```
import java.util.concurrent.Semaphore;

public class BarberSaloon extends Thread{

      //                ***         for panel control configs     ***

      static  final int nClients= 15;          // nr of clients
      static  final int nChairs=4;             // nr of chairs (barber chair
also)
      static  final int minTimeHairCut=2500;   // 25 minutes simulation
      static  final int variableTimeCut=150;   // 100ms => 1.5 minutes
      static  final int maxTimeBetweenCuts=300; // coffee pause
    //static  final int maxTimeBetweenClients=500;// busy saloon => 5 minutes
      static  final int maxTimeBetweenClients=6000;// calm saloon => 60 minutes
      // 1 chair for barber + 3 for clients waiting => 4 chairs permanently
available

      static int freeChairs= nChairs;

      //3 semaphores are needed

      Semaphore semafMutex=new Semaphore(1);
      // semafMutex controls access to shared variable freeChairs
      // no Resource semaphore(4): it doesn't guarantee an order FIFO

      Semaphore semafBarberReady = new Semaphore(0);
      // when Barber is ready for new client semafBarberReady:=1

      Semaphore semafSaloonNotEmpty=new Semaphore(0); //begin: it's false=0
      // if saloon has clients semafSaloonNotEmpty:=1


      public Semaphore getSemafMutex() {  return semafMutex;      }

      public Semaphore getSemafSaloonNotEmpty() {
            return semafSaloonNotEmpty;
      }

      public void setSemafSaloonNotEmpty(Semaphore semafSaloonNotEmpty) {
            this.semafSaloonNotEmpty = semafSaloonNotEmpty;
      }

      public void setSemafMutex(Semaphore semafMutex) {
            this.semafMutex = semafMutex;
      }

      public Semaphore getSemafBarberReady(){ return semafBarberReady;}

      public void setSemafBarberReady(Semaphore semafBarberReady) {
            this.semafBarberReady = semafBarberReady;
      }


      public BarberSaloon() { } //constructor


      public static void main(String[] args) {
            BarberSaloon bs= new BarberSaloon();
            bs.start();
      }

      public void run() {

        Client [] c =new Client[ nClients]; // thread pool
```

```
        Barber b= new Barber();

        b.start();

        for(int i=0;i< nClients;i++) { // begin: 1 client

             c[i]=new Client(i);

             try {Thread.sleep((int)(Math.random()*maxTimeBetweenClients));}
             catch(InterruptedException e) {};
             c[i].start();
        }

        System.out.println ("\n   ***    Summary     *** ");
        for(int i=0;i< nClients;i++) {
             try {
               c[i].join();
               System.out.println ("Client id: "+i+" finished => thread join.");
              } catch(InterruptedException e) {};
        }
      }



public class Barber extends Thread{


public Barber() {} //constructor


  private void cutHair() {

  System.out.print ("\n  Barber  begins hair cut. Freechairs =
"+freeChairs+".");

  try {
      Thread.sleep((int)(minTimeHairCut+ Math.random() * variableTimeCut));}
  catch(InterruptedException e) {};

  System.out.println (" Barber ends hair cut.");
 }


  public void run() {

      while(true) {

        try {Thread.sleep((int)(Math.random()*maxTimeBetweenCuts));}
        catch(InterruptedException e) {};

        try {getSemafBarberReady().release();} catch(Exception e) {};
        try {getSemafSaloonNotEmpty().acquire();}    catch(Exception e) {};
        cutHair();

      }
  }// end run()
}// end inner class Barber




 public class Client extends Thread{
```

```
      private int clientId;


      public int getClientId() {    return clientId;}
      public void setClientId(int clientId) {   this.clientId = clientId;}

      public Client(int i) {  clientId=i; }


      private void getCut() {

        System.out.print ("\n  Client id="+this.clientId+" begins hair cut.  ");

        try {Thread.sleep((int)(minTimeHairCut+ Math.random() *
variableTimeCut));}  catch(InterruptedException e) {};

        System.out.println ("  Client id="+this.clientId+" ends hair cut.");
      }


      public void run() {

        boolean cut=false;    //begin: cut is false
        int count=1;
        String s;

        while ((cut==false)&&(count<=3)) {

          try {
                  s=" Client id= "+this.clientId+" arrives";
                  if (count==1) {   s+=".";     }
                  else {      s+=" again.";     }
                  System.out.print (s);

                  getSemafMutex().acquire();

                  if (freeChairs>0) {

                   freeChairs--;
                   System.out.println ("  Client id="+this.clientId+" seats.
Freechairs = " + freeChairs + ". Trials: "+count+ ".");

                    try {getSemafSaloonNotEmpty().release();}    catch(Exception
e) {};

                    try {getSemafBarberReady().acquire();}  catch(Exception e)
{};

                    try {getSemafMutex().release();} catch(Exception e) {};

                    getCut();
                    cut=true; //hair is cut
                    try {getSemafMutex().acquire();} catch(Exception e) {};
                        freeChairs++;
                    try {getSemafMutex().release();} catch(Exception e) {};
                    System.out.print ("\n  Client id="+this.clientId+" says Bye.
Freechairs = " + freeChairs + ".");
                  }
                  else {// assumption: client comes a bit later, but only twice

                    try {getSemafMutex().release();} catch(Exception e) {};

                    System.out.println ("\n  Saloon full. Client
id="+this.clientId+" comes maybe later. Freechairs = " + freeChairs + ". Trials:
```

```
"+count+".");

                    try {Thread.sleep((int)( Math.random()*2*minTimeHairCut) );}
                    catch(Exception e) {};

                  }
                  count++;
          }
          catch(Exception e) {};
         }// end while

        System.out.println ("\n     *** Client id="+this.clientId+" departs.
Freechairs = " + freeChairs + ". Hair cut? "+cut+". Trials? "+ (int) (count-1) +
". ***      ");
       }// end run()

 }// end inner class Client

}// end BarberSaloon class
```