Preface

The objective of this program is to simulate the functionality of a fat based file system

Block Class

- The Block class is used to simplify the use of 128 byte segments. When a block is needed simply construct a block with the data requested. Then use getBlock to get the byte array from the block. setBlock is made private as to avoid unregistered changes to the Block.

Methods

- `Block(byte[] NewContents)` constructor for the Block class simply calls setBlock.
- `getBlock()` returns the Block as an array of 128 bytes.
- `setBlock(byte[] New Contents)` confirms that the array of bytes is less than or equal to the 128 bytes. If it is too high the block will not be made and will return a -1. If it is too small it will write the bytes to an block of size 128 bytes and return a 1. If it is the perfect size then the block will be created and return a 0.

PartitionControlBlock Class

The PartianControlBlock class is used to represent the PCB. The PCB holds: - a pointer to the Root Directory (represented by an int that represents the block it is held at) - the size of the FAT - the total number of data blocks - a pointer to the first free block in the FAT(the list of free blocks is its own class) - a linked list representation of the free space blocks

Methods

- `PartianControlBlock(int FirstBlock, int TotalSize)` This is the constructor for the PCB. First I remove the first to blocks the first for the reserved space, and the second for the PCB. The total number of blocks is used to determine the number of blocks in the FAT. This is then used to determine the number of blocks taken up by the FAT which are then also removed from the free blocks. Another free block is removed and acts as the pointer to the root directory. Finally the remaining free block us used as the start of the free space list
- `PartianControlBlock(Block PCBBlock)` This is the constructor for the PCB using a Block. It converts the block into a string and then parses it for information of the PCBon the disk.
- `toBlock()` the converts the PCB into a Block so that it can be stored on disk.
- `checkFirstFreeBlock()` returns the next free block without removing it from the list

- `UpdateFirstFreeBlock(freespaceList FreeSpace)` checks what the next available free block is
- `getFirstFreeBlock()` returns the index of the first free block and removes it from the list
- `getRootDir()` returns a pointer to the root directory(as an index in the FAT)
- `getFirstFreeBlock()` returns the firstFreeBlock and replaces it with the next free block
- `UpdateFirstFreeBlock(freespaceList FreeSpace)` Sets the first free block to the next block in the list
- `addFreeBlock(int IndexOfBlock)` Adds a block with the specified index
- `getTotalDataBlock()` Returns the number of total data blocks.
- `getSizeOfFAT()` Returns the size of the fat.

Private Class

- `freespaceList` This represents all of the free blocks as a linked list of ints representing the index of the next free block

Methods

- `freespaceList(int FirstFreeBlock, int lastBlock)` This is the constructor for the list. When called it initializes a linked list of blocks between the first and last blocks.
- `addBlock(int FreeBlockLocation)` adds a block to the free space list. The block is located at the specified location.
- `getFreeBlock()` gives the index of a free block by removing it.
- `checkFreeBlock()` gives the index of a free block without removing it

FileAllocationTable Class

The FileAllocationTable class is used to represent the FAT. The FAT holds:

- An entire table consisting of the location on the next block in this file
- A Block table consisting of the block associated with each location on the EntrieTable
- Total number of entries in the table

Methods

- `FileAllocationTable(int TableSize)` This is the FAT constructor when called it initializes both the block and entry array based on the specified size.

- `FileAllocationTable(Block[] updatedFATBlocks)` Converts an array of blocks to a new FileAllocationTable.
- `changeBlockAndEntrie(int Index, int Entrie, Block NewBlock)` Simultaneously changes the block and entry at the specified index.
- `setBlockTotable(int Index, Block NewBlock)` Block at the specified index.
- `StringtoEntrietable(String StringRep)` Updates the entire table based on the String entered.
- `getBlockFromTable(int Index)` Returns the block at the specified index
- `ChangeEntrie(int Index, int Entrie)` Changes the Entry at the specified index.
- `getEntrieFromTable(int Index)` Returns the entry at the specified index.
- `getEntryTable()` Returns the full entry table(represented as an array of ints)
- `toBlocks()` Converts the FAT into a block to be written on disk
- `toString()` Converts the FAT into a String

Directory Class

The Directory holds the following: - A FileAttributes which describes the directory - A linked list of FileAttributess that each represent a file or directory these attributes can then be used to find the appropriate directory using the FAT(this feature is not yet fully implemented)

Methods

- `Directory(String FileName, int startBlock, int fileSize)` This is the constructor for the directory when called it creates a directory with the specified attributes.
- `addFile()` When called it adds a file to this directories linked list
- `addDirectory()` When called it adds a new directory to this directories linked list. This is done by making a addition to the linked list and specifying it as a directory in its file attributes.
- `getFile(String fileName)` when called it looks for an entry in this directory with the specified name. When found the file attributes are returned and can be used to construct the file using the FAT. If this file is a directory(specified in the file attributes) the constructed file will be able to act as a directory.
- `findIndex(String fileName)` When called it looks for an entry in this directory with the specified name. When found it returns the index in the linked list that it is located at.
- `getDirectoryInfo()` Returns the file attributes of the this directory;

FileAttributes Class

The FileAttributes class holds the following information(based on the Project requirements): - If it is a file or directory - The name of the file/directory as an array of 15 bytes - The first block in the FAT for this file(if its a file) - The file name as a string - The length of the name - The size of the file/directory

Methods

- `FileAttributes(boolean isfile, String FileName, int startBlock, int fileSize)` When called it constructs the class with the specified information. If the name exceeds the specified 15 bytes in size it will be denied.
- `getName()` returns the file name as a string.

Demo

TODO: <05-01-20 Gavin Jaeger-Freeborn>