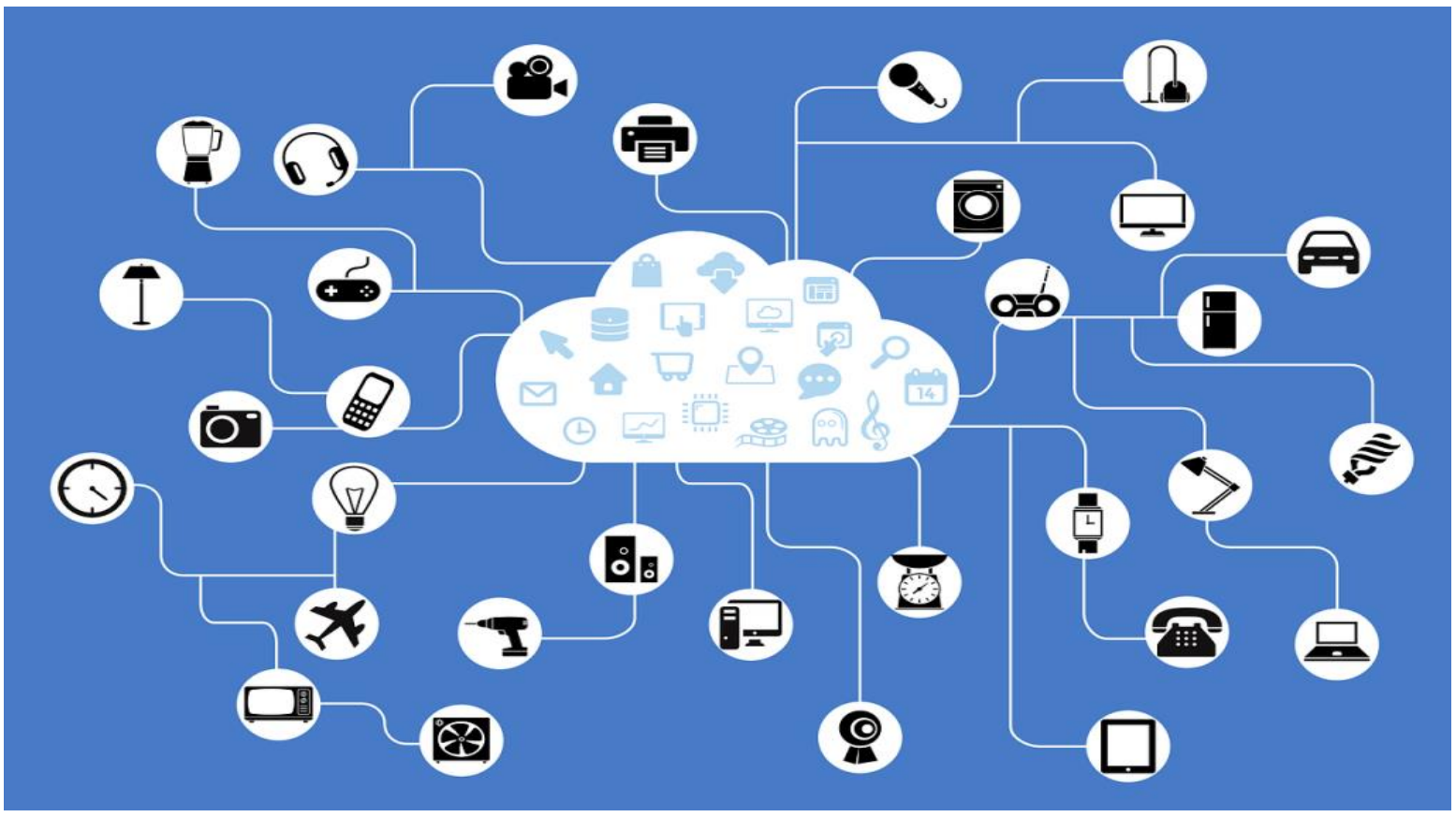
OPREATING SYSTEM PROJECT



Supervised by :

DR. Mohamed Ali Saleh

Team name:

OS Geeks

Team members:

Ahmed Sha'aban Ahmed

Reham Khalf Aabdelhamiid

Mohamed Hassan Saied

Mohamed saleh Hassanen

Mahmoud Yahia Mohamed

Part one of document written by:

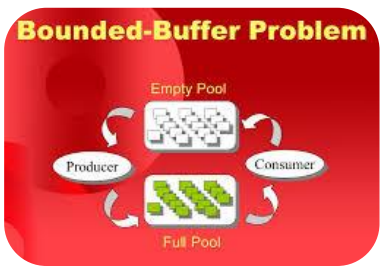
Reham Khalf Abdelhamid Mohamed

Part two of document written by:

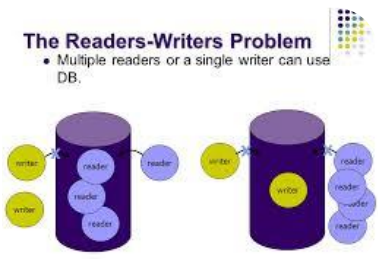
Mohamed Saleh Hassanien

Part one of documentation:

SYNCHRONISATION



Bounded buffer



writer/ reader

Dining philosophers

barber's shop

```
Author:                         Mahmoud Yahia
Date:                           1/5/2016
Version:                        1.0
Project ID:                     Synchronization module
CS Class:                       sec 4 OS subject 3_computer
Programming Language:                java
OS/Hardware dependencies:       None

Problem Description:            we have 4 problem's in this module :
                                   1- Bounded buffer : The problem describes two processes, the
                                      producer and the consumer, who share a common, fixed-size
                                      buffer used as a queue. The producer's job is to generate
                                      data, put it into the buffer, and start again. At the same
                                      time, the consumer is consuming the data , one piece at a
                                      time. The problem is to make sure that the producer won't
                                      try to add data into the buffer if it's full and that the
                                      consumer won't try to remove data from an empty buffer

                                   2- Reader / writer problem's : common computing problem in
                                      concurrency
                                      There are at least three variations of the problems, which
                                      deal with situations in which many threads try to access
                                      the same shared resource at one time. Some threads may
                                      read and some may write, with the constraint that no
                                      process may access the share for either reading or
                                      writing, while another process is in the act of writing to
                                      it. ( it is allowed for two or more readers to access the
                                      share at the same time.) A readers-writer lock is a data
                                      structure that solves one or more of the readers-writers
                                      problems.

                                   3- Sleeping Barber shop problem :
                                      The problem is analogous to that of keeping a barber
                                      working when there are customers, resting when there are
                                      none, and doing so in an orderly manner.
```

The analogy is based upon a hypothetical barber shop with one barber. The barber has one barber chair and a waiting room with a number of chairs in it. When the barber finishes cutting a customer's hair, he dismisses the customer and then goes to the waiting room to see if there are other customers waiting. If there are, he brings one of them back to the chair and cuts his hair. If there are no other customers waiting, he returns to his chair and sleeps in it.
Each customer, when he arrives, looks to see what the barber is doing. If the barber is sleeping, then the customer wakes him up and sits in the chair. If the barber is cutting hair, then the customer goes to the waiting room. If there is a free chair in the waiting room, the customer sits in it and waits his turn. If there is no free chair, then the customer leaves
Based on a naïve analysis, the above description should ensure that the shop functions correctly, with the barber cutting the hair of anyone who arrives until there are no more customers, and then sleeping until the next customer arrives. In practice, there are a number of problems that can occur that are illustrative of general scheduling problems.

The problems are all related to the fact that the actions by both the barber and the customer (checking the waiting room, entering the shop, taking a waiting room chair, etc.) all take an unknown amount of time.

4-dinig philosopher :
Five silent philosophers sit at a round table with bowls of spaghetti. Forks are placed between each pair of adjacent philosophers.

Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when he has both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher. After he finishes eating, he needs to put down both forks so they become available to others. A philosopher can take the fork on his right or the one on his left as they become available, but cannot start eating before getting both of them.

Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed.

The problem is how to design a discipline of behavior (a concurrent algorithm) such that no philosopher will starve each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think

---

.

How to build the program: by using netbeans program.java

Program Source:
_____
 # 1- bounded buffer:

**producer _ consumer:*

```
package osmain.ProducerConsumer;

import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Java program to solve Producer Consumer problem using wait and notify
 * method in Java. Producer Consumer is also a popular concurrency design pattern.
 *
 * @author Javin Paul
 */
public class ProducerConsumerSolution {

    public static void ProducerConsumer_Driver() {
```

```
        System.out.println("Executing in Bounded Buffer");
        Vector sharedQueue = new Vector();
        int size = 4;
        Thread prodThread = new Thread(new Producer(sharedQueue, size), "Producer");
        Thread consThread = new Thread(new Consumer(sharedQueue, size), "Consumer");
        prodThread.start();
        consThread.start();
    }
}

class Producer implements Runnable {

    private final Vector sharedQueue;
    private final int SIZE;

    public Producer(Vector sharedQueue, int size) {
        this.sharedQueue = sharedQueue;
        this.SIZE = size;
    }

    @Override
    public void run() {
        for (int i = 0; i < 7; i++) {
            System.out.println("Produced: " + i);
            try {
                produce(i);
            } catch (InterruptedException ex) {
                Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);
            }

        }
    }

    private void produce(int i) throws InterruptedException {

        //wait if queue is full
        while (sharedQueue.size() == SIZE) {
            synchronized (sharedQueue) {
                System.out.println("Queue is full " + Thread.currentThread().getName()
                                    + " is waiting , size: " + sharedQueue.size());

                sharedQueue.wait();
            }
        }

        //producing element and notify consumers
        synchronized (sharedQueue) {
            sharedQueue.add(i);
            sharedQueue.notifyAll();
        }
    }
}

class Consumer implements Runnable {

    private final Vector sharedQueue;
    private final int SIZE;

    public Consumer(Vector sharedQueue, int size) {
        this.sharedQueue = sharedQueue;
        this.SIZE = size;
    }

    @Override
    public void run() {
        while (true) {
            try {
                System.out.println("Consumed: " + consume());
                Thread.sleep(50);
            } catch (InterruptedException ex) {
                Logger.getLogger(Consumer.class.getName()).log(Level.SEVERE, null, ex);
            }

        }
    }

    private int consume() throws InterruptedException {
        //wait if queue is empty
        while (sharedQueue.isEmpty()) {
            synchronized (sharedQueue) {
                System.out.println("Queue is empty " + Thread.currentThread().getName()
                                    + " is waiting , size: " + sharedQueue.size());

                sharedQueue.wait();
            }
        }
```

```
        //Otherwise consume element and notify waiting producer
        synchronized (sharedQueue) {
            sharedQueue.notifyAll();
            return (Integer) sharedQueue.remove(0);
        }
    }
}
```

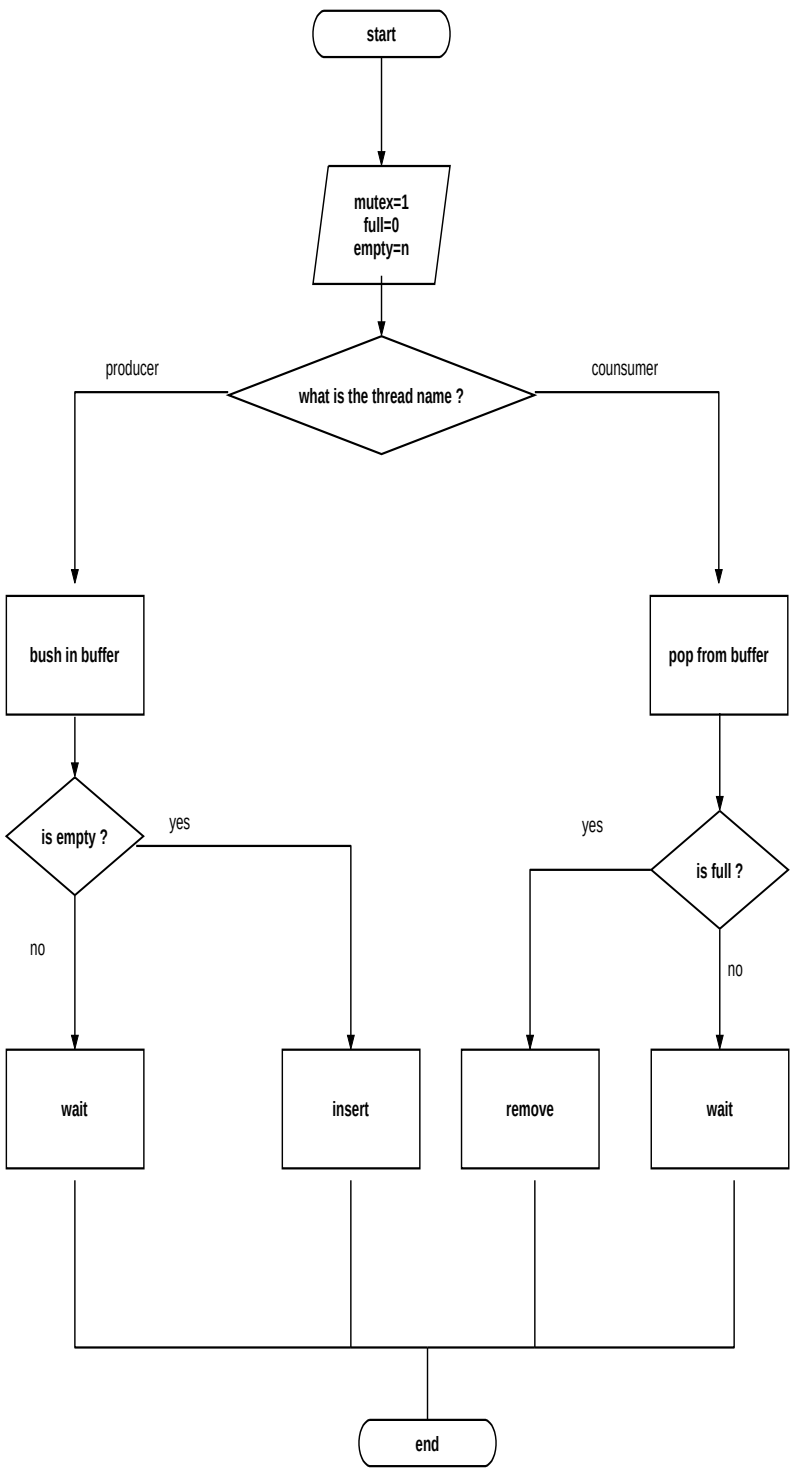
Additional Files: ***None.***

Results:

```
Executing in Bounded Buffer
Produced: 0
Produced: 1
Produced: 2
Produced: 3
Produced: 4
Queue is full Producer is waiting , size: 4
Produced: 5
Queue is full Producer is waiting , size: 4
Consumed: 0
Consumed: 1
Produced: 6
Queue is full Producer is waiting , size: 4
Consumed: 2
Consumed: 3
Consumed: 4
Consumed: 5
Consumed: 6
Queue is empty Consumer is waiting , size: 0
```

References: ***None.***

Flowchart and pseudo code:



Producer Process:

```
while (true)
{
    /* produce an item in nextProduced */
    while (counter == BUFFER_SIZE)
       ; /* do nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
    counter++;
}
```

Consumer Process:

```
while (true)
{
    while (counter == 0)
       ; /* do nothing */
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    counter--;
    /* consume the item in nextConsumed */
}
```

Program Source:

2- reader/writer :

```
package osmain.ReaderWriter;

import java.util.concurrent.Semaphore;

    public class ReaderWriterSolution{
      public static final int NUM_OF_READERS = 3;
      public static final int NUM_OF_WRITERS = 2;

       public static void ReaderWriter_Driver(){
         System.out.println("Executing in Reader Writer");
         RWLock database = new Database();

         Thread[] readerArray = new Thread[NUM_OF_READERS];
         Thread[] writerArray = new Thread[NUM_OF_WRITERS];

         for (int i = 0; i < NUM_OF_READERS; i++) {
            readerArray[i] = new Thread(new Reader(i, database));
            readerArray[i].start();
         }

         for (int i = 0; i < NUM_OF_WRITERS; i++) {
            writerArray[i] = new Thread(new Writer(i, database));
            writerArray[i].start();
         }
       }
   }

//*************************************************************


    interface RWLock{
       public abstract void acquireReadLock(int readerNum);
       public abstract void acquireWriteLock(int writerNum);
       public abstract void releaseReadLock(int readerNum);
       public abstract void releaseWriteLock(int writerNum);
   }

//*************************************************************



    class Database implements RWLock{
      private int readerCount;  // the number of active readers
      private Semaphore mutex;  // controls access to readerCount
      private Semaphore db;     // controls access to the database

       public Database() {
         readerCount = 0;
         mutex = new Semaphore(1);
         db = new Semaphore(1);
       }

       public void acquireReadLock(int readerNum) {
         try{
         //mutual exclusion for readerCount
            mutex.acquire();
         }
             catch (InterruptedException e) {}

         ++readerCount;

       // if I am the first reader tell all others
       // that the database is being read
         if (readerCount == 1){
            try{
               db.acquire();
            }
                catch (InterruptedException e) {}
         }

         System.out.println("Reader " + readerNum + " is reading. Reader count = " +
readerCount);
         //mutual exclusion for readerCount
         mutex.release();
       }

       public void releaseReadLock(int readerNum) {
         try{
```

```
         //mutual exclusion for readerCount
            mutex.acquire();
         }
             catch (InterruptedException e) {}

         --readerCount;

      // if I am the last reader tell all others
      // that the database is no longer being read
         if (readerCount == 0){
            db.release();
         }

         System.out.println("Reader " + readerNum + " is done reading. Reader count = " +
readerCount);

      //mutual exclusion for readerCount
         mutex.release();
      }

       public void acquireWriteLock(int writerNum) {
         try{
            db.acquire();
         }
             catch (InterruptedException e) {}
         System.out.println("Writer " + writerNum + " is writing.");
      }

       public void releaseWriteLock(int writerNum) {
         System.out.println("Writer " + writerNum + " is done writing.");
         db.release();
      }


   }

//*************************************************************


    class Reader implements Runnable
   {

      private RWLock database;
      private int readerNum;

       public Reader(int readerNum, RWLock database) {
         this.readerNum = readerNum;
         this.database = database;
      }

       public void run() {
         while (true) {
            SleepUtilities.nap();

            System.out.println("reader " + readerNum + " wants to read.");
            database.acquireReadLock(readerNum);

         // you have access to read from the database
         // let's read for awhile .....
            SleepUtilities.nap();

            database.releaseReadLock(readerNum);
         }
      }
   ;
   }

//*************************************************************

/**
 * Writer.java
 *
 * A writer to the database.
 *
 */
    class Writer implements Runnable
   {
      private RWLock database;
      private int writerNum;

       public Writer(int w, RWLock d) {
         writerNum = w;
         database = d;
      }

       public void run() {
```

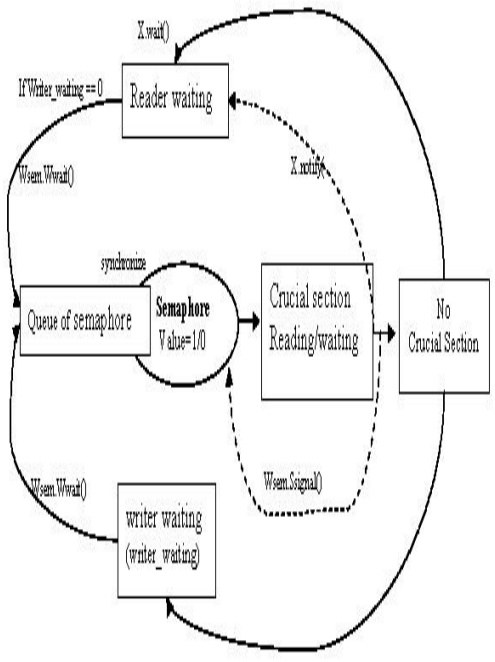
```
            while (true){
               SleepUtilities.nap();

               System.out.println("writer " + writerNum + " wants to write.");
               database.acquireWriteLock(writerNum);

            // you have access to write to the database
            // write for awhile ...
               SleepUtilities.nap();

               database.releaseWriteLock(writerNum);
            }
         }


      }

   //*********************************************************************


       class SleepUtilities
      {

          public static void nap() {
            nap(NAP_TIME);
         }

          public static void nap(int duration) {
            int sleeptime = (int) (NAP_TIME * Math.random() );
            try { Thread.sleep(sleeptime*1000); }
                catch (InterruptedException e) {}
         }

         private static final int NAP_TIME = 5;
      }
```

Additional Files: *None.*

Results:

```
reader 1 wants to read.
reader 2 wants to read.
Writer 1 is done writing.
Writer 0 is writing.
writer 1 wants to write.
reader 0 wants to read.
Writer 0 is done writing.
Reader 1 is reading. Reader count = 1
Reader 2 is reading. Reader count = 2
Reader 0 is reading. Reader count = 3
writer 0 wants to write.
Reader 1 is done reading. Reader count = 2
Reader 2 is done reading. Reader count = 1
Reader 0 is done reading. Reader count = 0
Writer 1 is writing.
Writer 1 is done writing.
Writer 0 is writing.
writer 1 wants to write.
reader 2 wants to read.
reader 0 wants to read.
Writer 0 is done writing.
Writer 1 is writing.
reader 1 wants to read.
```

Flowchart and pseudo code



Readers-Writers Problem

- The structure of a writer process

```
do {
    wait (wrt) ; //  writing is performed
    signal (wrt) ;
  } while (TRUE);
```

The structure of a reader process

```
do {
    wait (mutex) ;
    readcount ++ ;
    if (readcount == 1)
        wait (wrt) ;
    signal (mutex) //reading is performed
        readcount  - - ;
        if (readcount  == 0)
            signal (wrt) ;
        signal (mutex) ;
} while (TRUE);
```

wait (mutex) ;

Program Source:

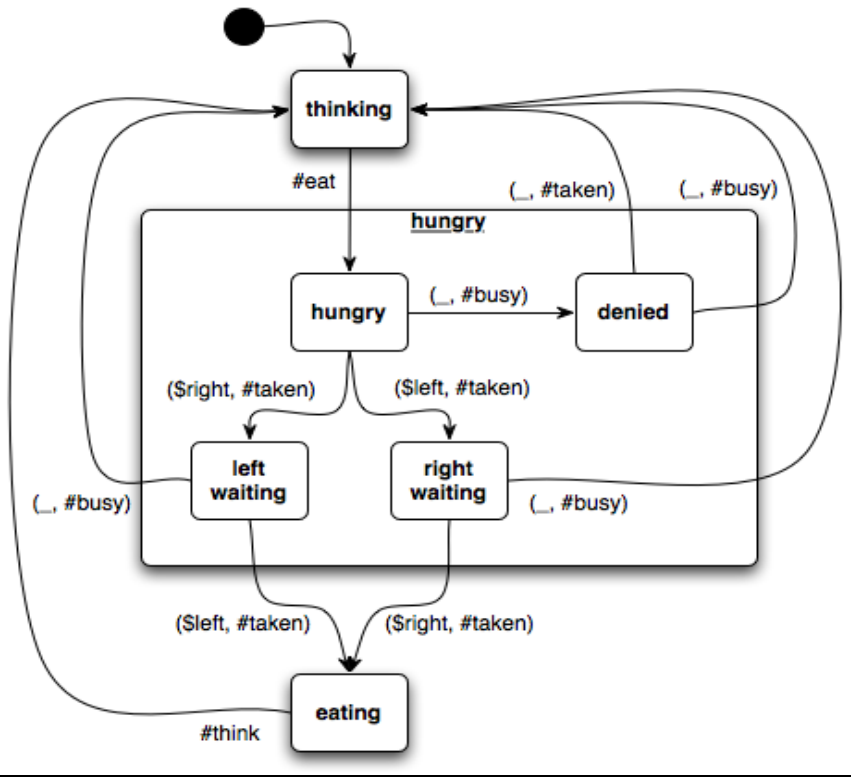
```
# 3- dining philosopher
package osmain.diningPhilosophers;
import osmain.diningPhilosophers.Philosopher;
import osmain.diningPhilosophers.Chopstick;
import java.util.concurrent.*;
import java.util.*;
public class DiningPhilosophers {

     public static void DiningPhilosophers_Driver(String[] args) throws Exception {
        int ponder = 5;
        if (args.length > 0) {
            ponder = Integer.parseInt(args[0]);
        }
        int size = 5;
        if (args.length > 1) {
            size = Integer.parseInt(args[1]);
        }
        ExecutorService exec = Executors.newCachedThreadPool();
        Chopstick[] sticks = new Chopstick[size];
        for (int i = 0; i < size; i++) {
            sticks[i] = new Chopstick();
        }
        for (int i = 0; i < size; i++) {
            exec.execute(new Philosopher(
                    sticks[i], sticks[(i + 1) % size], i, ponder));
        }
        if (args.length == 3 && args[2].equals("timeout")) {
            TimeUnit.SECONDS.sleep(5);
        } else {
            System.out.println("Press 'Enter' to quit");
            System.in.read();
        }
        exec.shutdownNow();
    }
}
```

Additional Files: None.

Results:

```
3
Philosopher 0 thinking
Philosopher 4 thinking
Philosopher 1 thinking
Philosopher 3 thinking
Press 'Enter' to quit
Philosopher 2 thinking
Philosopher 0 grabbing right
Philosopher 1 grabbing right
Philosopher 4 grabbing right
Philosopher 3 grabbing right
Philosopher 4 grabbing left
Philosopher 0 grabbing left
Philosopher 1 grabbing left
Philosopher 2 grabbing right
Philosopher 2 grabbing left
Philosopher 3 grabbing left
```

Flowchart and pseudo code




```
#define N 5                         /* number of philosophers */

void philosopher(int i)             /* i: philosopher number, from 0 to 4 */
{
    while (TRUE) {
        think();                    /* philosopher is thinking */
        take_fork(i);               /* take left fork */
        take_fork((i+1) % N);       /* take right fork; % is modulo operator */
        eat();                      /* yum-yum, spaghetti */
        put_fork(i);                /* put left fork back on the table */
        put_fork((i+1) % N);        /* put right fork back on the table */
    }
}
```

Program Source:

```
# 4- sleeping barber :
package osmain.SleepingBarber;
import java.util.concurrent.*;

public class SleepingBarber extends Thread {


    public static Semaphore customers = new Semaphore(0);
    public static Semaphore barber = new Semaphore(0);
    public static Semaphore accessSeats = new Semaphore(1);

  /* we denote that the number of chairs in this barbershop is 5. */

    public static final int CHAIRS = 5;

  /* we create the integer numberOfFreeSeats so that the customers
   can either sit on a free seat or leave the barbershop if there
   are no seats available */

   public static int numberOfFreeSeats = CHAIRS;


/* THE CUSTOMER THREAD */

class Customer extends Thread {


  int iD;
  boolean notCut=true;

  /* Constructor for the Customer */

  public Customer(int i) {
    iD = i;
  }

  public void run() {
    while (notCut) {  // as long as the customer is not cut
      try {
      accessSeats.acquire();  //tries to get access to the chairs
      if (numberOfFreeSeats > 0) {  //if there are any free seats
        System.out.println("Customer " + this.iD + " just sat down.");
        numberOfFreeSeats--;  //sitting down on a chair
        customers.release();  //notify the barber that there is a customer
        accessSeats.release();  // don't need to lock the chairs anymore
        try {
       barber.acquire();  // now it's this customers turn but we have to wait if the barber
is busy
        notCut = false;  // this customer will now leave after the procedure
        this.get_haircut();  //cutting...
        } catch (InterruptedException ex) {}
      }
      else  {  // there are no free seats
        System.out.println("There are no free seats. Customer " + this.iD + " has left the
barbershop.");
        accessSeats.release();  //release the lock on the seats
        notCut=false; // the customer will leave since there are no spots in the queue left.
       }
      }
       catch (InterruptedException ex) {}
    }
  }

  /* this method will simulate getting a hair-cut */

  public void get_haircut(){
    System.out.println("Customer " + this.iD + " is getting his hair cut");
    try {
    sleep(5050);
    } catch (InterruptedException ex) {}
  }

}


/* THE BARBER THREAD */


class Barber extends Thread {

  public Barber() {}
```

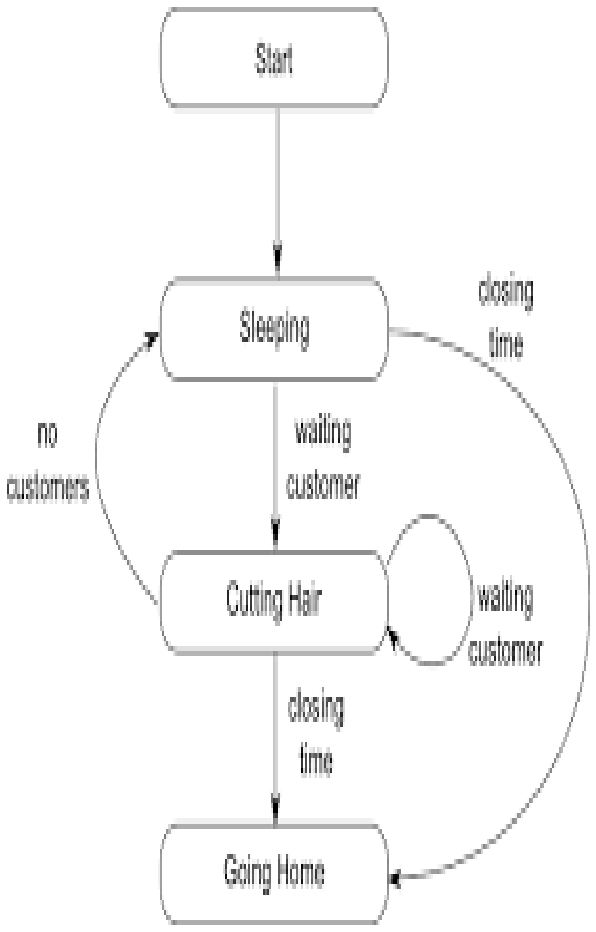
```
  public void run() {
    while(true) {  // runs in an infinite loop
      try {
      customers.acquire(); // tries to acquire a customer - if none is available he goes to
sleep
      accessSeats.release(); // at this time he has been awaken -> want to modify the number
of available seats
        numberOfFreeSeats++; // one chair gets free
      barber.release();  // the barber is ready to cut
      accessSeats.release(); // we don't need the lock on the chairs anymore
      this.cutHair();  //cutting...
    } catch (InterruptedException ex) {}
    }
  }

    /* this method will simulate cutting hair */

  public void cutHair(){
    System.out.println("The barber is cutting hair");
    try {
      sleep(5000);
    } catch (InterruptedException ex){ }
  }
}

  /* main method */

  public static void SleepingBarber_Driver() {
    System.out.println("Executing in Sleeping Barber");
    SleepingBarber barberShop = new SleepingBarber();  //Creates a new barbershop
    barberShop.start();  // Let the simulation begin
  }

  public void run(){
   Barber giovanni = new Barber();  //Giovanni is the best barber ever
   giovanni.start();  //Ready for another day of work

   /* This method will create new customers for a while */

   for (int i=1; i<16; i++) {
     Customer aCustomer = new Customer(i);
     aCustomer.start();
     try {
       sleep(2000);
     } catch(InterruptedException ex) {};
   }
  }
}
```

Additional Files: None.

Results:

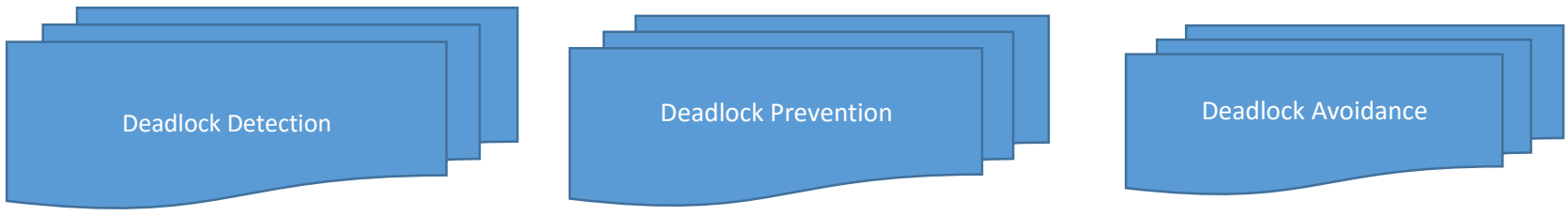
```
4
Executing in Sleeping Barber
Customer 1 just sat down.
The barber is cutting hair
Customer 1 is getting his hair cut
Customer 2 just sat down.
Customer 3 just sat down.
```

Flowchart and pseudo code




```
# The first two are mutexes (only 0 or 1 possible)
Semaphore barberReady = 0
Semaphore accessWRSeats = 1     # Хэрвээ 1 бол хүлээлгийн өрөөний сул суудлын тоо өсч буурна.
Semaphore custReady = 0         # Хүлээлгийн өрөөнд суугаад үсээ засуулахаа хүлээж буй үйлчлүүлэгчдийн тоо
int numberOfFreeWRSeats = N     # Хүлээлгийн өрөөн дэх нийт суудлын тоо

def Barber():
  while true:                   # хязгааргүй давталтаар давтана.
    wait(custReady)             # Үйлчлүүлэгчийнхээ үсийг засах гэж хүлээлгийн өрөө рүү явна , байхгүй бол унтана
    wait(accessWRSeats)         # Сэрээд хүлээлгийн өрөөний сандал руу явна, эсвэл унтана.
    numberOfFreeWRSeats += 1    # Хүлээлгийн өрөөний нэг сандал сулрана.
    signal(barberReady)         # Үс засахад бэлэн болно.
    signal(accessWRSeats)       # Дахин түгжрэл үүсгэх хэрэггүй болно.
    # (Үйлчлүүлэгчийн үсийг засна.)

def Customer():
  while true:                   # Олон үйлчлүүлэгч үүсгэхийн тулд хязгааргүй давталтаар давтана.
    wait(accessWRSeats)         # Хүлээлгийн өрөөний суудалд суухыг оролдоно.
    if numberOfFreeWRSeats > 0: # Хэрвээ сул суудал байвал:
      numberOfFreeWRSeats -= 1  #   Сандал дээр сууна
      signal(custReady)         #   Үсчин нь үйлчлүүлэгчгүй байгаа эсэхийг шалгана.
      signal(accessWRSeats)     #   Түгжрэл үүсгэхгүй болно
      wait(barberReady)         #   Үсчинийг бэлэн болохыг хүлээнэ.
      # (Үсээ засуулна.)
    else:                       # Эсрэгээрээ азгүйтэж, тэнд нь ямар ч суудал байхгүй бол
      signal(accessWRSeats)     #   Суудалд түгжрэл үүссэн бол түгжрэлээс гаргахаа мартваа
      # (Үсээ засуулахгүйгээр гарч одно.)
```

DEADLOCK



Author: ***Ahmed Sha'aban***
Date: ***1/5/2016***
Version: ***2.0***
Project ID: ***deadlock module***
CS Class: ***sec 1 OS subject 3_computer***
Programming Language: ***java***
OS/Hardware dependencies: ***None***

Problem Description: ***we have 3 problem's in this module :***

1- Deadlock prevention : when multiple processes must acquire more than one shared resource. If two or more concurrent processes obtain multiple resources indiscriminately, a situation can occur where each process has a resource needed by another process. As a result, none of the processes can obtain all the resources it needs, so all processes are blocked from further execution. This situation is called a deadlock. A deadlock prevention algorithm organizes resource usage by each process to ensure that at least one process is always able to get all the resources it needs

2- Deadlock ***detection : under deadlock detection, deadlocks are allowed to occur. Then the state of the system is examined to detect that a deadlock has occurred and subsequently it is corrected. An algorithm is employed that tracks resource allocation and process states, it rolls back and restarts one or more of the processes in order to remove the detected deadlock. Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler of the operating system***

3- Deadlock ***avoidance :*** Deadlock can be avoided if certain information about processes are available to the operating system before allocation of resources, such as which resources a process will consume in its lifetime. For every resource request, the system sees whether granting the request will mean that the system will enter an *unsafe* state, meaning a state that could result in deadlock. The system then only grants requests that will lead to *safe* states

.

How to build the program: ***by using netbeans program.java***

Program Source:

 # 2- Deadlock prevention ***:***

```
package osmain.DeadlockPrevention;

public class DeadlockPrevention {

    public static void DeadlockPrevention_Driver() {
        System.out.println("Executing in Deadlock Prevention");
        DeadlockPrevention test = new DeadlockPrevention();

        final A a = test.new A();
        final B b = test.new B();

           // Thread-1
Runnable block1 = new Runnable() {
    public void run() {
        synchronized (b) {
            try {
                // Adding delay so that both threads can start trying to
                // lock resources
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
```

```
            // Thread-1 have A but need B also
            synchronized (a) {
                System.out.println("Allocation success of 1");
            }
        }
    }
};

       // Thread-2
Runnable block2 = new Runnable() {
    public void run() {
        synchronized (b) {
            // Thread-2 have B but need A also
            synchronized (a) {
                System.out.println("Allocation Success of 2");
            }
        }
    }
};

        new Thread(block1).start();
        new Thread(block2).start();
    }

    // Resource A
    private class A {
        private int i = 10;

        public int getI() {
            return i;
        }

        public void setI(int i) {
            this.i = i;
        }
    }

    // Resource B
    private class B {
        private int i = 20;

        public int getI() {
            return i;
        }

        public void setI(int i) {
            this.i = i;
        }
    }
}
```

Additional Files: None.

References: None.

Program Source:

 # 2- Deadlock *detection :*

```
package osmain.DeadlockDetection;

import java.lang.management.ManagementFactory;
import java.lang.management.ThreadInfo;
import java.lang.management.ThreadMXBean;
import java.util.Map;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;


public class DeadlockDetection {


    public static void DeadlockDetection_Driver(){
       System.out.println("Executing in Dead Lock Detection");
       DeadlockDetector deadlockDetector = new DeadlockDetector(new
DeadlockConsoleHandler(), 5, TimeUnit.SECONDS);
       deadlockDetector.start();

final Object lock1 = new Object();
final Object lock2 = new Object();

Thread thread1 = new Thread(new Runnable() {
  @Override
```

```
  public void run() {
    synchronized (lock1) {
      System.out.println("Thread1 acquired lock1");
      try {
        TimeUnit.MILLISECONDS.sleep(500);
      } catch (InterruptedException ignore) {
      }
      synchronized (lock2) {
        System.out.println("Thread1 acquired lock2");
      }
    }
  }

});
thread1.start();

Thread thread2 = new Thread(new Runnable() {
  @Override
  public void run() {
    synchronized (lock2) {
      System.out.println("Thread2 acquired lock2");
      synchronized (lock1) {
        System.out.println("Thread2 acquired lock1");
      }
    }
  }
});
thread2.start();
    }

}

 interface DeadlockHandler {
  void handleDeadlock(final ThreadInfo[] deadlockedThreads);
}

 class DeadlockDetector {

  private final DeadlockHandler deadlockHandler;
  private final long period;
  private final TimeUnit unit;
  private final ThreadMXBean mbean = ManagementFactory.getThreadMXBean();
  private final ScheduledExecutorService scheduler =
  Executors.newScheduledThreadPool(1);

  final Runnable deadlockCheck = new Runnable() {
    @Override
    public void run() {
      long[] deadlockedThreadIds = DeadlockDetector.this.mbean.findDeadlockedThreads();

      if (deadlockedThreadIds != null) {
        ThreadInfo[] threadInfos =
        DeadlockDetector.this.mbean.getThreadInfo(deadlockedThreadIds);

        DeadlockDetector.this.deadlockHandler.handleDeadlock(threadInfos);
      }
    }
  };

  public DeadlockDetector(final DeadlockHandler deadlockHandler,
    final long period, final TimeUnit unit) {
    this.deadlockHandler = deadlockHandler;
    this.period = period;
    this.unit = unit;
  }

  public void start() {
    this.scheduler.scheduleAtFixedRate(
    this.deadlockCheck, this.period, this.period, this.unit);
  }
}

 class DeadlockConsoleHandler implements DeadlockHandler {

  @Override
  public void handleDeadlock(final ThreadInfo[] deadlockedThreads) {
    if (deadlockedThreads != null) {
      System.err.println("Deadlock detected!");

      Map<Thread, StackTraceElement[]> stackTraceMap = Thread.getAllStackTraces();
      for (ThreadInfo threadInfo : deadlockedThreads) {

        if (threadInfo != null) {

          for (Thread thread : Thread.getAllStackTraces().keySet()) {
```

```
            if (thread.getId() == threadInfo.getThreadId()) {
              System.err.println(threadInfo.toString().trim());

              for (StackTraceElement ste : thread.getStackTrace()) {
                  System.err.println("\t" + ste.toString().trim());
              }
            }
          }
        }
      }
    }
  }
}
```

Additional Files: None.

Results:

Program Source:

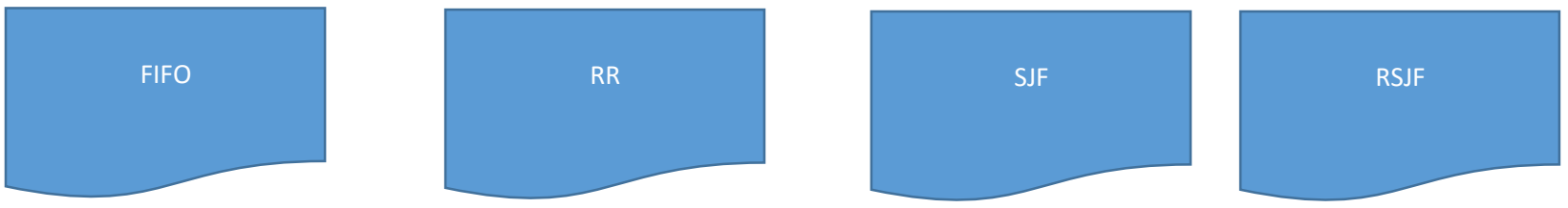
 # 2- Deadlock **avoidance:**

```
package osmain.DeadLockAvoidance;

import java.util.Scanner;


//using Banker's Algrithm

public class DeadlockAvoidance{
    private int need[][],allocate[][],max[][],avail[][],np,nr;

    private void input(){
     Scanner sc=new Scanner(System.in);
     System.out.print("Enter no. of processes and resources : ");
     np=sc.nextInt();  //no. of process
     nr=sc.nextInt();  //no. of resources
     need=new int[np][nr];  //initializing arrays
     max=new int[np][nr];
     allocate=new int[np][nr];
     avail=new int[1][nr];

     System.out.println("Enter allocation matrix -->");
     for(int i=0;i<np;i++)
          for(int j=0;j<nr;j++)
         allocate[i][j]=sc.nextInt();  //allocation matrix

     System.out.println("Enter max matrix -->");
     for(int i=0;i<np;i++)
          for(int j=0;j<nr;j++)
         max[i][j]=sc.nextInt();  //max matrix

        System.out.println("Enter available matrix -->");
        for(int j=0;j<nr;j++)
         avail[0][j]=sc.nextInt();  //available matrix

        sc.close();
     }

    private int[][] calc_need(){
       for(int i=0;i<np;i++)
         for(int j=0;j<nr;j++)  //calculating need matrix
          need[i][j]=max[i][j]-allocate[i][j];

       return need;
    }

    private boolean check(int i){
       //checking if all resources for ith process can be allocated
       for(int j=0;j<nr;j++)
       if(avail[0][j]<need[i][j])
          return false;

    return true;
    }

    public void isSafe(){
       input();
       calc_need();
       boolean done[]=new boolean[np];
       int j=0;

       while(j<np){  //until all process allocated
```

```
        boolean allocated=false;
        for(int i=0;i<np;i++)
         if(!done[i] && check(i)){  //trying to allocate
             for(int k=0;k<nr;k++)
             avail[0][k]=avail[0][k]-need[i][k]+max[i][k];
          System.out.println("Allocated process : "+i);
          allocated=done[i]=true;
                j++;
              }
           if(!allocated) break;  //if no allocation
        }
        if(j==np)  //if all processes are allocated
         System.out.println("\nSafely allocated");
        else
         System.out.println("All proceess cant be allocated safely");
    }

    public static void DeadLockAvoidance_Driver(){
        System.out.println("Execuint in Dead Lock Avoidance");
        new DeadlockAvoidance().isSafe();
    }
}
```

Additional Files: None.

Scheduling



Author: ***Mahmoud Yahia***
Date: ***1/5/2016***
Version: ***3.0***
Project ID: ***Scheduling module***
CS Class: ***sec 4 OS subject 3_computer***
Programming Language: ***java***
OS/Hardware dependencies: ***None***

Problem Description: ***we have 4 problem's in this module :***

1- FiFo : First In, First Out (FIFO), also known as First Come, First Served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue. This is commonly used for a task queue

2- ROUND ROBIN (RR) : is one of the algorithms employed by process and network schedulers in computing As the term is generally used, time slices are assigned to each process in equal portions and in circular order handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. It is an Operating System concept

3- SHORTEST JOB FIRST (SJF) :

also known as shortest job first (SJF) or shortest process next (SPN), is a scheduling policy that selects the waiting process with the smallest execution time to execute next.[1] SJN is a non-preemptive algorithm. Shortest remaining time is a preemptive variant of SJN.

Shortest job next is advantageous because of its simplicity and because it minimizes the average amount of time each process has to wait until its execution is complete. However, it has the potential for process starvation for processes which will require a long time to complete if short processes are continually added

4- REMAINING SHORTEST JOB FIRST (RSJF) : also known as shortest remaining time first (SRTF), is a scheduling method that is preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time Shortest remaining time is advantageous because short processes are handled very quickly. The system also requires very little overhead since it only makes a decision when a process completes or a new process is added, and when a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute e shortest job first, it has the potential for process starvation; long processes may be held off indefinitely if short processes are continually added. This threat can be minimal when process times follow a heavy-tailed distribution

.

How to build the program: by using netbeans program.java

Program Source:

_____

```
package osmain.fifo;


import java.util.Scanner;

public class FIFO {


    public static void Fifo_Driver() {
        System.out.println("Executing in FIFO");
        LinkedList list = new LinkedList();
        Scanner scan = new Scanner(System.in);

        String processName;
        int processNumber;
        int counter = 0;
        int time = 0;
        int timeOld = 0;
        int timeTemp = 0;

        System.out.println("please, enter the number of the processes");
        processNumber = scan.nextInt();

        for(int i = 0; i < processNumber; i++){
             processName = "P" + String.valueOf(i);
             System.out.println("please, enter the burst of " + processName);

             list.insert(scan.nextInt(), processName);
         }

        list.printList();

            System.out.println("\nApplying First-in First-out Algorithm");

            Link temp;

            while(true){
                if(counter == processNumber){
                counter = 0;
                }

                temp = list.iteration(counter);

                if(temp.burst == 0){
                    if(list.conditionBreak() == processNumber)
                        break;

                    continue;
                }

                timeTemp = temp.burst;
                list.iterateAndEdit(counter, 0);
                timeOld = time;
                time += timeTemp;

                System.out.println("process: " + temp.ProcessName + " executed from " +
timeOld + " to " + time);
                counter += 1;
            }

        System.out.println("\nDONE!");
    }
}
```

```
2
Enter 1 for FIFO
Enter 2 for SJF
Enter 3 for Round Robin
1
Executing in FIFO
please, enter the number of the processes
5
please, enter the burst of P0
5
please, enter the burst of P1
14
please, enter the burst of P2
7
please, enter the burst of P3
8
please, enter the burst of P4
6
List: {P0, 5} {P1, 14} {P2, 7} {P3, 8} {P4, 6}

Applying First-in First-out Algorithm
process: P0 executed from 0 to 5
process: P1 executed from 5 to 19
process: P2 executed from 19 to 26
process: P3 executed from 26 to 34
process: P4 executed from 34 to 40

DONE!
```

Results:

References: ***None.***

Program Source:
_____:

```
package osmain.RoundRobin;

import osmain.RoundRobin.LinkedList;
import osmain.RoundRobin.Link;
import java.util.Scanner;

public class RoundRobin {



    //DRIVER CLASS OF ROUND ROBIN ALGORITHM
    public static void RoundRobin_Driver() {
        System.out.println("Executing in RoundRobin");
         LinkedList list = new LinkedList();
         Scanner scan = new Scanner(System.in);

         String processName;
         int processNumber;
         int timeSlot;
         int counter = 0;
         int time = 0;
         int timeOld = 0;
         int timeTemp = 0;
         int checker;

         System.out.println("please, enter the number of process");
         processNumber = scan.nextInt();
         checker = processNumber;

         for(int i = 0; i < processNumber; i++){
             processName = "P" + String.valueOf(i);
             System.out.println("please, enter the burst of " + processName);

             list.insert(scan.nextInt(), processName);
         }

           list.printList();


            System.out.println("please, enter the time Slot");
            timeSlot = scan.nextInt();

            System.out.println("time slot value is: " + timeSlot);

            System.out.println("\nApplying Round Robin Algorithm");

            Link temp;

            while(true){
```

```
                    if(counter == processNumber){
                    counter = 0;
                    }

                    temp = list.iteration(counter);

                    if(temp.burst == 0){
                        counter += 1;
                        checker -= 1;

                        if(list.conditionBreak() == processNumber)
                            break;

                        continue;
                    }

                    if(temp.burst > timeSlot){
                        list.iterateAndEdit(counter, list.iteration(counter).burst - timeSlot);
                        //list.iteration(counter).burst = list.iteration(counter).burst -
timeSlot;
                        timeOld = time;
                        time += timeSlot;
                    }

                    else if(temp.burst < timeSlot){
                        timeTemp = temp.burst;
                        list.iterateAndEdit(counter, 0);
                        timeOld = time;
                        time += timeTemp;
                    }

                    else if(temp.burst == timeSlot){
                        list.iterateAndEdit(counter, 0);
                        timeOld = time;
                        time += timeSlot;
                    }

                    System.out.println("process: " + temp.ProcessName + " executed from " +
timeOld + " to " + time);
                    counter += 1;
                }

            System.out.println("\nDONE!");
        }
}
```

Additional Files: None.

References: None.

Program Source:

```
* SHORTEST JOB FIRST (SJF) :
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package osmain.SJF;

import osmain.SJF.LinkedList;
import osmain.SJF.Link;
import java.util.Scanner;

/**
 *
 * @author Mahmoud
 */
public class SJF {

    /**
     * @param args the command line arguments
     */
    public static void SJF_Driver() {
        System.out.println("Executing in SJF");
        LinkedList list = new LinkedList();
        Scanner scan = new Scanner(System.in);

        String processName;
        int processNumber;
        int time = 0;
        int timeOld = 0;
        int timeTemp = 0;
```

```
int minIndex = 0;

System.out.println("please, enter the number of the processes");
processNumber = scan.nextInt();

for(int i = 0; i < processNumber; i++){
     processName = "P" + String.valueOf(i);
     System.out.println("please, enter the burst of " + processName);

     list.insert(scan.nextInt(), processName);
 }

list.printList();

    System.out.println("\nApplying Shortest-Job First Algorithm");

    Link temp;

     while(true){
        minIndex = list.iteration(processNumber);
        temp = list.getNode(minIndex);

        if(temp.burst == 0){
                break;
        }

        timeTemp = temp.burst;
        list.iterateAndEdit(minIndex, 0);
        timeOld = time;
        time += timeTemp;

        System.out.println("process: " + temp.ProcessName + " executed from " +
timeOld + " to " + time);
             }

        System.out.println("\nDONE!");
    }
}
```

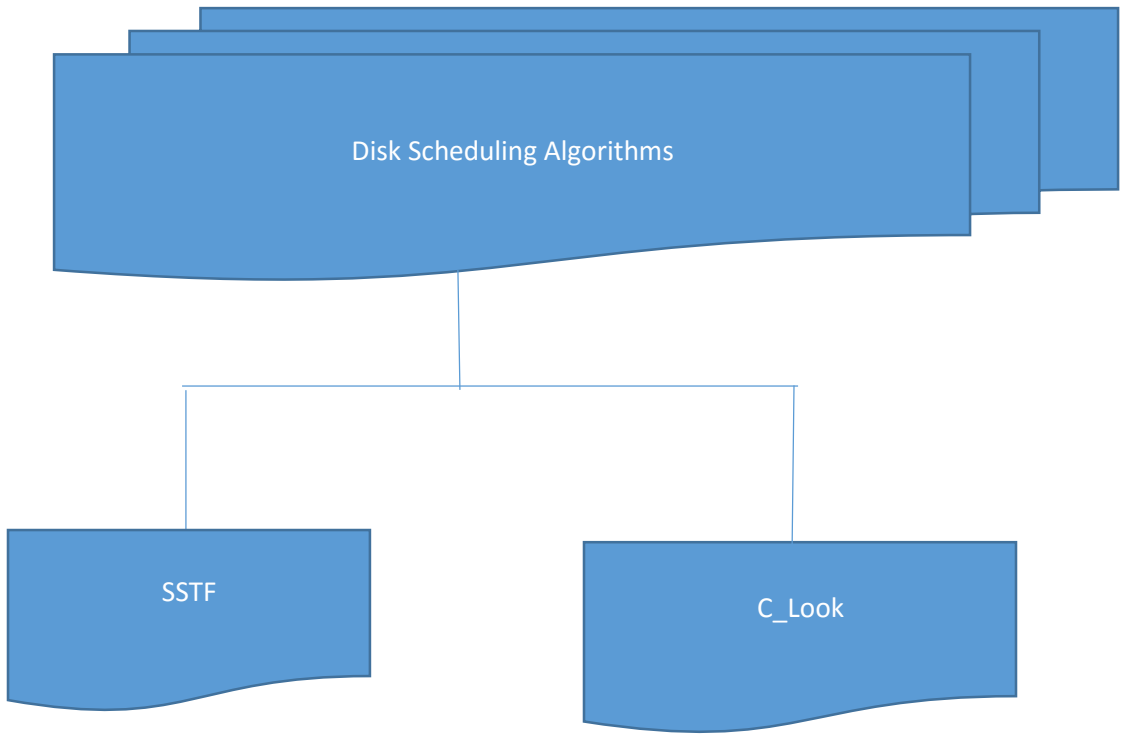
Additional Files: None.

Results:

```
2
Executing in SJF
please, enter the number of the processes
3
please, enter the burst of P0
10
please, enter the burst of P1
14
please, enter the burst of P2
12
List: {P0, 10} {P1, 14} {P2, 12}

Applying Shortest-Job First Algorithm
process: P0 executed from 0 to 10
process: P2 executed from 10 to 22
process: P1 executed from 22 to 36

DONE!
```

References: None.

FILE SYSTEM



Author: ***Reham Khalf Abdelhamiid***
Mohamed Hassan Saied
Date: ***8/5/2016***
Version: ***4.0***
Project ID: ***File system module***
CS Class: ***sec 4 OS subject 3_computer***
Programming Language: ***java***
OS/Hardware dependencies: ***None***

Problem Description: ***we have 1 problem in this module consist of 2 algorithm:***

1- sstf: This is a direct improvement upon a first-come first-served (FCFS) algorithm. The drive maintains an incoming buffer of requests, and tied with each request is a cylinder number of the request. Lower cylinder numbers indicate that the cylinder is closer to the spindle, while higher numbers indicate the cylinder is farther away. The shortest seek first algorithm determines which request is closest to the current position of the head, and then services that request next

---

2- C-look: The **LOOK** algorithm is the same as the **SCAN** algorithm in that it also honors requests on both sweep direction of the disk head, however, this algorithm "Looks" ahead to see if there are any requests pending in the direction of head movement. If no requests are pending in the direction of head movement, then the disk head traversal will be reversed to the opposite direction and requests on the other direction can be served. In LOOK scheduling, the arm goes only as far as final requests in each direction and then reverses direction without going all the way to the end**.**

One variant of LOOK is C-LOOK. (Circular LOOK) It is an effort to remove the bias in LOOK for track clusters at the edges of the platter. C-LOOK basically only scans in one direction. Either you sweep from the inside out, or the outside in. When you reach the end, you just swing the head all the way back to the beginning. This actually takes advantage of the fact that many drives can move the read/write head at high speeds if it's moving across a large number of tracks

---

.

How to build the program: by using netbeans program.java

Program Source:

SSTF ::

```
package osmain.DiskSchSSTF;
import java.util.Scanner;
import java.util.ArrayList;

public class Disk_SSTF {

    public static void SSTF_Driver() {
        System.out.println("Executing in SSTF Driver");
```

```
Scanner input = new Scanner(System.in);         // read cylinder requests
int headPos;        // starting head position

System.out.println("Enter Head position");
headPos = input.nextInt();

int index;
ArrayList<Integer> requests = new ArrayList<>();
System.out.println("Enter no of values");
index = input.nextInt();
for (int i = 0; i < index; i++) {
    System.out.println("Enter value" + (i + 1));

    requests.add(input.nextInt());

}
int scheduleCost = 0;
System.out.print(headPos);
while (requests.size() > 0) // SSTF implementation
{
    int closestPos = 0;
    for (int j = 1; j < requests.size(); j++) // find next closest position
    {
        if (Math.abs(headPos - requests.get(j))
                < Math.abs(headPos - requests.get(closestPos))) {
            closestPos = j;                         // closest position index
        }
    }
            }
            scheduleCost += Math.abs(headPos - requests.get(closestPos));
            headPos = requests.get(closestPos); // reposition the head

            System.out.print(" - " + headPos);
            requests.remove(closestPos);              // remove processed request
        }

        System.out.println("\n" + scheduleCost + " cylinder moves");
    }
}

class SSTF {

    int[] queue;
    int initialCylinder;

    public SSTF(int[] queue, int initialCylinder) {
        this.queue = queue;
        this.initialCylinder = initialCylinder;
    }

    public int serviceRequests() {
        int headMovement = 0;
        int prev = initialCylinder;
        int[] rpath = path();
        for (int i = 0; i < rpath.length; i++) {
            headMovement += Math.abs(rpath[i] - prev);
            prev = rpath[i];
        }
        return headMovement;
    }

    public int[] path() {
        int[] resultPath = new int[queue.length];
        int now = initialCylinder;
        int[] requests = new int[queue.length];
        for (int i = 0; i < queue.length; i++) {
            requests[i] = queue[i];
        }
        for (int i = 0; i < resultPath.length; i++) {
            int closest = closest(now, requests);
            resultPath[i] = closest;
            now = closest;
        }
        return resultPath;
    }

    int closest(int k, int[] requests) {
        int min = 5000000;
        int minPos = -1;
        for (int i = 0; i < requests.length; i++) {
            if (requests[i] == -1) {
                continue;
            } else if (Math.abs(k - queue[i]) < min) {
                minPos = i;
                min = Math.abs(k - queue[i++]);
            }
        }
```

```
        int nearestCylinder = requests[minPos];
        requests[minPos] = -1;
        return nearestCylinder;
    }

    public void println() {
        System.out.println("SSTF head movement = " + serviceRequests());

        System.out.print("SSTF Path = ");
        for (int i : path()) {
            System.out.print(i + " ");
        }
        System.out.println("");
    }
}
```

Output

```
Executing in SSTF Driver
pleas, enter the number of elements
5
please, enter the elements in the array
12
14
15
12
13
please, enter the initial head position
15
-----------------------------
15
15
14
13
12
12
```

C-look :

```
package clook;

import java.util.Scanner;

public class CLook {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int temp, size, minIndex, index = 0, counter = 0, initialHead;

        System.out.println("enter the size of the array");
        size = scan.nextInt();
        int[] array = new int[size];

        System.out.println("please, enter the elments of the array");
        for(int index1 = 0; index1 < size; index1++)
            array[index1] = scan.nextInt();

        System.out.println("please, enter the initial head");
        initialHead = scan.nextInt();

        System.out.println("the array after sorting it");

        for(int loc = 0; loc < size - 1; loc++)
        {
            minIndex = loc;
            for(int i = loc + 1; i < size; i++)
            {
                if(array[minIndex] > array[i])
                    minIndex = i;
            }

            temp = array[minIndex];
            array[minIndex] = array[loc];
            array[loc] = temp;
        }

        /*for(int index = 0; index < size; index++)
            System.out.println(array[index]);*/

        for(int i = 0; i < size; i++)
        {
            if(array[i] >= initialHead)
```

```
        {
            index = i;
            break;
        }
    }

    while(true)
    {
        System.out.println("----------------------");

        if(array[counter + index] == -1)
        {
            System.out.println("loop terminated");
            break;
        }

        System.out.println(array[counter + index]);

        array[counter + index] = -1;
        counter += 1;

        if((counter + index) == size)
        {
            System.out.println("returning to the beginning");
            index = 0;
            counter = 0;
        }
    }
}
```

Additional Files: None.

Results:

```
2
Executing in CLook File system
enter the size of the array
3
please, enter the elments of the array
25
14
18
please, enter the initial head
18
the array after sorting it
----------------------
18
----------------------
25
returning to the beginning
----------------------
14
----------------------
loop terminated
```

References: None.

PART TWO OF DOCUMENTATION:

Memory Management

Author: *Mohamed Saleh Hassanien*
Date: *11/5/2016*
Version: *1.0*
Project ID: *memory management module*
CS Class: *SEC3 3_computer*
Programming Language: *Java*
OS/Hardware dependencies: *None*

Problem Description:

the part is divided into 4 pieces:
memory Segmentation: take inputs calculate segments and print the segmentation table

best Fit : take inputs and calculate the fragmentation
Least recently Used replacement algorithm
First in first out replacement algorithm

Algorithms

Segmentation algorithm
1. *Make and initiate a linked list*
2. *Take inputs*

Best fit algorithm
1. *initiate arrays*
2. *take inputs*
3. *calculate the fragments*
4. *print output*

LRU replacement algorithm
1. *Take inputs*
2. *Initialize Frame and Recent array to -1*
3. *Check for Page Hit. If yes then Goto step 7 else Goto step 4*
4. *Create array of the coming paging as series sequence.*
5. *Page to be replaced is the first page from current array.*
6. *Replace this first page from frame array with current page.*
7. *print contents of frame array.*

FIFO replacement Algorithm
1. *Take inputs*
2. *Initialize Frame and Recent array to -1*
3. *Check for Page Hit. If yes then Goto step 7 else Goto step 4*
4. *Create array of least recently used pages and store it in recent.*
5. *Page to be replaced is the last page from recent array.*
6. *Replace this last page from recent array with current page.*
7. *print contents of frame array.*

How to build the program: *netBeans program*

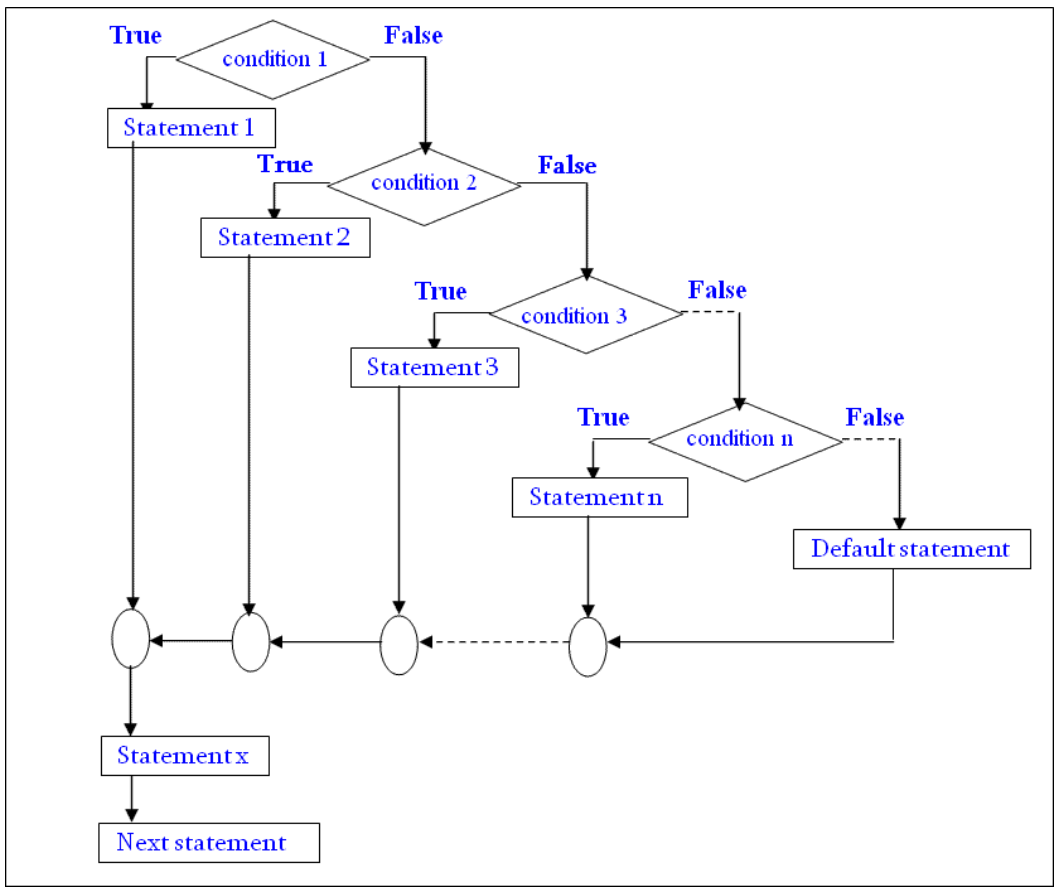
Program Source:

Segmentation code

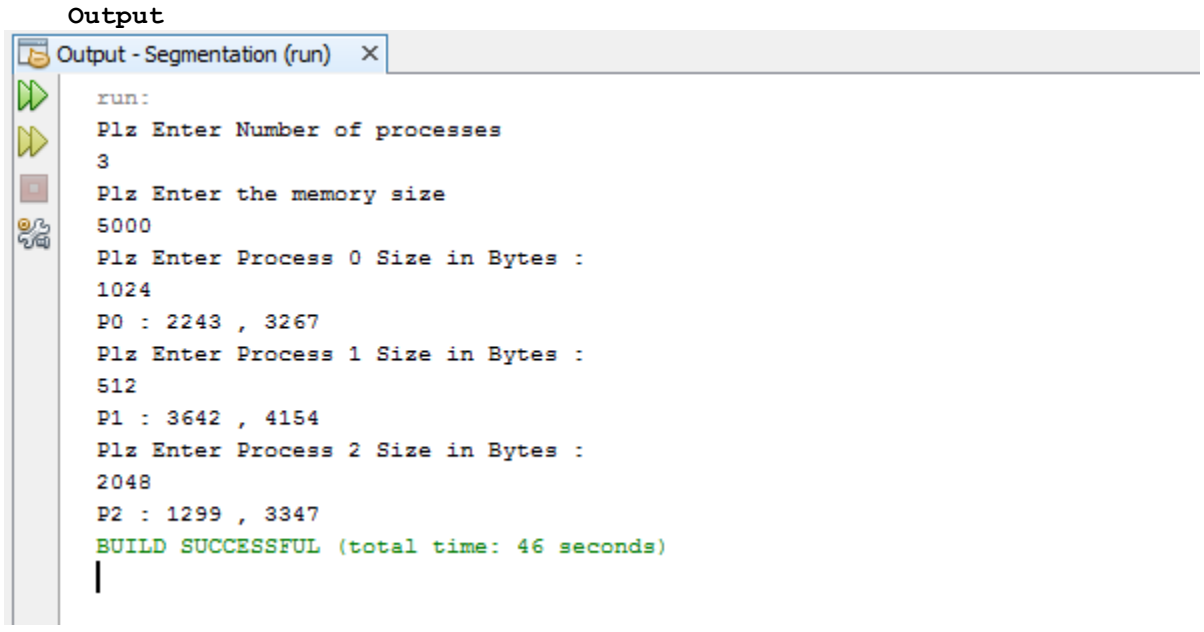
```
import java.util.LinkedList ;
import java.util.Random;
import java.util.Scanner;

    public static int GetBaseAddress (LinkedList <Segment> M , int Psize ,int Msize)
    {
        boolean flag ;
        int BaseAddress = -1;
        Random r = new Random ();
        do
        {
            flag = true;
```

```
        BaseAddress = r.nextInt(Msize-Psize);

            for (int i = 0; i < M.size(); i++) {
                Segment s = M.get(i);
                if (BaseAddress >= s.BaseAddress && BaseAddress <= s.LimitAddress ) {
                    //if (BaseAddress+Psize < s.BaseAddress || BaseAddress+Psize >s.LimitAddress) {
                    flag = false;
                   // }


                }
            }

        }
        while (flag == false);
    return BaseAddress ;
    }
    public static boolean CheckLimitAddress (LinkedList <Segment> M , int Psize ,int Msize , int
BaseAddress)
    {
        boolean flag ;
        int LimitAddress = BaseAddress+Psize;



            flag = true;


            for (int i = 0; i < M.size(); i++)
            {
                Segment s = M.get(i);
                if (
                        (LimitAddress==s.BaseAddress)
                        ||(LimitAddress >= s.BaseAddress &&LimitAddress <= s.LimitAddress)
                        ||(LimitAddress==s.LimitAddress) )
                {
                    flag = false;// Flag is false if the limit is invalid
                }
            }



return flag ;
}
public static void main(String[] args) {
    // TODO code application logic here
   LinkedList <Segment> Memory = new LinkedList<Segment>();

   Scanner sc = new Scanner(System.in) ;
   System.out.println("Plz Enter Number of processes");

   int Pnumber = sc.nextInt();
          System.out.println("Plz Enter the memory size");

   int memory_size=sc.nextInt();
   int available_memory=memory_size;

    for (int i = 0; i < Pnumber; i++) {
        System.out.println("Plz Enter Process "+i+" Size in Bytes :");
        int Psize = sc.nextInt();

        Segment s = new Segment();
        int temp_base_address;
        boolean flag_limit = false;

        do {

             temp_base_address = GetBaseAddress(Memory , Psize , memory_size);
              flag_limit = CheckLimitAddress(Memory, Psize, memory_size,
temp_base_address);
            if(flag_limit){ available_memory-=Psize;}



            }
            while(flag_limit==false&&available_memory>Psize);

                s.BaseAddress = temp_base_address;
                s.LimitAddress = s.BaseAddress+Psize;

                System.out.println("P"+i+" : "+ s.BaseAddress + " , "+ s.LimitAddress);

                    }


    }

}
```

Flow chart



Output



Best fit source code

```
import java.io.*;

class Bestfit

{

int fn,bn;

int file[],block[],bflag[],fragment[];

BufferedReader input=new BufferedReader(new InputStreamReader(System.in));

Bestfit() throws IOException

{

int i;

System.out.println("Enter the number of files");

fn=Integer.parseInt(input.readLine());

System.out.println("Enter the number of blocks");

bn=Integer.parseInt(input.readLine());
```

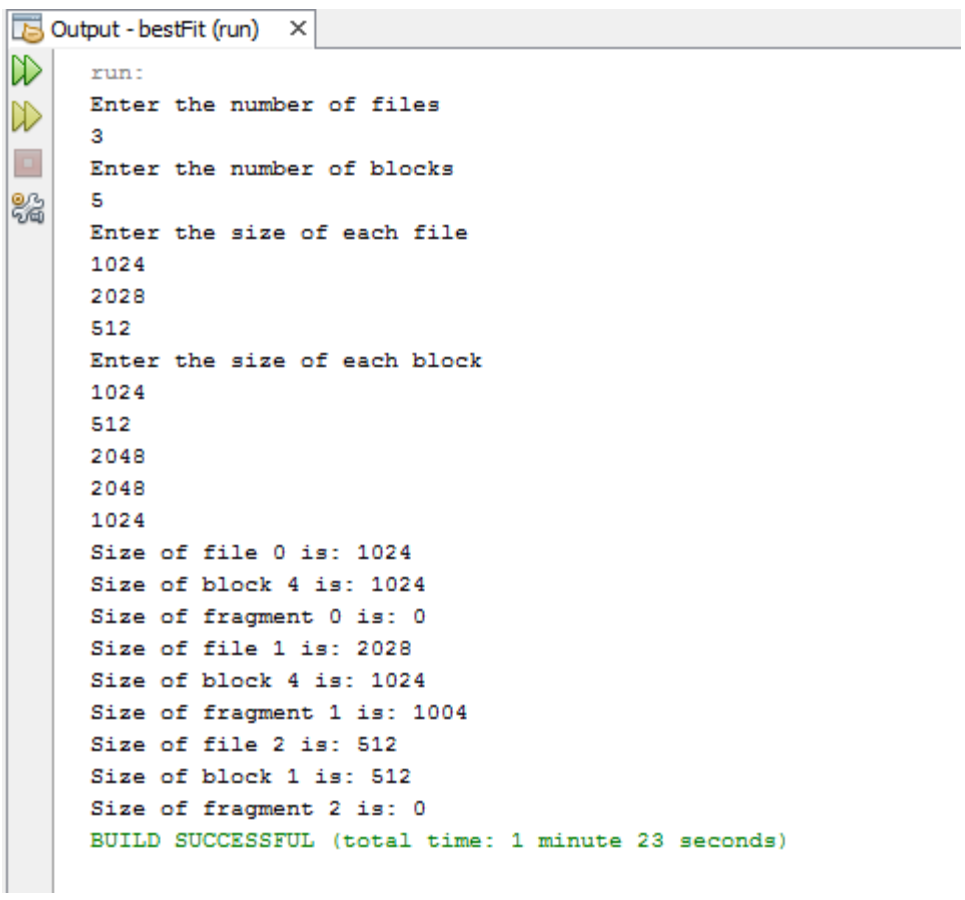
```
file=new int[fn+1];
block=new int[bn+1];
bflag=new int[bn+1];
fragment=new int[fn+1];
for(i=0;i<bn;i++)
bflag[i]=0;
}

void accept() throws IOException
{
int i;
System.out.println("Enter the size of each file");
for(i=0;i<fn;i++)
file[i]=Integer.parseInt(input.readLine());
System.out.println("Enter the size of each block");
for(i=0;i<bn;i++)
block[i]=Integer.parseInt(input.readLine());
}

void execute()
{
int i,j,min,p=-1;
for(i=0;i<fn;i++)
{
min=block[0];
for(j=0;j<bn;j++)
{
if((bflag[j]!=1)&&(file[i]<=block[j])&&(min>=block[j]))
{
min=block[j];
p=j;
}
}
}
fragment[i]=(Math.abs(file[i]-min));
bflag[j]=1;
System.out.println("Size of file " +i+" is: "+file[i]);
System.out.println("Size of block " +p+" is: "+block[p]);
System.out.println("Size of fragment " +i+" is: "+fragment[i]);
}
}

public static void main(String args[]) throws IOException
{
Bestfit bf=new Bestfit();
bf.accept();
bf.execute();
```

```
}

}
```

Best fit pseudo code:

```
    Best-Fit(request_size)
begin

node rover = head;
node min = rover;
// Assume that head is always a valid node. Also, head may be the only node in the FL, thus making:
// Head->next = tail.

do
{
diff = rover->size - request_size;
if (diff >= 0 && rover->size < min->size)
min = rover;
rover = rover->next;
} while (rover != tail);

if (min->size >= request_size)
// Block was found.
else
// Search was unsuccessful.
end
```

Output



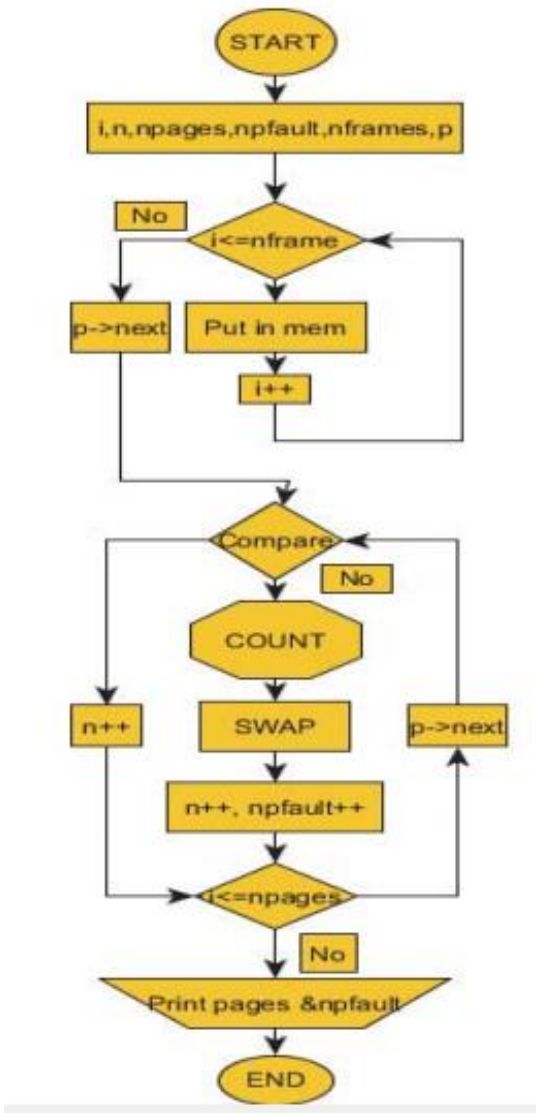
LRU source code:

```
          import java.io.*;
public class lru
{
    public static void LRU_Driver() throws Exception
    {
        System.out.println("Executing in least recently used");
        int f, p, num=0, pageHit=0, count=0, ptPage=0, pg=0;
        int pages[];
        int frame[];
        int recent[];
        boolean flag = true;
        boolean flag2 = true;

        BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter number of frames : ");
```

```
f = Integer.parseInt(br.readLine());
System.out.println("Enter number of pages : ");
p = Integer.parseInt(br.readLine());

frame = new int[f];
pages = new int[p];
recent = new int[f];

for(int i=0; i<f; i++)
{
    frame[i] = -1;
    recent[i] = -1;
}

System.out.println("Enter page number : ");
for(int i=0;i<p;i++)
    pages[i] = Integer.parseInt(br.readLine());

for(int i=0; i<p; i++)
{
    flag = true;
    int page = pages[i];

    for(int j=0; j<f; j++)
        recent[j] = -1;

    for(int j=0; j<f; j++)
        if(frame[j] == page)
        {
            flag = false;
            pageHit++;
            break;
        }
    }
    if(flag)
    {
        count = 0;
        if(i>=3)
        {
            ptPage = i-1;

            while(count < f)
            {
                Thread.sleep(1000);
                pg = pages[ptPage];
                flag2 = true;
                //for(int j=0; j<f; j++)
                //        System.out.print(recent[j]+" ");
                //System.out.println();
                for(int j=0; j<f; j++)
                {
                    if(pg == recent[j])
                    {
                        flag2 = false;
                        //System.out.println(pg+" "+recent[j]+" "+flag2+" "+i+" "+j+" "+f+"
"+ptPage);
                        break;
                    }

                }
                if(flag2)
                {
                    //System.out.println(pg+" "+recent[j]+" "+flag2+" "+i+" "+j+" "+f+"
"+ptPage);
                    recent[count] = pg;
                    count++;
                    ptPage--;
                }
                else
                    ptPage--;
            }

        /*    System.out.print("page : "+page+" recent : ");
            for(int j=0; j<f; j++)
                System.out.print(recent[j]+" ");
            System.out.println();
        */

            int replace = recent.length - 1;
            int pg_to_replace = recent[replace];
            int k=0;
            while(frame[k] != pg_to_replace)
                k++;

            frame[k] = page;
        }
        else
            frame[i] = page;
    }
    System.out.print("frame : ");
    for(int k=0; k<f; k++)
        System.out.print(frame[k]+" ");
    System.out.println();
}
System.out.println("No. of page hit : "+pageHit);
```

```
        }
    }
```

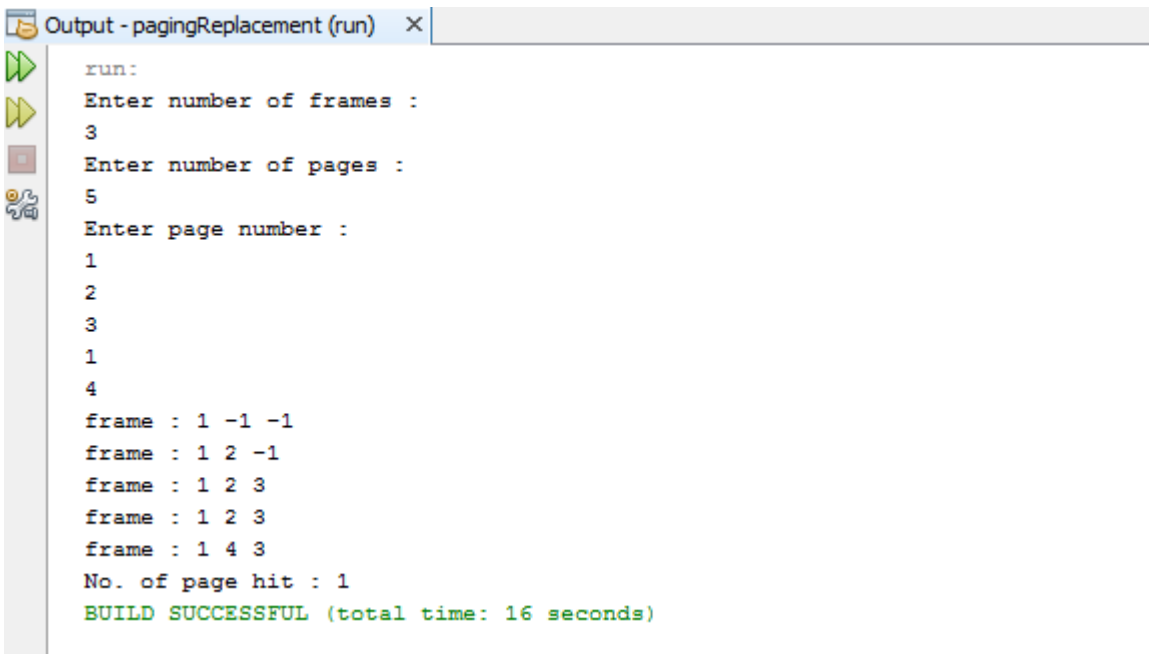
LRU flow chart



LRU pseudo code

```
Page array = -1;
Frame array = -1
Page array = coming page
If (coming page already exist )
Hit ++;
Else
    Replace the first came page with the coming page ;
```

Output

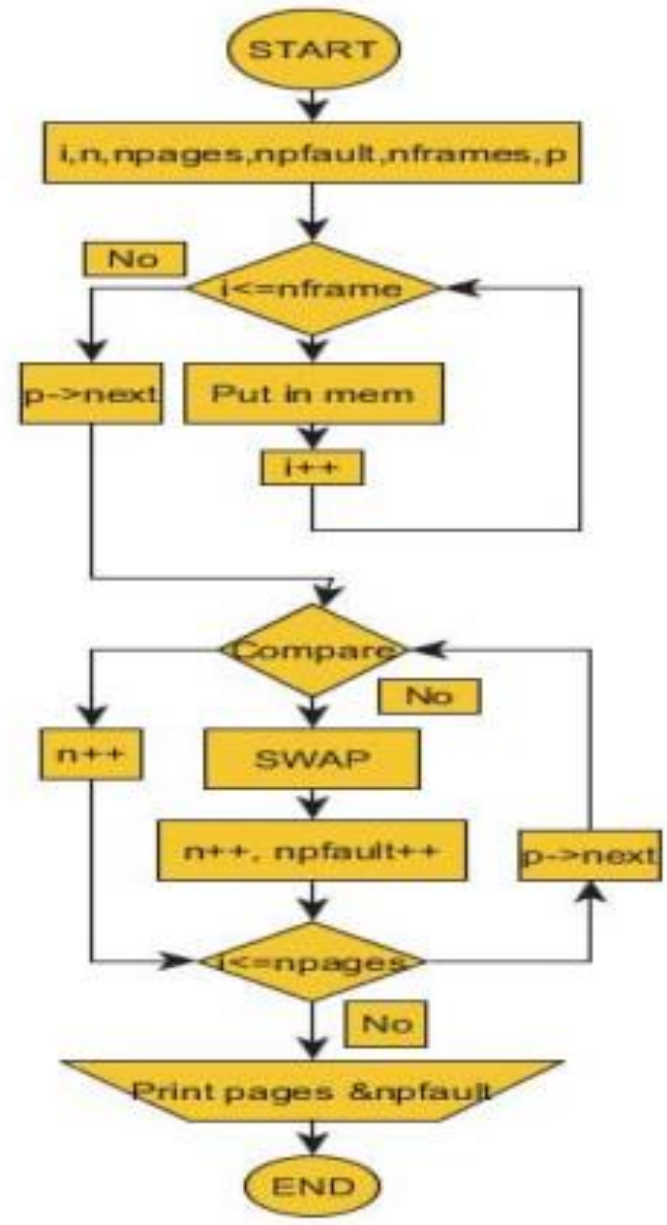


FIFO source code

```
  import java.io.*;
public class FirstInFirstOut
{
    public static void FirstComeFirstServed_Driver() throws Exception
    {
        System.out.println("Executing in first come first served dirver");
        int f,p,num=0, pageHit=0;
        int pages[];
        int frame[];
        boolean flag = true;
```

```
BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter number of frames : ");
f = Integer.parseInt(br.readLine());
System.out.println("Enter number of pages : ");
p = Integer.parseInt(br.readLine());

frame = new int[f];
pages = new int[p];

for(int i=0; i<f; i++)
{
    frame[i] = -1;
}

System.out.println("Enter page number : ");
for(int i=0;i<p;i++)
    pages[i] = Integer.parseInt(br.readLine());

for(int i=0; i<p; i++)
{
    flag = true;
    int page = pages[i];
    for(int j=0; j<f; j++)
    {
        if(frame[j] == page)
        {
            flag = false;
            pageHit++;
            break;
        }
    }
    if(num == f)
        num = 0;

    if(flag)
    {
        frame[num] = page;
        num++;
    }
    System.out.print("frame : ");
    for(int k=0; k<f; k++)
        System.out.print(frame[k]+" ");
    System.out.println();

}
System.out.println("No. of page hit : "+pageHit);
}
}
```

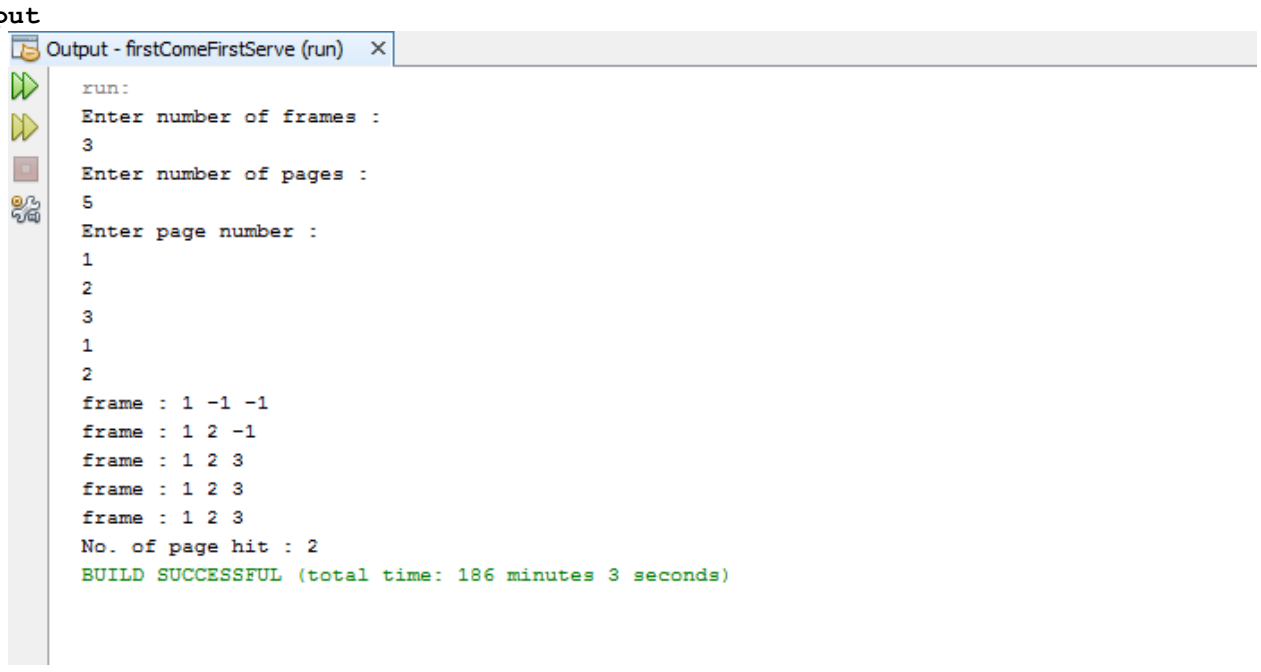
FIFO flow chart




```
 Pseudo code
      Page array = -1;
Frame array = -1
Page array = coming page
If (coming page already exist )
Hit ++;
Else
    Replace the first came page with the coming page ;
```

Output

Output



References: Brehob M. Wagner S. Torng E. Enbody R. (2004). Optimal replacement Is NP-hard for nonstandard caches.
nonstandard caches. IEEE Transactions on

Computers. Vol. 53. No. 1. pp. 73-76
Cormen T. H. Leiserson C. E. RivestR. L. Stein C. (2009). Introduction to algorithms. third edition.
MIT press Chavan A. Nayak K. R. Vora K. D. Purohit
M. D. Chawan P. M. (2011). A comparison of page Replacement algorithm. IACSIT International Journal of Engineering and Technology. Vol. 3. No. 2.
http://sourcecodesforfree.blogspot.com.eg/2013/05/5-best-fit-memory-allocation.html

Network

Author: *Mahmoud Yahia*
Date: *11/5/2016*
Version: *1.0*
Project ID: *network module*
CS Class: *SEC4 3_computer*
Programming Language: *Java*
OS/Hardware dependencies: *None*

Problem Description: *The Client that can be run both as a console or a GUI*
Algorithms :
1. initiate sockets for client and server
2 the client connect to the server
3 the server bind and listen the action if accent go to step 4 and if close go to 5
4 sending and receiving can be done
5 shutdown and close

Source code for client

```
package osmain.Chatting;
import osmain.Chatting.ClientGUI;
import osmain.Chatting.ChatMessage;
import java.net.*;
import java.io.*;
import java.util.*;

public class Client  {

	// for I/O
	private ObjectInputStream sInput;		// to read from the socket
	private ObjectOutputStream sOutput;		// to write on the socket
	private Socket socket;

	// if I use a GUI or not
	private ClientGUI cg;

	// the server, the port and the username
	private String server, username;
	private int port;


	Client(String server, int port, String username) {
		// which calls the common constructor with the GUI set to null
		this(server, port, username, null);
	}

	Client(String server, int port, String username, ClientGUI cg) {
		this.server = server;
		this.port = port;
		this.username = username;

		this.cg = cg;
	}


	public boolean start() {

		try {
			socket = new Socket(server, port);
		}

		catch(Exception ec) {
			display("Error connectiong to server:" + ec);
			return false;
		}

		String msg = "Connection accepted " + socket.getInetAddress() + ":" +
socket.getPort();
		display(msg);

		/* Creating both Data Stream */
		try
		{
			sInput  = new ObjectInputStream(socket.getInputStream());
			sOutput = new ObjectOutputStream(socket.getOutputStream());
		}
		catch (IOException eIO) {
			display("Exception creating new Input/output Streams: " + eIO);
			return false;
		}

		// creates the Thread to listen from the server
		new ListenFromServer().start();
		// Send our username to the server this is the only message that we
		// will send as a String. All other messages will be ChatMessage objects
		try
		{
```

```
                        sOutput.writeObject(username);
                }
                catch (IOException eIO) {
                        display("Exception doing login : " + eIO);
                        disconnect();
                        return false;
                }
                // success we inform the caller that it worked
                return true;
        }

        /*
         * To send a message to the console or the GUI
         */
        private void display(String msg) {
                if(cg == null)
                        System.out.println(msg);      // println in console mode
                else
                        cg.append(msg + "\n");         // append to the ClientGUI JTextArea (or
whatever)
        }

        /*
         * To send a message to the server
         */
        void sendMessage(ChatMessage msg) {
                try {
                        sOutput.writeObject(msg);
                }
                catch(IOException e) {
                        display("Exception writing to server: " + e);
                }
        }

        /*
         * When something goes wrong
         * Close the Input/Output streams and disconnect not much to do in the catch clause
         */
        private void disconnect() {
                try {
                        if(sInput != null) sInput.close();
                }
                catch(Exception e) {} // not much else I can do
                try {
                        if(sOutput != null) sOutput.close();
                }
                catch(Exception e) {} // not much else I can do
         try{
                        if(socket != null) socket.close();
                }
                catch(Exception e) {} // not much else I can do

                // inform the GUI
                if(cg != null)
                        cg.connectionFailed();

        }
        /*
         * To start the Client in console mode use one of the following command
         * > java Client
         * > java Client username
         * > java Client username portNumber
         * > java Client username portNumber serverAddress
         * at the console prompt
         * If the portNumber is not specified 1500 is used
         * If the serverAddress is not specified "localHost" is used
         * If the username is not specified "Anonymous" is used
         * > java Client
         * is equivalent to
         * > java Client Anonymous 1500 localhost
         * are eqquivalent
         *
         * In console mode, if an error occurs the program simply stops
         * when a GUI id used, the GUI is informed of the disconnection
         */
        public static void main(String[] args) {
                // default values
                int portNumber = 1500;
                String serverAddress = "localhost";
                String userName = "Anonymous";

                // depending of the number of arguments provided we fall through
                switch(args.length) {
                        // > javac Client username portNumber serverAddr
                        case 3:
                                serverAddress = args[2];
                        // > javac Client username portNumber
                        case 2:
                                try {
                                        portNumber = Integer.parseInt(args[1]);
                                }
                                catch(Exception e) {
                                        System.out.println("Invalid port number.");
                                        System.out.println("Usage is: > java Client [username]
[portNumber] [serverAddress]");
```

```
                                        return;
                                }
                        // > javac Client username
                        case 1:
                                userName = args[0];
                        // > java Client
                        case 0:
                                break;
                        // invalid number of arguments
                        default:
                                System.out.println("Usage is: > java Client [username] [portNumber]
{serverAddress]");
                        return;
                }
                // create the Client object
                Client client = new Client(serverAddress, portNumber, userName);
                // test if we can start the connection to the Server
                // if it failed nothing we can do
                if(!client.start())
                        return;

                // wait for messages from user
                Scanner scan = new Scanner(System.in);
                // loop forever for message from the user
                while(true) {
                        System.out.print("> ");
                        // read message from user
                        String msg = scan.nextLine();
                        // logout if message is LOGOUT
                        if(msg.equalsIgnoreCase("LOGOUT")) {
                                client.sendMessage(new ChatMessage(ChatMessage.LOGOUT, ""));
                                // break to do the disconnect
                                break;
                        }
                        // message WhoIsIn
                        else if(msg.equalsIgnoreCase("WHOISIN")) {
                                client.sendMessage(new ChatMessage(ChatMessage.WHOISIN,
"""));
                        }
                        else {                          // default to ordinary message
                                client.sendMessage(new ChatMessage(ChatMessage.MESSAGE, msg));
                        }
                }
                // done disconnect
                client.disconnect();
        }

        /*
         * a class that waits for the message from the server and append them to the JTextArea
         * if we have a GUI or simply System.out.println() it in console mode
         */
        class ListenFromServer extends Thread {

                public void run() {
                        while(true) {
                                try {
                                        String msg = (String) sInput.readObject();
                                        // if console mode print the message and add back the prompt
                                        if(cg == null) {
                                                System.out.println(msg);
                                                System.out.print("> ");
                                        }
                                        else {
                                                cg.append(msg);
                                        }
                                }
                                catch(IOException e) {
                                        display("Server has close the connection: " + e);
                                        if(cg != null)
                                                cg.connectionFailed();
                                        break;
                                }
                                // can't happen with a String object but need the catch anyhow
                                catch(ClassNotFoundException e2) {
                                }
                        }
                }
        }
}
```

Source for server

```
package osmain.Chatting;
import osmain.Chatting.ServerGUI;
import osmain.Chatting.ChatMessage;
import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.*;

/*
 * The server that can be run both as a console application or a GUI
 */
public class Server {
        // a unique ID for each connection
        private static int uniqueId;
```

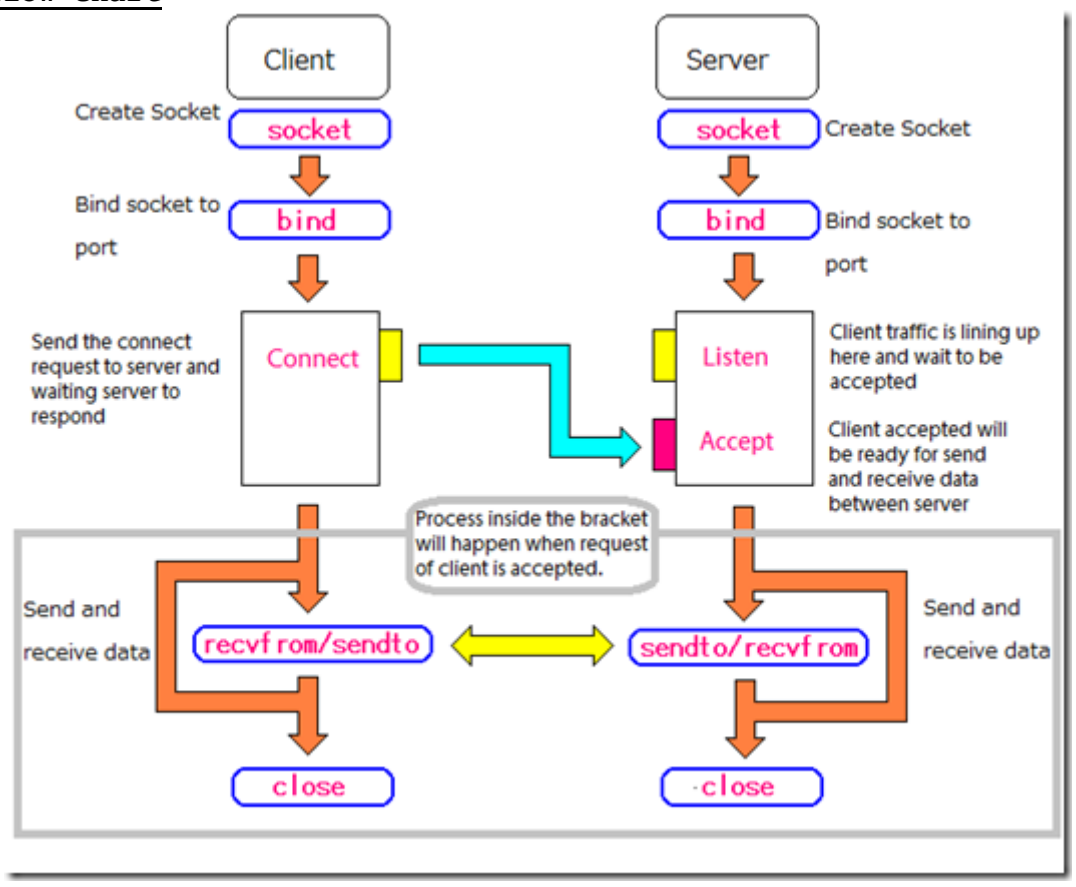
```
// an ArrayList to keep the list of the Client
private ArrayList<ClientThread> al;
// if I am in a GUI
private ServerGUI sg;
// to display time
private SimpleDateFormat sdf;
// the port number to listen for connection
private int port;
// the boolean that will be turned of to stop the server
private boolean keepGoing;


/*
 *  server constructor that receive the port to listen to for connection as parameter
 *  in console
 */
public Server(int port) {
        this(port, null);
}

public Server(int port, ServerGUI sg) {
        // GUI or not
        this.sg = sg;
        // the port
        this.port = port;
        // to display hh:mm:ss
        sdf = new SimpleDateFormat("HH:mm:ss");
        // ArrayList for the Client list
        al = new ArrayList<ClientThread>();
}

public void start() {
        keepGoing = true;
        /* create socket server and wait for connection requests */
        try
        {
                // the socket used by the server
                ServerSocket serverSocket = new ServerSocket(port);

                // infinite loop to wait for connections
                while(keepGoing)
                {
                        // format message saying we are waiting
                        display("Server waiting for Clients on port " + port + ".");

                        Socket socket = serverSocket.accept();        // accept connection
                        // if I was asked to stop
                        if(!keepGoing)
                                break;
                        ClientThread t = new ClientThread(socket);  // make a thread of it
                        al.add(t);
// save it in the ArrayList
                        t.start();
                }
                // I was asked to stop
                try {
                        serverSocket.close();
                        for(int i = 0; i < al.size(); ++i) {
                                ClientThread tc = al.get(i);
                                try {
                                tc.sInput.close();
                                tc.sOutput.close();
                                tc.socket.close();
                                }
                                catch(IOException ioE) {
                                        // not much I can do
                                }
                        }
                        }
                        catch(Exception e) {
                                display("Exception closing the server and clients: " + e);
                        }
                }
                // something went bad
                catch (IOException e) {
             String msg = sdf.format(new Date()) + " Exception on new ServerSocket: " + e + "\n";
                        display(msg);
                }
    }
/*
 * For the GUI to stop the server
 */
    protected void stop() {
            keepGoing = false;
            // connect to myself as Client to exit statement
            // Socket socket = serverSocket.accept();
            try {
                    new Socket("localhost", port);
            }
            catch(Exception e) {
                    // nothing I can really do
            }
    }

    private void display(String msg) {
```

```
        String time = sdf.format(new Date()) + " " + msg;
        if(sg == null)
                System.out.println(time);
        else
                sg.appendEvent(time + "\n");
}

private synchronized void broadcast(String message) {
        // add HH:mm:ss and \n to the message
        String time = sdf.format(new Date());
        String messageLf = time + " " + message + "\n";
        // display message on console or GUI
        if(sg == null)
                System.out.print(messageLf);
        else
                sg.appendRoom(messageLf);     // append in the room window

        // we loop in reverse order in case we would have to remove a Client
        // because it has disconnected
        for(int i = al.size(); --i >= 0;) {
                ClientThread ct = al.get(i);
                // try to write to the Client if it fails remove it from the list
                if(!ct.writeMsg(messageLf)) {
                        al.remove(i);
                        display("Disconnected Client " + ct.username + " removed from list.");
                }
        }
}

// for a client who logoff using the LOGOUT message
synchronized void remove(int id) {
        // scan the array list until we found the Id
        for(int i = 0; i < al.size(); ++i) {
                ClientThread ct = al.get(i);
                // found it
                if(ct.id == id) {
                        al.remove(i);
                        return;
                }
        }
}


public static void Server_Driver() {
        // start server on port 1500 unless a PortNumber is specified
        int portNumber = 1500;
        /*switch(args.length) {
                case 1:
                        try {
                                portNumber = Integer.parseInt(args[0]);
                        }
                        catch(Exception e) {
                                System.out.println("Invalid port number.");
                                System.out.println("Usage is: > java Server [portNumber]");
                                return;
                        }
                case 0:
                        break;
                default:
                        System.out.println("Usage is: > java Server [portNumber]");
                        return;

        }*/
        // create a server object and start it
        Server server = new Server(portNumber);
        server.start();
}

/** One instance of this thread will run for each client */
class ClientThread extends Thread {
        // the socket where to listen/talk
        Socket socket;
        ObjectInputStream sInput;
        ObjectOutputStream sOutput;
        // my unique id (easier for deconnection)
        int id;
        // the Username of the Client
        String username;
        // the only type of message a will receive
        ChatMessage cm;
        // the date I connect
        String date;

        // Constructore
        ClientThread(Socket socket) {
                // a unique id
                id = ++uniqueId;
                this.socket = socket;
                /* Creating both Data Stream */
                System.out.println("Thread trying to create Object Input/Output Streams");
                try
                {
                        // create output first
                        sOutput = new ObjectOutputStream(socket.getOutputStream());
                        sInput  = new ObjectInputStream(socket.getInputStream());
```

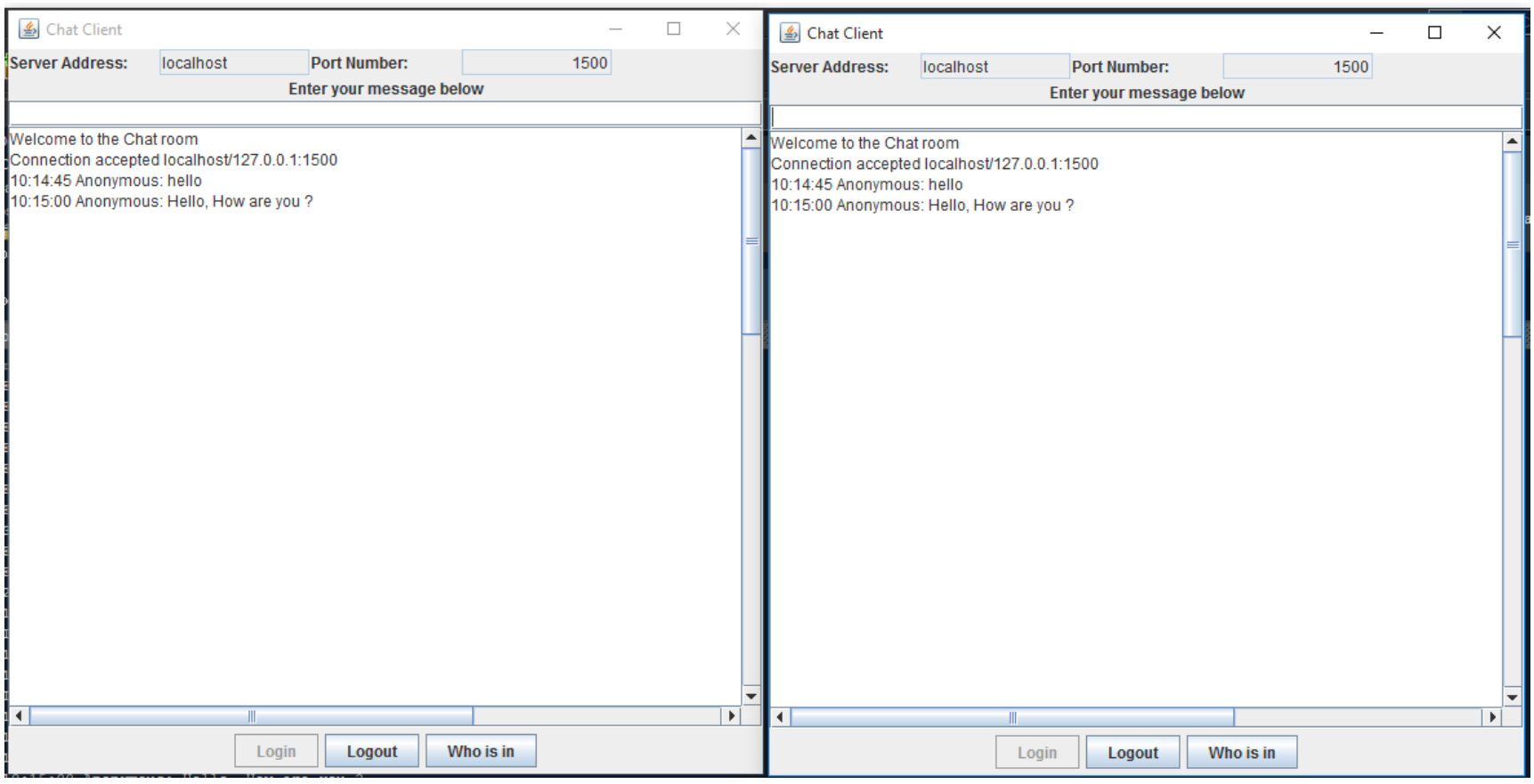
```
                        // read the username
                        username = (String) sInput.readObject();
                        display(username + " just connected.");
                }
                catch (IOException e) {
                        display("Exception creating new Input/output Streams: " + e);
                        return;
                }
                // have to catch ClassNotFoundException
                // but I read a String, I am sure it will work
                catch (ClassNotFoundException e) {
                }
        date = new Date().toString() + "\n";
        }

        // what will run forever
        public void run() {
                // to loop until LOGOUT
                boolean keepGoing = true;
                while(keepGoing) {
                        // read a String (which is an object)
                        try {
                                cm = (ChatMessage) sInput.readObject();
                        }
                        catch (IOException e) {
                                display(username + " Exception reading Streams: " + e);
                                break;
                        }
                        catch(ClassNotFoundException e2) {
                                break;
                        }
                        // the messaage part of the ChatMessage
                        String message = cm.getMessage();

                        // Switch on the type of message receive
                        switch(cm.getType()) {

                        case ChatMessage.MESSAGE:
                                broadcast(username + ": " + message);
                                break;
                        case ChatMessage.LOGOUT:
                                display(username + " disconnected with a LOGOUT message.");
                                keepGoing = false;
                                break;
                        case ChatMessage.WHOISIN:
                                writeMsg("List of the users connected at " + sdf.format(new
        Date()) + "\n");
                                // scan al the users connected
                                for(int i = 0; i < al.size(); ++i) {
                                        ClientThread ct = al.get(i);
                                        writeMsg((i+1) + ") " + ct.username + " since " +
        ct.date);
                                }
                                break;
                        }
                }
                // remove myself from the arrayList containing the list of the
                // connected Clients
                remove(id);
                close();
        }

        // try to close everything
        private void close() {
                // try to close the connection
                try {
                        if(sOutput != null) sOutput.close();
                }
                catch(Exception e) {}
                try {
                        if(sInput != null) sInput.close();
                }
                catch(Exception e) {};
                try {
                        if(socket != null) socket.close();
                }
                catch (Exception e) {}
        }


        private boolean writeMsg(String msg) {
                // if Client is still connected send the message to it
                if(!socket.isConnected()) {
                        close();
                        return false;
                }
                // write the message to the stream
                try {
                        sOutput.writeObject(msg);
                }
                // if an error occurs, do not abort just inform the user
                catch(IOException e) {
                        display("Error sending message to " + username);
                        display(e.toString());
                }
```

```
                    return true;
                }
            }
        }
```

Client server flow chart



Output:



INPUT/OUTPUT

Author: *Ahmed Sha'ban*
Date: *11/5/2016*
Version: *1.0*
Project ID: *input/output module*
CS Class: *SEC3 3_computer*
Programming Language: *Java*
OS/Hardware dependencies: *Arduino*

Problem description **writing on a serial port**

Source code

```
package osmain.IOArduinoWithJava;


import java.util.Scanner;

import javax.swing.JFrame;

import javax.swing.JSlider;

import com.fazecast.jSerialComm.*;



public class IO {

	public static void IO_Driver() {


		JFrame window = new JFrame();

		JSlider slider = new JSlider();


	slider.setMaximum(1023); //Arduino got 10-bit ADC

		slider.setPaintTicks(true);

	slider.setPaintTrack(true);

	slider.setPaintLabels(true);

	window.add(slider);

	window.pack();

		window.setVisible(true);


SerialPort[] ports = SerialPort.getCommPorts(); //array of ports that shows all ports on system

System.out.println("Select a port:");

int i = 1;

for(SerialPort port : ports) //special kind of for loop

	System.out.println(i++ +  ": " + port.getSystemPortName());

Scanner s = new Scanner(System.in);

int chosenPort = s.nextInt();


SerialPort serialPort = ports[chosenPort - 1];

if(serialPort.openPort())

	System.out.println("Port opened successfully.");

else {

	System.out.println("Unable to open the port.");

	return;
```

```
        }
        //serialPort.setComPortParameters(9600, 8, 1, SerialPort.NO_PARITY);
        serialPort.setComPortTimeouts(SerialPort.TIMEOUT_READ_BLOCKING, 0, 0);

        Scanner data = new Scanner(serialPort.getInputStream());
        int value = 0;
              System.out.println("Now you can change the POT value and it will be shown on the screen and slider");
       System.out.println("NOTE : there may be some sort of delay for 5 seconds at most, be patient please!");
while(data.hasNextLine()){
    System.out.println(data.nextLine());
    try{value = Integer.parseInt(data.nextLine());}catch(Exception e){}
                double analog_reading = (10000/1024)*value;
    slider.setValue(value);
        }
        System.out.println("Done.");


    }
}
```

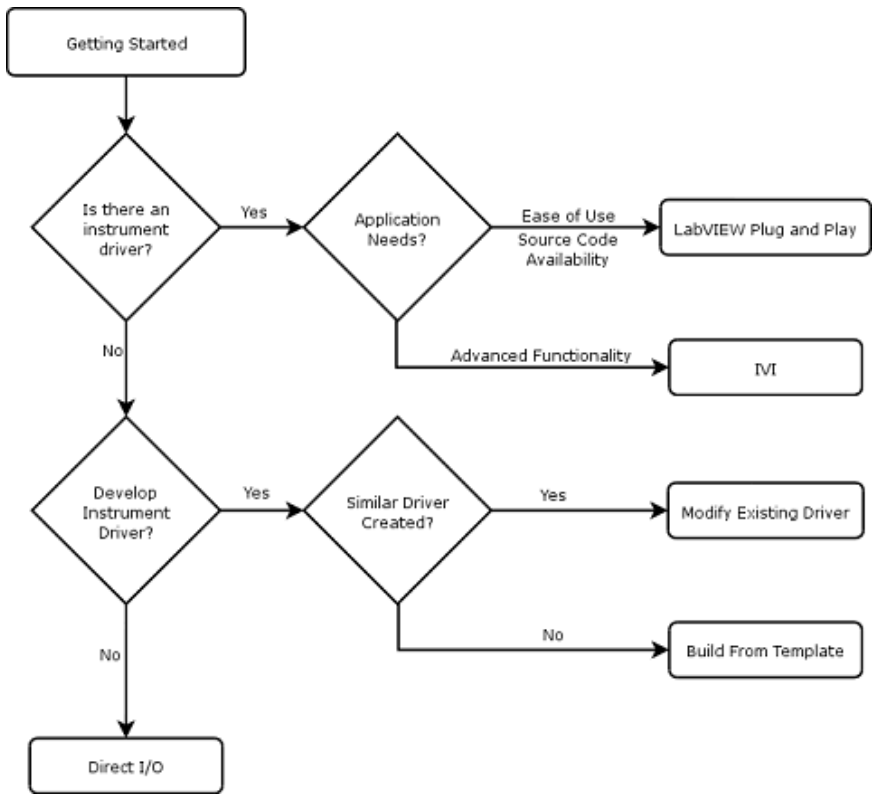
Pseudo code

```
select port ();
get port name ();
if(port selected )
      read  port value
else
       return;
```

Algorithm:

```
select port
Read from the selected port
```

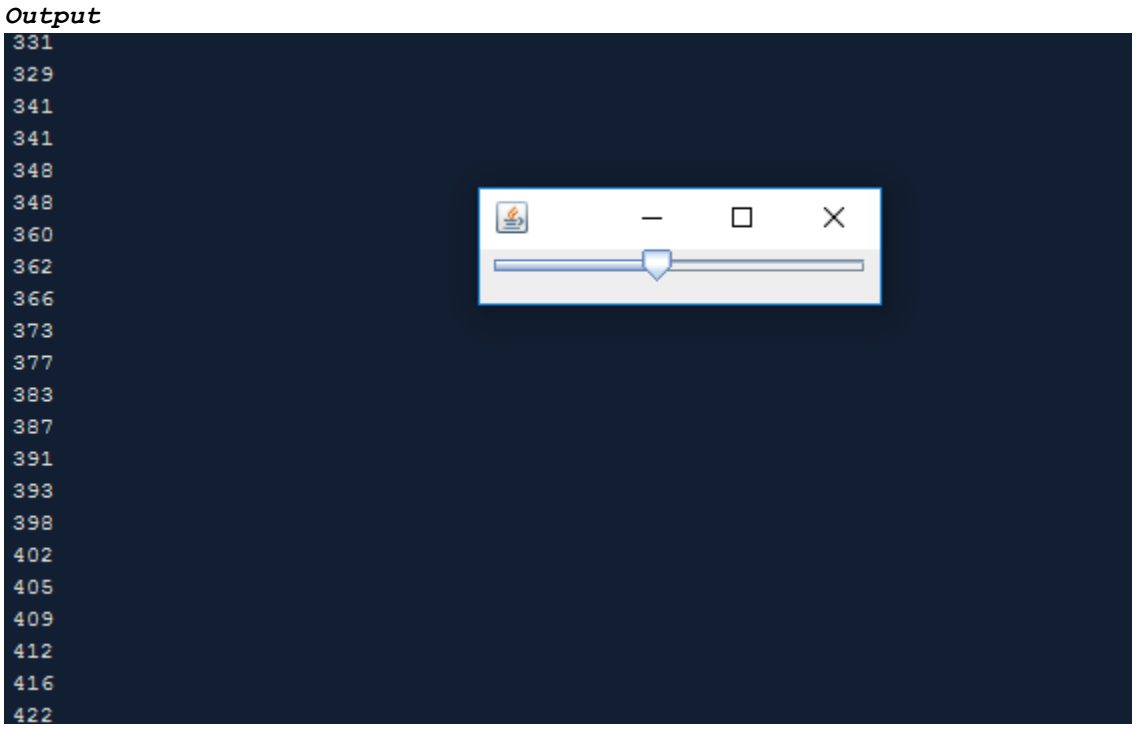
flow chart:



Additional files required: ***SerialComm-1.3.11.jar***

RXTX.jar

Output



This project runs completely from a unified interface

Output