Interfaces

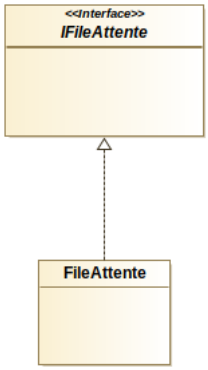
1 Extraction d'une interface FileAttente

Question 1. On suppose connue une classe `Personne` et ci-dessous on vous donne le code d'une classe représentant des files d'attente de personnes. Proposez une interface décrivant le type de cette classe, c'est-à-dire ce qu'elle peut exposer de manière publique aux autres classes qui veulent l'utiliser.

Programme 1 : *Classe FileAttente*

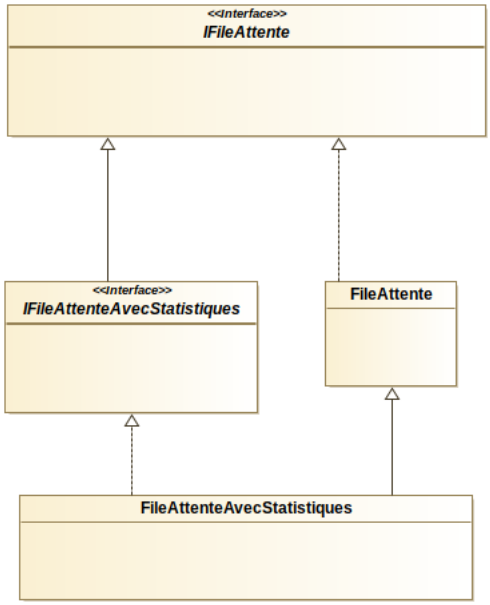
```
public class FileAttente {
        private String nomFile;
        private static String reglementationFile = "sans priorité";
        private ArrayList<Personne> contenu;
        public FileAttente(){
                contenu=new ArrayList<Personne>();
        }
        public void mettreEnFile(Personne p){
                contenu.add(p);
        }
        public Personne defiler(){
                Personne p=null;
                if (!contenu.isEmpty())
                {p=contenu.get(0);
                contenu.remove(0);}
                return p;
        }

        public boolean estVide(){
                return contenu.isEmpty();
        }
        public int taille(){
                return contenu.size();
        }

        public void vider(){
                while(!estVide()) {
                        this.defiler();
                }
        }

        public String toString(){
                return ""+descriptionContenu();
        }
        private String descriptionContenu(){
                String resultat = "";
                for (Personne p:this.contenu)
                        resultat += p + " ";
                return resultat;
        }
}
```

Question 2. Modifiez la classe `FileAttente` pour qu'elle implémente l'interface que vous avez créée.



Question 3. Proposez une interface pour représenter les files d'attente avec des statistiques. Des opérations supplémentaires permettent de connaître le nombre d'entrées et le nombre de sorties qui ont eu lieu depuis la création de la file.

Question 4. Proposez une classe qui implémente cette nouvelle interface (file d'attente avec statistiques).



2 Autour d'une interface de créneau horaire

On souhaite mettre en place pour une formation un emploi du temps représentant une semaine type d'enseignement.

Question 1. Mettez en place une interface CreneauHoraire (modélisation UML+ implémentation en Java) qui spécifie que l'on peut obtenir d'un créneau horaire :

- les heures du début du créneau (un entier de 0 à 23)
- les minutes du début du créneau (un entier de 0 à 59)
- le séparateur entre heures et minutes pour le formatage, ce séparateur est " :"
- l'heure de début formatée sous forme d'une chaîne de type HH :MN (où HH et MN sont à 2 digits) ; cela pourra être placé sous forme de méthode par défaut
- la durée du créneau en minutes
- un booléen déterminant si le créneau en chevauche un autre ; cela pourra être placé sous forme de méthode par défaut puisque le calcul à réaliser est (start <= endc) && (end>= startc), avec start et end le début et la fin du créneau, et startc et endc le début et la fin de l'autre créneau.

Question 2. On souhaite mettre en place une première implémentation de l'interface CreneauHoraire : CreneauHoraireFDS. Les créneaux horaires FDS, comme vous le savez, sont par tranches d'1h30, avec 15 minutes de pause entre chaque créneau, aux 7 horaires de début possibles : 8h, 9h45, 11h30, 13h15, 15h, 16h45 et 18h30. Les créneaux horaires FDS seront donc représentés par un numéro entre 1 et 7 indiquant l'horaire de début parmi les 7 horaires possibles, ainsi que le nombre de tranches consécutives de 90 minutes occupées. Pour implémenter la durée, pensez à rajouter, s'il y a plus d'une tranche de 90 minutes, le quart d'heure de pause comprise entre 2 tranches (2 tranches comptent pour une durée de 2*90+15 minutes). Le constructeur des créneaux horaire FDS prendra en paramètre l'horaire de début (sous forme d'un entier) et le nombre de tranches de 90 minutes du créneau.

Vous compléterez le modèle de la question précédente et donnerez l'implémentation en Java.

Question 3. Mettez en place une deuxième implémentation de créneaux plus classiques. Pour cela on se base sur la classe LocalTime (package java.time) de l'API Java pour l'heure de début du créneau, et un entier (en minutes) pour la durée du créneau. Le constructeur de ces créneaux prendra en paramètre 3 entiers : l'un pour les heures, l'autre pour les minutes, le dernier pour la durée.

On envisage de redéfinir en la spécialisant la méthode par défaut qui regarde si 2 créneaux se chevauchent, afin d'utiliser les méthodes de comparaison prévues dans la classe LocalTime. Est-ce possible ?

Vous compléterez le modèle de la question précédente et donnerez l'implémentation en Java.

Question 4. Mettez en place une classe Semainier. Pour chaque jour de la semaine (utiliser l'énumération java.time.DayOfWeek), on maintient une liste de créneaux qui ne doivent pas se chevaucher les uns les autres. Les créneaux peuvent être des créneaux FDS et/ou des créneaux classiques. Placez juste dans cette classe de quoi ajouter un créneau à un jour de la semaine, et de quoi afficher (sous forme de texte) le semainier.

Vous compléterez le modèle de la question précédente et donnerez l'implémentation en Java.

Question 5. Pour les plus rapides, proposez une méthode pour trier les créneaux de chaque journée par ordre chronologique de début de créneau, en vous inspirant du tri des palettes de couleurs ébauché dans les diapos de cours.