PROG 1 – TD 2

1. Tri par fusion

L'objectif de cet exercice est de trier un tableau d'éléments en utilisant l'algorithme de tri par fusion. Pour ce faire, nous allons utiliser la file que nous avons implémentée au TD précédent pour représenter des files d'éléments triés qui seront fusionnées pour obtenir une unique file d'éléments triés.

Question 1 : Fusion de deux files triées.
Écrivez une fonction générique *fusionFile* permettant de fusionner deux files triées dans l'ordre croissant (l'élément en tête de file est l'élément le plus petit, l'élément en fin de file est l'élément le plus grand). Cette fonction retournera une nouvelle file triée dans l'ordre croissant. Les éléments contenus dans les files implémenteront l'interface *Comparable<T>* et disposeront donc de la méthode *compareTo* permettant de comparer deux instances de ces éléments.

Question 2 : Création d'une file de files d'éléments à partir d'un tableau.
Écrivez une fonction générique *initialisationFiles* prenant un tableau d'éléments en paramètre et retournant une file contenant des files elles mêmes contenant un unique élément. Le rôle de cette fonction est de créer autant de files que d'éléments contenus dans le tableau fourni en paramètre. Chacune de ces files contiendra un unique élément du tableau d'origine. Ces différentes files seront stockées dans une file résultat comme présenté sur la Figure 1: Tableau d'origine et file de files d'éléments de ce tableau.

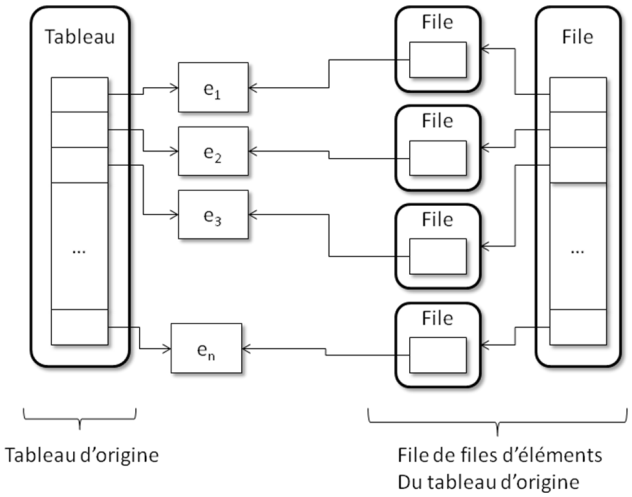

Figure 1: Tableau d'origine et file de files d'éléments de ce tableau

Question 3 : Tri par fusion de files.
Écrivez une fonction *fusionTriee* prenant en paramètre une file de file d'éléments (Cf. question précédente) et retournant une file contenant l'ensemble des éléments triés dans l'ordre croissant.

Question 4 : Tri par fusion de tableau.
Écrivez une fonction générique *triFusion* prenant en paramètre un tableau d'éléments et modifiant ce tableau afin que ce dernier soit trié dans l'ordre croissant.

2. Modélisation d'un self Service.

Nous allons modéliser le fonctionnement d'un self service, c'est-à-dire le fonctionnement des files d'attente liées au service (plats, caisses). Dans notre modèle, chaque client sera représenté par un entier et les files d'attentes par une file d'entiers. Chaque client, lorsqu'il se fait servir occupe la seule et unique place de ce

service. Il ne peut quitter ce service que lorsqu'il est servi et qu'il y a de la place dans la file d'attente du service suivant.

- L'entrée des clients est gérée par une file d'attente de capacité non bornée. Avant de pouvoir se servir en entrées, les clients doivent faire la queue dans cette file.
- Une fois servi en entrée, un client fait la queue dans une file d'une capacité de 20 personnes pour pouvoir prendre son plat de résistance.
- Une fois servi, ce client fait de nouveau la queue dans une file d'une capacité de 20 personnes pour prendre son dessert.
- Enfin, il fait la queue aux caisses. Dans ce self service, il y a deux caisses pour lesquelles les files d'attente ont une capacité de 10 personnes. Une fois qu'un client est servi en dessert, il choisit la file la moins remplie pour passer à la caisse.
- Une fois le prix son repas réglé, il fait la queue dans une file de capacité non bornée pour chercher une place dans le self service.

Un service (entrée, plat de résistance, dessert, caisse) est modélisé par le type abstrait suivant :

```
/*
 * Interface décrivant un service.
 */
public interface Service
{
        /* Servir un client
         * @param client Le numero du client à servir
         * @pre la place doit être libre
         */
        public void servir(int client) ;
        /* Permet de savoir si le service du client courant est terminé
         * @remarks la durée du service est variable
         * @return vrai si la place est libre
         */
        public boolean serviceFini() ;
        /* Permet de savoir si la place est libre
         * @return vrai si la place est libre
         */
        public boolean placeLibre() ;
        /* Libère le client actuellement en train d'être servi
         * @pre la place ne doit pas être libre et le service doit être terminé
         */
        public int libererClient() ;
}
```

Le principe de fonctionnement du service est le suivant : si l'unique place disponible pour un service donné est libre (méthode *placeLibre*), le client peut se faire servir (méthode servir). Une fois servi (méthode *serviceFini*), le client est libéré (méthode *libererClient*).

Question 1 : Faites un schéma présentant le fonctionnement du self service. Ce schéma devra faire apparaître les services ainsi que les files d'attente les reliant.

Question 2 : Rédigez une procédure *avancerFile* prenant deux files (file d'entrée, file de sortie) et un service en paramètres. Le rôle de cette procédure est de faire avancer les clients en fonction de l'état des files et de l'état du service.

Question 3 : Rédigez le programme principal permettant de simuler le fonctionnement du restaurant. Ce programme devra déclarer les différentes files associées à la modélisation du restaurant et remplir la file d'entrée du restaurant avec 100 clients. Il devra aussi créer les services (entrée, plat de résistance, dessert, caisses). Ensuite, ce programme devra faire avancer les clients dans les files jusqu'à ce que toutes les files soient vides (à l'exception de la file modélisant la recherche d'une place dans le self service).