Project information

```
disk                        (deals with Each disk, but again we only have 1 disk)
  |
diskmanager                 (for each disk in the system, but we only have 1 disk)
  |
  partionmanager                     (for each partition on a disk)
  |          |           |          |
FS1     FS2         FS3      FS4
|  |  |  |
a  b c  d
```

As a note, a block on the disk is hard coded to be 64 bytes.

Disk class

- Simulated disk using UNIX file system, normally there would be more than 1 disk possible, but we are only going to implement 1 disk.
- Creates file, divides it into blocks, read disk block (block#, buffer)

Diskmanager

- read/write block functions
- Also deals in partition tables (stored in Block #0 of each disk)
  - When the disk is initially created, the superblock is created (Disk block #0) with the partition information. If the disk already exists, then the information is read from the superblock.
- There will never be more than 4 partitions.
- Security as well, so that a file can't be written into the wrong partition.

Partition manager

- manages the allocation & deallocation of blocks for the partition
- partition goes through the disk manager
- uses bitmap (Bit vector) to keep track of blocks
- store bitmap in one block (**block 0 of each partition**) update the stored copy after each change in the bitmap Use 0 free/ 1 not avail
  - o In order to do this, it will need to use the Diskmanager to read and write the bitmap.
- When a partition is first initialized, you will need to think about the root directory, which is always stored in block 1 of each partition. This can be setup in the partition manager or in the filesystem manager. Don't forgot to set block 1 of the partition as not free.

Filesystem

- The filesystem is where you will write most of the code and spend most of your time debugging.
- The root directory will allows be stored in block 1 of each partition.

- A file can be opened multiple times and it will need to have an entry in the open file table (with a read write pointer and mode as well) for each time it is opened. The rw pointer and mode for each open file will likely have a different value.

-- File inode implementation

- name 1 byte
- type 1 byte: file/directory
- filesize 4 bytes:
- 3 direct address at 4bytes each = 12 bytes
- 1 indirect address (pointers to indirect inode) 4 bytes
- The rest of the space for your attributes.

-- Indirect inode

- 16 direct address pointers at 4 bytes each = total of 64 bytes.

-- Directory inode

- Filename 1byte
- Type file/directory 1byte
- Pointer to sub directory block or file inode 4bytes
- Repeated 9 more times
- Pointer to next directory inode to continue the information on this directory 4 bytes

Lastly, remember you can use a hex editor to read the DISK1 that is created. You should be able to see and read the information directly off the disk to verify everything is working correctly. The bit vector will be only block you can not read easily with a hex editor.