Programmation Objet - Cours II

Exercices

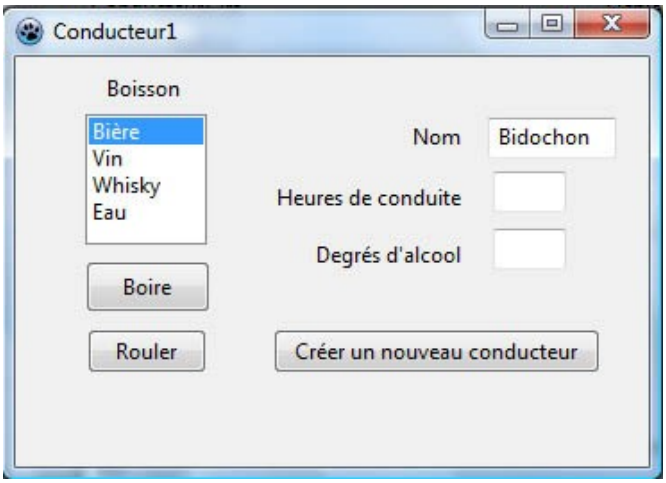
Auteur : E.Thirion - Dernière mise à jour : 25/02/2019

Les exercices suivants sont en majorité des projets à complèter. L'interface graphique de ces projets est déjà réalisée ce qui vous permettra un gain de temps important. Ces projets sont disponibles par téléchargement. Pour savoir comment y accéder cliquez ici.

D'autre part, ce document fait partie d'un ensemble de cours du même auteur (programmation procédurale et objet, programmation web, bases de données) auxquels vous pouvez accédez en cliquant ici.

Exercice 1 : La classe Conducteur

Projet: Exo-ProgObjet2/Conducteur1/ProjetConducteur1.lpi

Le Formulaire	Zones de texte et zones de liste
	**Zone de liste** Zl_Boisson **Zones de texte** Zt_Nom Zt_HC (heure de conduites) Zt_Alcool

Objectif

Ce projet simule un mauvais exemple de conducteur, qui se permet de consommer de l'alcool avant de conduire. Plus il boit, plus son degré d'alcool dans le sang augmente, et plus il a de chance d'avoir un accident. S'il a un accident, il meurt nécessairement.

Bien entendu, s'il est mort, il ne peut plus boire, ni rouler. Pour continuer la simulation, il est alors nécessaire de créer un nouveau conducteur.

Question 1 : Interface classe

Nous allons représenter le conducteur par une instance d'une classe nommé **Conducteur.**

Les attributs de cette classe sont les suivants:

- **Nom** : chaine de caractère contenant le nom du conducteur.
- **Alcool** : degré d'alcool dans le sang (nombre pas forcément entier).

- **Conduite** : heures de conduites(nombre entier).
- **Mort** : booléen indiquant si le conducteur est mort.

Et voici les méthodes:

- **Nouveau** : constructeur de la classe. Cette méthode génère un conducteur de nom donné (paramètre du constructeur).
- **AfficherAttributs** : affiche le nom, les heures de conduites et le degré d'alcool dans le sang d'un conducteur.
- **Boire** : augmente le degré d'alcool dans le sang d'un conducteur en fonction de la boisson absorbée. Le nom de la boisson est passé en paramètre.
- **Accident** : fonction aléatoire retournant la valeur vraie si le conducteur a eu un accident.
- **Rouler** : incrémente nombre d'heures de conduite du conducteur, sauf s'il est mort ou s'il a eu un accident.

Travail à faire: Ecrire l'interface de la classe conducteur ainsi que les méthodes vides (aucune instruction dans le corps) dans la partie implementation. Vérifiez que la compilation se passe bien.

Question 2 : Le bouton "Créer un Nouveau Conducteur"

Lorsque l'utilisateur clique sur ce bouton, une instance de la classe **Conducteur** est générée grâce à la méthode **Nouveau**. Elle est ensuite mémorisée dans une variable globale (**LeConducteur**) de type conducteur, puis ses attributs sont affichées.

Pour faire fonctionner ce bouton, il vous faudra donc d'abord écrire le constructeur de la classe ainsi que la méthode **AfficherAttributs**. Pour afficher un nombre non entier, utilisez la procédure **AfficherNombre** de **ETBib**.

Précisons qu'un nouveau conducteur est vivant. Son degrée d'alcool dans le sang est nul et il n'a pas d'heures de conduite.

Question 3 : Le bouton "Boire"

Lorsque l'utilisateur clique sur ce bouton, le degré d'alcool du conducteur augmente en fonction de la boisson sélectionnée (bière +0.2 , vin +0.4, whisky + 0.8) sauf bien sur, si le conducteur est mort, au quel cas un message d'erreur est affiché mentionnant le nom du conducteur.

L'instruction permettant de récupérer le nom de la boisson sélectionnée à partir de la zone de liste figure déjà dans la procédure évènementielle associée à ce bouton. Il vous suffira donc de complèter cette procédure et d'écrire la méthode **Boire**.

Question 4 : Le bouton "Rouler"

Ce qui se passera lorsque l'utilisateur cliquera sur ce bouton dépend de l'état du conducteur:

- s'il est mort, il ne peut évidemment pas rouler.
- s'il est vivant, on essaiera de le faire rouler une heure. Mais il peut avoir un accident (forcément mortel).

Pour faire fonctionner ce bouton, Il vous faudra donc écrire la méthode **Accident**. Quelques précisions à ce sujet:

- si le degré d'alcool dans le sang du conducteur est inférieur à 1, la probabilité d'avoir un accident est égale à 0.1.
- s'il est compris entre 1 et 2, elle vaut 0.5.
- s'il est compris entre 2 et 3, elle vaut 0.8.
- s'il est supérieur à 3, elle vaut 0.95.

Pour écrire cette fonction, utilisez la fonction **EvenementDeProbabilite**, dont le code figure dans le fichier source du projet. Cette fonction retourne la valeur **true** avec un probabilité donnée. Par exemple **EvenementDeProbabilite (0.5)** a une probabilité 0.5 de retourner la valeur **true**.

Exercice 2 : La classe Conducteur encapsulée

Projet: Exo-ProgObjet2/Conducteur1/ProjetConducteur2.lpi

Le formulaire du projet est le même qu'à l'exercice 2.

Objectif: réaliser une nouvelle version du projet **Conducteur1** dans laquelle la classe **Conducteur** est encapsulée.

Nouvelle organisation du projet

Le projet est désormais séparé en deux unités:

- l'unité **Conducteur2.pas** contient les procédures évènementielles gèrant l'interface graphique.
- l'unité **ClasseConducteur2.pas** est destinée à contenir la nouvelle version de la classe **Voiture**. Ce module contient la fonction **EvenementDeProbabilite**.

L'unité ClasseConducteur2.pas

Dans cette unité, déclarez l'interface de la classe **Voiture** de manière à ce qu'elle soit encapsulée.

Les attributs sont les mêmes que pour le projet **Conducteur1**.

Les méthodes sont:

- le constructeur **Nouveau** et la méthode **Accident** de l'ancienne classe Voiture. A vous de voir si vous pouvez reprendre ces méthodes telles qu'elles ou si vous devez les modifier.

  Les anciennes méthodes **AffichersAttributs**, **Boire** et **Rouler** de la classe **Conducteur** ont été supprimées. Les opérations effectuées par les méthodes **Boire** et **Rouler** seront à présent directement effectuées dans les procédures évènementielles associées aux boutons correspondants.

- les accesseurs qui permettrons d'accéder à la classe **Voiture** depuis l'unité **Conducteur2.pas**.

L'unité Conducteur2.pas

Comme la méthode **AfficherAttributs** n'existe plus, une nouvelle procédure nommée **AfficherLeConducteur** a été introduite. C'est une procédure sans paramètres qui affiche les attributs **Nom**,

Alcool et **Conduite** de l'instance **LeConducteur**. A vous de la complèter.

Le code des boutons **Boire**, **Rouler** et **Créer un nouveau conducteur**, doivent être modifié en tenant compte du faite que les méthodes **Boire** et **Rouler** n'existent plus et que la classe est à présent encapsulée.

Dans ces procédures évènementielles, vous utiliserez la procédure **AfficherAttributs** pour afficher l'état du conducteur.

Exercice 3 : Gestion de dates

Projet: Exo-ProgObjet2/Date/Projet_Date.lpi

Le Formulaire	Zones de texte et zones de liste
	**Zones de texte** Zt_Annee Zt_Num Zt_Mois **Zone de liste** Zl_Date

Objectif

Gèrer des dates représentées de deux manières: par une année (classe **DateA**) ou par une année et un mois (classe **DateM**). Du point de vue pédagogique : assimilation des notions de redéfinition de méthode et de polymorphisme.

Le tableau des dates

Toutes les dates sont rangées dans un tableau **TD** dont les éléments sont de type **DateA**. Le nombre de dates contenues dans ce tableau est défini par la valeur de l'entier **ND**.

TD et **ND** sont des variables globales (déjà déclarées dans le code). Voici leurs déclaratiions:

```
const
  MAX_ND = 10;
var
  TD : array [1..MAX_ND] of DateA;
  ND : integer;
```

Question 1: Adjonction d'une nouvelle date

1-A) Déclaration des interfaces

DateA possède un attribut **Annee** de type entier et deux méthodes: **Lire** et **Ajouter**. Ces deux méthodes sont des procédures sans paramètres.

DateM est une sous-classe de **DateA** possédant un attribut **Mois** de type entier et une méthode **Lire**

(procédure sans paramètres).

Travail à faire: écrivez la déclaration des interfaces de ces deux classes, puis vérifiez qu'elle est correcte en compilant le projet. Attention: il est nécessaire de déclarer également les squelettes (sous-programme privé de corps) des méthodes dans la partie implementation.

1-B) Implementation des méthodes

Méthodes de la classe DateA

- **Lire** : lit l'année depuis la zone de texte **Zt_Annee** et stocke sa valeur dans l'attribut **Annee**.
- **Ajouter**: ajoute la date dans le tableau **TD** et incrémente **ND**.

Méthodes de la classe DateM

- **Lire** : lit l'année et le mois depuis les zones de textes **Zt_Annee** et **Zt_Mois** et stocke leurs valeurs dans les attributs **Annee** et **Mois**.

Travail à faire: écrivez l'implémentation de ces trois méthodes en surchargeant la méthode **Lire**.

1-C) Utilisation des méthodes

Lorsque l'utilisateur clique sur le bouton **Ajouter**, il y a deux possibilités:

- la zone de texte **Zt_Mois** est vide.
  Dans ce cas, on considère que la date n'est définie que par son année. On rajoute donc cette date en tant qu'objet de la classe **DateA**.
- la zone de texte **Zt_Mois** n'est pas vide.
  Dans ce cas, on considère que la date est définie par une année et un mois. On rajoute donc cette date en tant qu'objet de la classe **DateM**.

Travail à faire: écrire le code de la procédure évènementielle associée au bouton **Ajouter**, en utilisant les méthodes **Ajouter** et **Lire**. Dans cette procédure évènementielle, utiliser la variable **d** déclarée et aucune autre variable.

Question 2: Affichage d'une date dans les zones de texte

2-A) La méthode AfficherZt

La méthode **AfficherZt** affiche une date dans les zones de textes:

- S'il s'agit d'une date de la classe **DateA**, elle affiche uniquement l'année dans la zone de texte **Zt_Annee**.
- S'il s'agit d'une date de la classe **DateM**, elle affiche son année et son mois dans les zones de texte **Zt_Annee** et **Zt_Mois**.

> **Travail à faire**: ajouter la méthode **AfficherZt** aux deux classes et écrivez les deux implémentation de cette méthode en redéfinissant celle de la classe **DateA**.

2-A) Le bouton Afficher

Lorsque l'utilisateur clique sur le bouton **Afficher**, le programme affiche la date dont le numéro est contenu dans la zone de texte **Zt_Num**.

> **Travail à faire**: écrivez le code de la procédure évènementielle associée au bouton **Afficher** en utilisant la méthode **AfficherZt**.

Question 3: Affichage des dates dans la zone de liste

3-A) La méthode NomDuMois

Ajoutez une nouvelle méthode, nommée **NomDuMois** à la classe **DateM**. Cette méthode est une fonction sans paramètre qui retourne une chaine de caractères ('Janvier' si Mois=1, 'Fevrier' si Mois=2, etc ...).

3-B) La méthode AfficherZl

La méthode **AfficherZl** affiche une date dans la zone de liste. Le résultat dépend de la classe à laquelle appartient la date:

- s'il s'agit d'une date de la classe **DateA**, seule l'année est affichée.
- s'il s'agit d'une date de la classe **DateM**, l'affichage de la date comprend le nom du mois suivi de l'année.

> **Travail à faire**: ajoutez la méthode **AfficherZl** (procédure sans paramètres) aux deux classes, puis écrivez les deux implémentations de cette méthode, en utilisant la méthode **NomDuMois** pour afficher une date de la classe **DateM**.

3-C) La procédure AfficherToutesLesDates

Complètez la procédure **AfficherToutesLesDates**, dont le squelette est déjà contenu dans le projet. Cette méthode doit afficher toutes les dates contenues dans le tableau **TD**, dans la zone de liste **Zl_Date**. Dans l'écriture de cette procédure utilisez impérativement la méthode **AfficherZl**.

Cette procédure est appelée dans la procédure évenementielle associée au bouton **Ajouter**, vous pouvez donc la tester en ajoutant des dates.

Question 4: Tri des dates

L'objectif de cette question est de trier les dates dans l'ordre chronologique croissant. Pour comparer une date **dm** de la classe **DateM** avec une date **da** de la classe **DateA**, on prendra les conventions suivantes:

- si les années de **dm** et **da** sont différentes, on compare les années (Juin 1961 est inférieure à 1962, 1935 est inférieure à Janvier 1938).

- si les années sont identiques, la date **dm** est la plus petite.
-

Pour trier les dates on utilisera l'algorithme de tri suivant (appelé "Bubble Sort" ou "Tri à Bulle"), qui permet de trier un tableau **T** de **N** éléments:

```
Algorithme Tri à Bulle

Var Stable : Booléen; i : entier;

Stable := Faux;

Tant que Non Stable Faire
    Stable := Vrai
    Pour i de 1 à N-1 Faire
         Si T[ i ] < T [ i + 1] Alors
              Echanger T[ i ] et T [ i + 1]
              Stable := Faux
         FinSi
    FinPour
Fin Tantque
```

4-A) La méthode InferieureA

Ajouter une méthode nommée **InferieureA** à la classe **DateA**, avec l'entête suivante:

```
Function InferieureA (d : DateA) : boolean;
```

En utilisant le polymorphisme, écrivez le code de cette méthode de manière à ce qu'elle permette de comparer deux dates de type quelconque (**DateA** avec **DateA**, **DateA** avec **DateM**, **DateM** avec **DateA** et **DateM** avec **DateM**).

La méthode doit retourner la valeur **true** si et seulement si la date à laquelle elle est appliquée est strictement inférieure à la date **d** passée en paramètre.

4-B) Le bouton Trier

En vous basant sur l'algorithme de tri à bulle, complèter la procédure évènementielle associée au bouton **Trier**, en utilisant la méthode **InferieureA** pour comparer deux dates.