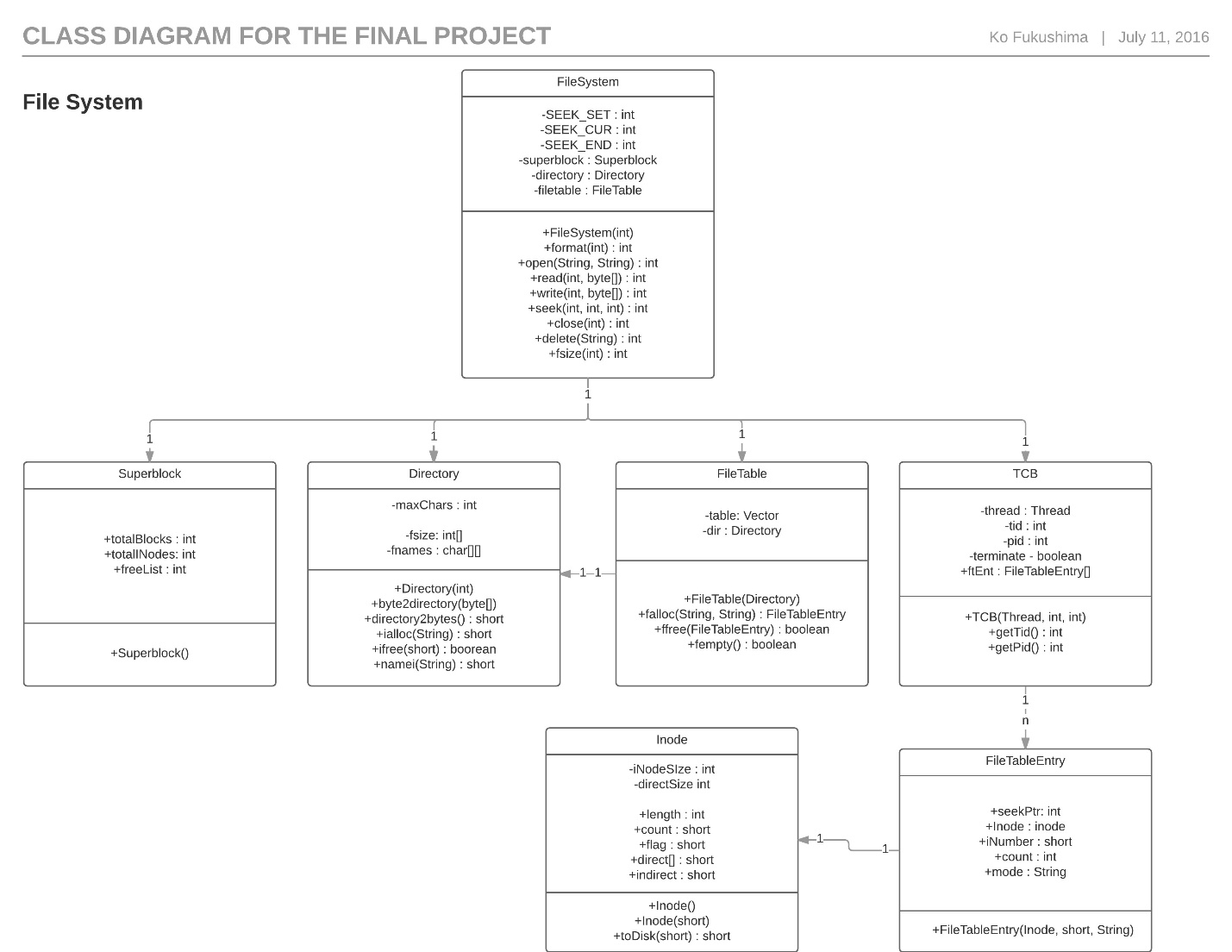
**Group 4**

Fukushima, Ko

Luo, Jesse

**Figure 1: Class Diagram for FileSystem**

This is a class diagram describing the work that needs to be completed for FileSystem. Each FileSystem has 1 Superblock, Directory, FileTable and TCB. The FileTable also has a Directory. The TCB has n amount of FileTableEntry, and each FileTableEntry has 1 Inode.

In the FileSystem, open returns a file descriptor. We use the file descriptor to access to the file table entry in the file descripter table in the TCB class. Each thread control block should have have the same amount of

**Class Descriptions**

**FileSystem**

**import** java.io.File;  
  
*/\*\*  
 \* Created by Ko Fukushima and Jesse Luo on 7/9/2016.  
 \*  
 \* This class manages the structure of a FileSystem by holding  
 \* the superblock, directory, and filetable.  
 \*  
 \*/***public class** FileSystem  
{  
 **private final int SEEK\_SET** = 0;  
 **private final int SEEK\_CUR** = 1;  
 **private final int SEEK\_END** = 2;  
  
 **private** Superblock **superblock**;  
 **private** Directory **directory**;  
 **private** FileTable **filetable**;  
  
 */\*\*  
 \* Constructs a new FileSystem.  
 \*  
 \** ***@param diskBlocks*** *size of the superblock  
 \*/* **public** FileSystem(**int** diskBlocks)  
 {  
 *// create new superblock, format disk with 64 inodes* **superblock** = **new** Superblock (diskBlocks);  
  
 *// create new directory, register "/" in directory entry 0* **filetable** = **new** FileTable(**directory**);  
  
 *// reconstruct the directory  
 /\*  
 FileTableEntry dirEnt = open("/", "r");  
 int dirSize = fsize(dirEnt);  
 if(dirSize > 0)  
 {  
 byte[] dirData = new byte[dirSize];  
 read(dirEnt, dirData);  
 directory.bytes2directory(dirData);  
 }  
 close(dirEnt);\*/* }  
  
 *// the description of sync will be added more info later  
  
 /\*\*  
 \* This method formats the disk  
 \*  
 \** ***@param files*** *the # files to be created  
 \** ***@return*** *0 on success, -1 otherwise  
 \*/* **public int** format(**int** files)  
 {  
 **return** 0; *// it needs to be modified later* }  
  
 *// the description of open will be added more info later  
  
 /\*\*  
 \* This method opens the file corresponding to the file  
 \* name in the given mode.  
 \*  
 \** ***@param fileName*** *\** ***@param mode*** *\** ***@return*** *int fd  
 \* between 3 to 31  
 \*/* **public int** open(String fileName, String mode)  
 {  
 **return** 0; *// it needs to be modified later* }  
  
 */\*\*  
 \* This method reads as many bytes as possible  
 \* or up to buffer.length the file  
 \* corresponding to the file descriptor  
 \*  
 \** ***@param fd*** *the file descriptor  
 \** ***@param buffer*** *the buffer  
 \** ***@return*** *the # bytes read or -1 if there is an error  
 \*/* **public int** read(**int** fd, **byte**[] buffer)  
 {  
 **return** 0; *// it needs to be modified later* }  
  
 */\*\*  
 \* This method writes the contents of the buffer to the  
 \* file corresponding to the file descriptor.  
 \*  
 \** ***@param fd*** *the file descriptor  
 \** ***@param buffer*** *the buffer  
 \** ***@return*** *\*/* **public int** write(**int** fd, **byte**[] buffer)  
 {  
 **return** 0; *// it needs to be modified later* }  
  
 */\*\*  
 \* This method updates the seek pointer corresponding to  
 \* the file descriptor.  
 \*  
 \** ***@param fd*** *the file descriptor  
 \** ***@param offset*** *the offset can be positive or negative  
 \** ***@param whence*** *the whence represents SEEK\_SET == 0,  
 \* SEEK\_CUR == 1, and SEEK\_END == 2  
 \** ***@return*** *\*/* **public int** seek(**int** fd, **int** offset, **int** whence)  
 {  
 **return** 0; *// it needs to be modified later* }  
  
 */\*\*  
 \* This method close the file corresponding to  
 \* the file descriptor  
 \*  
 \** ***@param fd*** *file descriptor  
 \** ***@return*** *0 in success, -1 false  
 \*/* **public int** close(**int** fd)  
 {  
 **return** 0; *// it needs to be modified later* }  
  
 */\*\*  
 \* This method deletes the file that  
 \* is specified by file name only when the  
 \* file is closed  
 \*  
 \** ***@param fileName*** *the file name  
 \** ***@return*** *0 if successful, -1 otherwise  
 \*/* **public int** delete(String fileName)  
 {  
 **return** 0; *// it needs to be modified later* }  
  
 */\*\*  
 \* This method returns the size in bytes  
 \* of the file indicated by file descriptor  
 \* and returns -1 when it detects an error  
 \*  
 \** ***@param fd*** *the file descriptor  
 \** ***@return*** *the file size  
 \*/* **public int** fsize(**int** fd)  
 {  
 **return** 0; *// it needs to be modified later* }  
}

**Superblock**

*/\*\*  
 \* Created by Ko Fukushima and Jesse Luo on 7/9/2016.  
 \*  
 \* This class is used to describe the number of disk blocks  
 \* , the number of inodes, and the block number of the  
 \* head block of the free list.  
 \*/***public class** Superblock  
{  
 **public int totalBlocks**; *// the number of disk blocks* **public int totalInodes**; *// the number of inodes* **public int freeList**; *// the block number of the free List's head  
  
 /\*\*  
 \* Class constructor that initializes the fields that are tatalBlocks,  
 \* totalInodes, and freeList  
 \*  
 \** ***@param diskSize*** *the disk size  
 \*/* **public** Superblock( **int** diskSize)  
 {  
 }  
}

**FileTable**

*/\*\*  
 \* Created by Ko Fukushima and Jesse Luo on 7/9/2016.  
 \*  
 \* This class manages file entry table by allocating for,  
 \* freeing, emptying the entry in the table  
 \*  
 \*/***import** java.util.Vector;  
  
**public class** FileTable  
{  
 **private** Vector **table**;  
 **private** Directory **dir**;  
  
 */\*\*  
 \* Class constructor that initializes the fields of table  
 \* and dir.  
 \*  
 \** ***@param directory*** *the directory  
 \*/* **public** FileTable(Directory directory)  
 {  
 **table** = **new** Vector();  
 **dir** = directory;  
 }  
  
 */\*\*  
 \* This method allocates a new file table entry for this file name  
 \* and it also allocate/retrive and register the corresoponding inode  
 \* using dir increment this inode's count immediately write back this  
 \* inode this inode to the disk  
 \*  
 \** ***@param filename*** *the file name  
 \** ***@param mode*** *the mode such as "r", "w", "w+", or "a"  
 \** ***@return*** *a reference to the file table entry  
 \*/* **public synchronized** FileTableEntry falloc(String filename, String mode)  
 {  
 **return null**; *// It needs to be modified later* }  
  
 */\*\*  
 \* This method receive a file table entry reference  
 \* and save the corresponding inode to the disk  
 \* and free this file table entry  
 \*  
 \** ***@param e*** *the file table entry  
 \** ***@return*** *True if this file entry found in the table,  
 \* false otherwise  
 \*/* **public synchronized boolean** ffree(FileTableEntry e)  
 {  
 **return false**; *// It needs to be modified later* }  
  
 */\*\*  
 \* This method clear all file table entry in the table  
 \* and it should be called before starting a format  
 \*  
 \** ***@return*** *True if table is empty and false otherwise  
 \*/* **public synchronized boolean** fempty( )  
 {  
 **return table**.isEmpty();  
 }  
}

**Directory**

*/\*\*  
 \* Created by Ko Fukushima and Jesse Luo on 7/9/2016.  
 \*  
 \* This class maintains each file in a different directory entry that  
 \* contains its file name and the corresponding inode number.  
 \*/***public class** Directory  
{  
 **private static int** *maxChars* = 30; *// max characters of each file name  
   
 // Directory entries* **private int fsize**[];  
 **private char fnames**[][];  
  
 */\*\*  
 \* Class constructor that initializes the fields that are fsize,  
 \* fnames, root, fsize.  
 \*  
 \** ***@param maxInumber*** *the max iNode number  
 \*/* **public** Directory(**int** maxInumber)  
 {  
 **fsize** = **new int**[maxInumber];  
 **fnames** = **new char**[maxInumber][*maxChars*];  
 String root = **"/"**;  
 **fsize**[0] = root.length();  
 root.getChars(0, **fsize**[0], **fnames**[0], 0);  
 }  
  
 */\*\*  
 \* This method initializes the Directory instance with this data[]  
 \*  
 \** ***@param data*** *the data[] received directory information from disk  
 \** ***@return*** *// It needs to be added later  
 \*/* **public int** bytes2directory(**byte** data[])  
 {  
   
 **return** 0; *// It needs to be modified later* }  
  
 */\*\*  
 \* This method converts and return Directory information into a plain  
 \* byte array that will be wrritten back to disk.  
 \*  
 \** ***@return*** *the meaningfull Directory information  
 \*/* **public byte**[] directory2bytes()  
 {  
 **return null**; *// It needs to be modified later* }  
  
 */\*\*  
 \* This methods creates the one of a file, and  
 \* allocates a new inode number for it.  
 \*  
 \** ***@param filename*** *the file name  
 \** ***@return*** *a new inode number // This might need to be modified later  
 \*/* **public short** ialloc(String filename)  
 {  
 **return** 0; *// It needs to be modified later* }  
  
 */\*\*  
 \* This method deallocate this inumber (inode number) and  
 \* also deallocate the corresponding file  
 \*  
 \** ***@param iNumber*** *the inode number  
 \** ***@return*** *True if find the inode number and deallocates  
 \* the file with that inumber, and False otherwise  
 \*/* **public boolean** ifree(**short** iNumber)  
 {  
   
 **return false**; *// It needs to be modified later* }  
  
 */\*\*  
 \* This method returns the inumber corresoponding to this filename  
 \*  
 \** ***@param filename*** *the file name  
 \** ***@return*** *the inode number corresponding to this file name  
 \*/* **public short** namei(String filename)  
 {  
 **return** 0; *// It needs to be modified later* }  
}

**TCB**

*/\*\*  
 \* Created by Ko Fukushima and Jesse Luo on 7/9/2016.  
 \*  
 \* This class represents a Thread control block that  
 \* manages up to 32 open files  
 \*  
 \*/***public class** TCB  
{  
 **private** Thread **thread** = **null**;  
 **private int tid** = 0;  
 **private int pid** = 0;  
 **private boolean terminate** = **false**;  
   
 *// User file descriptor table:  
 // each entry pointing to a file (structure) table entry* **public** FileTableEntry[] **ftEnt** = **null**;  
  
 */\*\*  
 \*  
 \* Class constructor that initializes the parameters: thread, tid, pid  
 \* , terminated, and FileTableEntry  
 \*  
 \** ***@param thread*** *a thread  
 \** ***@param tid*** *a thread id  
 \** ***@param pid*** *a process id  
 \*/* **public** TCB(Thread thread, **int** tid, **int** pid)  
 {  
 **this**.**thread** = thread;  
 **this**.**tid** = tid;  
 **this**.**pid** = pid;  
 **terminate** = **false**;  
   
 *// The following code is added for the file system* **ftEnt** = **new** FileTableEntry[32];  
 }  
  
 */\*\*  
 \* This method returns a thread id  
 \*  
 \** ***@return*** *tid the id for a thread  
 \*/* **public int** getTid()  
 {  
 **return tid**;  
 }  
  
 */\*\*  
 \* This method returns a process id  
 \*  
 \** ***@return*** *pid the id for a process  
 \*/* **public int** getPid()  
 {  
 **return pid**;  
 }  
  
}

**FileTableEntry**

*/\*\*  
 \* Created by Ko Fukushima and Jesse Luo on 7/9/2016.  
 \*  
 \* This class is shared among all user threads  
 \*/***public class** FileTableEntry  
{  
 **public int seekPtr**;  
 **public final** Inode **inode**;  
 **public final short iNumber**;  
 **public int count**;  
 **public final** String **mode**;  
  
 */\*\*  
 \* Class constructor that initializes the fields that are seekPtr,  
 \* inode, iNumbers, count, mode.  
 \*  
 \** ***@param inode*** *the inode  
 \** ***@param iNumber*** *the inode number  
 \** ***@param mode*** *the mode such as "r", "w", "w+", or "a"  
 \*/* **public** FileTableEntry(Inode inode, **short** iNumber, String mode)  
 {  
 **seekPtr** = 0; *// a file seek pointer* **this**.**inode** = inode; *// a reference to its inode* **this**.**iNumber** = iNumber; *// am inode number* **count** = 1; *// # threads sharing this entry* **this**.**mode** = mode; *// "r", "w", "w+", or "a"* **if** (**this**.**mode**.compareTo(**"a"**) == 0)  
 {  
 **seekPtr** = **this**.**inode**.**length**;  
 }  
 }  
   
}

**Inode**

*/\*\*  
 \* Created by Ko Fukushima and Jesse Luo on 7/9/2016.  
 \*  
 \* This class describes a file, and this inode is a  
 \* Simplified version of the UnixInode  
 \*/***public class** Inode  
{  
 **private final static int *iNodeSize*** = 32; *// fix to 32 bytes* **private final static int *directSize*** = 11; *// # direct pointers* **public int length**; *// file size in bytes* **public short count**; *// # file-table entries pointing on this* **public short flag**; *// 0 = unused, 1 = used, ...* **public short direct**[] = **new short**[***directSize***]; *// direct pointers* **public short indirect**; *// a indirect pointer  
  
 /\*\*  
 \* Class constructor that initializes the fields that are length,  
 \* count, flag, direct, and indirect.  
 \*/* Inode()  
 {  
 **length** = 0;  
 **count** = 0;  
 **flag** = 1;  
 **for** (**int** i = 0; i < ***directSize***; i++)  
 {  
 **direct**[i] = -1;  
 }  
 **indirect** = -1;  
 }  
  
 */\*\*  
 \* This method retrieves the inode from disk  
 \*  
 \** ***@param iNumber*** *\*/* Inode(**short** iNumber)  
 {  
  
 }  
  
 */\*\*  
 \* This method saves to disk as the i-th inode  
 \*  
 \** ***@param iNumber*** *\** ***@return*** *\*/* **int** toDisk(**short** iNumber)  
 {  
 **return** 0; *// It needs to be modified later* }  
}

**Work Items To Be Completed**

Ko:

Jesse: