

Zadanie 1:

Synchronizacja programu "Wyścig" przy pomocy wait() i notify()

```
3 import static java.lang.Thread.sleep;
4
5 public class main {
6
7
8     public static void main(String args[]){
9         MyCounter counter=new MyCounter();
10        int num=1000000;
11        DecCntr dc=new DecCntr(counter,num);
12        IncCntr ic=new IncCntr(counter,num);
13        System.out.println("START!");
14        new Thread(dc).start();
15        new Thread(ic).start();
16        try { sleep(num); } catch (InterruptedException e){}
17        System.out.println("                FINISHED : " + counter.val);
18
19
20
21    }
22 }
```

```
1     public class MyCounter {
2         int val;
3         MyCounter() { this.val=0; }
4
5
6
7         public synchronized void inc() {
8             if (this.val > 0) {
9                 try {
10                     wait();
11                 } catch (InterruptedException e) {
12                     System.out.println("    INTERRUPTED    ");
13                 }
14             }
15             this.val++;
16             System.out.println(val + " ++ ");
17             notify();
18         }
19
20         public synchronized void dec() {
21             if (this.val <= 0) {
22                 try {
23                     wait();
24                 } catch (InterruptedException e) {
25                     System.out.println("    INTERRUPTED    ");
26                 }
27             }
28             this.val--;
29             System.out.println(val + "  --");
30             notify();
31         }
32     }
```

```

1 public class IncCtr implements Runnable{
2     MyCounter cnt;
3     int num;
4
5     IncCtr(MyCounter c, int n){ this.cnt=c; this.num=n;}
6
7     public void run() {
8         System.out.println("inc");
9         for (int i = 0; i < num; i++) {
10             cnt.inc();
11         }
12         System.out.println(cnt.val+" | finished +++");
13     }
14 }
15
16 }

```

```

1 public class DecCtr implements Runnable{
2     MyCounter cnt;
3     int num;
4
5     DecCtr(MyCounter c, int n){ this.cnt=c; this.num=n; }
6
7     public void run() {
8         System.out.println("dec");
9         for (int i = 0; i < num; i++) {
10             cnt.dec();
11         }
12         System.out.println(cnt.val+" | finished ---");
13     }
14 }

```

DecCtr.java x IncCtr.java x Semafor.java x MyCounter.java x main.java x

```

1 public class Semafor {
2     private boolean stan;
3
4     public Semafor ( ) { stan=true; }
5
6
7
8     public synchronized void P ( ) {
9         while (!stan) {
10             try { wait(); } catch (InterruptedException e) { }
11         }
12         stan = false;
13     }
14
15     public synchronized void V ( ) {
16         if(!stan) stan=true;
17         notify();
18     }
19 }
20

```

Run: main x main x

```

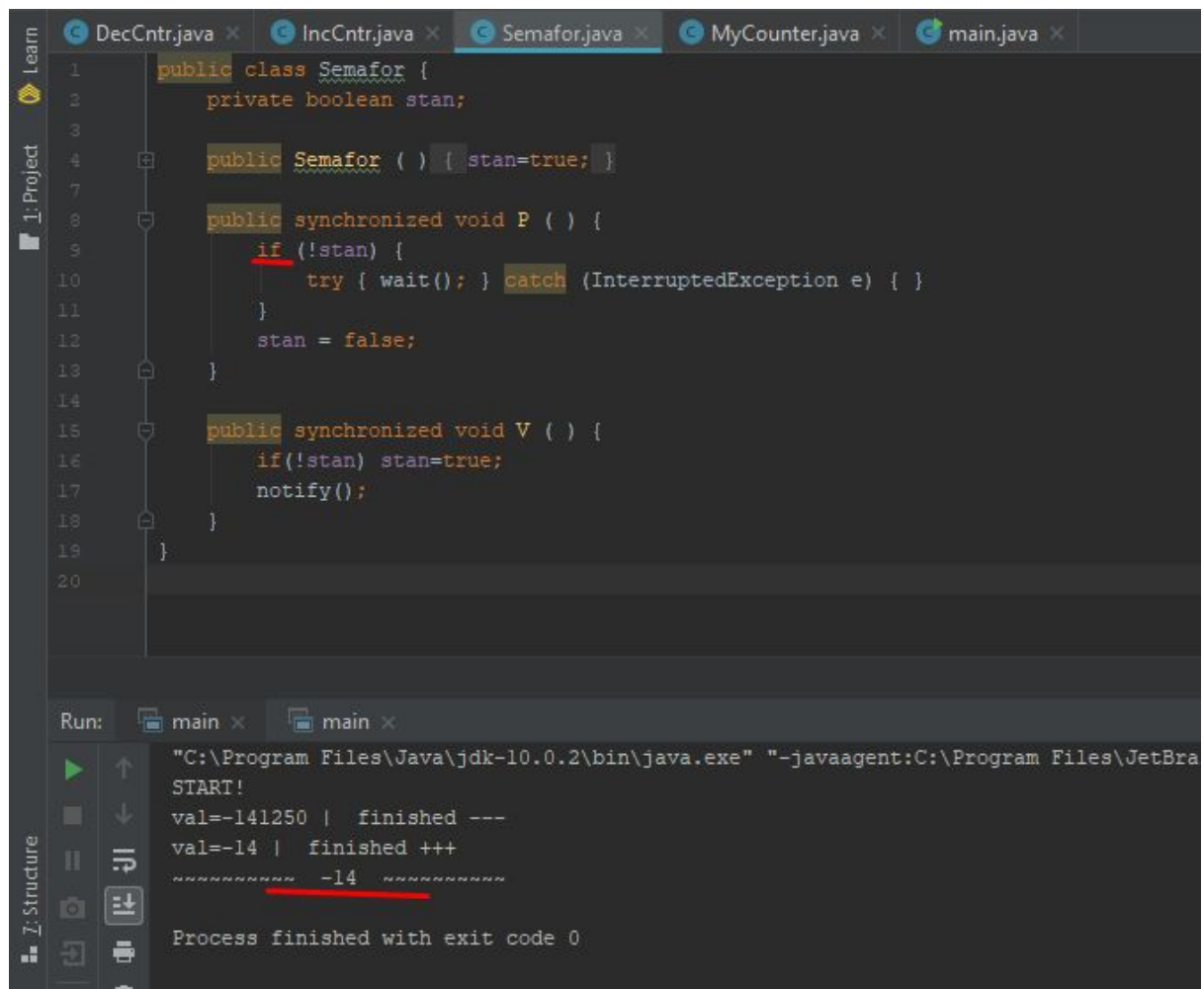
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
START!
val=0 | finished ---
val=155973 | finished +++
~~~~~ 0 ~~~~~
Process finished with exit code 0

```

Zadanie 2:

Synchronizacja programu “Wyścig” przy pomocy wait() i notify()

Nie możemy wykorzystać if ponieważ w momencie kiedy okazuje się że stan=true może on zostać zmieniony innym wątkiem, musimy zapewnić jeszcze jedno sprawdzenie i tylko wtedy możemy wykonać działanie.



```
1 public class Semafor {
2     private boolean stan;
3
4     public Semafor ( ) { stan=true; }
5
6
7
8     public synchronized void P ( ) {
9         if (!stan) {
10             try { wait(); } catch (InterruptedException e) { }
11         }
12         stan = false;
13     }
14
15     public synchronized void V ( ) {
16         if(!stan) stan=true;
17         notify();
18     }
19 }
20
```

Run: main x main x

```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBra
START!
val=-141250 | finished ---
val=-14 | finished +++
~~~~~ -14 ~~~~~
Process finished with exit code 0
```

Widzimy że w przypadku wykorzystania IF w wyniku nie mamy 0.

Zadanie 3:

Zaimplementować semafor licznikowy

Semafor binarny nie jest szczególnym przypadkiem semafora ogólnego.

Semafor binarny nie pamięta liczby operacji podniesienia. Próba podnieść już podniesiony semafor binarny jest błędem. Semafor binarny może zastąpić semafor licznikowy jeśli realizuje wzajemne wykluczanie.

```
1  import java.util.ArrayList;
2
3  public class CountSem {
4      private int resources=0;
5      private int available=0;
6      private ArrayList<Semafor> semafors;
7
8      CountSem(int r){
9          resources=r;
10         semafors=new ArrayList<>();
11         for(int i=0;i<r;i++){
12             Semafor sem=new Semafor();
13             semafors.add(sem);
14             available++;
15         }
16     }
17
18     public synchronized void P ( ) {
19         while (available==0) {
20             try { wait(); } catch (InterruptedException e) { }
21         }
22         available--;
23         semafors.get(available).P();
24     }
25
26     public synchronized void V ( ) {
27         if(available!=resources) {
28             semafors.get(available).V();
29             available++;
30         }
31     }
32 }
33
```