```
package com.ehelpy.brihaspati4.GC;
//Glue Code Pseudo Code:

//  Authentication Manager = am
//  Communication Manager = cm
//  Indexing Manager = im
//  Routing Manager = rm
//  Web Server Module = ws
//  Web Module = web
//  VoIP = voip
//  DFS = dfs
//  UFS = ufs
//  Mail = mail
//  Message = sms

//import the required packages...
import java.util.LinkedList;



public class GlueCode {
//Start thread of Authentication Manager.
//Start thread of Routing Manager.
//Start thread of Communication Manager.
//Start thread of Indexing Manager.
//Start thread of Web Server.
```

//Initialize a status_skip_flag corresponding to each input buffer (one input buffer for each module) of the Glue Code with 1. (This status_skip_flag will contain values in the powers of 2 such as 1, 2, 4, etc., upto some specific number limit such as 16 or 32.) (The default value for the status_skip_flag for each module at starting will be 1.)

```
   private int status_skip_flag_am = 1;
   private int status_skip_flag_rm = 1;
   private int status_skip_flag_cm = 1;
   private int status_skip_flag_im = 1;
   private int status_skip_flag_ws = 1;
   private int max_limit_of_skip_flags = 32;
   private int count = 0;
   // input module buffer output module
```

```
LinkedList gc_buffer_am = new LinkedList();
LinkedList gc_buffer_rm = new LinkedList();
LinkedList gc_buffer_cm = new LinkedList();
LinkedList gc_buffer_im = new LinkedList();
LinkedList gc_buffer_ws = new LinkedList();
//LinkedList gc_buffer_web = new LinkedList();
LinkedList internal_process_queue = new LinkedList();
```

//Initialize a current_skip_flag corresponding to each input buffer (one input buffer for each module) of the Glue Code with 1. (This current_skip_flag will contain the current value; ie., after how many rotations the input buffer corresponding to that respective module will be checked.) (The default value for the current_skip_flag for each module at starting will be 1.)

```
private int current_skip_flag_am = 1;
private int current_skip_flag_rm = 1;
private int current_skip_flag_cm = 1;
private int current_skip_flag_im = 1;
private int current_skip_flag_ws = 1;
```

//The input buffers of the Glue Code will be checked periodically (only if the current_skip_flag's value is 1. More about this in the next point.) for all the modules in Round Robin fashion. Each input buffer will be checked for 5 seconds or until all the data/query are extracted, whichever time is less.

//If the current_skip_flag value is 1, then only the corresponding module's input buffer is checked. Otherwise, the current_skip_flag's value is decremented by 1.

```
if(current_skip_flag_am == 1)
{
   //check output buffer of authentication manager;
   //if (data found in the output buffer of authentication manager)
   if ((gc_buffer_am.size())>0)
   {
      status_skip_flag_am = 1;
      current_skip_flag_am = 1;
      count=0;
      while ( (gc_buffer_am.size()>0) && (count<1000) )
```

redundant

```
    {
      internal_process_queue.add(gc_buffer_am.pollFirst());
      count = count+1;
    }
  }
  //else if (no query found in the output buffer of Authentication Manager)
  //else if (gc_buffer_am.size()==0)
  else
  {
    if (status_skip_flag_am < max_limit_of_skip_flags)
    {
      status_skip_flag_am = status_skip_flag_am * 2;
    }
    current_skip_flag_am = status_skip_flag_am;
  }
}
else
{
  current_skip_flag_am--;
}


if(current_skip_flag_rm ==1)
{
  //check output buffer of routing manager;
  //if (data found in the output buffer of routing manager)
  if ((gc_buffer_rm.size())>0)
  {
    status_skip_flag_rm = 1;
    current_skip_flag_rm = 1;
    count=0;
    while ( (gc_buffer_rm.size()>0) && (count<1000) )
    {
      internal_process_queue.add(gc_buffer_rm.pollFirst());
      count = count+1;
    }
  }
  //else if (no query found in the output buffer of Routing Manager)
  //else if (gc_buffer_rm.size()==0)
```

→ You Can make into a function

Call this function for am, rm, and so on.

Code lines will be less. Any logic change in buffer reading will happen for all modules uniformly. No manual Code change at all the places.

```
   else
   {
      if (status_skip_flag_rm < max_limit_of_skip_flags)
      {
         status_skip_flag_rm = status_skip_flag_rm * 2;
      }
      current_skip_flag_rm = status_skip_flag_rm;
   }
}
else
{
   current_skip_flag_rm--;
}


if(current_skip_flag_cm ==1)
{
   //check output buffer of communication manager;
   //if (data found in the output buffer of communication manager)
   if ((gc_buffer_cm.size())>0)
   {
      status_skip_flag_cm = 1;
      current_skip_flag_cm = 1;
      count=0;
      while ( (gc_buffer_cm.size()>0) && (count<1000) )
      {
         internal_process_queue.add(gc_buffer_cm.pollFirst());
         count = count+1;
      }
   }
   //else if (no query found in the output buffer of Communication Manager)
   //else if (gc_buffer_cm.size()==0)
   else
   {
      if (status_skip_flag_cm < max_limit_of_skip_flags)
      {
         status_skip_flag_cm = status_skip_flag_cm * 2;
      }
      current_skip_flag_cm = status_skip_flag_cm;
```

```
    }
}
else
{
    current_skip_flag_cm--;
}


if(current_skip_flag_im ==1)
{
    //check output buffer of indexing manager;
    //if (data found in the output buffer of indexing manager)
    if ((gc_buffer_im.size())>0)
    {
        status_skip_flag_im = 1;
        current_skip_flag_im = 1;
        count=0;
        while ( (gc_buffer_im.size()>0) && (count<1000) )
        {
            internal_process_queue.add(gc_buffer_im.pollFirst());
            count = count+1;
        }
    }
    //else if (no query found in the output buffer of Indexing Manager)
    //else if (gc_buffer_im.size()==0)
    else
    {
        if (status_skip_flag_im < max_limit_of_skip_flags)
        {
            status_skip_flag_im = status_skip_flag_im * 2;
        }
        current_skip_flag_im = status_skip_flag_im;
    }
}
else
{
    current_skip_flag_im--;
}
```

```
if(current_skip_flag_ws ==1)
{
   //check output buffer of web server;
   //if (data found in the output buffer of web server)
   if ((gc_buffer_ws.size())>0)
   {
      status_skip_flag_ws = 1;
      current_skip_flag_ws = 1;
      count=0;
      while ( (gc_buffer_ws.size()>0) && (count<1000) )
      {
         internal_process_queue.add(gc_buffer_ws.pollFirst());
         count = count+1;
      }
   }
   //else if (no query found in the output buffer of Web Server)
   //else if (gc_buffer_ws.size()==0)
   else
   {
      if (status_skip_flag_ws < max_limit_of_skip_flags)
      {
         status_skip_flag_ws = status_skip_flag_ws * 2;
      }
      current_skip_flag_ws = status_skip_flag_ws;
   }
}
else
{
   current_skip_flag_ws--;
}
```

//If data/query is found in the input buffer of Glue Code (output buffer of the respective module), then both the status_skip_flag and the current_skip_flag values are set to 1.

//If no data/query is found, then the status_skip_flag value is multiplied by 2 if the status_skip_flag value is less than the number limit which is set before (something like 16 or

32), else, the status_skip_flag value is kept the same as the number limit (16 or 32). Also, the current_skip_flag is made equal to the status_skip_flag.

//If any query is found in any input buffer, read it, analyse it, and then do the appropriate operation.

//The required appropriate operation may be to give information (publish) to any module or to get some information (query) from any module.

//In such a case, the Glue Code will call the respective module directly via an API call.

//The modules will respond to such calls instantly with the required data/information (in case of a query). (As no response is needed in case of a publish).

//After the Glue Code receives the required information, it will send back the information to the module which initially demanded for it.

//In this case the information is directly put in the input buffer of that respective module which initially demanded for that information.

//All these while, the Glue Code will also update all the required statistics and data to a web browser using the web-server module.
//(This will be the user interface; to be done using React JS).

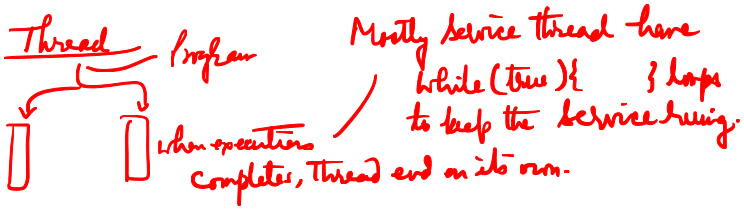
//The below threads will be started based on demand… (These will always have a Singleton Thread.)

↳ Any data which is to be retained should be saved on disk when thread finishes

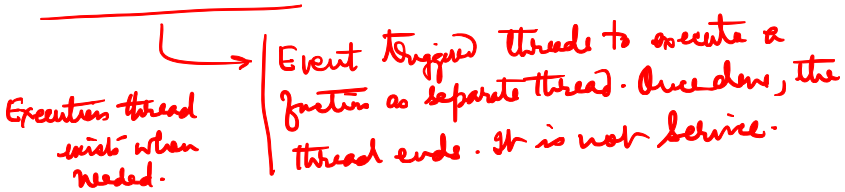
//(Singleton Thread - Whenever a thread needs to be started, we will check whether there is a thread already running or not. If yes, we will process that thread only. Otherwise, we will start a new thread. And whenever the requirement of the module ends, we will stop the thread.)

//Message Service
//Start thread of Messaging System.
//Condition for Starting Thread:
//(i) Message comes from other node via Communication Manager. This information

reaches the Glue Code. The Glue Code starts the singleton Messaging thread, and hands over the Message to the Messaging module. The messaging module will read the messages and write it on the hard disk. Then the messaging module will create a pop-up that a message has been received and then the messaging module thread will be closed.
//(ii) Whenever the user clicks on the pop-up, the messaging module thread will run again, and the Messaging UI Window will be opened. … The messaging thread will keep running until the user manually closes the messaging window.
//(iii) When the user opens the messaging service manually, the messaging module thread will run again, and the Messaging UI Window will be opened. … The messaging thread will keep running until the user manually closes the messaging window.



//Mailing Service
//Start thread of Mailing System.
//Condition for Starting Thread:
//(i) Mail comes from other node via Communication Manager. This information reaches the Glue Code. The Glue Code starts the singleton Mailing thread, and hands over the Mail to the Mailing module. The mailing module will create a pop-up that a mail has been received and then the mailing module thread will be closed.
//(ii) Whenever the user clicks on the pop-up, the mailing module thread will run again, and the Mail UI Window will be opened. The mailing thread will keep running until the user manually closes the Mail UI Window.
//(iii) When the user opens the mailing service manually, the mailing module thread will run again, and the Mail UI Window will be opened. The mailing thread will keep running until the user manually closes the Mail UI Window.



//VOIP Service
//Start thread of VoIP.
//Condition for starting Thread:
//(i) When the user manually starts the VoIP, the VoIP module thread will start. … When the user manually closes the VoIP window, the VoIP module will be closed.
//(ii) When VoIP call request arrives from the other node, then the request arrives through the Communication Manager to the Glue Code, and then the VoIP thread will start running, the VoIP window will be opened requesting user to accept the call.
//(a) If the user accepts the call, then the call will continue until the user disconnects the call. …*** (After the call has been set–up, then the VoIP service directly communicates with the Communication Manager… That means, the Glue Code is bypassed…) When the user disconnects the call, the VoIP thread will close.)

//(b) If the user does not accept the call, then the ring will happen for about 30 seconds, and then a pop-up of missed call will be shown the window by the VoIP, and the VoIP module thread will be closed.
//(c) When the user rejects the call, the VoIP service thread will be closed.
//(iii) When the user clicks on the pop-up, the VoIP module thread will start. When the user manually closes the VoIP window, the VoIP module will be closed.

//Storage Service
//Start service of DFS Storage.
//Start service of UFS Storage.

//Web Service
//Start service of Web.

//Search Service
//Start thread of Search.

//Address Book Service
//Start thread of Address Book.
}