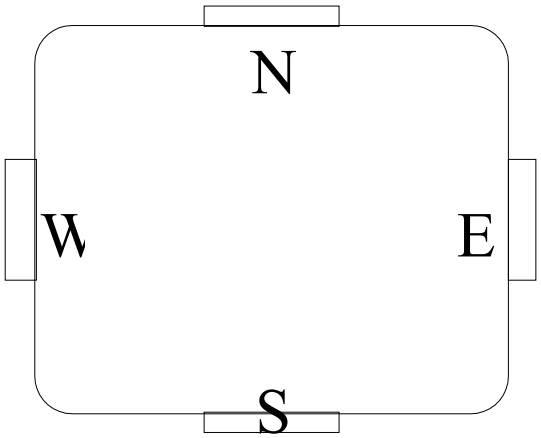
semafori-monopoli

La città di Monopoli ha quattro stazioni ferroviarie chiamate Nord, Sud, Est, Ovest (N,S,E,W).
La linea ferroviaria è circolare a binario semplice e solo alle stazioni esiste un secondo binario per consentire sorpassi o incroci tra treni.
Per motivi di sicurezza anche se due treni procedono nella stessa direzione uno solo di essi può ocuupare la tratta ferroviaria fra due stazioni, l'altro deve aspettare che il primo abbia raggiunto la stazione successiva per poter entrare nella tratta.



A monopoli esistono tre treni:
- il locale che ferma in tutte le stazioni
- il treno espresso N/S
- il treno espresso E/W.

Tutti i treni viaggiano nella direzione N->E->S->W (senso orario).
I treni espresso hanno la precedenza sul locale: se il locale è fermo in stazione e un espresso o è fermo nella stessa stazione o è sulla tratta ferroviaria immediatamente precedente allora il locale deve attendere che l'espresso abbia superato la stazione prima di poter ripartire.

Sono possibili casi di deadlock? PERCHÉ?
Sono possibili casi di starvation? PERCHÉ?
Scrivere il monitor Ferrovia
Scrivere le funzioni staz.* facendo uso di semafori
Se il treno locale percorresse il tragitto in senso antiorario cambierebbe la situazione in merito ai casi di deadlock? PERCHÉ?
E in merito ai casi di starvation? PERCHÉ?
(facoltativo) Scrivere un monitor staz che risolva il problema in questo caso (con il treno locale in senso antiorario e i treni espresso in senso orario).

```
process locale {
  staz.init(LOCALE,N);
  while(true) {
     ... sosta in stazione....
     staz.exit(LOCALE, N);
     staz.enter(LOCALE, E);
     ... sosta in stazione....
     staz.exit(LOCALE, E);
     staz.enter(LOCALE, S);
     ... sosta in stazione....
     staz.exit(LOCALE, S);
     staz.enter(LOCALE, W);
     ... sosta in stazione....
     staz.exit(LOCALE, W);
     staz.enter(LOCALE, N);
   }
}

process expr_ns {
  staz.init(EXPR_NS,S);
  while(true) {
     ... sosta in stazione....
     staz.exit(EXPR_NS, S);
     staz.enter(EXPR_NS, W);
     staz.exit(EXPR_NS, W);
     staz.enter(EXPR_NS, N);
     ... sosta in stazione....
     staz.exit(EXPR_NS, N);
     staz.enter(EXPR_NS, E);
     staz.exit(EXPR_NS, E);
     staz.enter(EXPR_NS, S);
   }
}

process expr_ew {
  staz.init(EXPR_EW,E);
  while(true) {
     ... sosta in stazione....
     staz.exit(EXPR_EW, E);
     staz.enter(EXPR_EW, S);
     staz.exit(EXPR_EW, S);
     staz.enter(EXPR_EW, W);
     ... sosta in stazione....
     staz.exit(EXPR_EW, W);
     staz.enter(EXPR_EW, N);
     staz.exit(EXPR_EW, N);
     staz.enter(EXPR_EW, E);
   }
}
```

```
// Costanti

static final int LOCALE   = 0
static final int EXPRESS  = 1
static final int EXPR_NW  = 2
static final int EXPR_SE  = 3

static final int NORD  = 0
static final int EST   = 1
static final int SUD   = 2
static final int OVEST = 3
```

Soluzione

```
--- Soluzione tramite await -------------------------------------
// Variabili
int mfree[] = new int[4];
int bfree[] = new int[4];

class Ferrovia
{
  void enter(int treno, int stazione)
  {
    await < mfree[stazione] > 0 ->
      bfree[(stazione-1)%4]++;
      mfree[stazione]--;
    >
  }

  void exit(int treno, int stazione)
  {
    await <  bfree[stazione] > 0 &&
             (treno != LOCALE || mfree[stazione] = 1) ->
      bfree[stazione]--;
      mfree[stazione]++;
  }
}

--- Traduzione await ---------------------------------------------------

Per ogni stazione, dovremmo avere: 1 condizione x entrare in stazione, 1
condizione x lasciare la stazione (espressi), 1 condizione x lasciare la
stazione (locali)

Semaphore[] msem = new Semaphore[4]; // Tutti inizializzati a zero
int[] mwaiting = new Semaphore[4]; // Tutti inizializzati a zero

Semaphore[][] bsem = new Semaphore[4][2]; // Tutti inizializzati a zero
int[][] bwaiting = new int[4][2]; // Tutti inizializzati a zero

Semaphore mutex = new Semaphore(1);

void enter(int treno, int stazione)
{
  mutex.P();
  if (mfree[stazione] == 0) {
    mwaiting[stazione]++;
    mutex.V();
    msem[stazione].P();
  }
  bfree[(stazione-1)%4]++;
  mfree[stazione]--;
  SIGNAL();
}
```

```
void exit(int treno, int stazione)
{
  mutex.P();
  if (treno == LOCALE) {
    if (bfree[stazione] == 0 || mfree[stazione] == 0)) {
      bwaiting[stazione][LOCALE]++;
      mutex.V();
      bsem[stazione][LOCALE].P();
    }
  } else {
    if (bfree[stazione] == 0) {
      bwaiting[stazione][EXPRESS]++;
      mutex.V();
      bsem[stazione][EXPRESS].P();
    }
  }
  bfree[stazione]--;
  mfree[stazione]++;
  SIGNAL();
}

void SIGNAL()
{
  // 12 possibili casi, più ultimo caso else, un po' troppi
}
```

Meglio lavorare direttamente con gli indici:

```
// Codice per signal in enter
if (bwaiting[(stazione-1)%4][EXPRESS] > 0) {
  // Precedenza agli espressi
  bwaiting[(stazione-1)%4][EXPRESS]++;
  bsem[(stazione-1)%4][EXPRESS].V();
} else if (bwaiting[(stazione-1)%4][LOCAL] > 0) {
  // Precedenza agli espressi
  bwaiting[(stazione-1)%4][LOCAL]++;
  bsem[(stazione-1)%4][LOCAL].V();
} else {
  mutex.V();
}

// Codice per signal in exit
if (mwaiting[stazione] > 0) {
  mwaiting[stazione]++; msem[stazione].V();
} else {
  mutex.V();
}
```