Ecole nationale Supérieure d'Informatique (ESI)-Alger 08/06/2024

UEF4.3. Programmation Orientée Objet

Examen Final (1h45)

Documents/Téléphones interdits

Exercice 1 (12 pts)

Nous souhaitons concevoir un système de gestion des patients pour un service d'urgence d'un hôpital. Le système doit permettre de gérer les informations des patients, de suivre leur état et de prioriser les traitements en fonction de la gravité de leur état.

À l'arrivée d'un patient au service d'urgence, le service enregistre son nom, prénom, date de naissance, sexe, ses conditions médicales (ensemble de maladies pour lesquelles le patient est en cours de traitement, s'il en existe), et la gravité de son état (de 1 à 10, 10 étant le plus grave). La gravité de l'état, combinée au temps d'arrivée au service (date/heure), détermine la priorité de passage du patient à la salle de soins, calculée comme suit :

*priorité = - (gravité d'état*la durée passée en salle d'attente)*

Le système dispose également de la liste du personnel soignant disponible au service. Chaque membre du personnel est caractérisé par son nom, sa spécialité, son identifiant unique ainsi que le numéro de la salle de soins qu'il occupe actuellement.

Après le passage d'un patient à la salle de soins, le système enregistre les actes médicaux qu'il a subis. Ces actes peuvent être des diagnostics ou des soins. Un diagnostic a un nom, il est établi par un personnel soignant à une certaine date et détermine une maladie avec un pourcentage de fiabilité. La maladie fait partie d'un ensemble fini de valeurs (exemple : Grippe, Covid, Hypertension). Un soin a un nom, il est réalisé par un personnel soignant, à une certaine date en utilisant un produit médical donné. Certains soins nécessitent la prescription d'une ordonnance contenant des médicaments, précisant pour chacun une posologie (quantité entière) et une fréquence (nombre de prises du médicament par jour).

1. Proposez une modélisation orientée objet du système, en précisant sur le diagramme les types des classes et de relations entre les classes, ainsi que les attributs principaux de chaque classe.
2. Les patients en salle d'attente sont classés par priorité (la plus petite valeur est la plus prioritaire). On vous propose d'utiliser un *PriorityQueue*[1] pour les gérer. Expliquez comment mettre en œuvre cette solution :
   - Donnez l'instruction java permettant de déclarer et instancier cette collection, et précisez son emplacement.
   - Expliquez les conditions nécessaires pour le bon fonctionnement du programme et donnez le code java des méthodes nécessaires en précisant leurs emplacements.
   - Peut-on utiliser une autre collection pour assurer le même fonctionnement ? Si oui expliquez laquelle.

[1] L'élément récupéré en tête de queue est le plus petit

3. Afin de réaliser des statistiques sur l'activité de l'hôpital, nous souhaitons enregistrer l'historique des patients traités dans le service d'urgence et les actes médicaux réalisés. Pour chacun des cas ci-dessous :
   - Expliquez quelle est la collection la plus adéquate.
   - Donnez l'instruction java permettant de déclarer et instancier cette collection et précisez son emplacement.
   - Expliquez les conditions nécessaires pour le bon fonctionnement du programme et donnez le code java des méthodes nécessaires et leurs emplacements.

Cas 3.1. Les patients traités dans le service sont classés par personnel soignant, ordonnés par ordre de passage dans le temps (date/heure).
Cas 3.2. Les patients traités sont classés par acte, puis par date de traitement.

Nous souhaitons ajouter au système une fonctionnalité permettant de générer un rapport d'activité par le service d'urgence ou par les membres du personnel soignant. Le service indiquera dans son rapport le nombre de malades traités alors qu'un membre du personnel soignant indiquera le nombre de patients qu'il a traités pour une maladie donnée.

4. Quelle solution orientée objet proposez-vous pour compléter votre diagramme de telle sorte que la portion code suivante puisse s'exécuter :

```
public void afficherRapport(Rapporteur r){
        System.out.println(r.genererRapport());  }
```

Annexe :

A. Opérations basiques de l'interface java.util.Collection<E> :

- int size(): retourne le nombre d'éléments dans la collection
- boolean isEmpty(): retourne true si la collection est vide
- boolean contains (Object o) : retourne true si la collection contient l'objet o (basé sur la méthode equals de la classe E)
- boolean add (E o): ajoute l'élément o à la collection. Retourne false si doubles interdits
- boolean remove(Object object): supprime l'objet o
- Iterator iterator(): retourne l'itérateur de la collection

B. Opérations basiques de l'interface java.util.Map<K,V>

- V put(K k, V v) : ajouter v avec la clé k : retourne l'ancienne valeur associée à la clé si la clé existait déjà
- V get(Object k) : retourne la valeur associée à la clé ou null
- V remove(Object k) : supprimer l'entrée associée à k
- boolean containsKey(Object k) : retourne true si la clé k est utilisée
- boolean containsValue(Object v) : retourne true si v existe dans la map
- Set<Map.Entry<K, V>> entrySet() : récupère les entrées (paires clé-valeur) sous forme de Set<Map.Entry<K,V>>
- Collection<V> values() Récupère les valeurs sous forme de Collection<V>
- Set<K> keySet() Récupère les clés sous forme de Set<K>
- int size() : retourne le nombre d'entrées dans la table
- boolean isEmpty() : retourne true si la map est vide
- void clear() : vide la map