est : Test

=== Menu Principal ===

1. Empiler
2. Dépiler
3. Récupérer à X
0. Quitter
Votre choix : |
    
```

# Thermomètre

Créer une classe **Thermometre** avec :

- un seul attribut `double _temperatureKelvin` contenant la température du thermomètre
- 3 propriétés permettant d'accéder et modifier cet attribut en **Kelvin**, **Celsius** et **Fahrenheit** (chercher les conversions correspondantes)
- 1 constructeur prenant en paramètre une température et une unité de type enum **UniteTemperature** (Kelvin, Celsius ou Fahrenheit)
- Réaliser une **IHM** pour tester l'application
- Si vous voulez aller plus loin, ajouter 2 membres utilisant le mot clé [operator](#) permettant de changer la température en Kelvin du thermomètre de cette manière :

```
Thermometre th = new Thermometre(10, UniteTemperature.Celcius)

th = th + 20.5;
```

## Travailleur et Scientifique

1. Créer une classe **Personne**, idéalement abstract, contenant le nom de la personne, son prénom, son numéro de téléphone, et son email. Une méthode ToString pour afficher les données de la personne.
2. Créer une classe **Travailleur**, la classe Travailleur hérite de la classe personne et étend avec les attributs nom d'entreprise, adresse entreprise et téléphone professionnel. Une méthode ToString pour afficher les données.
3. Créer une classe **Scientifique**, la classe Scientifique hérite de la classe Travailleur et étend avec les disciplines (physique, chimie, mathématique, ...) et types du scientifique (théorique, expérimental, informatique...) Une méthode ToString pour afficher les données
4. Réaliser une **IHM** pour tester l'application

# Forum

## Sujet

1. Créer une classe **Forum** ayant les attributs suivants : **nom**, **dateCreation**, **abonnes[]**, **nouvelles[]** et **modérateur**
2. Créer une classe **Modérateur** et **Abonné** héritants d'une classe abstraite **Utilisateur** contenant les attributs suivants : **prenom**, **nom**, **age**.  
L'**abonné** peut **ajouter** et **consulter** une nouvelle.  
Le **modérateur** peut **supprimer une nouvelle**, **bannir un abonné**, **ajouter un abonné** ou **lister les abonnés et les nouvelles**.
3. Créer une classe **Nouvelle** contenant les attributs suivant : **sujet**, **descriptif**
4. Réaliser une **IHM** pour tester l'application

## Exemple

```
C:\Users\gharr\source\repos x + v
=== Premiere étape : Création du Forum ===

Quel est le nom de ce forum ? Test
Combien d'abonnés ce forum aura-t-il ? 10
Combien de nouvelles ce forum aura-t-il ? 10
Ce forum aura-t-il un modérateur ? Y/nY
=== Affectation d'un modérateur au forum ===

Quel est le nom du modérateur ? Modo
Quel est le prénom du modérateur ?
Quel est l'âge du modérateur ? 29
=== Menu Principal ===

1. Voir les abonnés
2. Ajouter un abonné
3. Bannir un abonné
4. Voir les nouvelles
5. Consulter une nouvelle
6. Ajouter une nouvelle
7. Répondre à une nouvelle
8. Supprimer une nouvelle
0. Quitter le programme
|
```

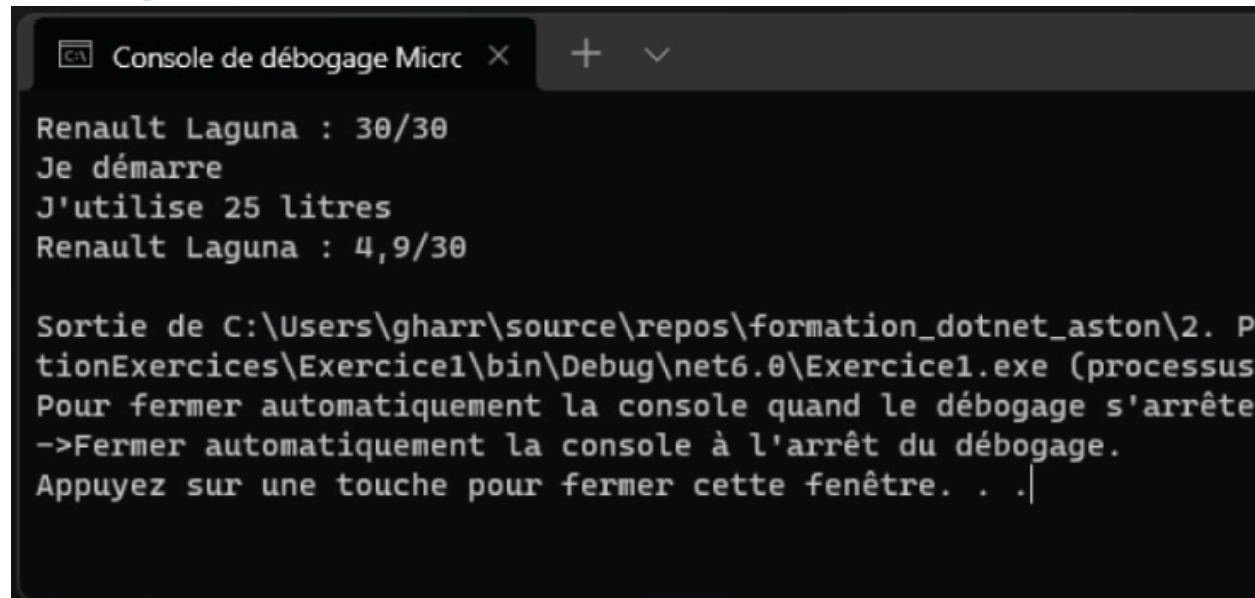
# Voiture

## Sujet

1. Créer une classe abstraite **Véhicule** contenant les attributs suivants et les initialiser à l'aide du constructeur : marque et modèle  
Ajouter les méthodes suivantes à la classe Véhicule : **démarrer(): bool**, **arrêter(): void**, **faireLePlein(double): void**
2. Créer une classe **Moteur** contenant les attributs suivants : volumeReservoir, volumeTotal, estDemarre.  
Cette classe possédera les méthodes de véhicule. démarrer() consommera cependant 1/10 de litre
3. Créer une classe abstraite héritant de Voiture nommée **VéhiculeAMoteur** ayant une propriété moteur qui servira à déléguer les trois méthodes héritées de Véhicule

4. Créer une classe héritant de VéhiculeAMoteur nommée **Voiture** qui servira à la construction d'une Peugeot 206
5. Créer une IHM pour tester le fonctionnement de l'application

## Exemple



```

Console de débogage Micrc
Renault Laguna : 30/30
Je démarre
J'utilise 25 litres
Renault Laguna : 4,9/30

Sortie de C:\Users\gharr\source\repos\formation_dotnet_aston\2. P
tionExercices\Exercice1\bin\Debug\net6.0\Exercice1.exe (processus
Pour fermer automatiquement la console quand le débogage s'arrête
->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre. . .
  
```

# L'hôtel

## Sujet

1. Créer une classe **Client** possédant : un **identifiant**, un **nom**, un **prénom** et un **numéro de téléphone**
2. Créer une classe **Chambre** ayant : un **numéro**, un **statut** (libre/occupé/en nettoyage de type enum), un **nombre de lits** et un **tarif**.
3. Créer une classe **Réservation** possédant : un **identifiant**, un **statut** (prévu/en cours/fini/annulé), une **liste de chambres** et un **client**
4. Créer une classe **Hotel** comportant : une **liste de clients**, une **liste de chambres** et une **liste de réservations**
5. Créer une **IHM** pour tester l'application

## Exemple

```

C:\Users\gharr\source\repos x + v
Quel est le nom de l'hôtel ? Excelsior
Excelsior créé avec succès !
=== Menu Principal ===

1. Ajouter un client
2. Afficher la liste des clients
3. Afficher les réservations d'un client
4. Ajouter une réservation
5. Annuler une réservation
6. Afficher la liste des réservations
0. Quitter
Votre choix : 1
=== Ajout d'un client ===

Quel est le nom du client ? MARTIN
Quel est le prénom du client ? Albert
Quel est le téléphone du client ? 0123456789
Client ajouté avec succès !

=== Menu Principal ===

1. Ajouter un client
2. Afficher la liste des clients
3. Afficher les réservations d'un client
4. Ajouter une réservation
5. Annuler une réservation
6. Afficher la liste des réservations
0. Quitter
Votre choix : |
  
```

# Caisse enregistreuse

## Sujet

1. Créer une classe **Caisse** possédant une collection de produits et une collections de ventes (Listes)
2. Créer une classe **Produit** comportant un numéro, un nom, un prix et un stock
3. Créer une classe **Vente** possédant un numéro, une liste de produits, un état (enum EnCours, Validée, Annulée), une méthode permettant de valider la vente, ce qui confirmera le paiement et changera le stock des produits et une méthode permettant d'annuler la vente
4. Créer une classe **PaiementCB** et une classe **PaiementEspece** dérivant de **Paiement**, qui auront un identifiant de référence, une date ainsi qu'une méthode pour payer
5. Créer une IHM pour tester l'application

## Exemple d'IHM incomplète

```
C:\Users\gharr\source\repos: x + v

=== Menu Principal ===

1. Voir les produits
2. Ajouter un produit dans la caisse
3. Faire une vente
0. Quitter
Votre choix : 2
Quel est le nom du produit ? Pomme
Quel est le prix du produit ? 2
Quel est le stock du produit ? 15
Le produit a été ajouté avec succès !

=== Menu Principal ===

1. Voir les produits
2. Ajouter un produit dans la caisse
3. Faire une vente
0. Quitter
Votre choix : 1
=== Liste des produits ===

1. Pomme : 2 Euros (15 restants)

=== Menu Principal ===
```



## Vols

Dans cet exercice on s'intéresse à créer des classes pour gérer les vols d'une compagnie aérienne qui organise des vols entre des villes.

Plus précisément on s'intéressera aux plans de vol entre les différentes villes.

Càd les vols disponibles ainsi que l'heure de départ.

Créer une classe VolDirect qui représentera un vol direct entre deux villes (pas d'escale dans une ville intermédiaire), on doit :

- Définir le constructeur de cette classe qui a quatre attributs :
  - Dep et Arr qui désigne respectivement la ville de départ et la ville d'arrivée
  - Jour qui désigne le jour de la semaine (lundi, mardi, ...)
  - Heure (un entier entre 0 et 24 qui représente l'heure de départ )
- Écrire une méthode affiche() qui affiche une chaîne bien formatée de la forme :  
« Ce vol part de Paris vers Marseille le lundi à 9 heure »

Créer une classe Vols qui représente tous les vols le long de la semaine en utilisant la classe VolDirect. Pour ce faire on doit :

- Définir le constructeur de cette classe avec un seul attribut qui est une liste de VolDirect
- Écrire une méthode Liste\_successeurs qui retourne une liste contenant les villes arrivées d'une ville de départ passée comme paramètre
- Écrire une méthode Appartient qui vérifie si une ville appartient au plan du vol que ce soit comme ville d'arrivée ou de départ
- Écrire une méthode Affiche qui affiche tous les vols directs.

## Vol (suite)

Écrire un programme principal permettant de :

1. Créer une liste LV d'objets Vol\_direct, on suppose avoir définie les 3 fonctions suivantes :  
Saisie\_Jour qui retourne un jour valide,  
Saisie\_Heure qui retourne une heure valide  
Saisie\_Ville qui retourne un nom de ville valide.
2. Créer un objet Vol nommé V à partir de la liste déjà créée
3. Afficher tous les vols
4. Saisir une ville qui doit appartenir au plan du vol puis calculer et afficher la liste de ses successeurs

## Exemple

```
=== Liste des vols ===  
Ce vol part de Paris vers Marseille le 17 à 4 heure  
Ce vol part de Paris vers Lyon le 21 à 8 heure  
Ce vol part de Marseille vers Lyon le 11 à 17 heure  
Ce vol part de Paris vers Bruxelles le 4 à 20 heure  
  
La ville Paris fait partie du plan de vol !  
La ville Bruxelles fait partie du plan de vol !  
La ville Bordeaux ne fait pas partie du plan de vol !  
  
La liste des destinations à partir de Paris est : {'Lyon', 'Marseille', 'Bruxelles'}
```