Tema Java – Un parcheggio coperto è dotato di 11 piani (incluso piano terra). Al piano terra vengono smistati gli utenti in salita su due code, una (**P**) per quelli che devono accedere ai piani da 1 a 5, una (**S**) per quelli dei piani da 6 a 10. Per la salita è a disposizione un ascensore di capienza **CAP;** quando l'ascensore sta caricando al piano terra, per motivi di efficienza 'preleva' utenti dalla coda S con priorità e, se c'è ancora spazio, successivamente dalla coda P. L'ascensore parte se è pieno oppure se ha atteso MAXATT unità da tempo dall'ultimo utente caricato (e pertanto ne contiene almeno uno).
Utilizzando il **Monitor di Java** come unico strumento di sincronizzazione, si realizzi la classe **`Salita`** che simula il sistema di controllo dell'accesso agli ascensori, e comprende, tra l'altro:

- Le variabili di conteggio **`prenP, prenS, ticketP, ticketS, servP, servS`** che rappresentano rispettivamente il numero di utenti in P e in S, e le corrispondenti variabili per il sistema di *ticketing* per mantenere l'ordine di arrivo;
- Le variabili **`num`** e **`arrivato`** che rappresentano rispettivamente l'occupazione (progressiva) dell'ascensore e il suo piano corrente di fermata;
- Il vettore **`pren`** che rappresenta con **`pren[i]`** quanti utenti in una corsa di ascensore vogliono salire al piano **`i`**;
- La variabile booleana **`aperto`** che è *true* quando l'ascensore sta caricando al piano terra;
- La classe interna **`Utente`** che riceve nel costruttore un indice identificativo e il piano a cui l'utente è diretto; nel suo metodo principale simula l'inserimento dell'utente nella coda di pertinenza chiamando uno dei due metodi **`inCodaP()`** o **`inCodaS()`** e, successivamente, simula l'ingresso nell'ascensore e la salita chiamando il metodo **`entra()`**;
- La classe interna **`Ascensore`** che, nel suo metodo principale, in un ciclo indefinito, simula l'attesa per caricamento al piano terra nonché la salita ai vari piani con la chiamata al metodo **`servizio()`**, seguita dalla discesa dall'ultimo piano visitato al piano terra con la chiamata al metodo **`discesa()`**.;
- I metodi (sospensivi) **`void inCodaP(int piano)`** e **`void inCodaS(int piano)`** che rappresentano rispettivamente l'eventuale attesa ordinata nella coda P e nella coda S nonché la prenotazione del piano rappresentato dal parametro di chiamata;
- Il metodo **`void entra(int piano)`** che simula l'ingresso di un utente nell'ascensore, di durata ENTRATIM, e l'attesa fino al momento in cui l'utente arriva al piano richiesto;
- Il metodo **`int minPren(int pr[])`** che restituisce l'indice del piano inferiore che ha prenotazioni pendenti come indicato dal parametro **`pr`**;
- Il metodo **`void servizio()`** che simula l'attesa dell'ascensore per il carico seguita dalla visita nell'ordine di tutti i piani prenotati; ogni viaggio ha una durata pari a m*PIANOT+DISCESA dove m≥1 è il numero di piani attraversati e la costante DISCESA è il tempo consentito per la discesa degli utenti al piano;
- Il metodo **`void discesa()`** che simula la discesa dell'ascensore dall'ultimo piano visitato al piano terra; la discesa ha una durata pari a *n*PIANOT* con *n* piano di partenza nella discesa e PIANOT costante corrispondente alla discesa di un piano;
- Il metodo **`main`** di collaudo che, dopo aver inizializzato l'istanza della classe **`Salita`**, e attivato l'ascensore, genera con tempi casuali, in un intervallo i cui estremi sono forniti sulla linea di comando, un certo numero di utenti ciascuno con il piano di destinazione scelto casualmente.

Suggerimento; si ricorda che, se si vuole avere *sincronizzato* (cioè con mutex) solo un segmento di codice e non un intero metodo, si può utilizzare il costrutto Java **`synchronized(Object) { . . . }`**.

Soluzione

```
import os.Util;

/**{c}
 * Salita, Monitor di Java
 * @author M.Moro DEI UNIPD
 * @version 1.00 2018-05-22
 */
public class Salita
{
    private static final int PIANOT = 1000;
    private static final int DISCESA = 1000;
    private static final int MAXWAIT = 3000;
    private static final int ENTRATIM = 500;
    private static final int CAP = 6;  // per la simulazione
    private int prenP=0, prenS=0, ticketP=0, ticketS=0, servP=0, servS=0;
    private int num=0, arrivato=0;
    private boolean aperto = true;
    private int pren[];

    /**[c]
     */
    public Salita()
    {
        int i;
```

```
    pren = new int[11];
      // i piani vanno da 1 a 10

    for (i=0; i<=10; i++)
        pren[i] = 0;

}


/**{c}
 * thread Utente
 */
private class Utente extends Thread
{
    private int idx;
      // indice cliente
    private int piano;
      // piano di salita

    /**[c]
     * @param idx  indice utente
     * @param pi  piano di salita
     */
    public Utente(int idx, int pi)
    {
        super("Utente_"+(pi<=5?"P_":"S_")+idx);
        System.out.println("Crea "+getName());
        this.idx = idx;
        piano = pi;
    }

    /**[m]
     * attende ingresso, eventuale entrata ascensore e salita
     */
    public void run()
    {
        System.out.println("Attivato "+getName()+" ora va in coda, piano="+piano);
        if (piano<=5)
            inCodaP(piano);
        else
            inCodaS(piano);
        System.out.println("1> Ora "+getName()+" va all'ascensore");
        entra(piano);
        System.out.println("2> "+getName()+" e' arrivato");
    } //[m] run

} // {c} Utente

/**{c}
 * thread Ascensore
 */
private class Ascensore extends Thread
{

    /**[c]
     */
    public Ascensore()
    {
        super("Ascensore");
        System.out.println("Crea "+getName());
    }

    /**[m]
```

```
     * attende riempimento e salita, discesa
     */
    public void run()
    {
        System.out.println("Attivato "+getName());
        for(;;)
        {
            servizio();
            discesa();
        } // for
    } //[m] run

} // {c} Ascensore


/**[m]
 * coda P prima parte
 * @param piano  indice piano di salita (<= 5)
 */
public synchronized void inCodaP(int piano)
{
    int myTicket = ticketP++;
    prenP++;
      // prenota
    System.out.println("1*** CodaP "+Thread.currentThread().getName()+
      " ticket="+myTicket);
    while (!((servP==myTicket) && (prenS==0) && (num<CAP) && aperto) )
    {
        // l'ingresso e' possibile se l'ascensore e' aperto e
        // non pieno, e se non ci sono prenotazioni prioritarie
        try { wait(); } catch (InterruptedException e) {};
    }
    prenP--;
      // elimina prenotazione
    servP++;  // puo' servire il prossimo
    num++;  // anticipa occupazione ascensore
    pren[piano]++;  // prenota piano
    System.out.println("2*** CodaP "+Thread.currentThread().getName()+
      " puo' entrare ticket="+myTicket+" num="+num);
} //[m]

/**[m]
 * coda S seconda parte
 * @param piano  indice piano di salita (> 5)
 */
public synchronized void inCodaS(int piano)
{
    int myTicket = ticketS++;
    prenS++;
      // prenota
    System.out.println("1^^^ CodaS "+Thread.currentThread().getName()+
      " ticket="+myTicket);
    while (!((servS==myTicket) && (num<CAP) && aperto) )
    {
        // l'ingresso e' possibile se l'ascensore e' aperto e
        // non pieno
        try { wait(); } catch (InterruptedException e) {};
    }
    prenS--;
      // elimina prenotazione
    servS++;  // puo' servire il prossimo
    num++;  // anticipa occupazione ascensore
    pren[piano]++;  // prenota piano
    System.out.println("2*** CodaS "+Thread.currentThread().getName()+
```

```
      " puo' entrare ticket="+myTicket+" num="+num);
} //[m]

/**[m]
 * ingresso in ascensore
 * @param piano  indice piano di salita
 */
public synchronized void entra(int piano)
{
    Util.sleep(ENTRATIM);
      // di norma non si mettono attese in mutex pero'
      // qui il tempo e' breve, non crea problemi in mutex
    while (arrivato != piano)
    {
        // attende il momento dell'uscita al piano richiesto
        try { wait(); } catch (InterruptedException e) {};
    }
    // arrivato
} //[m]

/**[m]
 * piano minimo
 * @param pr  array prenotazioni
 * @returns indice piano inferiore da raggiungere >0 o -1
 */
public int minPren(int pr[])
{
    int i;

    for(i=1; i<=10; i++)
    {
        // inizia dal pinao 1 in su
        if (pr[i]!=0)
            return i;
    }
    return -1;
} //[m]


/**[m]
 * simulazione carico e movimento ascensore
 */
public void servizio()
{
    int quanti, min;


    quanti = 0;
    num = 0;
    System.out.println("1=== servizio "+Thread.currentThread().getName()+
      " attende riempimento");
    for(;;)
    {
        // si noti che le sincronizzazioni avvengono
        // sull'unica istanza di classe Salita
        synchronized(this) {
            // da' modo agli utenti in eventuale attesa di entrare
            notifyAll();
        }

        if (prenP+prenS==0 && num < CAP)
            // al momento non ci sono prenotazioni per ingresso
            // ma ancora spazio
            Util.sleep(MAXWAIT);
```

```
        if ((prenP+prenS==0 && num!=0) || num == CAP)
            // dopo l'attesa non ci sono prenotazioni
            // ma ci sono gia' utenti nell'ascensore
            // oppure l'ascensore e' pieno: deve partire
            break;

    }

    System.out.println("2=== servizio "+Thread.currentThread().getName()+
      " si muove");
    aperto = false;
      // ingressi bloccati

    // inizia salita
    for (;;)
    {
        if ((min = minPren(pren)) == -1)
            break;
        System.out.println("3=== servizio "+Thread.currentThread().getName()+" da "+arrivato+
          " sale al piano "+min+" e scarica "+pren[min]+" utenti");
        Util.sleep((min-arrivato)*PIANOT+DISCESA);
          // simula una tratta
        // raggiunto un piano richiesto
        pren[min]=0;
        arrivato = min;
        synchronized(this) {
            // consente agli utenti che devono scendere
            // di farlo
            notifyAll();
        }
    }
    System.out.println("4=== servizio "+Thread.currentThread().getName()+
      " completato servizio");
} //[m]


/**[m]
 * simulazione discesa ascensore
 */
public void discesa()
{
    if (arrivato!=0)
    {
        // scende
        System.out.println(Thread.currentThread().getName()+" scende");
        Util.sleep(arrivato*PIANOT);
          // simula la discesa senza soste
        synchronized(this) {
            // questa mutua esclusione non e' strettamente necessaria
            arrivato = 0;
            aperto = true;
        }
    }
} //[m]


/**[m][s]
 * main di collaudo
 * @param args[0] minT
 * @param args[1] maxT
 */
public static void main(String[] args)
{
```

```
        // senza controlli
        Salita st = new Salita();
        st.new Ascensore().start();
        for (int i=0; i<100; i++)
        {
            Util.rsleep(Long.parseLong(args[0]),Long.parseLong(args[1]));
            st.new Utente(i, Util.randVal(1, 10)).start();
        }
    } //[m][s] main

} //{c} Salita
```