TP : Tableaux à plusieurs dimensions, généricité et collections

Ce TP vise à explorer divers concepts tels que les tableaux à plusieurs dimensions, la généricité et les collections.

Exercice 1 : classe générique

L'objectif de cet exercice est de vous familiariser avec la notion de généricité en Java en créant une classe générique appelée **Pair**. Cette classe devra être capable de stocker deux objets de types différents.

1. **Définir la classe générique Pair :**
    a) Créez une nouvelle classe Java appelée **Pair**.
    b) Utilisez des génériques pour définir deux paramètres de type, généralement notés **T** et **U**.
    c) La classe doit avoir deux variables d'instance de type générique **T** et **U** pour stocker les deux objets.
    d) Ajoutez un constructeur à la classe **Pair** prenant deux paramètres, un de type **T** et l'autre de type **U**. Le constructeur devra initialiser les variables d'instance.
    e) Ajoutez des méthodes d'accès (getters) pour obtenir les valeurs des deux objets stockés dans la paire.
2. **Test de la classe Pair avec différents types :**
    a) Créez une classe de test distincte, appelée par exemple **PairTest**.
    b) Dans la classe de test, créez plusieurs instances de la classe **Pair** en utilisant différents types pour les objets stockés. Par exemple, une paire de **Integer** et de **String**.
    c) Utilisez les méthodes d'accès pour récupérer et afficher les valeurs stockées dans chaque paire.

Exercice 2 : Méthodes génériques

Dans cet exercice, l'objectif est d'implémenter une méthode générique **swap** qui permet d'échanger les positions de deux éléments dans un tableau. Cette méthode devra être testée avec des tableaux de différents types.

1. **Définition de la méthode générique swap :**
    - Créez une nouvelle classe ou utilisez une classe existante dans laquelle vous allez implémenter la méthode générique **swap**. La méthode **swap** doit prendre en paramètre le tableau (array) et les indices des deux éléments à échanger. Utilisez un type générique pour déclarer le type des éléments du tableau.
    - Écrivez le code nécessaire pour échanger les positions des deux éléments dans le tableau.

2. **Test de la méthode swap avec différents types :**
    - Créez un programme de test (classe de test) dans lequel vous instancierez différents tableaux de types variés (par exemple, **Integer**, **String**, **Double**).
    - Utilisez la méthode **swap** pour échanger les positions de deux éléments dans chaque tableau.
    - Affichez le tableau avant et après l'échange pour vérifier la fonctionnalité de la méthode **swap** avec différents types.
3. **Gestion des exceptions :**
    - Ajoutez une gestion des exceptions pour éviter tout débordement d'indices ou tout autre problème potentiel lors de l'échange des éléments. Utilisez des clauses **try-catch** pour gérer les exceptions appropriées.

Exercice 3 : Héritage et généricité

Cet exercice vise à explorer l'héritage et les génériques en Java. Vous devrez créer une classe générique appelée **Box** capable de contenir un objet de n'importe quel type, puis étendre cette classe avec une nouvelle classe appelée **ColoredBox** qui ajoute une propriété de couleur spécifique au contenu.

1. **Définir la classe générique Box :**
    - Créez une nouvelle classe Java appelée **Box**.
    - Utilisez la généricité pour déclarer un attribut de type, conventionnellement noté **T**. La classe **Box** doit avoir une variable d'instance pour stocker l'objet de type **T**.
2. **Implémenter le constructeur et les méthodes d'accès dans la classe Box :**
    - Ajoutez un constructeur à la classe **Box** prenant un paramètre de type **T** pour initialiser la variable d'instance.
    - Intégrez des méthodes d'accès et de modification (getters et setters) pour obtenir et afficher le contenu de la boîte.
3. **Créer la classe héritée ColoredBox :**
    - Créez une nouvelle classe Java appelée **ColoredBox** qui hérite de la classe **Box**.
    - Ajoutez une variable d'instance pour stocker la couleur de la boîte.
4. **Implémenter le constructeur dans la classe ColoredBox :**
    - Ajoutez un constructeur à la classe **ColoredBox** qui prend deux paramètres : un de type **T** pour le contenu et un de type **String** pour la couleur.
    - Appelez le constructeur de la classe **Box** pour initialiser le contenu, puis initialisez la couleur.

5. **Utiliser les classes avec différents types et couleurs :**
    - Dans une classe de test distincte (par exemple, **BoxTest**), créez des instances de **Box** avec différents types d'objets (par exemple, **Integer**, **String**, **Double**) et affichez le contenu.
    - Créez des instances de **ColoredBox** avec différents types d'objets et couleurs, puis affichez le contenu et la couleur.

Exercice 4 : Interface générique

Cet exercice a pour objectif de vous familiariser avec les interfaces génériques en Java. Vous serez amené à définir une interface générique appelée **Calculator**, comprenant des méthodes pour effectuer des opérations mathématiques de base, puis à l'implémenter pour les types **Integer**, **Double**, et **BigDecimal**.

1. **Définir l'interface générique Calculator :**
    - Créez une nouvelle interface Java appelée **Calculator**.
    - Utilisez des génériques pour déclarer les types des paramètres et du résultat des opérations mathématiques.
2. **Méthodes d'opérations mathématiques :**
    - Dans l'interface **Calculator**, déclarez des méthodes génériques pour les opérations mathématiques de base, telles que l'addition, la soustraction, la multiplication et la division. Utilisez les génériques pour les types des paramètres et des résultats.
3. **Implémentation pour le type Integer :**
    - Créez une classe appelée **IntegerCalculator** qui implémente l'interface **Calculator** pour le type **Integer**.
    - Implémentez les méthodes de l'interface pour effectuer les opérations mathématiques avec des objets de type **Integer**.
4. **Implémentation pour le type Double :**
    - Créez une classe appelée **DoubleCalculator** qui implémente l'interface **Calculator** pour le type **Double**.
    - Implémentez les méthodes de l'interface pour effectuer les opérations mathématiques avec des objets de type **Double**.
5. **Implémentation pour le type BigDecimal :**
    - Créez une classe appelée **BigDecimalCalculator** qui implémente l'interface **Calculator** pour le type **BigDecimal**.
    - Implémentez les méthodes de l'interface pour effectuer les opérations mathématiques avec des objets de type **BigDecimal**.

6. **Test de chaque implémentation :**
    - Dans une classe de test (par exemple, **CalculatorTest**), créez des instances des classes **IntegerCalculator**, **DoubleCalculator**, et **BigDecimalCalculator**.
    - Effectuez quelques opérations mathématiques en utilisant les méthodes de l'interface et affichez les résultats.

Exercice 5 : Collections génériques

Cet exercice a pour but de vous familiariser avec les collections génériques en Java en implémentant une classe générique **Stack** qui représente une pile. Vous serez amené à ajouter des méthodes pour empiler et dépiler des éléments, puis à tester cette classe avec différents types.

1. **Définir la classe générique Stack :**
    - Créez une nouvelle classe Java appelée **Stack**.
    - Utilisez un type générique, généralement noté **T**.
    - La classe **Stack** doit avoir une structure de données interne pour stocker les éléments de manière à respecter le principe Last-In-First-Out (LIFO) d'une pile.
2. **Méthodes d'empilement et de dépilement :**
    - Ajoutez des méthodes à la classe **Stack** pour permettre l'empilement (**push**) et le dépilement (**pop**) d'éléments. Assurez-vous de gérer correctement la structure de données interne.
3. **Test de la classe Stack avec différents types :**
    - Dans une classe de test distincte (par exemple, **StackTest**), créez des instances de **Stack** avec différents types d'objets (par exemple, **Integer**, **String**, **Double**).
    - Utilisez les méthodes d'empilement et de dépilement pour manipuler la pile, puis affichez les résultats pour vérifier le bon fonctionnement de la classe **Stack** avec différents types.
4. **Gestion des exceptions :**
    - Ajoutez une gestion des exceptions pour éviter tout débordement ou sous-débordement de la pile lors des opérations d'empilement et de dépilement. Utilisez des clauses **try-catch** pour gérer les exceptions appropriées.

Exercice 6 : Collections génériques – Arbres binaires

Cet exercice a pour but de vous familiariser avec les collections génériques en Java, en implémentant une classe générique représentant un arbre binaire. Vous serez amené à créer un arbre binaire avec des méthodes d'insertion, de parcours, et à le tester avec différents types.

1. Définir la classe générique `BinaryTree<T>`

- Créez une nouvelle classe Java appelée `BinaryTree` avec une structure générique (`T`).
- Définissez une classe interne `Node<T>` représentant un nœud de l'arbre, contenant :
  - Une valeur de type `T`
  - Un sous-arbre gauche (`left`)
  - Un sous-arbre droit (`right`)
- La classe `BinaryTree<T>` doit contenir une racine (`root`) et les méthodes associées à l'arbre.

2. Méthodes de manipulation de l'arbre

Implémentez les méthodes suivantes :

- `void insert(T value)` : insère un élément dans l'arbre (ex : insertion en respectant l'ordre pour un arbre binaire de recherche).
- `void inorder()` : affiche les éléments selon un parcours infixe.
- `void preorder()` : affiche les éléments selon un parcours préfixe.
- `void postorder()` : affiche les éléments selon un parcours postfixe.

3. Test de la classe `BinaryTree` avec différents types

- Créez une classe de test appelée `BinaryTreeTest`.
- Créez des instances de `BinaryTree` avec des types différents (`Integer`, `String`, `Double`).
- Insérez plusieurs valeurs, puis affichez les résultats des différents parcours pour chaque type.

4. Gestion des cas particuliers

- Vérifiez et traitez les cas où l'arbre est vide (pas de racine).
- Vérifiez que les types utilisés soient **comparables** si vous appliquez un tri (implémentation d'un arbre binaire de recherche).
- Gérez les erreurs éventuelles avec des blocs `try-catch`.

Exercice 7 : Interface générique

Cet exercice a pour objectif de vous familiariser avec les interfaces génériques en Java. Vous serez amené à définir une interface générique appelée Calculator, comprenant des méthodes pour effectuer des opérations mathématiques de base, puis à l'implémenter pour les types Integer, Double, et BigDecimal.

1. **Définir l'interface générique Calculator :**

   - Créez une nouvelle interface Java appelée **Calculator**.
   - Utilisez des génériques pour déclarer les types des paramètres et du résultat des opérations mathématiques.

2. **Méthodes d'opérations mathématiques :**
    - Dans l'interface **Calculator**, déclarez des méthodes génériques pour les opérations mathématiques de base, telles que l'addition, la soustraction, la multiplication et la division. Utilisez les génériques pour les types des paramètres et des résultats.
3. **Implémentation pour le type Integer :**
    - Créez une classe appelée **IntegerCalculator** qui implémente l'interface **Calculator** pour le type **Integer**.
    - Implémentez les méthodes de l'interface pour effectuer les opérations mathématiques avec des objets de type **Integer**.
4. **Implémentation pour le type Double :**
    - Créez une classe appelée **DoubleCalculator** qui implémente l'interface **Calculator** pour le type **Double**.
    - Implémentez les méthodes de l'interface pour effectuer les opérations mathématiques avec des objets de type **Double**.
5. **Implémentation pour le type BigDecimal :**
    - Créez une classe appelée **BigDecimalCalculator** qui implémente l'interface **Calculator** pour le type **BigDecimal**.
    - Implémentez les méthodes de l'interface pour effectuer les opérations mathématiques avec des objets de type **BigDecimal**.
6. **Test de chaque implémentation :**
    - Dans une classe de test (par exemple, **CalculatorTest**), créez des instances des classes **IntegerCalculator**, **DoubleCalculator**, et **BigDecimalCalculator**.
    - Effectuez quelques opérations mathématiques en utilisant les méthodes de l'interface et affichez les résultats.

Exercice 8 : Tri générique

Cet exercice vous amènera à écrire un algorithme de tri générique (par exemple, tri à bulles ou tri rapide) capable de trier un tableau d'objets de n'importe quel type. Vous serez ensuite invité à tester cet algorithme avec des types différents.

1. **Définir une méthode de tri générique :**
    - Créez une nouvelle classe ou utilisez une classe existante pour implémenter une méthode de tri générique. Vous pouvez nommer cette classe **GenericSort**.
    - Utilisez un type générique pour déclarer le type des éléments dans le tableau à trier.

2. **Implémenter l'algorithme de tri :**
    - Choisissez un algorithme de tri (par exemple, tri à bulles, tri par sélection, tri rapide, tri fusion) et implémentez-le dans votre méthode générique. Assurez-vous que l'algorithme fonctionne pour n'importe quel type d'objet.
3. **Test de la méthode de tri générique :**
    - Dans une classe de test distincte (par exemple, **SortTest**), créez des tableaux de différents types d'objets (par exemple, **Integer**, **String**, **Double**).
    - Utilisez la méthode de tri générique pour trier ces tableaux.
    - Affichez les tableaux avant et après le tri pour vérifier la précision de l'algorithme de tri générique.
4. **Gestion des comparaisons :**
    - Assurez-vous que votre algorithme de tri générique utilise des comparaisons appropriées pour trier les éléments, en particulier si les éléments du tableau ne sont pas nécessairement comparables nativement.
5. **Discussion sur la performance :**
    - Discutez de la performance de l'algorithme de tri générique. Considérez la complexité temporelle et spatiale de l'algorithme choisi.
6. **Variations et optimisations :**
    - Expérimentez avec d'autres algorithmes de tri ou des optimisations spécifiques pour les différents types d'objets.