Exercices dirigés
séance n°9 - corrigé

Exercice 1 : piles et files

Un système muti-tâches peut exécuter *n* tâches en quasi parallélisme. Chaque tâche est munie d'une priorité et d'un numéro. Elles sont rangées, dans l'ordre de leur arrivée, sur une *pile*. A chaque niveau de priorité est associée une *file*. Les tâches sont ensuite dispatchées sur l'une des files selon leur priorité de manière à ce que, pour une même priorité, la plus ancienne soit la première traitée.

Pour tester ce système, on considérera 3 niveaux de priorité. Le résultat du test sera un affichage de chaque tâche (sous la forme (heure, numéro)) rangés selon un ordre de priorité .

L'analyse de ce problème montre la nécessité de deux types de données à construire: les classes `Pile` et `File`.

Question 1

Construire la classe `Tache`.

Question 2

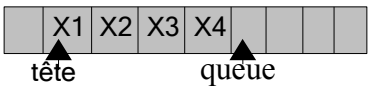
Construire la classe `Pile`. Les opérations possibles sont :

- `empiler` une tâche (ajouter une tâche au sommet de la pile)
- `depiler` une tâche (ce qui provoque son élimination de la pile)
- `sommet` (consulter le sommet de la pile sans le supprimer de la pile)

On s'inspirera de la classe `Liste` vue en cours pour construire la classe `Pile`. Il s'agira donc d'une représentation chaînée.

Question 3

Une file est accessible par une tête et une queue. Les éléments sont ajoutés en queue et retirés en tête (si elle n'est pas vide).



Construire la classe `File`. On choisira une représentation de la classe `File` sous la forme d'un tableau de taille suffisante. Les opérations possibles sont :

- enfiler une tâche (ajouter un élément dans la file selon l'ordre de priorité). On suppose que 2 tâches ne peuvent pas arriver en même temps (dates différentes). On n'envisagera pas de gestion circulaire du tableau.
- défiler, retourner l'élément en tête de file (c'est-à-dire le plus prioritaire).
- tester si elle est pleine
- tester si elle est vide

Question 4

Comment transformer ce système en exploitant les possibilités de l'Orienté Objet ?

Question 5

Ecrire un programme de test qui générera aléatoirement un certain nombre de tâches, les rangera selon leur priorité puis les affichera sous la forme (numéro, priorité) selon l' ordre de priorité et d'arrivée décroissante.

Rappel :

la méthode `int nextInt(int i)` de la classe `java.util.Random` génère un entier aléatoirement dans l'intervalle[0,i[.

--- Exercice 1 ------ Solutions ----------

Question 1

```
public class Tache{
    private int priorite;
    private int numero;

    public Tache(){}
    public Tache(int priorite, int numero){
        this.priorite = priorite;
        this.numero = numero;
    }

    public long getNumero(){
        return numero;
    }

    public int getPriorite(){
        return priorite;
    }

    public void setNumero( long msecs ){
        this.msecs = msecs;
    }

    public void setPriorite( int priorite ){
        this.priorite = priorite;
    }

    public void println(){
        System.out.print("[priorité : "+priorite);
        System.out.println(", date : "+msecs+"]");
    }
}
```

Question 2

```
public class Pile{
   Tache tache;
   Pile suivant;

   public Pile(){}
   public Pile( Tache tache, Pile suivant ){
       this.tache = tache;
       this.suivant = suivant;
    }

    public Pile empiler( Tache tache ){
        return new Pile(tache,this);
    }

    public Pile depiler(){
        return suivant;
    }

    public Tache sommet(){
        return tache;
    }
```

```
    public void println(){
       Pile ref = this;
       while( ref!=null ){
           (ref.sommet()).println();
           ref = ref.depiler();
      }
      System.out.println();
   }
}
```

Question 3

```
import static java.lang.System.*;
public class File{
   private Tache[] taches;
   private int queue;
   private int tete;
   private int taille=0;
   private boolean fileVide; // filePleine si gestion circulaire

   public File(){}
   public File( int taille ){
       this.taille = taille;
       taches = new Tache[taille];
       queue = 0;
       tete = 0;
       fileVide = true;
       // filePleine = false;
    }

    public void enfiler( Tache tache )throws FileException{
      // if ( !filePleine ) { si gestion circulaire
        file[queue] = x;
        queue = queue + 1;
      // (queue=queue+1)%taille si gestion circulaire
        if ( fileVide ) fileVide = false;
      // if ( tete == queue ) filePleine = true;
      // si gestion circulaire
      }
      // else throw new FileException( "file pleine" );

    }

    public Tache defiler()throws FileException{
      if ( !fileVide ) {
            Tache x = file[tete];
            tete = tete + 1;
        //tete = (tete + 1)%taille si gestion circulaire
        //  if( filePleine ) filePleine = false;
            if ( tete == queue ) fileVide = true;
            return x;
      }
      else throw new FileException( "file vide" );
    }

    public boolean vide () { return fileVide; }

    public boolean pleine () { return filePleine;  }
```

```
    public void println(){
      if ( fileVide == true ) System.out.println("Vide");
      else{
        System.out.print(file[0].getPriorite()+" : ");
        for ( int i=tete;i<queue;i++ )
            System.out.print(file[i].getNumero()+" ");
        System.out.println();
       }
     }
   }
}
```

Question 4

A condition de renommer les méthodes et de modifier la signature des méthodes de la classe File :

```
interface PileFile{
      public PileFile ajouter( Tache tache );
      public  PileFile supprimer();
      public Tache tete();
}

public class Pile implements PileFile{
      Tache tache;
      Pile suivant;

      public Pile(){}
      public Pile( Tache tache, Pile suivant ){
                  this.tache = tache;
                  this.suivant = suivant;
       }
      public PileFile ajouter( Tache tache ){…}
      public  PileFile supprimer(){…}
      public Tache tete(){…}
      ….
}

public class File implements PileFile{
      private Tache[] taches;
      private int queue;
      private int tete;

      public FilePriorite(){}
      public FilePriorite( int taille ){
                  taches = new Tache[taille];
                  queue = 0;
                  tete = 0;
       }

      public PileFile ajouter( Tache tache ){…}
      public  PileFile supprimer(){…}
      public Tache tete(){…}
      ….
}
```

Question 5

```
public class TestTaches{
   public static void main(String[] args){
        int numero = 1;
        int priorite = (new java.util.Random()).nextInt(4);
        Tache tache = new Tache(priorite,numero);
        Pile pile = new Pile(tache,null);
        for( int i=2;i<=15;i++){
            priorite = (new java.util.Random()).nextInt(3);
            tache = new Tache(priorite,i);
            pile = pile.empiler(tache);
         }
         // affichage de la pile
         pile.println();
         // transformation en 3 files de priorité
         File[] file = new File[3];
         for( int i=0;i<3;i++ )
             file[i]=new File(15);

         while( pile!=null ){
             tache = pile.sommet();
             switch( (int)tache.getPriorite() ){
                 case 0 : file[0].enfiler( tache );break;
                 case 1 : file[1].enfiler( tache );break;
                 case 2 : file[2].enfiler( tache );break;
                 default : break;
             }
             pile = pile.depiler();
         }
         System.out.println();
         System.out.println(" ------ les files ------ ");
         System.out.println();
         for( int i=0;i<3;i++ )
             file[i].println();

    }

}
```