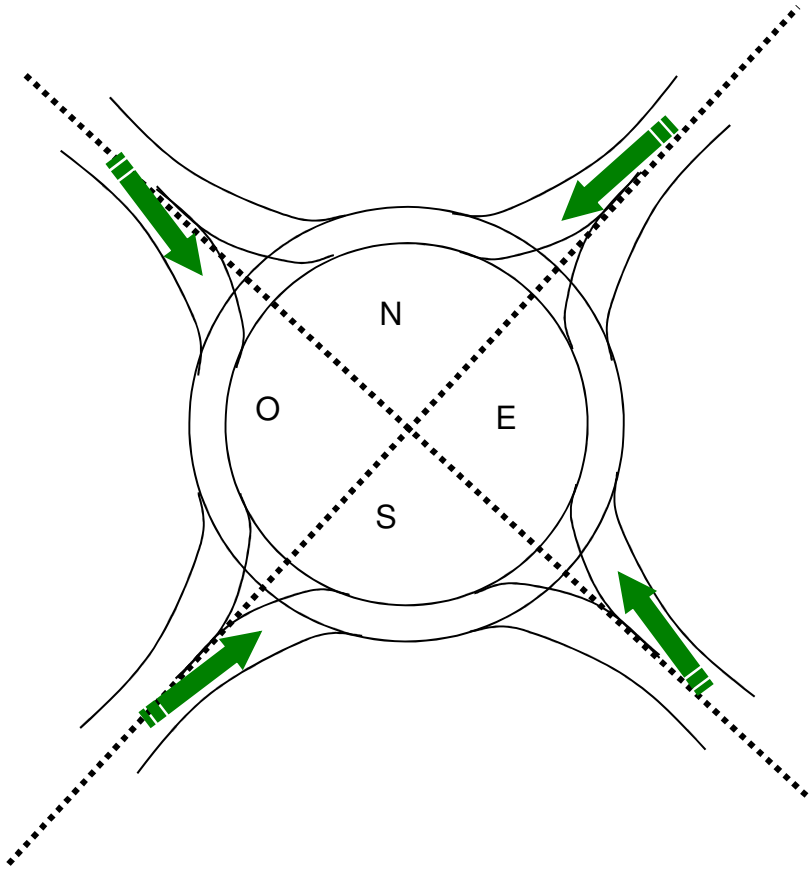
SISTEMI OPERATIVI

a.2

UNIU

Exercises

Roundabout



Vehicles must give precedence to a vehicle inside the roundabout and every sector can be occupied by at most one vehicle

```
public class RotatA
{
    public static final int SETTORI = 4;
    public static final int SETT_NORD = 0;
    public static final int SETT_OVEST = 1;
    public static final int SETT_SUD = 2;
    public static final int SETT_EST = 3;

    private boolean occupato[];  // stato del segmento
    // true occupato

    private int succ(int s)
    {
        // successore circolare
        return (s+1)%SETTORI;
    }

    private int prec(int s)
    {
        // successore circolare
        return (s-1+SETTORI)%SETTORI;
    }
```

```
public RotatA()
{
    occupato = new boolean[SETTORI];
    for (int i=0; i<SETTORI; occupato[i++] = false);
    // rotatoria vuota
} //[c]

public synchronized int entra(int settore)
{
    while(occupato[settore] || occupato[prec(settore)])
    {
        // e` occupato o il settore d'ingresso o
        // quello precedente
        try {
            wait();
        } catch(InterruptedException e) {}

    } // while
    occupato[settore] = true;
    // non ha senso liberare altri processi
    return settore;
} //[m] entra
```

```
public synchronized int prossimo(int settore)
{
    int nuovoSett = succ(settore);

    while(occupato[nuovoSett])
    {
        // e` occupato il settore successivo
        try {
            wait();
        } catch(InterruptedException e) {}

    } // while
    occupato[settore] = false;
    occupato[nuovoSett] = true;

    notifyAll();  // bisognerebbe liberarli in ordine
    return nuovoSett;
} //[m] prossimo
```


DEI UNIV PD © 2019

```
        public synchronized void esce(int settore)
        {
            occupato[settore] = false;
            notifyAll();  // bisognerebbe liberarli in ordine
        } //[m] esce


    } //{c} RotatA


    public class Veicolo extends Thread
    {
        private RotatA rot;
        int veicolo, settore;
        // settore all'inizio rappresenta l'ingresso

        public Veicolo(RotatA r, int ve, int se)
        {
            rot = r;
            veicolo = ve;
            settore = se;
        } //[c]
```


DEI UNIV PD © 2019

```
public void run() {
    settore = rot.entra(settore);
        // entra
    int set = Util.randVal(1, 2*RotatA.SETTORI);
      // massimo 2 giri
    for(int i=1; i<=set; i++)
    {
        Util.rsleep(500, 4000);
        settore = rot.prossimo(settore);
    } // while(true)
    // esce
    rot.esce(settore);
} //[m] run
```


DEI UNIV PD © 2019

```
    public static void main(String args[])
    {
        final int MAXVEICOLI = 10;

        RotatA rt = new RotatA();
        for (int i=1; i<= MAXVEICOLI; i++)
        {
            Veicolo v = new Veicolo(rt, i, Util.randVal(0, 3));
            v.start();
            Util.rsleep(500, 4000);
        }
    } //[m][s] main

} //{c} Veicolo
```


DEI UNIV PD © 2019

```
public class RotatX extends Monitor
{
    public static final int SETTORI = 4;
    public static final int SETT_NORD = 0;
    public static final int SETT_OVEST = 1;
    public static final int SETT_SUD = 2;
    public static final int SETT_EST = 3;

    private boolean occupato[];
      // stato del segmento
      // true occupato
    private Condition attDentro[];
      // condition di attesa in rotatoria
    private Condition attFuori[];
      // condition di attesa in immissione
    private int veic[];
      // veicolo occupante, non richiesto
```

```
private int succ(int s)
{
    // successore circolare
    return (s+1)%SETTORI;
}

private int prec(int s)
{
    // successore circolare
    return (s-1+SETTORI)%SETTORI;
}
```


DEI UNIV PD © 2019

```
        public RotatX()
        {
            occupato = new boolean[SETTORI];
            attDentro = new Condition[SETTORI];
            attFuori = new Condition[SETTORI];
            for (int i=0; i<SETTORI; occupato[i++] = false)
            {
                attDentro[i] = new Condition();
                attFuori[i] = new Condition();
            }
            // rotatoria vuota
        } //[c]

        public int ingresso(int settore)
        {
            mEnter();
            if(occupato[settore] || occupato[prec(settore)])
                // e` occupato o il settore d'ingresso o
                // quello precedente
                attFuori[settore].cWait();
            occupato[settore] = true;
            // non ha senso liberare altri processi
            mExit();
            return settore;
        } //[m] ingresso
```

```
public int passa(int settore)
{
    int nuovoSett = succ(settore);
    mEnter();
    if(occupato[nuovoSett])
        // e` occupato il settore successivo
        attDentro[settore].cWait();
    occupato[settore] = false;
    occupato[nuovoSett] = true;
    // priorita' a chi e' dentro
    attDentro[prec(settore)].cSignal();
    if (!occupato[settore] && !occupato[prec(settore)])
        // il settore e' ancora occupabile
        attFuori[settore].cSignal();
    mExit();
    return nuovoSett;
} //[m] passa
```


DEI UNIV PD © 2019

```
        public void fuori(int settore)
        {
            mEnter();
            occupato[settore] = false;

            // priorita' a chi e' dentro
            attDentro[prec(settore)].cSignal();
            if (!occupato[settore] && !occupato[prec(settore)])
                // il settore e' ancora occupabile
                attFuori[settore].cSignal();
            mExit();
        } //[m] fuori

    } //{c} RotatX
```


DEI UNIV PD © 2019

```
public class VeicoloMon extends Thread
{

    private RotatX rot;
    int veicolo, settore;
    // settore all'inizio rappresenta l'ingresso

    public VeicoloMon(RotatX r, int ve, int se)
    {
        rot = r;
        veicolo = ve;
        settore = se;
    } //[c]
```


DEI UNIV PD © 2019

```
public void run()
{
    settore = rot.ingresso(settore);
        // entra
    int set = Util.randVal(1, 2*RotatX.SETTORI);
      // massimo 2 giri
    for(int i=1; i<=set; i++)
    {
        Util.rsleep(500, 4000);
        settore = rot.passa(settore);
    } // while(true)
    // esce
    rot.fuori(settore);
} //[m] run
```


DEI UNIV PD © 2019

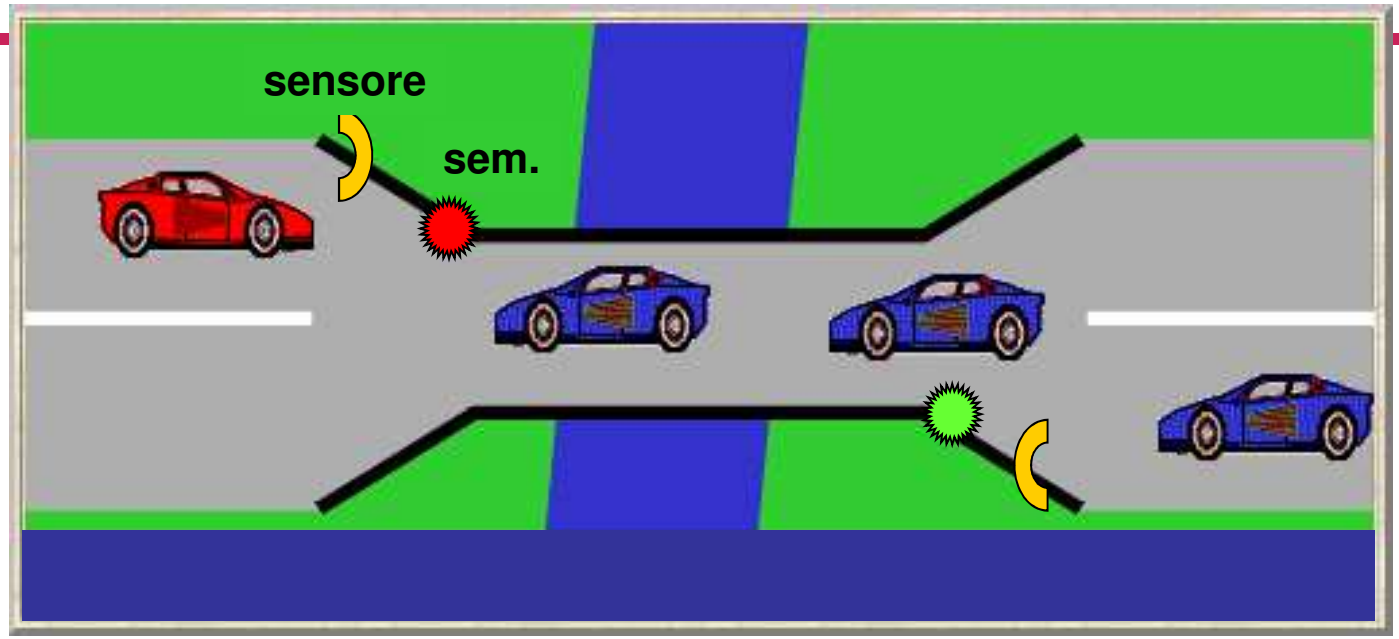
```
    public static void main(String args[])
    {
        final int MAXVEICOLI = 10;

        RotatX rt = new RotatX();
        for (int i=1; i<= MAXVEICOLI; i++)
        {
            VeicoloMon v = new
              VeicoloMon(rt, i, Util.randVal(0, 3));
            v.start();
            Util.rsleep(500, 4000);
        }
    } //[m][s] main

} //{c} VeicoloMon
```


DEI UNIV PD © 2019

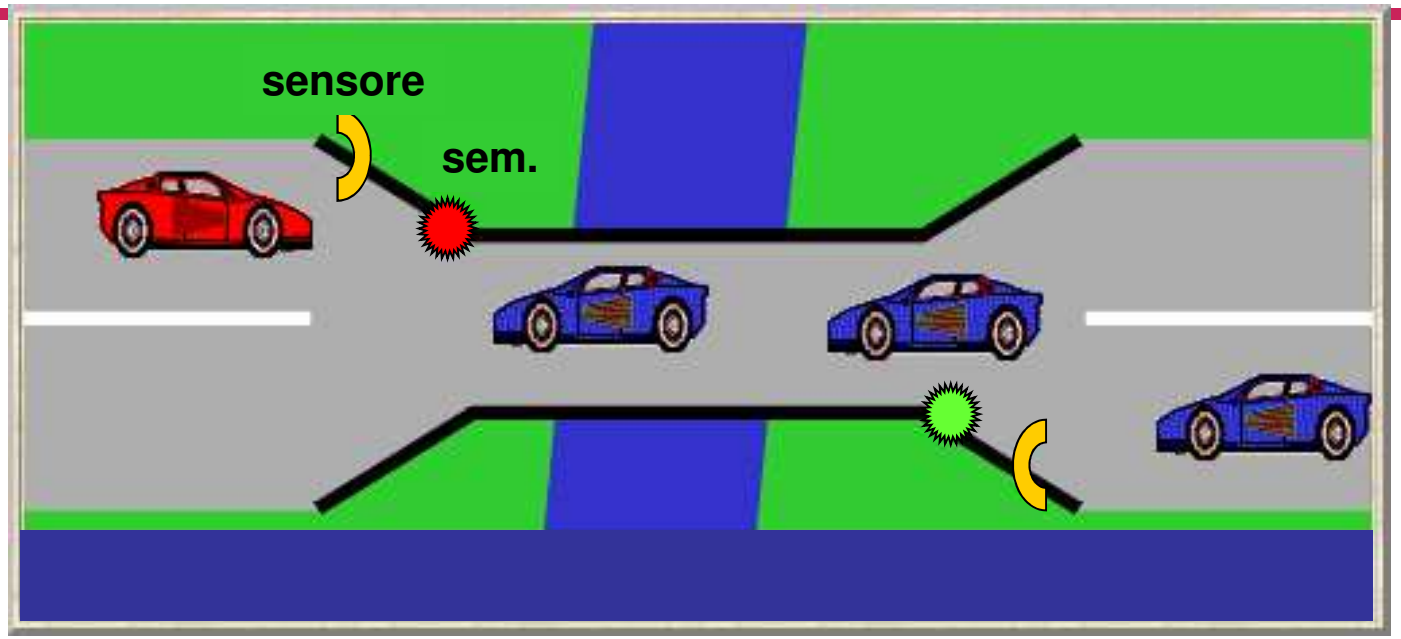
The bridge



On a bridge the road narrows into one lane.
The system is managed through 2 traffic lights and 2 sensors able to sense cars coming from each one of the two directions. The control must guarantee:

- If a car is crossing the bridge in one direction, this gives precedence to other cars waiting to cross the bridge in the SAME direction (see the cars' color in the picture);
- Only when no other car is waiting to cross the bridge in the same direction, the crossing in the other direction is allowed;
- No car should wait for an indefinite amount of time.

The bridge



Problems to solve:

- Before switching a traffic light to green for a waiting car, previously crossing car must free the bridge
- What to do with a continuos queue of cars in one direction?
- What to do if two cars come almost in the same moment from opposite directions?

DEI UNIV PD © 2019

The end

a.2

Exercises