Computer Organization
Name: *Divyansh Rastogi*
Roll. No.: *2019464*

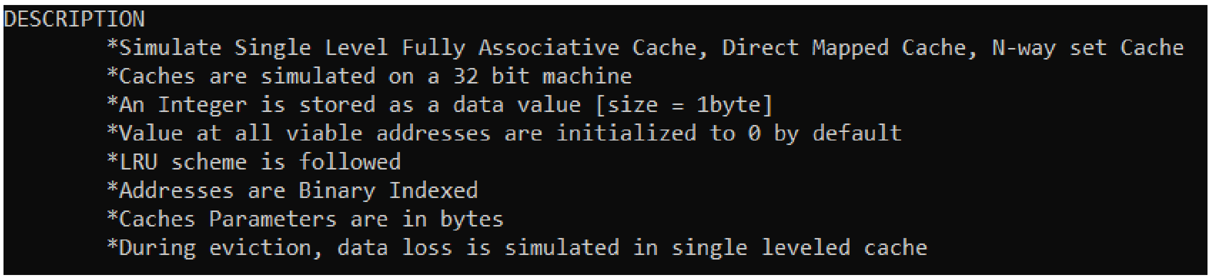
CACHE TERMINAL

Documentation

Project Description && Implementation Presuppositions

Single level [1] Fully Associative Cache, Direct Mapped Cache, N-way Set Cache are implemented in a highly user-interactive terminal interface. The programming language of the aforementioned implementation is Java (based on OOPS).
The implemented caches simulate to a very high extent the functionality with extensive showcase of oops modularity to model a real-world computer's inbuilt cache. The programming done is very user-friendly, error-free and the application is very convenient to use and easy to get familiar with the inbuilt functionalities.

Implementation presuppositions: -



```
DESCRIPTION
        *Simulate Single Level Fully Associative Cache, Direct Mapped Cache, N-way set Cache
        *Caches are simulated on a 32 bit machine
        *An Integer is stored as a data value [size = 1byte]
        *Value at all viable addresses are initialized to 0 by default
        *LRU scheme is followed
        *Addresses are Binary Indexed
        *Caches Parameters are in bytes
        *During eviction, data loss is simulated in single leveled cache
```

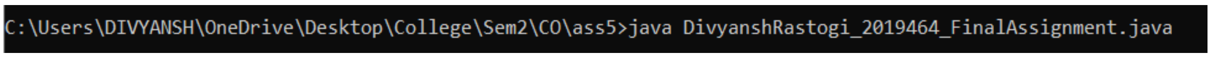
Dependencies

Java (TM) SE Runtime Environment (build 13.0.1+9)

Usage/ Code Functionality

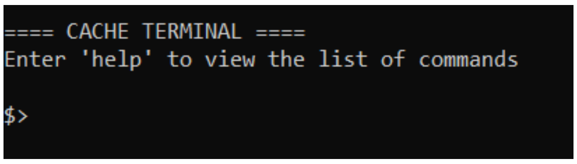
Initialization

Direct the program "DivyanshRastogi_2019464_FinalAssignment.java" into a suitable directory, and load your command prompt into the same directory and execute the program.



```
C:\Users\DIVYANSH\OneDrive\Desktop\College\Sem2\CO\ass5>java DivyanshRastogi_2019464_FinalAssignment.java
```

The following will be displayed on the terminal:



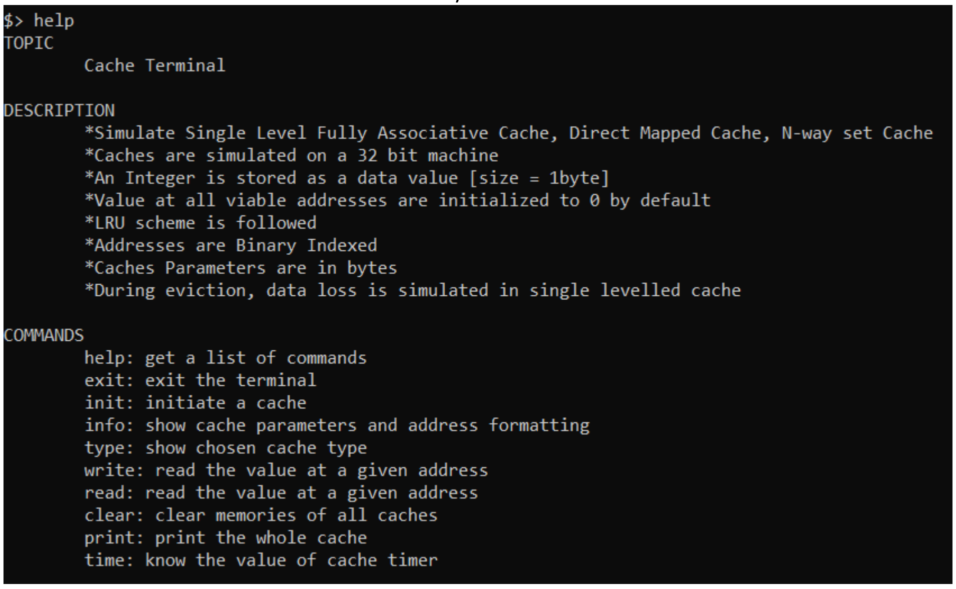
```
==== CACHE TERMINAL ====
Enter 'help' to view the list of commands

$>
```

[1] *Hyperlinks are created to reference the workings of the given caches @* https://www.sciencedirect.com/topics/computer-science/

Computer Organization
Name: *Divyansh Rastogi*
Roll. No.: *2019464*

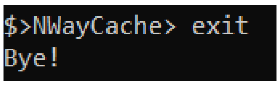
Commands

- *help*
  - Get an overview of the functionality of the caches and various commands.



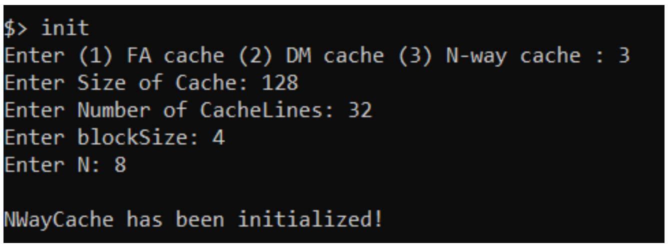
```
$> help
TOPIC
        Cache Terminal

DESCRIPTION
        *Simulate Single Level Fully Associative Cache, Direct Mapped Cache, N-way set Cache
        *Caches are simulated on a 32 bit machine
        *An Integer is stored as a data value [size = 1byte]
        *Value at all viable addresses are initialized to 0 by default
        *LRU scheme is followed
        *Addresses are Binary Indexed
        *Caches Parameters are in bytes
        *During eviction, data loss is simulated in single levelled cache

COMMANDS
        help: get a list of commands
        exit: exit the terminal
        init: initiate a cache
        info: show cache parameters and address formatting
        type: show chosen cache type
        write: read the value at a given address
        read: read the value at a given address
        clear: clear memories of all caches
        print: print the whole cache
        time: know the value of cache timer
```

- *exit*
  - Exit the cache terminal.



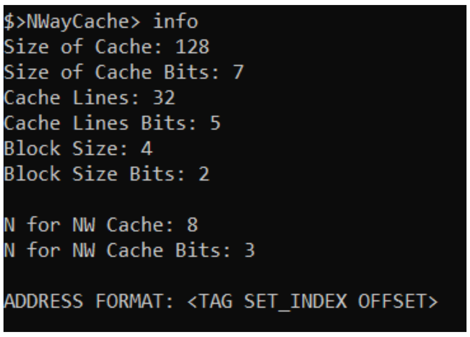
```
$>NWayCache> exit
Bye!
```

- *init*
  - Initialize a cache, when a cache is initialized, the program simulates a directory behaviour.



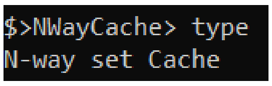
```
$> init
Enter (1) FA cache (2) DM cache (3) N-way cache : 3
Enter Size of Cache: 128
Enter Number of CacheLines: 32
Enter blockSize: 4
Enter N: 8

NWayCache has been initialized!
```

- *info*
  - Show initialized cache parameters, the bits required for each parameter and the address format breakdown a cache simulates for accessing a data value.



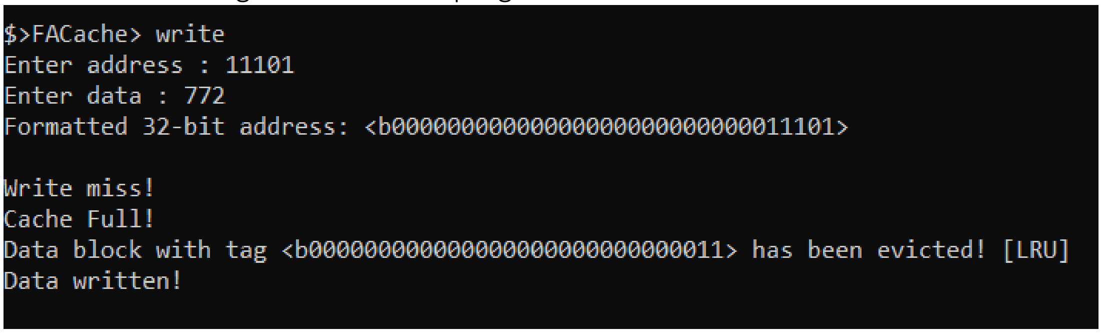
```
$>NWayCache> info
Size of Cache: 128
Size of Cache Bits: 7
Cache Lines: 32
Cache Lines Bits: 5
Block Size: 4
Block Size Bits: 2

N for NW Cache: 8
N for NW Cache Bits: 3

ADDRESS FORMAT: <TAG SET_INDEX OFFSET>
```

- *type*
    - Show initialized cache type.

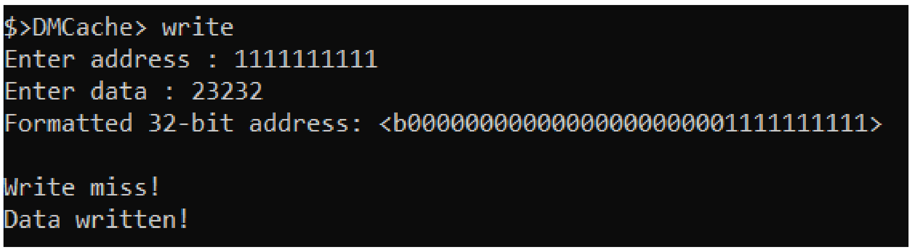


```
$>NWayCache> type
N-way set Cache
```

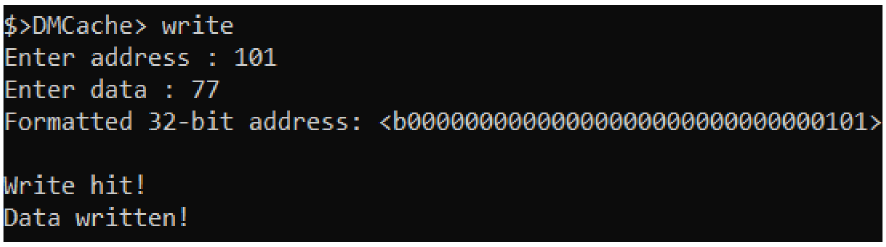
- *write*
    - Write a data value at a specific address.
    - Input data must be an integer value. [1 Byte]
    - The input address is binary and shall not be inputted compulsorily as a 32bit address, as the program auto completes the address.
    - The program outputs a write hit or a write miss.
    - For FA cache and NW cache, LRU schemes are followed (when caches are full) in which the data block with the oldest timer count is evicted. As the caches are single level, evicted data block is reinitialized to 0.
    - For DM cache, in case of hit, the data is overwritten at the input address but in case of a miss, the block at the index calculated from the address is evicted and reinitialized to 0.
    - The written block gets allotted the program's timer count.



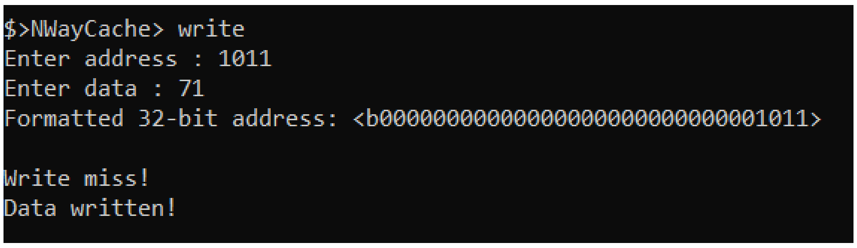
```
$>FACache> write
Enter address : 11101
Enter data : 772
Formatted 32-bit address: <b00000000000000000000000000011101>

Write miss!
Cache Full!
Data block with tag <b000000000000000000000000000011> has been evicted! [LRU]
Data written!
```



```
$>DMCache> write
Enter address : 1111111111
Enter data : 23232
Formatted 32-bit address: <b00000000000000000000001111111111>

Write miss!
Data written!
```

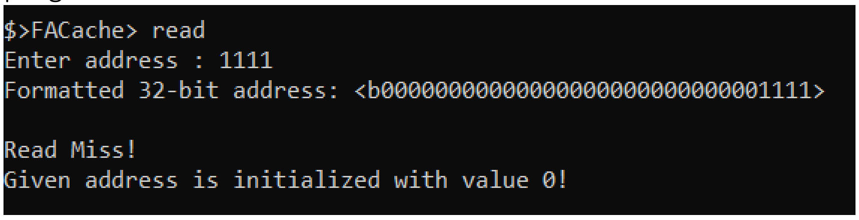


```
$>DMCache> write
Enter address : 101
Enter data : 77
Formatted 32-bit address: <b00000000000000000000000000000101>

Write hit!
Data written!
```

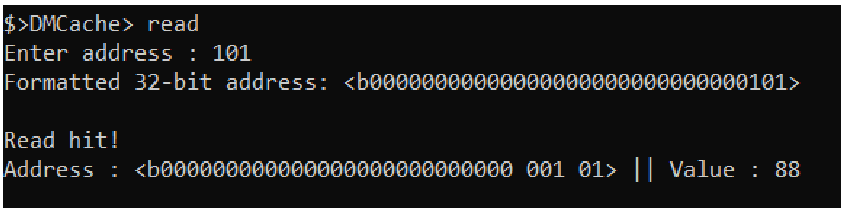


```
$>NWayCache> write
Enter address : 1011
Enter data : 71
Formatted 32-bit address: <b00000000000000000000000000001011>

Write miss!
Data written!
```

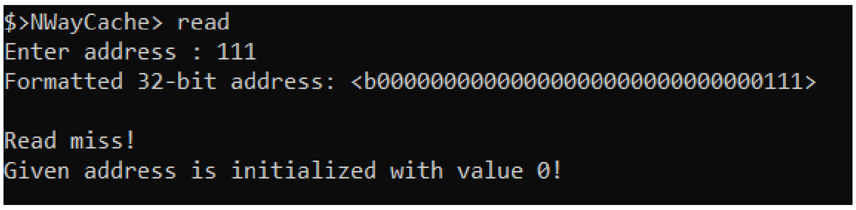
- *read*
  - Read the value at a specific address.
  - The input address is binary and shall not be inputted compulsorily as a 32bit address, as the program auto completes the address.
  - The program outputs a read hit or read miss.
  - In case of a read hit, the program outputs the respective cache's formatted address with the data value and updates the timer count of the read block with the program's timer count.
  - In case of read miss, the block is loaded(written) into the cache initialized with default value i.e. 0. The loaded block's timer count is updated with the program's timer count.



```
$>FACache> read
Enter address : 1111
Formatted 32-bit address: <b00000000000000000000000000001111>

Read Miss!
Given address is initialized with value 0!
```

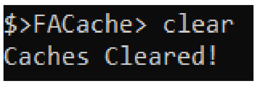


```
$>DMCache> read
Enter address : 101
Formatted 32-bit address: <b00000000000000000000000000000101>

Read hit!
Address : <b000000000000000000000000000 001 01> || Value : 88
```

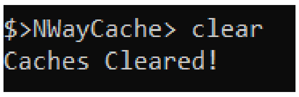


```
$>NWayCache> read
Enter address : 111
Formatted 32-bit address: <b00000000000000000000000000000111>

Read miss!
Given address is initialized with value 0!
```

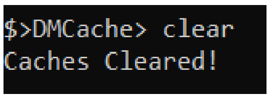
- *clear*
  - Refresh your caches, reinitialize them by setting respective parameters to default.
  - Return to the root directory.



```
$>FACache> clear
Caches Cleared!
```

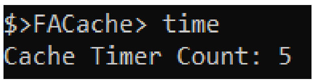


```
$>NWayCache> clear
Caches Cleared!
```



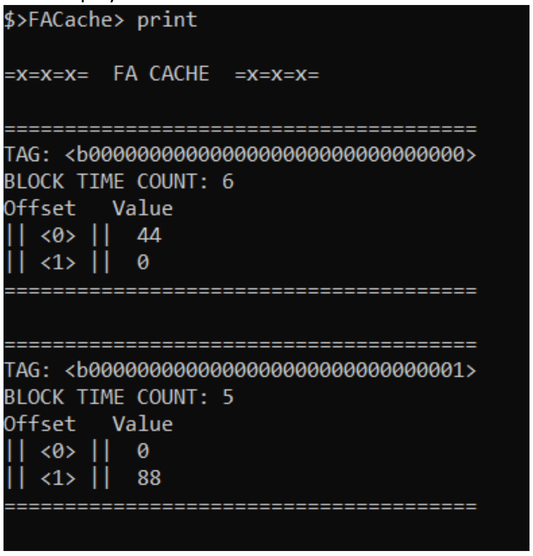
```
$>DMCache> clear
Caches Cleared!
```

- *time*
  - Know the value of cache timer which is used to time the block.
  - The cache timer is incremental and resets to 0 when cleared.
  - Forms the basis of our LRU policy.

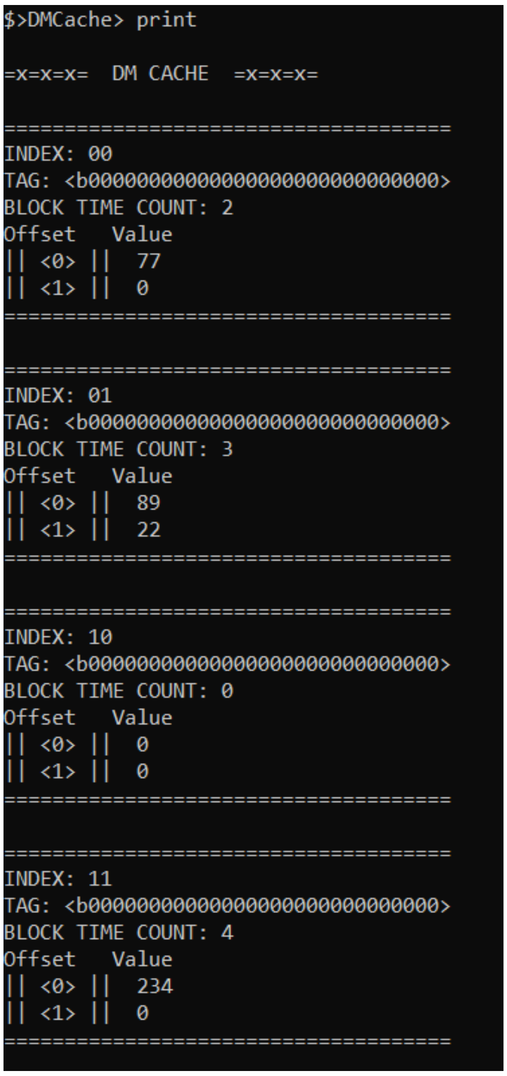


```
$>FACache> time
Cache Timer Count: 5
```

- *print*
  - Print the initialized cache.
  - Printing is done according to address formatting to aid to user convenience.
  - DM cache is initialized by default value and address while FA and NW are set to empty.

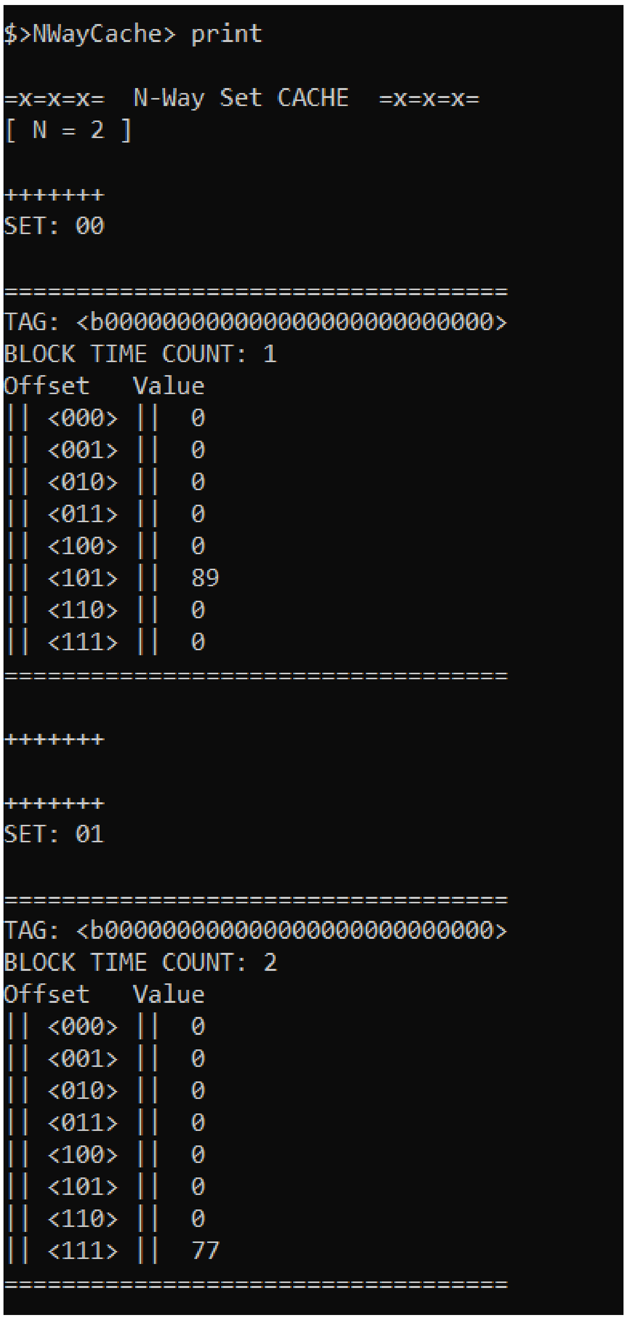


```
$>FACache> print

=x=x=x=  FA CACHE  =x=x=x=

=======================================
TAG: <b0000000000000000000000000000000>
BLOCK TIME COUNT: 6
Offset   Value
|| <0> ||  44
|| <1> ||  0
=======================================

=======================================
TAG: <b0000000000000000000000000000001>
BLOCK TIME COUNT: 5
Offset   Value
|| <0> ||  0
|| <1> ||  88
=======================================
```

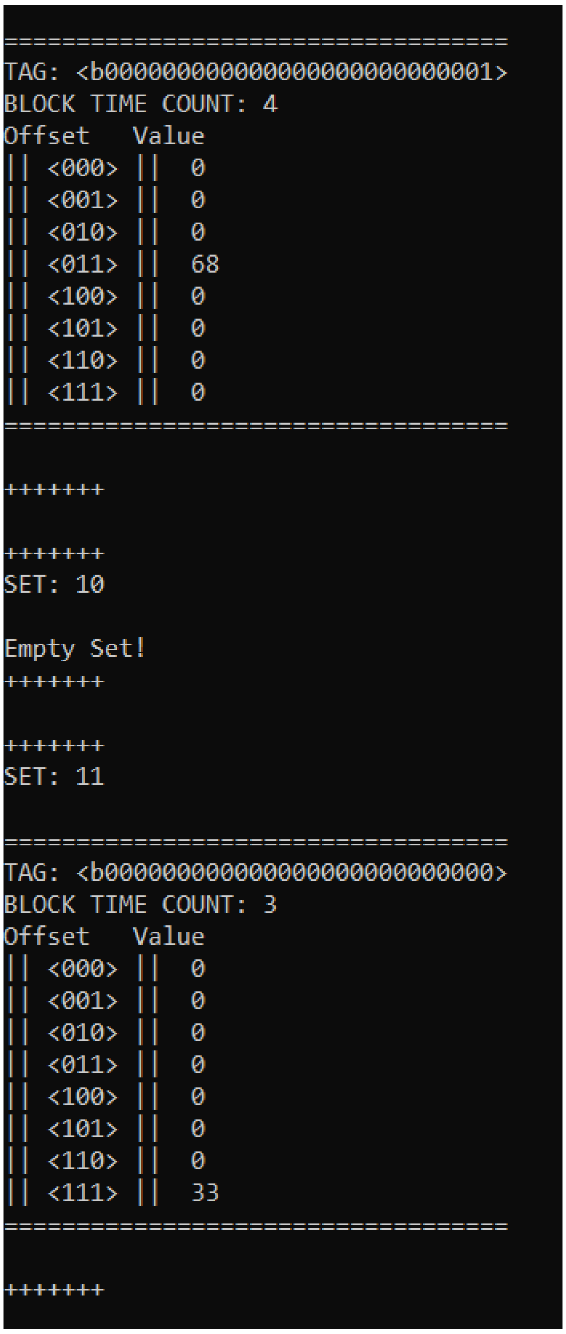


```
$>DMCache> print

=x=x=x=  DM CACHE  =x=x=x=

=====================================
INDEX: 00
TAG: <b00000000000000000000000000000>
BLOCK TIME COUNT: 2
Offset   Value
|| <0> ||  77
|| <1> ||  0
=====================================

=====================================
INDEX: 01
TAG: <b00000000000000000000000000000>
BLOCK TIME COUNT: 3
Offset   Value
|| <0> ||  89
|| <1> ||  22
=====================================

=====================================
INDEX: 10
TAG: <b00000000000000000000000000000>
BLOCK TIME COUNT: 0
Offset   Value
|| <0> ||  0
|| <1> ||  0
=====================================

=====================================
INDEX: 11
TAG: <b00000000000000000000000000000>
BLOCK TIME COUNT: 4
Offset   Value
|| <0> ||  234
|| <1> ||  0
=====================================
```

Computer Organization
Name: *Divyansh Rastogi*
Roll. No.: *2019464*



```
$>NWayCache> print

=x=x=x=  N-Way Set CACHE  =x=x=x=
[ N = 2 ]

+++++++
SET: 00

===================================
TAG: <b000000000000000000000000000>
BLOCK TIME COUNT: 1
Offset   Value
|| <000> ||  0
|| <001> ||  0
|| <010> ||  0
|| <011> ||  0
|| <100> ||  0
|| <101> ||  89
|| <110> ||  0
|| <111> ||  0
===================================

+++++++

+++++++
SET: 01

===================================
TAG: <b000000000000000000000000000>
BLOCK TIME COUNT: 2
Offset   Value
|| <000> ||  0
|| <001> ||  0
|| <010> ||  0
|| <011> ||  0
|| <100> ||  0
|| <101> ||  0
|| <110> ||  0
|| <111> ||  77
===================================
```

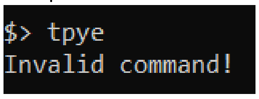


```
===================================
TAG: <b000000000000000000000000001>
BLOCK TIME COUNT: 4
Offset   Value
|| <000> ||  0
|| <001> ||  0
|| <010> ||  0
|| <011> ||  68
|| <100> ||  0
|| <101> ||  0
|| <110> ||  0
|| <111> ||  0
===================================

+++++++

+++++++
SET: 10

Empty Set!
+++++++

+++++++
SET: 11

===================================
TAG: <b000000000000000000000000000>
BLOCK TIME COUNT: 3
Offset   Value
|| <000> ||  0
|| <001> ||  0
|| <010> ||  0
|| <011> ||  0
|| <100> ||  0
|| <101> ||  0
|| <110> ||  0
|| <111> ||  33
===================================

+++++++
```

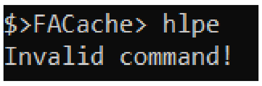
Error Handling/Reporting

Various error handling measures have been introduced in the program to handle unexpected inputs and user responses. Error reporting/handling in the program is showcased below: -

- For a command, that isn't within the set of commands, a user gets the following output: -

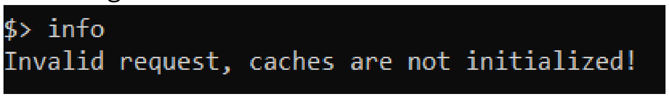


```
$> tpye
Invalid command!
```

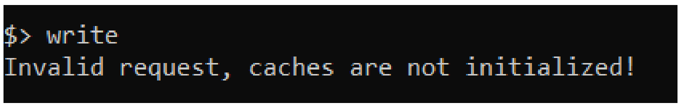


```
$>FACache> hlpe
Invalid command!
```

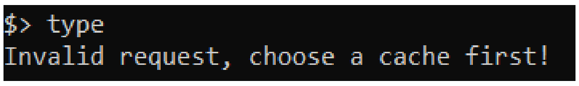
- For inputs such as 'info', 'time', 'type', 'write', 'read', 'print' in which cache initialization is required, if the caches are not initialized, the program outputs the following: -



```
$> info
Invalid request, caches are not initialized!
```

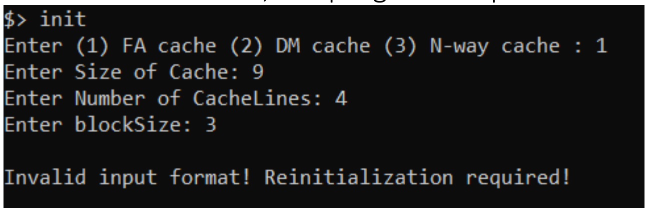


```
$> write
Invalid request, caches are not initialized!
```



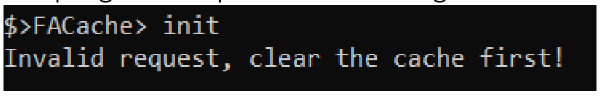
```
$> type
Invalid request, choose a cache first!
```

- When a cache is initialized, the user is requested for cache parameters which must be powers of 2 and satisfying the condition of S = CL x B with a valid cache type. If the above isn't followed, the program outputs the following: -



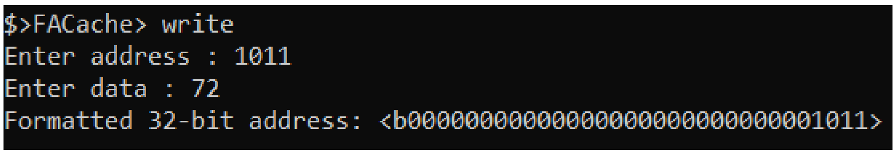
```
$> init
Enter (1) FA cache (2) DM cache (3) N-way cache : 1
Enter Size of Cache: 9
Enter Number of CacheLines: 4
Enter blockSize: 3

Invalid input format! Reinitialization required!
```

- Once initialized a cache, if you request to initialize the cache again before clearing, the program outputs the following: -



```
$>FACache> init
Invalid request, clear the cache first!
```

- The input format of the addresses while reading and writing aren't expected to always be 32 bits. To aid to user convenience, the program autocompletes the address and shows the formatted 32-bit address: -

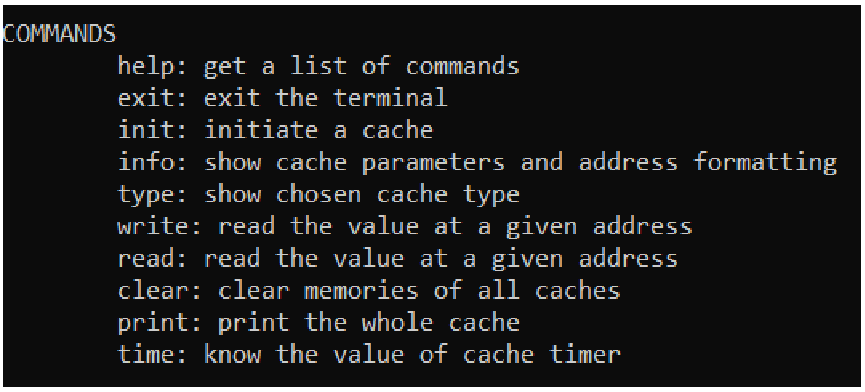


```
$>FACache> write
Enter address : 1011
Enter data : 72
Formatted 32-bit address: <b00000000000000000000000000001011>
```

Computer Organization
Name: *Divyansh Rastogi*
Roll. No.: *2019464*

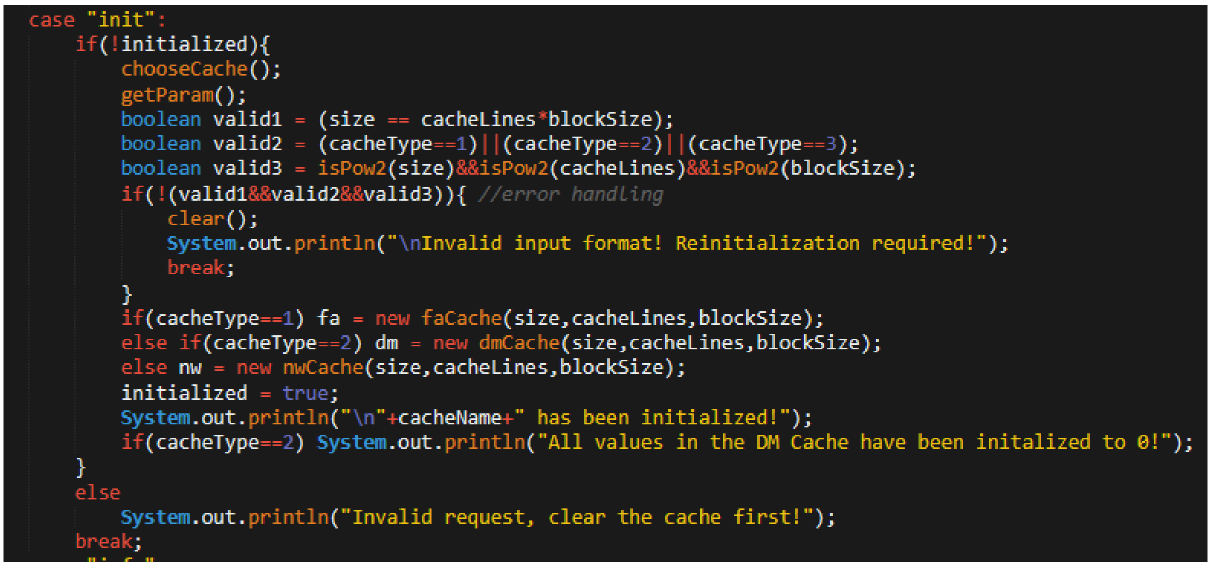
Code Overview of Cache functionalities

The programming language of choice was JAVA. Taking advantage of its modularity, caches and their functionalities are very distinctly represented. The program is menu driven with very user-friendly commands and functionalities.



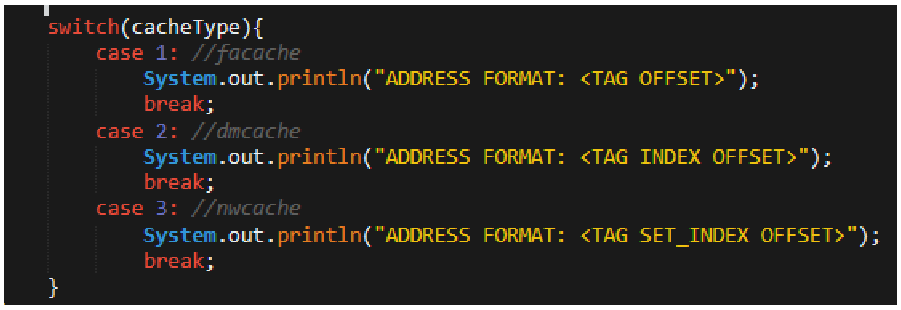
```
COMMANDS
        help: get a list of commands
        exit: exit the terminal
        init: initiate a cache
        info: show cache parameters and address formatting
        type: show chosen cache type
        write: read the value at a given address
        read: read the value at a given address
        clear: clear memories of all caches
        print: print the whole cache
        time: know the value of cache timer
```

The caches are required to be initiated first with cache parameters (in bytes).



```java
  case "init":
      if(!initialized){
          chooseCache();
          getParam();
          boolean valid1 = (size == cacheLines*blockSize);
          boolean valid2 = (cacheType==1)||(cacheType==2)||(cacheType==3);
          boolean valid3 = isPow2(size)&&isPow2(cacheLines)&&isPow2(blockSize);
          if(!(valid1&&valid2&&valid3)){ //error handling
              clear();
              System.out.println("\nInvalid input format! Reinitialization required!");
              break;
          }
          if(cacheType==1) fa = new faCache(size,cacheLines,blockSize);
          else if(cacheType==2) dm = new dmCache(size,cacheLines,blockSize);
          else nw = new nwCache(size,cacheLines,blockSize);
          initialized = true;
          System.out.println("\n"+cacheName+" has been initialized!");
          if(cacheType==2) System.out.println("All values in the DM Cache have been initalized to 0!");
      }
      else
          System.out.println("Invalid request, clear the cache first!");
      break;
```

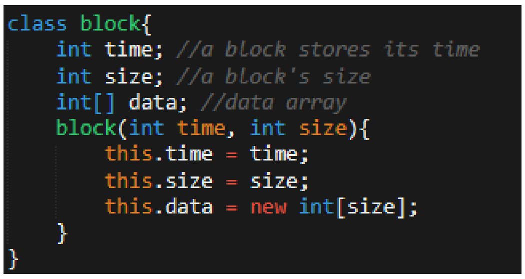
After initiation, for reading and writing address inputs and data inputs are required. Presupposition states that an Integer takes one byte in our system. The addresses are binary indexed. After input, the address is formatted according to cache requirements.



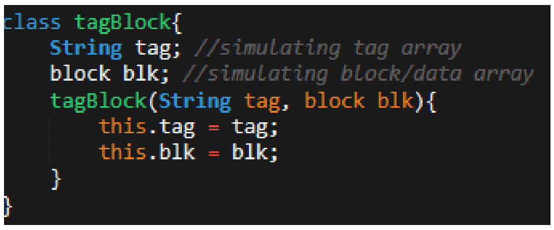
```java
switch(cacheType){
    case 1: //facache
        System.out.println("ADDRESS FORMAT: <TAG OFFSET>");
        break;
    case 2: //dmcache
        System.out.println("ADDRESS FORMAT: <TAG INDEX OFFSET>");
        break;
    case 3: //nwcache
        System.out.println("ADDRESS FORMAT: <TAG SET_INDEX OFFSET>");
        break;
}
```

The fundamental units, block of an array and the mapping of each tag to a block have been atomized to classes to showcase their fundamental nature.

Computer Organization
Name: *Divyansh Rastogi*
Roll. No.: *2019464*

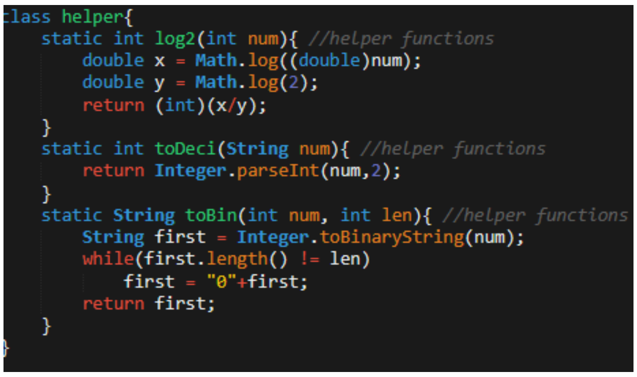


```
class block{
    int time; //a block stores its time
    int size; //a block's size
    int[] data; //data array
    block(int time, int size){
        this.time = time;
        this.size = size;
        this.data = new int[size];
    }
}
```

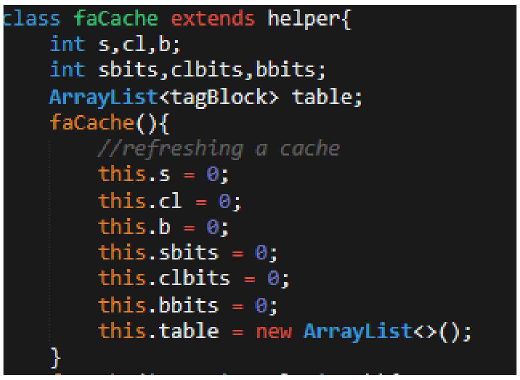


```
class tagBlock{
    String tag; //simulating tag array
    block blk; //simulating block/data array
    tagBlock(String tag, block blk){
        this.tag = tag;
        this.blk = blk;
    }
}
```

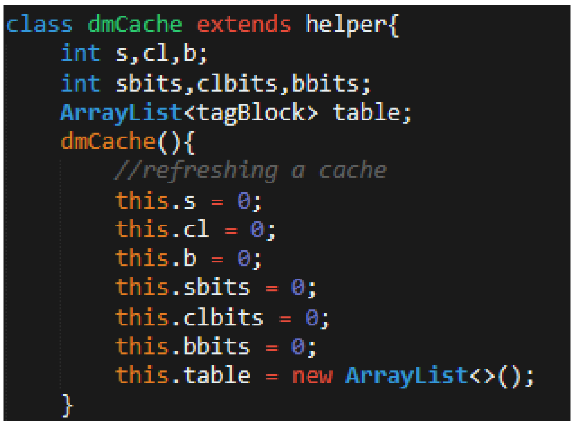
Separate classes for each of the caches have been implemented to modularize their functionality with a super main file class driving the program. A helper class has been created in order to extract helper functions such as binary and decimal conversion.



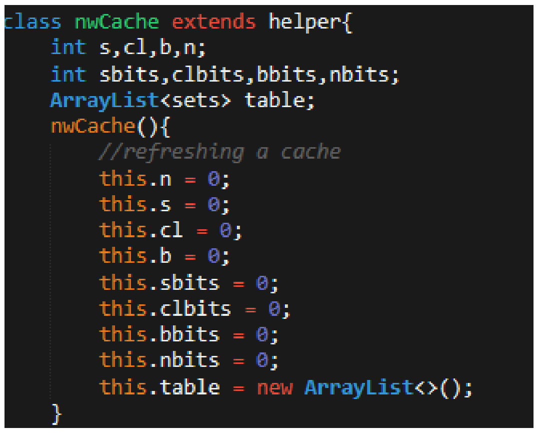
```
class helper{
    static int log2(int num){ //helper functions
        double x = Math.log((double)num);
        double y = Math.log(2);
        return (int)(x/y);
    }
    static int toDeci(String num){ //helper functions
        return Integer.parseInt(num,2);
    }
    static String toBin(int num, int len){ //helper functions
        String first = Integer.toBinaryString(num);
        while(first.length() != len)
            first = "0"+first;
        return first;
    }
}
```



```
class faCache extends helper{
    int s,cl,b;
    int sbits,clbits,bbits;
    ArrayList<tagBlock> table;
    faCache(){
        //refreshing a cache
        this.s = 0;
        this.cl = 0;
        this.b = 0;
        this.sbits = 0;
        this.clbits = 0;
        this.bbits = 0;
        this.table = new ArrayList<>();
    }
```



```
class dmCache extends helper{
    int s,cl,b;
    int sbits,clbits,bbits;
    ArrayList<tagBlock> table;
    dmCache(){
        //refreshing a cache
        this.s = 0;
        this.cl = 0;
        this.b = 0;
        this.sbits = 0;
        this.clbits = 0;
        this.bbits = 0;
        this.table = new ArrayList<>();
    }
```



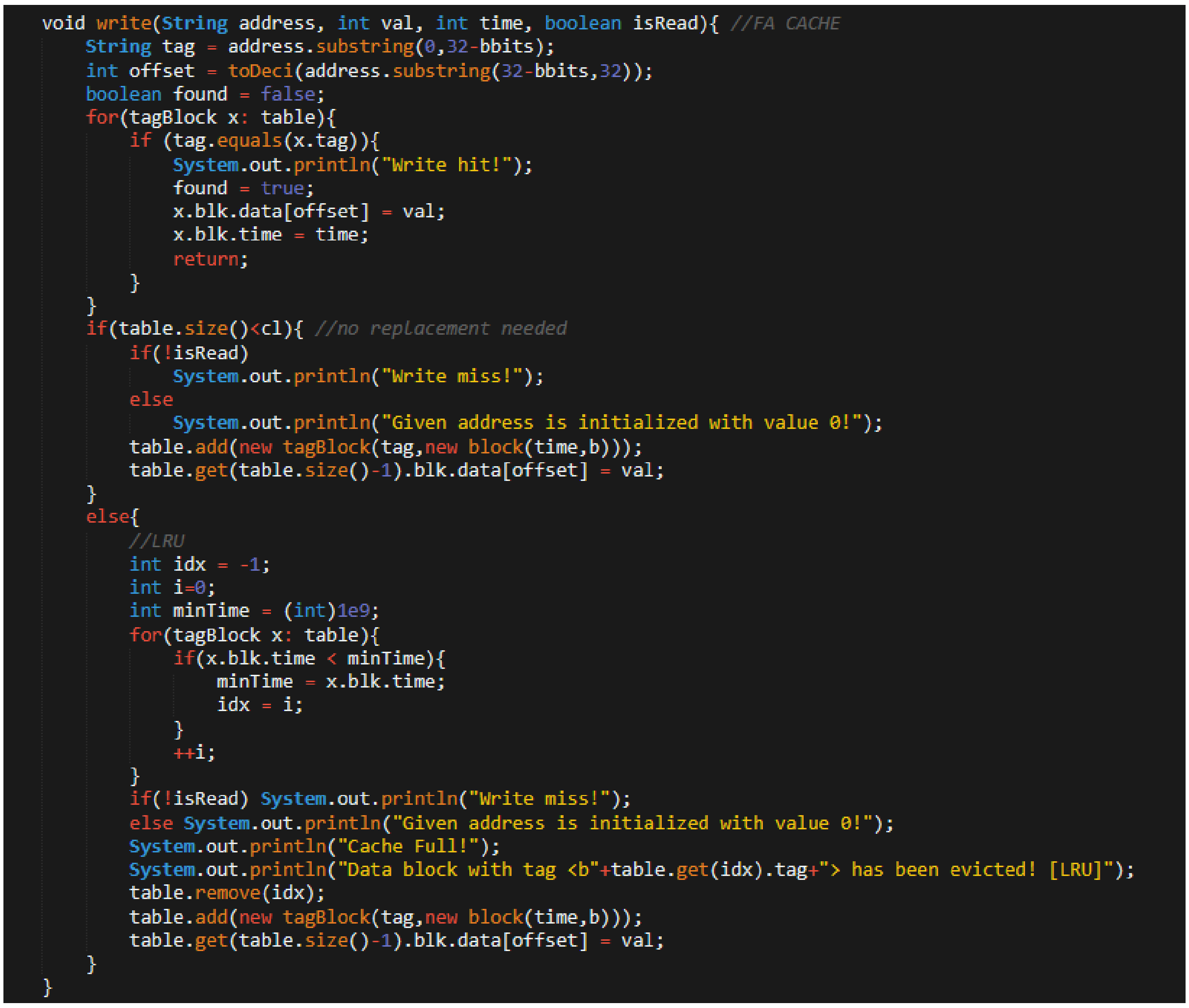
```
class nwCache extends helper{
    int s,cl,b,n;
    int sbits,clbits,bbits,nbits;
    ArrayList<sets> table;
    nwCache(){
        //refreshing a cache
        this.n = 0;
        this.s = 0;
        this.cl = 0;
        this.b = 0;
        this.sbits = 0;
        this.clbits = 0;
        this.bbits = 0;
        this.nbits = 0;
        this.table = new ArrayList<>();
    }
```

The writing and reading are simulated as in a real cache.

- For the FA cache, first the cache is empty, the input address is broken down into tag and offset and then tags are compared individually and if found, it's written/read else write/read miss it passed and the block is loaded into the cache. When the cache is full LRU policy is catered to which works in accordance with the timer set for the caches which increments on every read and write.
- For the DM cache, first the cache is filled with default addresses starting from 0. For every input address, it's broken down to tag, index and offset. Read and write miss only occur when at the current index, the tag doesn't match, else a read/write hit is passed and the data is overwritten/read.

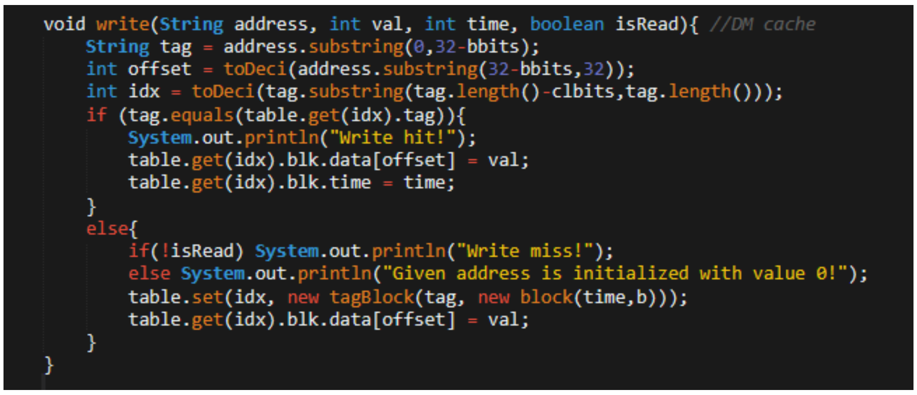
- For NW cache, combination of policies is followed, first all sets are empty, then set is found according to policies of DM cache, while reading and writing inside a set is done by FA policies (LRU in a set).

Basis of FA Cache write and read: -



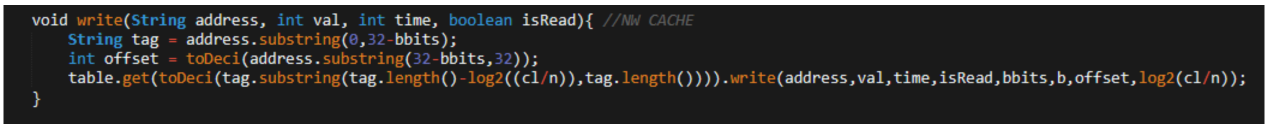
```java
void write(String address, int val, int time, boolean isRead){ //FA CACHE
    String tag = address.substring(0,32-bbits);
    int offset = toDeci(address.substring(32-bbits,32));
    boolean found = false;
    for(tagBlock x: table){
        if (tag.equals(x.tag)){
            System.out.println("Write hit!");
            found = true;
            x.blk.data[offset] = val;
            x.blk.time = time;
            return;
        }
    }
    if(table.size()<cl){ //no replacement needed
        if(!isRead)
            System.out.println("Write miss!");
        else
            System.out.println("Given address is initialized with value 0!");
        table.add(new tagBlock(tag,new block(time,b)));
        table.get(table.size()-1).blk.data[offset] = val;
    }
    else{
        //LRU
        int idx = -1;
        int i=0;
        int minTime = (int)1e9;
        for(tagBlock x: table){
            if(x.blk.time < minTime){
                minTime = x.blk.time;
                idx = i;
            }
            ++i;
        }
        if(!isRead) System.out.println("Write miss!");
        else System.out.println("Given address is initialized with value 0!");
        System.out.println("Cache Full!");
        System.out.println("Data block with tag <b"+table.get(idx).tag+"> has been evicted! [LRU]");
        table.remove(idx);
        table.add(new tagBlock(tag,new block(time,b)));
        table.get(table.size()-1).blk.data[offset] = val;
    }
}
```

Basis of DM Cache Write and Read: -

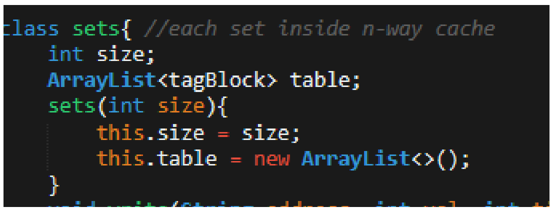


```java
void write(String address, int val, int time, boolean isRead){ //DM cache
    String tag = address.substring(0,32-bbits);
    int offset = toDeci(address.substring(32-bbits,32));
    int idx = toDeci(tag.substring(tag.length()-clbits,tag.length()));
    if (tag.equals(table.get(idx).tag)){
        System.out.println("Write hit!");
        table.get(idx).blk.data[offset] = val;
        table.get(idx).blk.time = time;
    }
    else{
        if(!isRead) System.out.println("Write miss!");
        else System.out.println("Given address is initialized with value 0!");
        table.set(idx, new tagBlock(tag, new block(time,b)));
        table.get(idx).blk.data[offset] = val;
    }
}
```

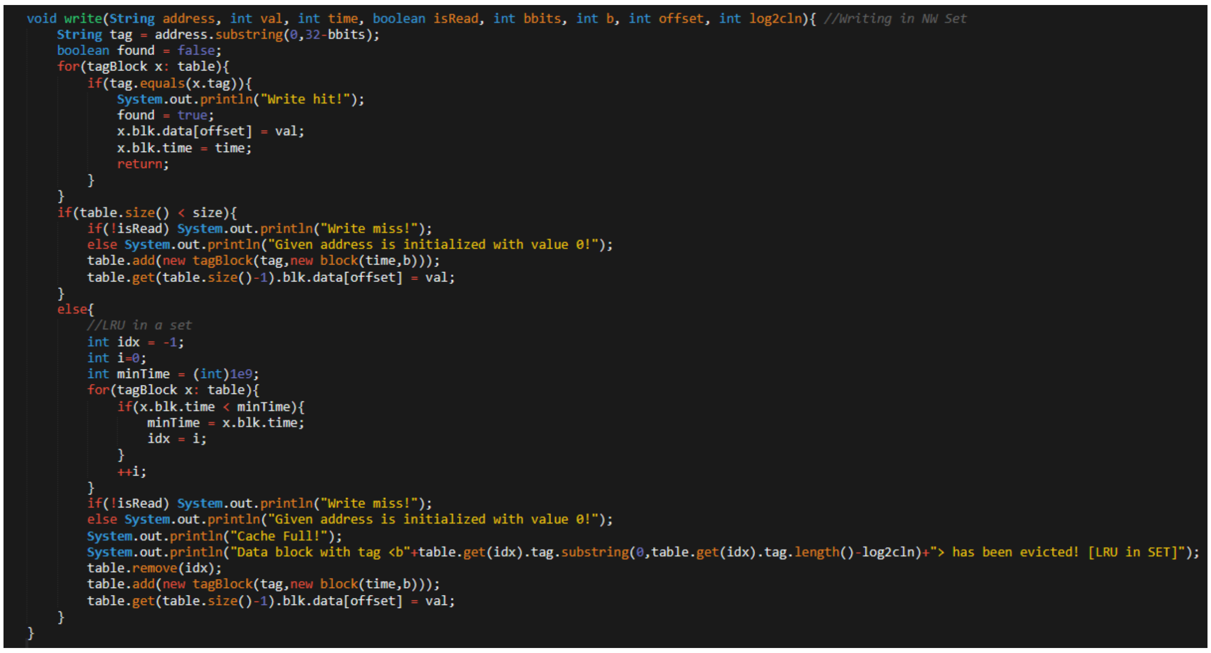
Basis of NW Cache Write and Read: -



```
void write(String address, int val, int time, boolean isRead){ //NW CACHE
    String tag = address.substring(0,32-bbits);
    int offset = toDeci(address.substring(32-bbits,32));
    table.get(toDeci(tag.substring(tag.length()-log2((cl/n)),tag.length()))).write(address,val,time,isRead,bbits,b,offset,log2(cl/n));
}
```

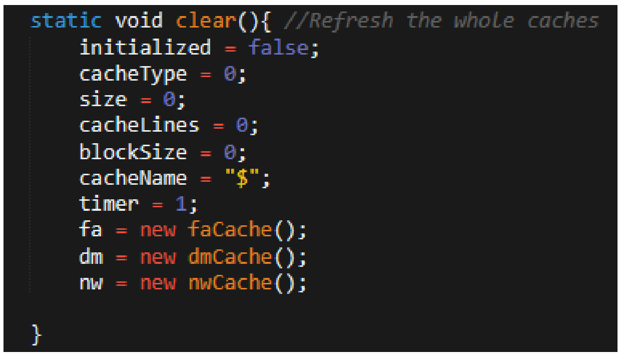


```
lass sets{ //each set inside n-way cache
    int size;
    ArrayList<tagBlock> table;
    sets(int size){
        this.size = size;
        this.table = new ArrayList<>();
    }
```



```
void write(String address, int val, int time, boolean isRead, int bbits, int b, int offset, int log2cln){ //Writing in NW Set
    String tag = address.substring(0,32-bbits);
    boolean found = false;
    for(tagBlock x: table){
        if(tag.equals(x.tag)){
            System.out.println("Write hit!");
            found = true;
            x.blk.data[offset] = val;
            x.blk.time = time;
            return;
        }
    }
    if(table.size() < size){
        if(!isRead) System.out.println("Write miss!");
        else System.out.println("Given address is initialized with value 0!");
        table.add(new tagBlock(tag,new block(time,b)));
        table.get(table.size()-1).blk.data[offset] = val;
    }
    else{
        //LRU in a set
        int idx = -1;
        int i=0;
        int minTime = (int)1e9;
        for(tagBlock x: table){
            if(x.blk.time < minTime){
                minTime = x.blk.time;
                idx = i;
            }
            ++i;
        }
        if(!isRead) System.out.println("Write miss!");
        else System.out.println("Given address is initialized with value 0!");
        System.out.println("Cache Full!");
        System.out.println("Data block with tag <b"+table.get(idx).tag.substring(0,table.get(idx).tag.length()-log2cln)+"> has been evicted! [LRU in SET]");
        table.remove(idx);
        table.add(new tagBlock(tag,new block(time,b)));
        table.get(table.size()-1).blk.data[offset] = val;
    }
}
```

When command 'clear' is passed, every parameter is initialized to default, the user is rooted back to the main directory and all caches are refreshed.



```
static void clear(){ //Refresh the whole caches
    initialized = false;
    cacheType = 0;
    size = 0;
    cacheLines = 0;
    blockSize = 0;
    cacheName = "$";
    timer = 1;
    fa = new faCache();
    dm = new dmCache();
    nw = new nwCache();

}
```

The documentation covers the concepts and functionalities of all the basic commands of the cache and its workings. To get an even further detailed overview, please refer the code file.