Programmation orientée objet (UAA14)

Exercices : série 6

Objets d'apprentissage :

- ✔ Modéliser une logique de programmation orientée objet.
- ✔ Déclarer une classe.
- ✔ Instancier une classe.
- ✔ Utiliser les méthodes de l'objet instancié.
- ✔ Traduire un algorithme dans un langage de programmation.
- ✔ Commenter des lignes de code.
- ✔ Tester le programme conçu.
- ✔ Caractériser les attributs dans une classe (encapsulation).
- ✔ Caractériser les méthodes dans une classe (encapsulation).
- ✔ Décrire la création d'un constructeur.
- ✔ Extraire d'un cahier des charges les informations nécessaires à la programmation.
- ✔ Programmer en recourant aux instructions et types de données nécessaires au développement d'une application.
- ✔ Corriger un programme défaillant.
- ✔ Développer une classe sur la base d'un cahier des charges en respectant le paradigme de la programmation orientée objet.
- ✔ Programmer en recourant aux classes nécessaires au développement d'une application orientée objet.
- ✔ Améliorer un programme pour répondre à un besoin défini.

Exercice 1

Etape 1

Crée une classe `Personne` caractérisée par un nom, un prénom, un sexe et une date de naissance.

Etape 2

Crée un setter / getter pour chaque variable d'instance.

Etape 3

Prévois un constructeur (à 6 arguments) qui permet de créer des personnes à partir d'un nom, un prénom, un sexe, une année, un mois et un jour de naissance.

Etape 4

Mets à jour le setter du sexe de telle sorte qu'il génère une exception de type `SexeIncorrectException` lorsque le sexe renseigné n'est pas correct. Les seules valeurs permises pour le sexe sont `'m'`, `'M'`, `'f'`, `'F'`, `'x'` et `'X'`.

Etape 5

Fais en sorte que le constructeur propage l'erreur du setter. En effet, ce n'est pas à lui de gérer cette exception, mais au script PHP qui utilise le constructeur.

Etape 6

Prévois également la méthode `toString()` qui retourne le prénom suivi du nom.

Etape 7

Réalise les deux étapes suivantes :

1. Essaye (**try**) de créer une personne avec un sexe incorrect
2. Capture (**catch**) l'exception et affiche à l'écran le message correspondant à l'erreur

Etape 8

Mets à jour ton code :

- lors de la création (réussie) d'une personne, un message de confirmation doit apparaître dans la console ;
- lorsque la création lance une exception (échoue) , un message d'erreur doit apparaître dans la console.

Etape 9

Fais en sorte que les données de création d'une personne soient obtenues auprès de l'utilisateur.

Les informations à demander sont :

- le nom ;
- le prénom ;
- le sexe ;
- l'année de naissance ;

- le mois de naissance ;
- le jour de naissance.

Etape 10

Fais en sorte que l'utilisateur puisse créer autant de personnes qu'il le souhaite. Pour cela, stocke chaque personne créée en session.

Etape 11

Mets en place un bouton cliquable permettant d'afficher le pourcentage de filles inscrites.

Exercice 2

L'objectif de cet exercice est de simuler une connexion à une base de données contenant des informations sur des Streamers.

Étape 1

Crée une classe `Streamer` permettant de créer des streamers. Un streamer est caractérisé par :

- son pseudo ;
- sa plateforme de jeu (PC, PS5, XBOX, etc.) ;
- son nombre d'abonnés.

Prévois un constructeur pour cette classe ainsi que des getters/setters.

Étape 2

Crée une classe `ConnexionBD` modélisant une connexion à une base de données. Les informations nécessaires pour se connecter à une base de données sont :

- une adresse (`host`) ;
- un nom d'utilisateur (identifiant) ;
- un mot de passe ;
- un tableau contenant un ensemble de streamers (autrement dis, les données).

Prévois également un constructeur : lorsqu'une `ConnexionDB` est instanciée, quelques streamers sont automatiquement ajouté au tableau de streamers (pour simuler une vraie base de données).

Étape 3

Il est possible que l'établissement d'une connexion à une base de données échoue, et ce pour plusieurs raisons :

- l'adresse renseignée ne redirige vers aucune base de données ;
- le nom d'utilisateur n'existe pas ;
- le mot de passe est incorrect.

Fais en sorte que le constructeur de la classe `ConnexionBD` lance une exception `ConnexionException` dans les cas suivants :

- l'adresse renseignée n'est pas `localhost:3306/streamers`
- le nom d'utilisateur n'est pas `admin`
- le mot de passe n'est pas `root`

Étape 4

Ajoute les méthodes suivantes à la classe `ConnexionBD` :

- `getStreamer` : retourne le streamer identifié par le pseudo reçu en argument
- `getAllStreamers` : retourne un tableau contenant l'ensemble des streamers présent dans la base de données

- `addStreamer` : ajoute le streamer reçu en argument dans le tableau des streamers
- `removeStreamer` : supprime le streamer du tableau identifié par le pseudo reçu en argument

Attention, des exception doivent être lancées dans certains cas :

- toutes les méthodes lancent possiblement une `ConnexionException`, dans le cas où la connexion avec la base de données est interrompue (ex. coupure internet) ;
- les méthodes `getStreamer`, `getAllStreamers` et `removeStreamer` lancent possiblement une `NoResultException` dans le cas où aucun streamer n'est trouvé dans la base de données ;
- la méthode `addStreamer` lance possiblement une `StreamerAlreadyExistsException` dans le cas où on tente d'ajouter un streamer qui existe déjà (deux pseudos identiques).

Teste l'ensemble de tes méthodes.

Exercice 3

L'objectif de cet exercice est de créer un gestionnaire de fichier qui permet d'écrire et de lire dans un fichier.

Étape 1

Crée la classe `GestionnaireFichier` permettant d'interagir avec des fichiers.

Étape 2

Ajoute à la classe `GestionnaireFichier` la méthode `lireFichier` qui retourne le contenu d'un fichier localisé au chemin reçu en argument.

Pour lire un fichier ligne par ligne, utilise la classe `LecteurFichier` disponible à l'adresse : https://github.com/LucasEmbrechts/UAA14-POO-PHP/tree/main/OperationsFichier

Attention : des cas d'erreur (exceptions) sont à prévoir :

- Si le fichier n'existe pas, le constructeur de la classe `LecteurFichier` lance une `LectureException`
- En cas d'erreur de lecture (ex. le fichier est supprimé pendant que le programme lit les lignes), la fonction `lireLigne` lance également une `LectureException`

Fais en sorte que la méthode `lirefichier` lance une exception nommée `FichierInexistantException` contenant le message « Le fichier `<chemin>` n'existe pas ou a été supprimé » lorsqu'une de ces deux erreurs est détectée.

Étape 3

Ajoute à la classe `GestionnaireFichier` la méthode `écrireDansFichier` qui écrit une chaine de caractères dans un fichier situé au chemin spécifié. La chaine de caractères et le chemin sont tous les deux reçus en argument.

Pour écrire une ligne dans un fichier, utilise la classe `EcritureFichier` disponible à l'adresse : https://github.com/LucasEmbrechts/UAA14-POO-PHP/tree/main/OperationsFichier

Attention : des cas d'erreur (exceptions) sont à prévoir : le constructeur de la classe `EcritureFichier` ainsi que la méthode `ecrire` lancent une `EcritureException` lorsqu'une erreur d'écriture dans le fichier est détectée

Fais en sorte que la méthode `écrireDansFichier` lance une exception nommée `AccèsFichierException` contenant le message « Accès refusé au fichier `<chemin>` » lorsqu'une erreur d'écriture est détectée.

Etape 4

Teste les deux méthodes de la classe `GestionnaireFichier` en essayant notamment de :

- lire le contenu d'un fichier inexistant ;
- écrire dans un fichier verrouillé.

Références

Les présents exercices ont été élaborés à l'aide des ressources suivantes :

- https://www.pierre-giraud.com/php-mysql-apprendre-coder-cours/introduction-programmation-orientee-objet/
- Cours de « Développement d'applications WEB », Hénallux (2019)