# Elaboration „Thread 2 (Thread Communication)“

## Critical Section

The solution to these problems lies in the concept of a **critical section**. A critical section is a block of code that accesses a shared resource and can’t be executed by more than one thread at the same time. To help programmers implement critical sections, Java (and almost all programming languages) offers synchronization mechanisms. When a thread wants access to a critical section, it uses one of these synchronization mechanisms to find out whether there is any other thread executing the critical section.

If not, the thread enters the critical section. If yes, the thread is suspended by the synchronization mechanism until the thread that is currently executing the critical section ends it. When more than one thread is waiting for a thread to finish the execution of a critical section, JVM chooses one of them and the rest wait for their turn.

Java language offers two basic synchronization mechanisms:

* The synchronized keyword
* The Lock interface and its implementations

In this article, we explore the use of synchronized keyword method to perform synchronization mechanism in Java.

## Semaphore

A semaphore controls access to a shared resource through the use of a counter. If the counter is greater than zero, then access is allowed. If it is zero, then access is denied. What the counter is counting are permits that allow access to the shared resource. Thus, to access the resource, a thread must be granted a permit from the semaphore.

## Semaphore in Java

private final Semaphore available = new Semaphore(MAX\_AVAILABLE, true);

public void acquire()

throws [InterruptedException](https://docs.oracle.com/javase/7/docs/api/java/lang/InterruptedException.html)

//Acquires a permit from this semaphore, blocking until one is available, or the thread is [interrupted](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html#interrupt()).

public void release()

// Releases a permit, returning it to the semaphore.