CAS VALIERS

PRÉSENTATION DU THÈME

Le club hippique de Valiers organise tous les ans une vingtaine de concours de sauts d'obstacles. Le président du club vous a recruté(e) afin de participer au développement d'une application de gestion de l'ensemble de ces concours. Cette application permettra d'informer la FFE (Fédération française d'équitation) des résultats détaillés de ces compétitions.

Règles d'organisation d'un concours de saut d'obstacles (CSO)

Un concours est constitué d'un ensemble d'épreuves. Une épreuve consiste à effectuer un parcours au cours duquel chevaux et cavaliers sont amenés à sauter des obstacles démontables dont le nombre, la largeur et la hauteur varient. Il faut effectuer le parcours proposé avec un minimum de pénalités (refus du cheval face à un obstacle, chute d'une barre de l'obstacle lors de son passage, etc.).

Pour chaque épreuve, un parcours est mis en place et comporte entre onze et quinze obstacles qui sont numérotés selon l'ordre de franchissement de la première phase.

Une épreuve peut comporter une ou deux phases. Pour les épreuves à deux phases, les concurrents qui se qualifient à la première phase (appelée « manche ») peuvent prendre part à la seconde (appelée « barrage »). Le parcours sur lequel se déroule le barrage est composé à partir du parcours sur lequel s'est déroulée la manche correspondante, en sélectionnant certains obstacles, parcourus dans le même ordre.

Dossier 3	Le traitement des engagements à une épreuve

Annexe à utiliser : annexe 5

La partie de l'application permettant la saisie des résultats d'une épreuve va être développée dans un langage à objets et vous êtes chargé(e) de réaliser quelques méthodes préparant notamment l'affichage de la liste des engagés à l'épreuve en cours. Vous disposez pour cela des classes définies en **annexe 5**.

	Travail à faire
3.1	En utilisant les méthodes définies dans l'**annexe 5**, écrire la méthode ***compterEngagés(uneAnnée)*** de la classe *Epreuve* permettant de compter le nombre des cavaliers engagés à l'épreuve qui ont obtenu leur licence au cours de l'année fournie en paramètre.

Annexe à utiliser : annexe 6

Afin de personnaliser l'affichage de la liste des cavaliers engagés comme indiqué dans l'**annexe 6.A**, il a été décidé de créer les classes *CavalierEnfant* et *CavalierAdulte* qui héritent de la classe *Cavalier* (**annexe 6.B**).

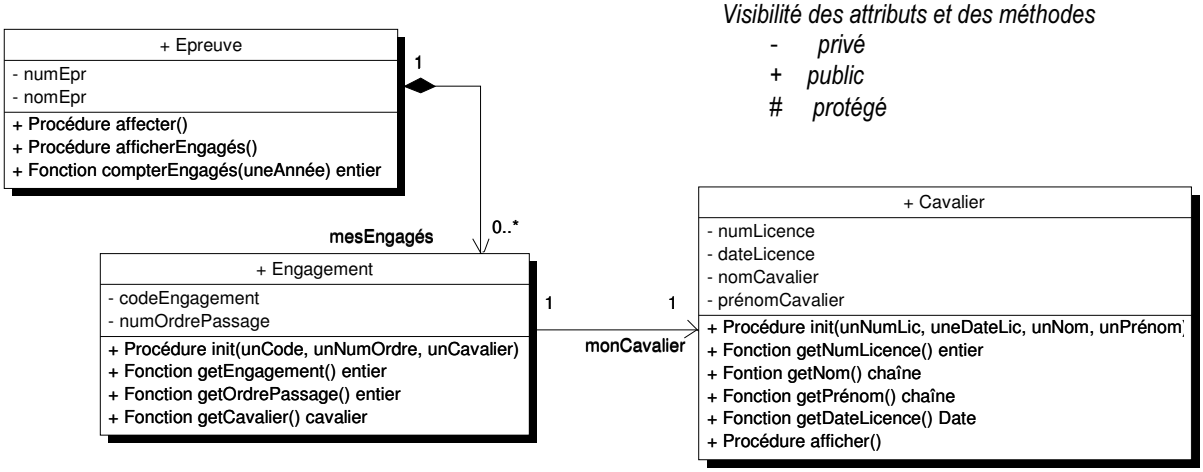
Il vous est demandé de modifier votre application en commençant par écrire les méthodes nécessaires des classes *CavalierEnfant* et *CavalierAdulte*, pour permettre le fonctionnement, sans modification, de la méthode *afficherEngagés()* de la classe *Epreuve* décrite en **annexe 6.C**.

	Travail à faire
3.3	Écrire les algorithmes correspondant aux méthodes des classes *CavalierEnfant* et *CavalierAdulte*, méthodes exploitées lors de l'exécution de la méthode *afficherEngagés()* de la classe *Epreuve*.

M. Dugazeau, responsable du centre équestre et passionné d'informatique, vient de lire un article intitulé « 10 conseils pour développer un site web dynamique » et vous a avoué se trouver démuni devant l'avalanche de termes techniques qu'il a rencontrés.

	Travail à faire
3.4	Rédiger à l'intention de M. Dugazeau une note de synthèse expliquant *sur la base d'un schéma* le principe d'une architecture à 3 niveaux.

Annexe 5 – Classes définies pour le traitement des engagements



- La classe Engagement représente l'engagement d'un cavalier à l'épreuve, elle permet de connaître le code d'engagement de chaque cavalier ainsi que le numéro d'ordre de passage.
- Pour l'épreuve, il y a plusieurs engagements (multiplicité *).
- Un engagement ne concerne que l'épreuve (multiplicité 1).

Dans les descriptions ci-dessous, les procédures *init()* jouent le rôle de constructeur.

➢ Classes métier

Classe Epreuve

Privé

numEpr : Entier
nomEpr : Chaîne de caractères
mesEngagés : Collection de Engagement
// les cavaliers sont classés dans la collection suivant le numéro d'ordre de passage

Public

Procédure affecter()
Procédure afficherEngagés()
Fonction compterEngagés(uneAnnée : Entier) : Entier
…

FinClasse

Classe Engagement

Privé

codeEngagement : Entier
numOrdrePassage : Entier
monCavalier : Cavalier

Public

Procédure init(unCode : Entier, unNumOrdre : Entier, unCavalier : Cavalier)
Fonction getEngagement() : Entier
Fonction getOrdrePassage() : Entier
Fonction getCavalier() : Cavalier
…

FinClasse

Classe Cavalier

Privé

numLicence : Entier
dateLicence : Date // Date d'obtention de la licence
nomCavalier : Chaîne de caractères
prénomCavalier : Chaîne de caractères

Public

Procédure init(unNumLicence : Entier, uneDateLicence : Date, unNom : Chaîne de caractères, unPrénom : Chaîne de caractères)
Fonction getNumLicence() : Entier
Fonction getNom() : Chaîne de caractères
Fonction getPrénom() : Chaîne de caractères
Fonction getDateLicence() : Date
Procédure afficher()
…

Fin Classe

- **Classes techniques**

Classe Collection

// classe générique : un objet de la classe Collection permet de gérer un ensemble d'objets.

Privé

…

Public

```
Fonction cardinal() : Entier                          // renvoie le nombre d'éléments de la collection
Fonction existe(unObjet : Objet) : Booléen            // teste si un objet identique existe dans la collection
Fonction index(unObjet : Objet) : Entier              // renvoie l'index d'un objet de la collection, le
                                                      // premier objet de la collection a pour index 1
Fonction extraireObjet(unIndex : Entier) : objet      // accède à un objet contenu dans la collection
Procédure ajouter(unObjet : Objet, unIndex : Entier)  // ajoute un objet à la collection, à l'index passé en
                                                      // paramètre
Procédure enlever(unIndex : Entier)                   // supprime un objet de la collection
Procédure vider()                                     // vide le contenu de la collection
```

FinClasse

// Pour instancier une collection

uneCollection : Collection de <classe> // La collection instanciée contiendra des objets de la classe <classe>

L'avantage de la classe Collection est d'offrir des services d'ajout et de suppression plus simples que la gestion d'un tableau. En outre le problème du dimensionnement de la structure n'est pas à la charge du développeur.

Classe Date

Privé

…

Public

Fonction année() : Entier // renvoie l'année
Fonction mois() : Entier // renvoie le mois
Fonction jour() : Entier // renvoie le jour
…

FinClasse

Annexe 6 – Éléments pour améliorer le traitement des engagements

6.A : Exemple d'affichage

Numéro Engagement	Ordre de passage	Engagé
145	1	Monsieur Jacky Durand
157	2	Madame Hannabelle Durand
158	3	Isabelle Durand - 15 ans - 1,45 m
167	4	Alain Durand - 13 ans - 1,49 m
…		

6.B : Description des classes héritant de Cavalier

Classe CavalierAdulte hérite de Cavalier
    Privé
        sexe : Entier // 1 pour masculin, 2 pour féminin
    Public
        Fonction getSexe() : Entier
        Fonction getCivilité() : Chaîne de caractères // retourne "Monsieur" ou "Madame"
        …
FinClasse

Classe CavalierEnfant hérite de Cavalier
    Privé
        dateNaissance : Date
        taille : Réel
    Public
        Fonction getAge() : Entier
        Fonction getTaille() : Entier
        …
FinClasse

6.C : Description de la méthode *afficherEngagés()* de la classe *Epreuve*

```
Procédure Epreuve::afficherEngagés()
Début
    nb, i  : Entier
    unEngagement : Engagement

    // Nombre de cavaliers engagés dans l'épreuve
    nb ← mesEngagés.cardinal()

    Pour i de 1 à nb
        unEngagement ← extraireObjet(i)
        écrire unEngagement.getEngagement(), unEngagement.getOrdrePassage()
        unEngagement.getCavalier().afficher()
        passerALaLigne
    Fin pour
Fin

Procédure Cavalier::afficher()
Début
    écrire getPrénom(), " ", getNom()    // affiche le prénom et le nom du cavalier
Fin
```