Projet TP Java 2014

Simulateur d’une clé USB

Ce projet, consiste à écrire un simulateur de clé USB. Partant d’une interface décrivant un appareil de stockage de données, il faut tout d’abord implémenter une classe représentant une clé USB et ensuite écrire une interface graphique (figure 1) permettant de voir précisément le contenu interne de la clé USB et ajouter et supprimer des fichiers.

On considère un modèle de mémoire simplifié dans le cadre de ce projet. La mémoire d’un appareil de stockage est divisée en blocs adjacents. Certains de ces blocs contiennent les données de fichiers tandis que d’autres blocs ne contiennent rien, ce sont des blocs vides.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Simulateur de clé USB | | | | | |  |  |
| f1  50 | f2  124 | 75 | 85 | f4  250 | f5  175 | 100 | f8  90 | 85 |
| f7  f8  Ajouter  Supprimer  90  Name Taille Index | | | | | | | | |

***A/ L’ajout d’un fichier :***

Lorsqu’il faut ajouter un fichier sur l’appareil de stockage, trois situations différentes peuvent se produire:

1. Il y a un bloc vide assez grand pour contenir le fichier. Ce bloc est divisé en deux parties, le fichier se place dans la première tandis que la seconde est un bloc vide plus petit. Voici par exemple comment se passe l’ajout d’un fichier f1 de taille 200 :

|  |  |  |  |
| --- | --- | --- | --- |
| vide (500) |  | f1 (200) | vide (300) |

1. Il n’y a pas de bloc vide assez grand mais il reste assez de place en tout sur la mémoire. Elle doit être défragmentée, c’est-à-dire que tous les blocs vides sont rassemblés. Supposons que l’on souhaite ajouter un fichier f3 de taille 200. Il n’y a pas de bloc vide assez grand, il faut donc d’abord défragmenter la mémoire :

|  |  |  |
| --- | --- | --- |
| f1 (100) vide (150) f2 (100) vide (150) |  | f1 (100) f2 (100) vide (300) |

Une fois la mémoire défragmentée, le fichier peut maintenant être ajouté sur la mémoire :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| f1 (100) | f2 (100) | vide (300) |  | f1 (100) | f2 (100) | f3 (200) | vide (100) |

1. Dernière possibilité : il n’y a pas assez de place sur la mémoire, même si on l’a défragmentait. Dans ce cas, la mémoire n’est pas défragmentée et le fichier n’est pas ajouté. Supposons que l’on souhaite ajouter un fichier f3 de taille 350:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f1 (100) | f2 (100) | vide (300) |  | f1 (100) | f2 (100) | vide (300) |

***B/ La suppression d’un fichier :***

On doit également pouvoir supprimer un fichier de la mémoire. Le bloc contenant le fichier devient tout simplement un bloc vide. Voici un exemple où l’on supprime le fichier f2 :

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| f1 (100) | vide (150) | f2 (100) | vide (150) |  | f1 (100) | vide (150) | vide (100) | vide (150) |

Vous remarquerez que la suppression du fichier f2 fait qu’il y a maintenant trois blocs vides consécutifs.

Ceux-ci ne vont pas être fusionnés en un bloc vide plus grand (ce serait bien entendu une optimisation possible mais qu’on ne va pas implémenter dans le cadre de ce projet).

***C/ Structure du programme :***

Les différentes classes et interfaces utilisées dans ce projet. La classe MemoireUSB est celle qui représente la clé USB et que vous devez implémenter. Elle utilise une classe interne BlocVide qui représente un bloc vide.

Les deux classes suivantes sont des programmes :

* La classe TestMemoireUSB teste la classe MemoireUSB.
* La classe SimulateurUSB créé une interface graphique qui permet de manipuler une clé USB.

**Implémentation**

1. La classe Fichier : Un fichier est un Bloc et est identifié par un nom et un contenu (tous les deux sont des String). La longueur d’un fichier est le nombre de caractères de son contenu. Voici le squelette de la classe :

public class Fichier implements Bloc

{

private String nom, data ;

public Fichier (String nom , String data ) { /\* À compléter \*/ }

}

Pensez également définir une méthode getNom qui permet d’obtenir le nom du fichier et à redéfinir la méthode toString.

1. La classe MemoireUSB : elle implémente l’interface MemoireStockage.

La mémoire est représentée comme une liste de Blocs. Aussi utilisez deux variables d’instance pour la capacité et l’espace utilisé sur la clé USB. N’oubliez pas non plus de définir la classe interne BlocVide représentant un bloc vide d’une certaine taille.

public class MemoireUSB implements MemoireStockage {

private List <Bloc > blocs ;

private final int capacité ;

private int espaceUtilisé ;

public MemoireUSB (int capacité ) { /\* À compléter \*/ }

// À compléter

private static class BlocVide implements Bloc {

private int taille ;

public BlocVide ( int taille ) { /\* À compléter \*/ }

} }

1. Une fois la classe MemoireUSB implémentée, il faut la tester. Écrivez une classe TestMemoireUSB avec une méthode main qui crée une clé USB et utilise toutes les méthodes de la classe MemoireUSB. Redéfinissez la méthode toString dans la classe MemoireUSB qui affiche la liste des blocs (vides et fichiers) qui sont sur la clé. Prévoir aussi les Exceptions (voir Annexe).
2. La dernière partie du projet consiste à écrire une petite interface graphique qui permet de consulter les blocs d’une clé USB et de lui ajouter ou supprimer des fichiers.

***D/ Remise du travail :*** Le travail devra être remis sur document papier et sur un support numérique (CD avec nom et prénom inscrit dessus).

* *Le document papier précise :* les noms du binôme, la version du compilateur utilisé, l'ensemble des classes et méthodes avec définition et rôle de chaque paramètre et tout choix supplémentaire, des prises d’écran avec des exemples d’exécution, la conclusion doit faire une synthèse et proposer une amélioration de ce travail.
* *Le code source :*doit être bien organisé, choisir des noms significatifs pour les variables, les commentaires sont obligatoires.
* *Important :* Le travail doit être fait en binôme ou trinôme (pas de monôme).
* La date de remise : au plus tard le 15/05/2014

**Annexes**

public interface Bloc {

public int getTaille ();

}

public interface MemoireStockage {

//retourne la capacité de la mémoire de stockage

public int getCapacité ();

// retourne la taille de l’espace utilisée

public int getEspaceUtilisé ();

// Vérifier si la mémoire est pleine

public boolean estPleine ();

// Ajout d’un fichier si la clé n’est pas pleine

public void ajout (Fichier f) throws FullException ;

// retourne un fichier (la recherche d’un fichier dans l’USB)

public Fichier getFichier ( String nom ) throws FileNotFoundException ;

// Suppression d’un fichier s’il existe

public Fichier suppression ( String name ) throws FileNotFoundException ;

}

//classe FullException

/\* Exception propagée en essayant d'ajouter un fichier à un périphérique de stockage complet \*/

@SuppressWarnings (" serial ")

public final class FullException extends RuntimeException

{

public FullException ( String msg ){

super (msg);

}

}

// classe FileNotFoundException

/\* Exception propagée en essayant d'accéder à un fichier non existant sur ​​une mémoire de stockage \*/

@SuppressWarnings (" serial ")

public final class FileNotFoundException extends Exception {}