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 oeuvre 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:

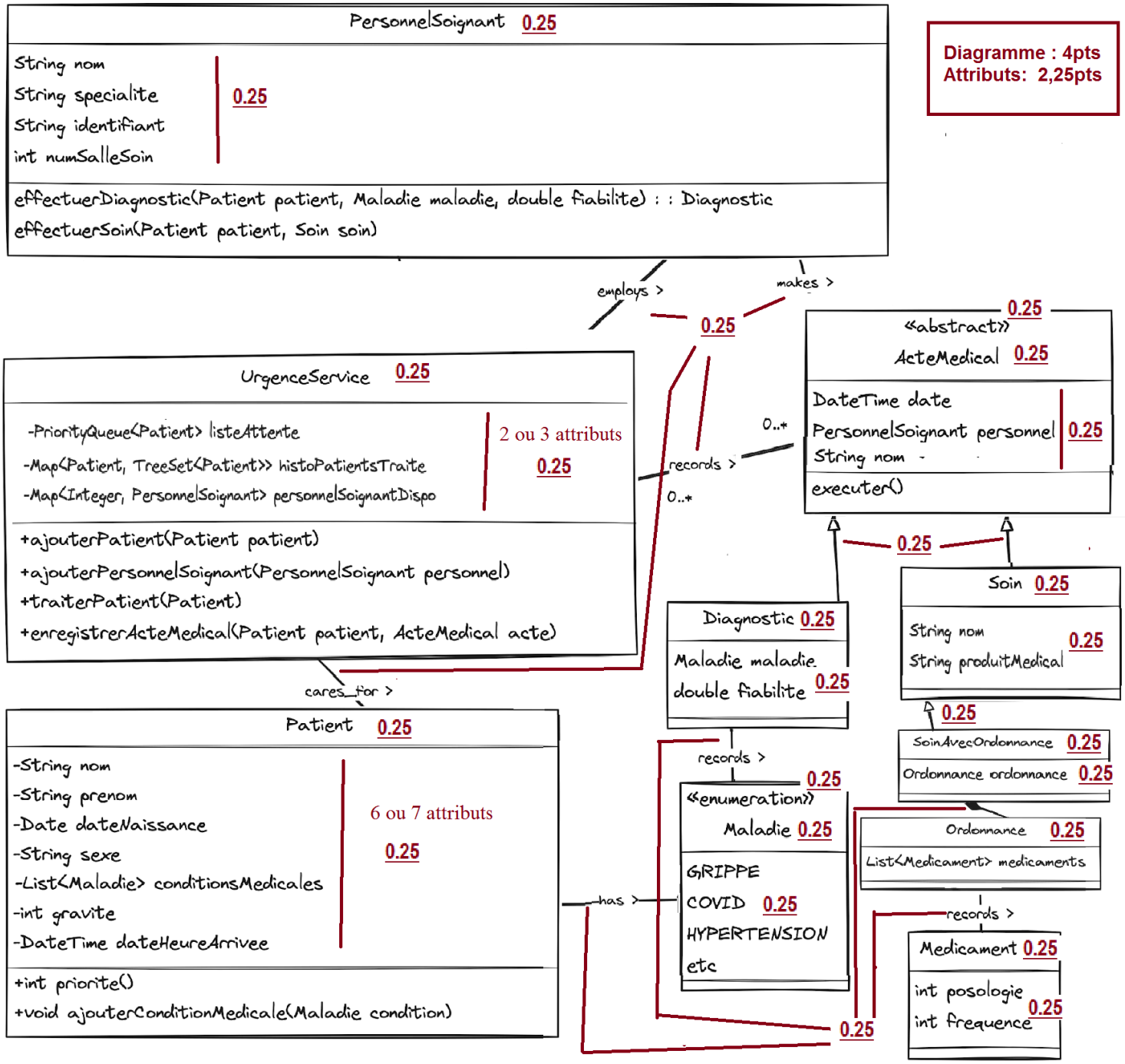
A. Opérations basiques de l'interface java.util.Collection<E> :

- 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

UEF4.3. Programmation Orientée Objet



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 PriorityDeque pour gérer les patients en salle d'attente. Expliquez comment mettre en oeuvre cette solution : **2pts**

- Donnez l'instruction java permettant de déclarer et instancier cette collection, et précisez son emplacement.

Dans la classe UrgenceService. 0,25
PriorityDeque<Patient> listeAttente = new PriorityDeque<Patient>(). 0,25

- Expliquez les conditions nécessaires pour le bon fonctionnement du programme et donnez le code java des méthodes nécessaires en précisant leurs emplacements.

La classe Patient doit implémenter Comparable de sorte que la priorité la plus petite soit mise en tête de la file. 0,25

..class Patient implements Comparable<Patient>{ **0,25**
…
public int compareTo(Patient p){ **0,25**
If (this.calculPriorite()==p.calculPriorite()) return 0;
else if (this.calculPriorite()<p.calculPriorite()) return -1;
else return 1;
}
public int calculPriorite() { **0,25**
ZonedDateTime zonedDateTime = heureArrivee.atZone(ZoneId.of("Africa/Algiers"));
return -(gravite * (int) (System.currentTimeMillis() - zonedDateTime.toInstant().toEpochMilli()));
}}

Dans la classe UrgenceService, la méthode ajouterPatient doit utiliser la méthode add de priorityQue, et pour faire passer un patient utiliser poll pour le retirer de la file et le mettre après dans la liste des patients traités.

- Peut-on utiliser une autre collection pour assurer le même fonctionnement? Si oui expliquez laquelle. **Oui, on peut utiliser un TreeSet avec les mêmes conditions. 0,5**

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). 1,75

Collection la plus adéquate **: HashMap<PersonnelSoignant, TreeSet<Patient>>: clé personnel car classé par PersonnelSoignant, puis trié par heure traitement patient qu'on peut enregistrer dans patient, sinon une autre HashMap<Double/Time, Patient>, avec Double ou Time est le temps de traitement 0,25**
Emplacement: Dans la classe UrgenceService. 0,25
Instruction : 0,25 **HashMap<PersonnelSoignant, TreeSet<Patient>>patientTraite=new HashMap<PersonnelSoignant, TreeSet<Patient>>().** **Ou** **HashMap<PersonnelSoignant, HashMap<Double/Time, Patient>>patientTraite=new HashMap<PersonnelSoignant, HashMap<Double/Time, Patient>>()**
Conditions 0,5 **PersonnelSoignant doit redéfinir equals et hashCode** **Si on utilise le TreeSet, Patient doit ajouter un attribut tempsPassage et implémenter Comparable selon ce temps de passage.**
Code 0,5 **..class PersonnelSoignant{** **…** **public int hashCode(){ return identifiant.hashCode();}** **public boolean equals(Object o){** **return ((PersonnelSoignant)o).getIdentifiant().equals(identifiant);}}** **rq: on peut aussi accepter d'ajouter une collection de patients traités dans PersonnelSoignant: HashMap<Double/Time, Patient> ou TreeSet<Patient>.**

Cas 3.2. L'historique des patients sont classés par acte, puis par date. 1,75

***Collection la plus adéquate* : HashMap<ActeMedical, HashMap<Date,HashSet<Patient>>>: clé ActeMedical car classé par Acte, puis HashMap dont la clé et la date est la valeur est l'ensemble des patients traité en cet date ou par date/time et valeur patient**
Emplacement: Dans la classe UrgenceService. 0,25
Instruction : 0,25 **HashMap<ActeMedical, HashMap<Date,HashSet<Patient>>>patientTraite=new HashMap<ActeMedical, HashMap<Date,HashSet<Patient>>>(). (Tout dépend l'explication)**
Conditions 0,5 **ActeMedical doit redéfinir equals et hashCode** **Selon la valeur et son explication:** **Si on utilise le HashSet de Patient, la classe Patient doit redéfinir equals et hashCode.** **Si on utilise le TreeSet de Patient, la classe Patient doit ajouter un attribut tempsPassage et implémenter Comparable selon ce temps de passage.**
Code 0,5 **..class ActeMedical{** **…** **public int hashCode(){ return date.hashCode()+personnel.hashCode();}** **public boolean equals(Object o){** **return** **(((ActeMedical)o).getDate().equals(date))&&(((ActeMedical)o).getPersonnel().equals(personnel));}}**

4. Ajouter une interface "Rapporteur" comportant une méthode public String genererRapport() qui sera implémentée par les classes ServiceUrgence et Personnel soignant. **1pt** (ServiceUrgence peut avoir le nombre de patients traité à partir de l'attribut de la q3. Personnel Soignant doit avoir un attribut pour avoir l'info (exemple Map<nom_acte, set<patient>>)).

Nom :…….. Prénom:……..……..... Groupe :……

Exercice 2 (8 pts): Merci de répondre directement sur la copie

Supposons que vous ayez un objet et que vous souhaitiez en créer une copie exacte (cloner l'objet). Comment feriez-vous? Tout d'abord, vous devez créer un nouvel objet de la même classe. Ensuite, vous devez parcourir tous les champs de l'objet d'origine et copier leurs valeurs dans le nouvel objet. Mais certains objets ne peuvent pas être copiés de cette façon, car certains champs de l'objet peuvent être privés et non visibles de l'extérieur. Dans le code qui suit, on propose un extrait d'une solution qui permet de cloner des formes géométriques (*Shape*) et de les enregistrer dans un registre (*ShapeRegistry*). Nous supposons disposer des formes *Rectangle* et *Circle*.

Analysez ce code et **complétez** les parties manquantes (le code de la classe *Circle* n'est pas demandé), en respectant les indications suivantes :

- Les attributs de chacune des classes ne sont accessibles que dans la classe
- L'égalité des formes géométriques est basée uniquement sur le type concret de l'objet et sa couleur.
- L'ajout d'une forme géométrique égale à une forme existante au registre provoquera le lancement d'une exception *ShapeExistException*.
- Si on essaye de remplacer une forme géométrique qui n'existe pas dans le registre, une exception *ShapeNotExistException* sera lancée.

```
public abstract class 0.25 Shape {
    private int x;
    private int y;
    private String color;
    public Shape(int x, int y, String color) {
          this.x = x;  this.y = y; this.color = color;  }
    public Shape(Shape target) { if (target != null) {
                                    this.x = this.x = target.x;;
                                    this.y = this.y = target.y;  0.25 ;
                                    this.color = this.color =target.color; ; }
}
    public abstract Shape 0.25 clone();
    public String toString() {return "Shape [x="+x+", y="+y+", color="+color+"]";}
    public boolean hasColor(String color){
      return return this.color.equals(color)  0.25 }
     public int hashCode() {
          {return getClass().hashCode()+ color.hashCode();  0.25  }
     public boolean equals(Object object2) {
         if (!(object2 instanceof Shape)) return false;
     Shape shape2 = (Shape) object2;  return shape2.color.equals(color);  0.5
       }
}
```

```
public class Rectangle extends Shape {
      private  0.25 int width;
      private int height;
     public Rectangle(int x, int y, String color, int width, int height) {
      super(x, y, color);
      this.width = width;  0.25
      this.height = height;
    }
    public Rectangle(Rectangle target) {
      super(target);  0.25
    if (target != null) {this.width = target.width;  this.height = target.height;  }
    }
```

```
 public Shape clone() {
      return new Rectangle(___this    0.25_____);}

    public boolean equals(_____Object object2_____) {
      if ((____object2_____instanceof Rectangle) && ___super.equals(object2)____)
          return ___true____;

          return ___false____;    }      0.25

    public String toString() {
     return super.toString()+"Rectangle [width=" + width + ", height=" + height + "]";
    }
}
```

```
public class ShapeRegistry {

private static ShapeRegistry instance;

private Set<Shape> shapeRegistry;

private ShapeRegistry() { shapeRegistry = _________ new HashSet<Shape>();  _______}

  public ____static ShapeRegistry  0.25 ____ getInstance(){

       if (instance==null) instance=new ShapeRegistry();
       return instance;
   }
  public void addShape(_Shape shape) throws ShapeExistException{      0.5
       _      if(!(shapeRegistry.add(shape))) throw new ShapeExistException();  0.5_.
   }
  /* Méthode pour remplacer une forme géométrique par une autre qui lui est égale */
  public void replaceShape(Shape shape)___ throws ShapeNotExistException{     0.5_
   __if(!(shapeRegistry.contains(shape))) throw new ShapeNotExistException();   0.5

     else { shapeRegistry.remove(shape); shapeRegistry.add(shape);         0.5
   }__.
  /* Méthode pour cloner une forme géométrique de type C et de couleur color,
existante
     dans le registre ; retourne null si elle n'existe pas */

 public Shape GetCloneShapeByColor(String color, Class c){
       _for (Shape shape : shapeRegistry)

        if (shape.hasColor(color)&& shape.getClass()==c) return shape.clone();

       return null;    }       1


 public String toString() {return "ShapeRegistry [shapeRegistry="+shapeRegistry+"]";}
}
```

```
public class Demo {

    public static void main(String[] args) {

      ShapeRegistry shapeRegistry= ShapeRegistry.getInstance();
      Shape circle=new Circle(10, 20, "red", 15);
      Shape circleClone= circle.clone();

   /* Donner les instructions pour ajouter circle et circleClone dans le registre
     de sorte que le programme puisse toujours s'exécuter */

     try{  shapeRegistry.addShape(circle);
           shapeRegistry.addShape(circleClone);
      }catch(ShapeExistException e){}        0.5

     ____________________________________________________________
     ____________________________________________________________
     ____________________________________________________________

     Rectangle rectangle = new Rectangle(10, 20,"bleu", 30, 30);

   /* Donner les instructions pour ajouter rectangle dans shapeRegistry puis
      le remplacer avec new Rectangle(10,20,"red",50,60) de sorte que le programme
      puisse toujours s'exécuter */

     try{  shapeRegistry.addShape(rectangle);
         shapeRegistry.replaceShape(new Rectangle(10, 20,"red", 50, 60));
      }catch(ShapeExistException e){}
       catch(ShapeNotExistException e){}    0.5

     ____________________________________________________________
     ____________________________________________________________

     //récupérer de shapeRegistry un clone de type rectangle avec une couleur bleu
     shapeRegistry.GetCloneShapeByColor("bleu",Rectangle.class));      0.5
     ____________________________________________________________


     }
}
 public class ShapeExistException extends Exception {}

 public class ShapeNotExistException extends Exception {}
```