TP 10 : Héritage, Généricité et Exceptions

Consignes :

- Avant d'écrire du code sur votre machine, vous allez faire **chaque exercice sur une feuille de papier sans utiliser d'ordinateur**. C'est un bon entraînement pour vérifier si vous comprenez bien les concepts et pour vous préparer pour l'examen. Vous pouvez consulter vos notes de cours pendant ce temps.
- Le code de chaque exercice doit résider dans un package différent.

Date limite de rendu de votre code sur votre dépôt GitLab : **dimanche 2 juin à 23h00**

Exercice 1 - Salle d'attente

Dans cet exercice on se propose de modéliser le principe d'une salle d'attente. Vous allez simuler le comportement de la collection `PriorityQueue` du package `java.util` en écrivant votre propre version de file de priorité. On suppose que la classe `Personne` suivante est donnée (et vous ne devriez pas la modifier).

```
public class Personne {
    private String nom;

    public Personne (String s) {
        nom = s;
    }

    @Override
    public String toString() {
        return nom;
    }
}
```

Avant de créer une vraie salle d'attente, nous devons dans un premier temps créer une classe `PersonnePriorisee` qui permet d'associer une priorité à une Personne.

1. Écrivez une classe `PersonnePriorisee`. Un objet de type `PersonnePriorisee` est défini par une `Personne` ainsi que par une priorité (une donnée de type `int`).
   (a) Définissez le/les attributs de la classe `PersonnePriorisee`.
   (b) Définissez un constructeur qui prend en paramètres une référence vers un objet `Personne` et un entier, et qui initialise les attributs de manière correspondante.
   (c) Ajoutez les *getters* correspondants.
   (d) Redéfinissez la méthode `public String toString()` afin qu'elle retourne l'ensemble des valeurs des attributs sous forme de chaîne de caractères.
2. On se propose maintenant de créer une classe `FilePriorite` qui gérera des objets `PersonnePriorisee` :
   (a) Définissez un attribut de type `ArrayList<PersonnePriorisee>`. C'est dans cette liste qu'on stockera les éléments de la file de priorité.
   (b) Proposez un constructeur sans paramètres `public FilePriorite()` initialisant la file de priorité.

(c) Proposez une méthode `public void ajouter(PersonnePriorisee d)` qui ajoute un objet `PersonnePriorisee` à la `FilePriorite`.

(d) Proposez une méthode `public PersonnePriorisee enlever()` qui retire un élément **de plus petite priorité** de la file et le retourne. Si plusieurs éléments ont la même priorité, vous pourrez en prendre une au hasard. Si la file de priorité est vide, vous vous contenterez de lever une *exception non-contrôlée* (en anglais *unchecked exception*) avec l'instruction

```
throw new ArrayIndexOutOfBoundsException("La liste est vide");
```

ce qui permettra d'arrêter le programme avec un message d'erreur approprié.

3. On propose enfin de créer une classe `SalleAttente` gérant une file de priorité (objet de type `FilePriorite`) :

(a) Définissez le/les attributs de la classe `SalleAttente`.

(b) Proposez un constructeur sans paramètres `public SalleAttente()`.

(c) Proposez une méthode `public void entrer(Personne p, int priorite)` qui ajoute une personne à la salle d'attente en lui affectant la priorité correspondante.

(d) Proposez une méthode `public Personne sortir()` qui retire la prochaine personne devant passer de la salle d'attente, c'est-à-dire une des personnes ayant **le plus petit indice de priorité**. Cette méthode devra retourner la référence correspondante.

4. Dessinez le diagramme de classes de votre application en y indiquant les relations entre classes, les attributs et les méthodes.

5. Voici le code de la classe principale qui vous est donné :

```
public class App {
    public static void main(String[] args) {
        // méthode à compléter
    }
}
```

Créez dans la méthode `main(String args[])` une `SalleAttente` ainsi que deux objets `p1` et `p2` de type `Personne`. Ensuite, faites entrer ces deux personnes dans la salle d'attente, en donnant à `p1` la priorité 4 et à `p2` la priorité 3. Enfin, sortez ces deux personnes dans l'ordre de priorité.

6. D'après ce qui vous a été demandé, la file de priorité de la salle d'attente organise les éléments dans l'ordre croissant des priorités. Imaginons qu'il vous est demandé de rendre le programme plus flexible afin de permettre à l'utilisateur de *décider* le choix de l'algorithme de gestion de la file de priorité à la création de la salle d'attente dans la classe cliente `App`. Par exemple, l'utilisateur pourrait choisir comme critère la priorité minimale, maximale ou une autre priorité qui n'est pas encore définie (mais le sera peut-être un jour)... Proposez une refactorisation de votre code (restructuration du diagramme de classes) afin que le programme respecte cette demande et dessinez le diagramme de classes.
Remarque : Pour cette question, vous ne devrez pas modifier les classes `Personne` et `PersonnePriorisee`. Également, vous n'allez pas utiliser les interfaces `Comparable` ou `Comparator`.

Exercice 2 - Gestion d'exceptions

On s'intéresse maintenant à gérer une salle d'attente à capacité d'accueil limitée, et sur laquelle des erreurs peuvent se produire lorsqu'on veut y ajouter ou faire sortir une personne.

1. Déclarez une classe `SalleCapaciteLimitee` qui "enveloppe" une `SalleAttente` (c'est-à-dire contient un attribut `SalleAttente` comme délégué). Ajoutez-y un constructeur qui prend en paramètre la capacité maximale d'une telle salle.

2. Dans un sous-package `exceptions`, déclarez une classe `SalleCapaciteLimiteeException`, qui représente une nouvelle catégorie d'`Exception`. Cette classe possède un constructeur à un argument de type `String`, qui initialise le message qui correspondra à l'exception. Ce message est généralement stocké dans la classe de base `Exception`, donc dans le constructeur de votre classe, vous vous contenterez avec l'appel du constructeur de la classe mère `Exception` en lui passant en paramètres la chaîne de caractères correspondante.

3. Déclarez `CapaciteMaximaleAtteinteException`, sous-classe de `SalleCapaciteLimiteeException`. Cette nouvelle classe possède un constructeur à un argument `n` de type `int`, qui représente la capacité maximale d'une salle, et appelle le constructeur de la classe de base avec le message "*Cette salle est pleine, elle est limitée à* " + *n* + " *personnes.*".
Rappel : toutes les classes d'exception doivent être définies dans le package `exceptions`.

4. Ajoutez à la classe `SalleCapaciteLimitee` une méthode `public void entrer(...)` avec les mêmes paramètres que celle d'une `SalleAttente` mais qui annonce qu'elle est susceptible de lever une exception de type `CapaciteMaximaleAtteinteException`. L'implémentation de cette méthode vérifie que le nombre actuel de personnes dans la salle est inférieur au nombre maximum autorisé dans cette salle, auquel cas elle appelle la méthode `public void entrer(...)` de la `SalleAttente` qu'elle contient. Si par contre la capacité maximale est atteinte, il faudra lever une `CapaciteMaximaleAtteinteException`.

5. Dans votre `main(String args[])`, faites entrer des personnes jusqu'à atteindre la capacité maximale d'une `SalleCapaciteLimitee` et gérez (`catch`) une `SalleCapaciteLimiteeException` pour afficher le message de l'`Exception` (la méthode `getMessage()` de cette classe).

6. Définissez maintenant une nouvelle `SalleVideException`, dont le constructeur (sans argument) définit le message "*Cette salle est vide.*".

7. Ajoutez à la classe `SalleCapaciteLimitee` une méthode `public Personne personneSuivante()` (comme chez le médecin) qui annonce qu'elle peut lever une `SalleVideException`. L'implémentation de cette méthode ne teste pas le nombre actuel de personnes dans la salle, mais appelle normalement la méthode `public Personne sortir()` de la `SalleAttente` interne. D'après l'exercice 1, cette méthode est susceptible de lever une `ArrayIndexOutOfBoundsException`. Vous allez maintenant gérer cette exception dans `public Personne personneSuivante()` pour lever une `SalleVideException`.

8. Constatez que votre `main(String args[])` de la classe `App` fonctionne correctement si vous faites sortir une personne d'une `SalleCapaciteLimitee` vide.

Exercice 3 - Généricité

Remarquez que la classe `PersonnePriorisee` associant une priorité à une personne, peut également faire sens pour une *tâche* à faire, une *voiture* sur un parking, un *produit de fabrication* sur une chaîne d'assemblage dans une usine, ou tout autre type d'objet qu'on souhaiterait prioriser.

1. Proposez une classe générique `ObjetPriorise`, correspondant à la version générique de la classe `PersonnePriorisee`. A vous d'ajouter les `<T>` aux endroits qui vous semblent nécessaires et de nommer correctement les attributs et les méthodes de cette nouvelle classe.

2. Réécrivez la classe `SalleAttente` en une nouvelle classe `SalleAttenteGenerique`, qui modélise toujours une salle d'attente mais qui peut être une salle d'attente de personnes, ou une salle d'attente de voitures, ou tout autre type. À la place de `PersonnePriorise` cette classe utilisera le type `ObjetPriorise`.
3. Complétez la classe principale `App` en instanciant plusieurs objets de type `SalleAttenteGenerique` : une pour les personnes, une pour des voitures (vous créerez une classe `Voiture` pour cela).