

I- INITIATION A LA PROGRAMMATION C-SHARP

Exercice 1 :

Objectif :

- les variables
- les conditions
- les boucles
- structure du code.

Énoncé:

Ecrire un programme qui réalise la conversion du Celsius vers Fahrenheit ou l'inverse. Le programme demande à l'utilisateur quelle conversion nous souhaitons effectuer, Celsius vers Fahrenheit ou l'inverse.

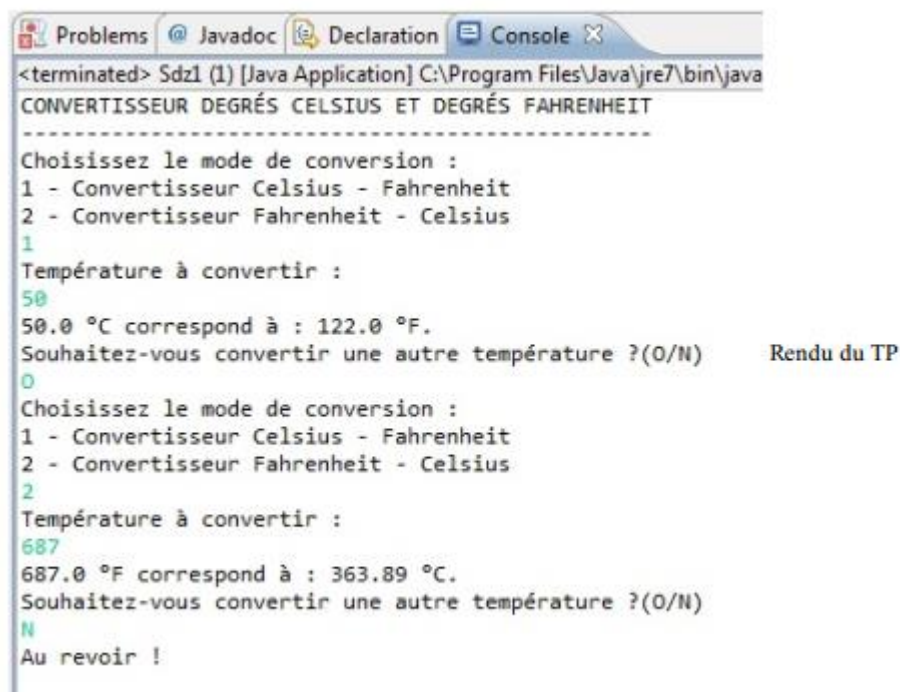
on n'autorise que les modes de conversion définis dans le programme (un simple contrôle sur la saisie fera l'affaire) ;

enfin, on demande à la fin à l'utilisateur s'il veut faire une nouvelle conversion, ce qui signifie que l'on doit pouvoir revenir au début du programme !

À toutes fins utiles, voici la formule de conversion pour passer des degrés Celsius en degrés Fahrenheit :

$$F = \frac{9}{5} * C + 32$$

La figure suivante est un aperçu du rendu



II- INITIATION A LA PROGRAMMATION ORIENTEE OBJECT (POO)

EXERCICE 1 :

Objectif:

- Créer une classe dérivée.
- Ajouter des méthodes à une classe dérivée.
- Redéfinir des méthodes dans une classe dérivée.

Énoncé:

Définir une classe **Vecteurs2D** caractérisée par l'abscisse **X** et l'ordonnée **Y**, ainsi qu'un attribut qui renseigne sur le nombre de vecteurs créés lors de l'exécution du programme.

Définir les constructeurs (par défaut, d'initialisation et de recopie), et les accesseurs aux différents attributs.

Définir les méthodes **ToString** et **Equals**, la première retourne une chaîne de caractères représentant l'abscisse et l'ordonnée du vecteur comme suit : $X = 1.5 - Y = 2$, la deuxième retourne **True** lorsque l'abscisse et l'ordonnée des deux vecteurs sont égaux.

1. Définir une méthode **Norme** qui retourne la norme d'un vecteur, cette méthode peut être redéfinie dans les classes dérivées.
2. Définir une classe **Vecteurs3D** dérivée de la classe **Vecteur2D** qui contient, en plus des propriétés de la classe de base, la propriété **Z** symbolisant la troisième dimension.
3. Définir les constructeurs (par défaut, d'initialisation et de recopie) de cette classe, ainsi que les accesseurs des propriétés.
4. Redéfinir les méthodes **ToString** et **Equals** pour intégrer la troisième propriété.
5. Redéfinir la méthode **Norme** pour qu'elle retourne la norme d'un vecteur dans l'espace.
6. Définir un programme de test permettant de créer deux objets de type **Vecteurs2D** et deux autres de type **Vecteurs3D**. Afficher les informations de tous les objets, et tester les méthodes **Equals** et **Norme**. Afficher le nombre d'objets créés.

Exemple d'exécution :

```
****Vecteur 2D****
Vecteur 1:
Donner X: 1
Donner Y: 2
X=1 Y=2
La norme est: 2,23606797749979
Vecteur 2:
Donner X: 1
Donner Y: 2
X=1 Y=2
La norme est: 2,23606797749979
Les deux vecteurs sont identiques
```

```

Les deux vecteurs sont identiques
****Vecteur 3D****
Vecteur 1:
Donner X: 1
Donner Y: 1
Donner Z: 1
X=1 Y=1 Z=1
La norme est: 1,73
Vecteur 2:
Donner X: 1
Donner Y: 2
Donner Z: 1
X=1 Y=2 Z=1
La norme est: 2,44
Non identiques
Le nombre de vecteurs créés est: 4

```

EXERCICE 2 :

Soit les informations suivantes :

- Un compte bancaire possède à tout moment une donnée : son solde. Ce solde peut être positif (compte créditeur) ou négatif (compte débiteur).
- Chaque compte est caractérisé par un code incrémenté automatiquement.
- le code et le solde d'un compte sont accessibles en lecture seulement.
- A sa création, un compte bancaire a un solde nul et un code incrémenté.
- Il est aussi possible de créer un compte en précisant son solde initial.
- Utiliser son compte consiste à pouvoir y faire des dépôts et des retraits. Pour ces deux opérations, il faut connaître le montant de l'opération.
- L'utilisateur peut aussi consulter le solde de son compte par la méthode ToString().
- Un compte Epargne est un compte bancaire qui possède en plus un champ « Taux Intérêt=6 » et une méthode calculIntérêt() qui permet de mettre à jour le solde en tenant compte des intérêts.
- Un ComptePayant est un compte bancaire pour lequel chaque opération de retrait et de versement est payante et vaut 5 DH.

Questions :

1. Définir la classe Compte.
2. Définir la classe CompteEpargne.
3. Définir la classe ComptePayant.
4. Créer un programme permettant de tester ces classes avec les actions suivantes:

- Créer une instance de la classe Compte, une autre de la classe CompteEpargne et une instance de la classe ComptePayant
- Faire appel à la méthode déposer() de chaque instance pour déposer une somme quelconque dans ces comptes.
- Faire appel à la méthode retirer() de chaque instance pour retirer une somme quelconque de ces comptes.
- Faire appel à la méthode calculInterêt() du compte Epargne.
- Afficher le solde des 3 comptes.

EXERCICE 3 :

Objectif:

- Mettre en place le concept du polymorphisme.

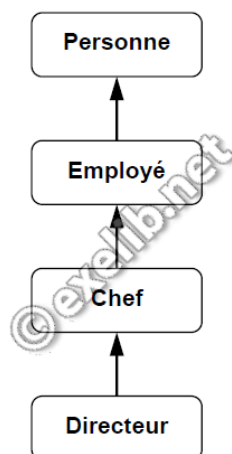
Énoncé:

une classe **Personne** qui comporte trois champs privés, nom, prénom et date de naissance. Cette classe comporte un constructeur pour permettre d'initialiser les données. Elle comporte également une méthode polymorphe Afficher pour afficher les données de chaque personne.

une classe **Employé** qui dérive de la classe Personne, avec en plus un champ *Salaire* accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.

une classe **Chef** qui dérive de la classe Employé, avec en plus un champ *Service* accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.

une classe **Directeur** qui dérive de la classe Chef, avec en plus un champ *Société* accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.



Travail à faire:

1. Ecrire les classe Personne, Employé, Chef et Directeur.

2. créez un programme de test qui comporte tableau de huit personnes avec cinq employés, deux chefs et un directeur (8 références de la classe Personne dans lesquelles ranger 5 instances de la classe Employé, 2 de la classe Chef et 1 de la classe Directeur).

Affichez l'ensemble des éléments du tableau à l'aide de for.

Affichez l'ensemble des éléments du tableau à l'aide de foreach.

EXERCICE 4:

Objectif:

- Créer une classe abstraite.
- Dériver une classe abstraite.
- Implémenter une méthode abstraite.

Énoncé:

Un parc auto se compose des voitures et des camions qui ont des caractéristiques communes regroupées dans la classe Véhicule.

- Chaque véhicule est caractérisé par son matricule, l'année de son modèle, son prix.
- Lors de la création d'un véhicule, son matricule est incrémenté selon le nombre de véhicules créés.
- Tous les attributs de la classe véhicule sont supposés privés. ce qui oblige la création des accesseurs (get...) et des mutateurs (set....) ou les propriétés.
- La classe Véhicule possède également deux méthodes abstraites démarrer() et accélérer() qui seront définies dans les classes dérivées et qui afficheront des messages personnalisés.
- La méthode ToString() de la classe Véhicule retourne une chaîne de caractères qui contient les valeurs du matricule, de l'année du modèle et du prix.
- Les classes Voiture et Camion étendent la classe Véhicule en définissant concrètement les méthodes accélérer() et démarrer() en affichant des messages personnalisés.

Travail à faire:

- Créer la classe abstraite Véhicule.
- Créer les classes Camion et Voiture.
- Créer une classe Test qui permet de tester la classe Voiture et la classe Camion

EXERCICE 5 :

Soit la classe abstraite Employé caractérisée par attributs suivants :

- Matricule
- Nom
- Prénom
- Date de naissance

La classe Employé doit disposer des méthodes suivantes :

- un constructeur d'initialisation
- des propriétés pour les différents attributs
- la méthode ToString
- une méthode abstraite GetSalaire.

Un ouvrier est **un employé** qui se caractérise par sa date d'entrée à la société.

- Tous les ouvriers ont une valeur commune appelée SMIG=2500 DH
- L'ouvrier a un salaire mensuel qui est : $\text{Salaire} = \text{SMIG} + (\text{Ancienneté en année}) * 100$.
- De plus, le salaire ne doit pas dépasser $\text{SMIG} * 2$.

Un cadre est **un employé** qui se caractérise par un indice.

- Le cadre a un salaire qui dépend de son indice :
 - 1 : salaire mensuel 13000 DH
 - 2 : salaire mensuel 15000 DH
 - 3 : salaire mensuel 17000 DH
 - 4 : salaire mensuel 20000 DH

Un patron est **un employé** qui se caractérise par un chiffre d'affaire et un pourcentage.

- Le chiffre d'affaire est commun entre les patrons.
- Le patron a un salaire annuel qui est égal à x% du chiffre d'affaire : $\text{Salaire} = \text{CA} * \text{pourcentage} / 100$

Travail à faire:

1. Créer la classe abstraite Employé.
2. Créer la classe Ouvrier, la classe Cadre et la classe Patron qui héritent de la classe Employé, et prévoir les Constructeurs et la méthode ToString de chacune des 3 classes.
3. Implémenter la méthode **GetSalaire()** qui permet de calculer le salaire pour chacune des classes.

EXERCICE 5 :

III- PROGRAMMATION EVENEMENTIELLE

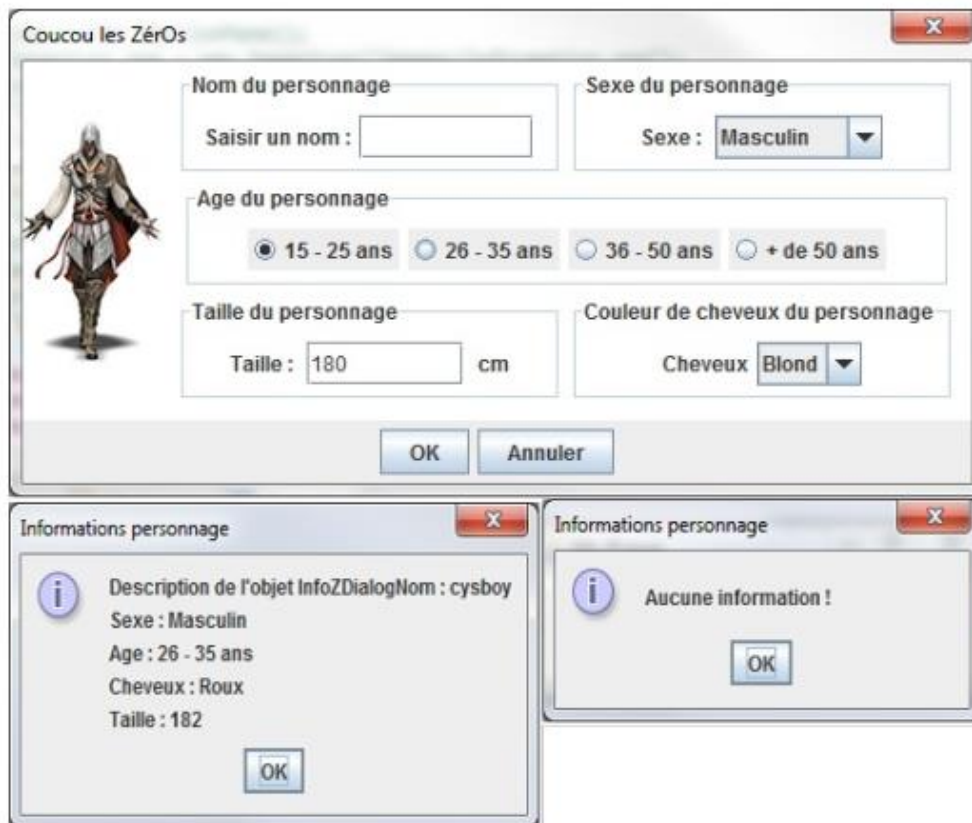
EXERCICE 1 :

Ecrire une classe calculatrice basique permettant d'effectuer la somme de deux entiers Nombre1 et Nombre2 lorsqu'on clique sur le bouton Ajouter.



Exercice 2 :

Ecrire une classe donnant les informations d'un personnage. Les affiches ou renvoi un message d'erreur.



Différentes copies

d'écran de test

Activer Window