**SLEEPING BARBER PROBLEM**

import java.util.concurrent.\*;

class SleepingBarber extends Thread {

/\* PREREQUISITES \*/

/\* we create the semaphores. First there are no customers and

the barber is asleep so we call the constructor with parameter

0 thus creating semaphores with zero initial permits.

Semaphore(1) constructs a binary semaphore, as desired. \*/

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 {

/\* we create the integer iD which is a unique ID number for every customer

and a boolean notCut which is used in the Customer waiting loop \*/

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 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) {};

}

}

}

public class SleepingBarber

{

public static void main(String args[]) {

SleepingBarber barberShop = new SleepingBarber(); //Creates a new barbershop

barberShop.start(); // Let the simulation begin

}

}