الجمهـورية الجزائريـة الديمقراطيـة الشعبيـة

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
Université 8 Mai 1945 Guelma
Faculté de mathématiques et de l'informatique et de sciences de la matière

وزارة التـعليم العالي والبحث العلمي
جـــامعـة 8 مـــاي 1945 قالمـــة
كلية الرياضيات ، الاعلام الآلي و علوم المادة

Module : Programmation orientée objet II
TP2 : Concepts avancés de la POO

Note : Les exercices de cette série de travaux pratiques doivent être implémentés en langage Java. Un diagramme de classes doit être élaboré et validé avant de commencer la programmation.

Partie I : Exceptions et flux d'entrées/sorties

Exercice 1 :

Développer une classe *Calculateur* capable d'effectuer les opérations arithmétiques de base, notamment l'addition, la soustraction, la multiplication et la division. Chaque méthode de la classe doit prendre deux nombres en paramètres. La classe *Calculateur* doit également implémenter une gestion robuste des exceptions pour gérer divers scénarios invalides, tels que la division par zéro. Créer un objet de la classe *Calculateur* dans le programme principal pour tester la validité du code ainsi que la gestion appropriée des différentes exceptions. L'utilisateur devra saisir deux nombres et un opérateur à partir du clavier. Le programme devrait utiliser le résultat de l'opération précédente comme premier opérande pour l'opération courante. Assurer que le programme offre une expérience utilisateur conviviale, en demandant à l'utilisateur de corriger toute saisie incorrecte et en gérant les exceptions de manière appropriée. Le programme continue à accepter de nouvelles opérations jusqu'à ce que l'utilisateur choisisse de quitter en entrant "ESC".

Exercice 2 :

Écrire un programme qui invite l'utilisateur à saisir un mot de passe. Créez une exception personnalisée *WeakPasswordException* qui sera déclenchée si le mot de passe ne respecte pas les critères suivants : une longueur minimale (8), la nécessité d'avoir des chiffres, des lettres majuscules, et au moins un caractère spécial.

Exercice 3 :

Développer un programme qui invite l'utilisateur à saisir un mot. Le programme doit rechercher ce mot dans un fichier texte spécifié et afficher les lignes du fichier, ainsi que leurs numéros, qui contiennent le mot saisi. Si le mot est présent dans plusieurs lignes, toutes les occurrences avec leurs numéros respectifs doivent être affichées.

Exercice 4 :

Écrire un programme qui prend une image en entrée, effectue une ou plusieurs opérations (par exemple, inversion des couleurs, mise à l'échelle et rotation) et enregistre la nouvelle image dans un fichier de sortie. Utiliser des blocs *try-catch* pour gérer les exceptions liées à la manipulation d'images.

Partie II : Structures de données complexes, réflexivité et généricité

Exercice 5 :

Concevoir un système de gestion d'utilisateurs où chaque utilisateur a un identifiant unique, un nom et un mot de passe. Utiliser une table de hachage pour stocker les utilisateurs en fonction de leur identifiant. Ajouter des fonctionnalités telles que l'ajout, la suppression et la recherche d'utilisateurs.

Exercice 6 :

Développer une application de gestion de bibliothèque où les livres (titre, auteur), les magazines (titre, numéro d'issue), sont des classes héritant d'une classe de base. Utiliser la généricité pour créer des collections qui peuvent stocker différents types de médias, et la réflexivité pour inspecter dynamiquement les attributs.

Exercice 7 :

Concevoir une classe générique *DataProcessor* capable de manipuler des collections d'objets génériques. Ajouter des méthodes pour trier, filtrer et effectuer d'autres opérations sur les données. Tester la classe avec différents types d'objets, y compris des types primitifs et des types utilisateurs tels que des voitures avec un modèle spécifique, afin de garantir la polyvalence et la robustesse de la classe pour différentes applications.

Exercice 8 :

Développer une application de gestion de réseau social baptisée *AmiConnect*. Dans ce réseau social, chaque utilisateur est identifiable par un nom unique, et des relations d'amitié sont établies entre eux. La classe *AmiConnect* joue le rôle de gestionnaire central, prenant en charge l'ensemble des utilisateurs et de leurs relations. Deux stratégies de recherche sont mises en œuvre, fondées sur la recherche en largeur (BFS) et la recherche en profondeur (DFS), permettant d'explorer le réseau social à partir d'un utilisateur spécifique. L'application de test, *AmiConnectApp*, offre à l'utilisateur la possibilité d'interagir avec le réseau social via le clavier, en effectuant la recherche de la liste des amis directs d'un utilisateur spécifique en utilisant une méthode recherche particulière. Une gestion rigoureuse des erreurs, la capacité de charger le réseau social depuis un fichier de configuration, ainsi qu'une conception orientée objet, sont des éléments clés garantissant la robustesse et la flexibilité de *AmiConnect*.