SEMAINE 6 : TP2 - ORDONNANCEUR (1ÈRE PARTIE)

Dans cette séance, on voit l'utilité d'utiliser les interfaces dans un exemple plus complexe.

1 ORDONNANCEUR GÉNÉRIQUE

On souhaite écrire une classe qui va permettre de gérer le traitement d'objets prioritaires.

Un objet est **prioritaire** si et seulement si il dispose d'une méthode permettant de connaître sa priorité.

On va donc définir une interface `Prioritaire` comme suit.

```
public interface Prioritaire{
  /** retourne un entier représentant la priorité de l'objet */
       public int getPriorite();
}
```

Recopier l'interface `Prioritaire` dans un fichier `Prioritaire.java`.

Tout objet qui implémente cette interface doit pouvoir être géré par notre *ordonnanceur*.

Un ordonnanceur est une boîte dans laquelle on entre des objets, et qui peut sortir à la demande l'objet à traiter en priorité.

Il existe plusieurs manières différentes d'implémenter un ordonnanceur, c'est pourquoi on définit aussi une interface `Ordonnanceur`.

Les spécifications de l'interface `Ordonnanceur` sont les suivantes :

```
 public interface Ordonnanceur{

   /** ajoute un objet prioritaire à l'ordonnanceur */
   public void ajouteObjet(Prioritaire p);

   /** extrait l'objet qui doit sortir le plus tôt et le supprime de l'ordonnanceur */
  public Prioritaire plusPrioritaire();

   /** retourne vrai s'il n'y a plus d'objets dans l'ordonnanceur */
   public boolean estVide();
 }
```

Recopier l'interface `Ordonnanceur` dans un fichier `Ordonnanceur.java`.

2 ORDONNANCEMENT SANS PRIORITÉ

Une 1ère implémentation possible pour un ordonnanceur est une ***file*** (**FIFO** : First In First Out) .

Note : pour créer une file, on encapsule une `ArrayList`*, en ajoutant tout nouvel élément en fin de liste et en retournant le 1ier élément lorsqu'on appelle la méthode* `plusPrioritaire()`*. Ainsi, c'est toujours le premier1ier entré qui est le 1ier sorti.*
Autrement dit ***la priorité n'est pas prise en compte dans ce 1er exercice :*** *les objets de priorités différentes sont traités de la même manière, la méthode* `getPriorite()` *de* `Prioritaire` *n'est pas utilisée.*

1. **Ecrire la classe** `OrdoFile` correspondante, qui implémente l'interface `Ordonnanceur`.

2. Pour tester votre ordonnanceur :
   1. **Ecrire une classe** `Tache` qui implémente l'interface `Prioritaire` avec comme attributs :

      - une chaîne représentant l'intitulé de la tâche à réaliser,
      - une priorité, définie à l'instanciation, et modifiable ultérieurement.
        (priorité sans effet pour cette première partie)

   2. **Ecrire une classe** `EssaiOrdonnanceur` contenant uniquement la méthode `main()` qui instancie des `Tache` pour mettre en œuvre l'ordonnanceur `OrdoFile.`

3 ORDONNANCEMENT AVEC PRIORITÉ

Une implémentation plus fine permet de gérer les priorités différentes.

Pour cela, on va créer une classe `OrdoAvecPriorite` qui utilise un tableau de `nbMax` objets de type `OrdoFile`.

On a une file par niveau de priorité.

Un élément entré à l'aide de la méthode `ajouteObjet()` est placé dans la file correspondant à sa priorité.

On considère que les objets de priorité 0 sont les plus prioritaires, puis les priorités vont par ordre décroissant.

Dans un premier temps, voici la stratégie qui sera utilisée pour la méthode `plusPrioritaire()` :

- on retourne l'élément le plus ancien de la file la plus prioritaire.

Comment peut-on être sûr qu'un élément entré a bien une priorité ?

Ecrire la classe `OrdoAvecPriorite`**.**