Unit 2. Task Synchronization

Communication mechanisms

By the use of concurrency, we can make threads to cooperate to accomplish some goal, for this, we need them to **communicate between themselves**, and to be **synchronized**.

To be able to communicate between themselves, there are two essential mechanisms:

- **Shared memory:** By the use of a shared variable that establishes who is accessing each resource, we can implement mechanisms to stop any thread to access a resource being accessed by other thread.
- **Message Exchange:** By the use of messages between threads that can be in separate spaces of the memory, we can ensure the proper coordination of tasks.

Synchronization

There are two main types of synchronization, that will be helpful to avoid possible race conditions.

Mutual exclusion

By establishing that only one thread can access simultaneously a resource, we avoid situations that are prone to errors.

Conditional synchronization

Other way to orchestrate threads is by using conditional synchronization. Basically threads have to wait a certain condition is fulfilled to be able to access a resource, otherwise, they will wait for that condition to be fulfilled.

Critical section

Race conditions appear in critical sections, i.e. sections with shared variables that can be accessed simultaneously.

It is in this section where we have to apply some of the previous mechanisms to protect the shared variables.

Locks

Locks are objects that can only have two states, **open or closed**, and they are used to allow the access to a resource, there are two situations:

Already Open

→ Close the access

→ Access the resource

→ Open the lock

Closed

→ Wait until opened

→ Close lock

→ Operate

→ Close again

With this procedure, accession to the critical section becomes an atomic action, as it is a indivisible operation for the threads that access the data.

Implementation in java

First of all, the methods that will be inside the critical section **must** be labeled with the `synchronized` label, this will create an implicit lock over each of these methods. Java will internally manage the coordination of the threads that call these methods.

Also, if instead of complete methods you want to declare `synchronized statements`, you can also do it, as it is specified in the file **"Concurrency with Java"**.