TITLE
Laboratory 7. Monitors and condition synchronization

PREREQUISITES
- OOP basic knowledge
- Java programming language basic knowledge
- Algorithms

RESOURCES
- Course slides
- Java Tutorial
- Java Concurrency

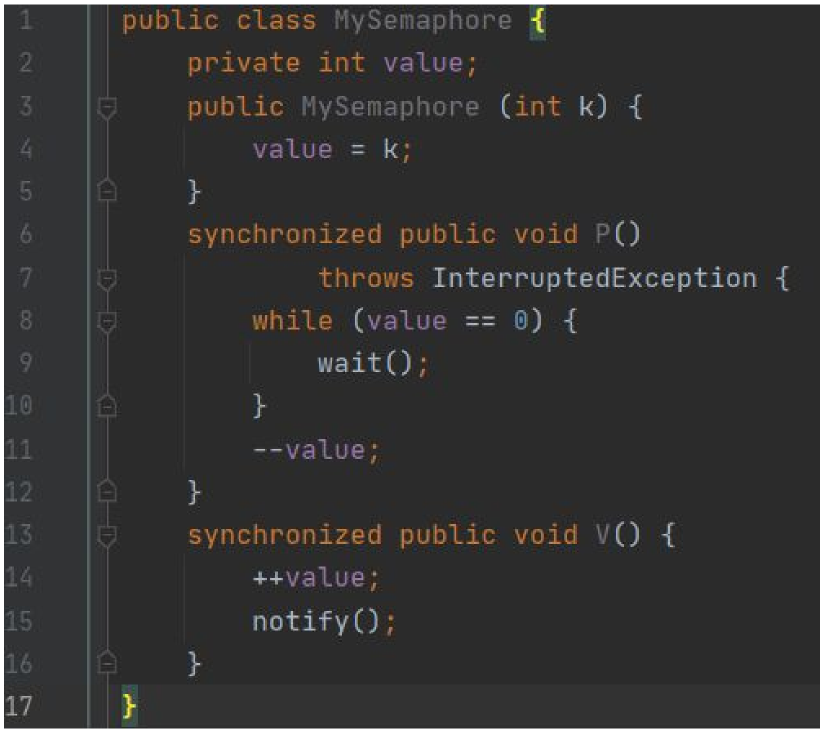
LABORATORY INSTRUCTIONS
Each object in Java has associated an *intrinsic lock* that controls the access of the threads to that object. Any thread that wants to access (exclusively) the object, must take control of this lock, and then release the lock, after the termination of access to the object.

We presented ***synchronized*** methods and blocks in Lab 4. You can find other details in Chapter 5 slides.

public final void wait() throws InterruptedException
It forces the thread that executes it to release the lock and to add it to the thread queue of this object. It is similar to the *waitC* method [!see Chapter 5 slides].

public final void notify()
This method takes a thread from the thread queue (if not empty) and puts it in the *ready* state. It is similar to the *signalC* method [!see Chapter 5 slides].

public final void notifyAll()
Extracts all threads from the thread queue (if not empty) and puts them in the *ready* thread.



```java
public class MySemaphore {
    private int value;
    public MySemaphore (int k) {
        value = k;
    }
    synchronized public void P()
            throws InterruptedException {
        while (value == 0) {
            wait();
        }
        --value;
    }
    synchronized public void V() {
        ++value;
        notify();
    }
}
```

Fig. 1 A semaphore implementation using monitors.

LABORATORY TASKS

1. Implement in Java the Producer-Consumer example in Chapter 5 slides.
2. Implement the “Dining philosophers problem” (see Chapter 4 slides) using monitors.