Nel negozio di un barbiere vi è una sala d'attesa con un divano contenente al massimo 5 persone [e un'area dove i clienti possono attendere in piedi. Per motivi di sicurezza, all'interno del negozio possono sostare al massimo N persone contemporaneamente].
Al negozio lavorano tre barbieri, ciascuno con la propria poltrona per il taglio dei capelli.
Se un cliente entra nel negozio [quando la sala d'attesa è piena] va via senza attendere il proprio turno. Altrimenti, [dapprima si accomoda nell'area di attesa, poi] si accomoda nel divano e [infine] attende di essere chiamato da un barbiere.
Il barbiere serve per primo il cliente in attesa da più tempo, gli taglia i capelli, e provvede al pagamento e al rilascio della ricevuta. [Poiché vi è soltanto un registro di cassa, i clienti prima di andare via devono aspettare il proprio turno per pagare].
Si implementi la sincronizzazione tra i clienti e i barbieri usando il costrutto Monitor. Si discuta inoltre se la soluzione proposta può presentare rinvio indefinito e/o deadlock, e se sì, discutere eventuali modifiche per evitarli.

Main.java

```
public class Main {
	public static void main( String [] args ) {

		Sala sala = new Sala();

		int nBarbieri = 3;
		Barbiere barbieri[] = new Barbiere[nBarbieri];

		for(int i=0; i<nBarbieri; i++){
			barbieri[i] = new Barbiere(sala, i);
			barbieri[i].start();
		}

		int nClienti = 1000;
		Cliente clienti[] = new Cliente[nClienti];

		for(int i=0; i<nClienti; i++){
			clienti[i] = new Cliente( sala, i);
			clienti[i].start();
			try{
				Thread.sleep( ( int) ( Math.random() * 501 ) );
			} catch (InterruptedException exception) {
				exception.printStackTrace();
			}
		}

	}
 }
```

Sala.java

```
import java.util.LinkedList;

public class Sala {

  private int servito;
      private int NUM_POSTI = 5;
      private LinkedList<Integer> divano = new LinkedList<Integer>();

      public synchronized void mettiInCoda(int id)
      {
            System.err.println( "== arriva il cliente " + id );

    if ( divano.size() <= NUM_POSTI ){
      divano.add(id);
      notifyAll();
    } else {
           System.out.println("DIVANO PIENO " + divano + " - il cliente " + id
+ " va via.");
        }
      }


      public synchronized void serviCliente(int id)
      {
   while(divano.size()==0){
     try{
       System.out.println("Barbiere " + id+ " nessun cliente da servire: wait");
       wait();
     } catch (InterruptedException e){
       e.printStackTrace();
     }
   }
   System.out.println("Barbiere " + id + " ottiene il cliente "  +
divano.getFirst());
   servito = divano.removeFirst();
   try{
     Thread.sleep( ( int) ( Math.random() * 101 ) );
   } catch (InterruptedException exception) {
     exception.printStackTrace();
   }
       System.out.println("Barbiere " + id+ " ha servito il cliente " +
servito);

      }
}
```

Barbiere.java

```
public class Barbiere extends Thread {
	private Sala sala;
	private int id;

	public Barbiere( Sala s, int i)
	{
	super ( "Barbiere" );
	sala = s;
	id = i;
	}

	public void run()
	{
		while ( true )
		{
	System.out.println("Barbiere " + id + " pronto per servire un cliente");
	sala.serviCliente(id);

			try
			{
				Thread.sleep( ( int ) ( Math.random() * 5001 ) );
			}

			catch ( InterruptedException exception )
			{
				exception.printStackTrace();
			}
		}
	}
}
```

Cliente.java

```
public class Cliente extends Thread{

	private Sala sala;
	private int id;

	public Cliente( Sala s, int i)
	{
		super( "Cliente" );
		sala = s;
		id=i;
	}

	public void run()
	{
		try
		{
			Thread.sleep( ( int) ( Math.random() * 0 ) );
			sala.mettiInCoda(id);
		}	catch ( InterruptedException exception )
		{
			exception.printStackTrace();
		}
	}
}
```