Object Oriented Programming with Java (Subject Code: BCS-403)

Unit 2

Lecture 20

Lecture 20

- Synchronizing Threads
- Inter-thread Communication

Synchronizing Threads

- Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.
- In order to overcome this problem, we have thread synchronization.

Synchronization means coordination between multiple processes/threads.

Why use Java Synchronization?

Java Synchronization is used to make sure by some synchronization method that only one thread can access the resource at a given point in time.

Java Synchronized Blocks

Java provides a way of creating threads and synchronizing their tasks using synchronized blocks.

A synchronized block in Java is synchronized on some object.

All synchronized blocks synchronize on the same object and can only have one thread executed inside them at a time.

All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.

General Form of Synchronized Block

- Only one thread can execute at a time.
- sync_object is a reference to an object
- whose lock associates with the monitor.
- The code is said to be synchronized on
- the monitor object

```
synchronized(sync_object)
{
   // Access shared variables and other
   // shared resources
}
```

Types of synchronization

There are two types of synchronization that are as follows:

- Process synchronization
- Thread synchronization

1. Process Synchronization in Java

Process Synchronization is a technique used to coordinate the execution of multiple processes. It ensures that the shared resources are safe and in order.

2. Thread Synchronization in Java

Thread Synchronization is used to coordinate and ordering of the execution of the threads in a multi-threaded program.

There are two types of thread synchronization are mentioned below:

- Mutual Exclusive
- Cooperation (Inter-thread communication in Java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data.

Inter-thread Communication in Java

- Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.
- Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.

It is implemented by following methods of Object class:

- wait()
- notify()
- notifyAll()

1) wait() method

- The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
- The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

2) notify() method

The notify() method wakes up a single thread that is waiting on this object's monitor.

If any threads are waiting on this object, one of them is chosen to be awakened.

The choice is arbitrary and occurs at the discretion of the implementation.

Syntax:

```
public final void notify()
```

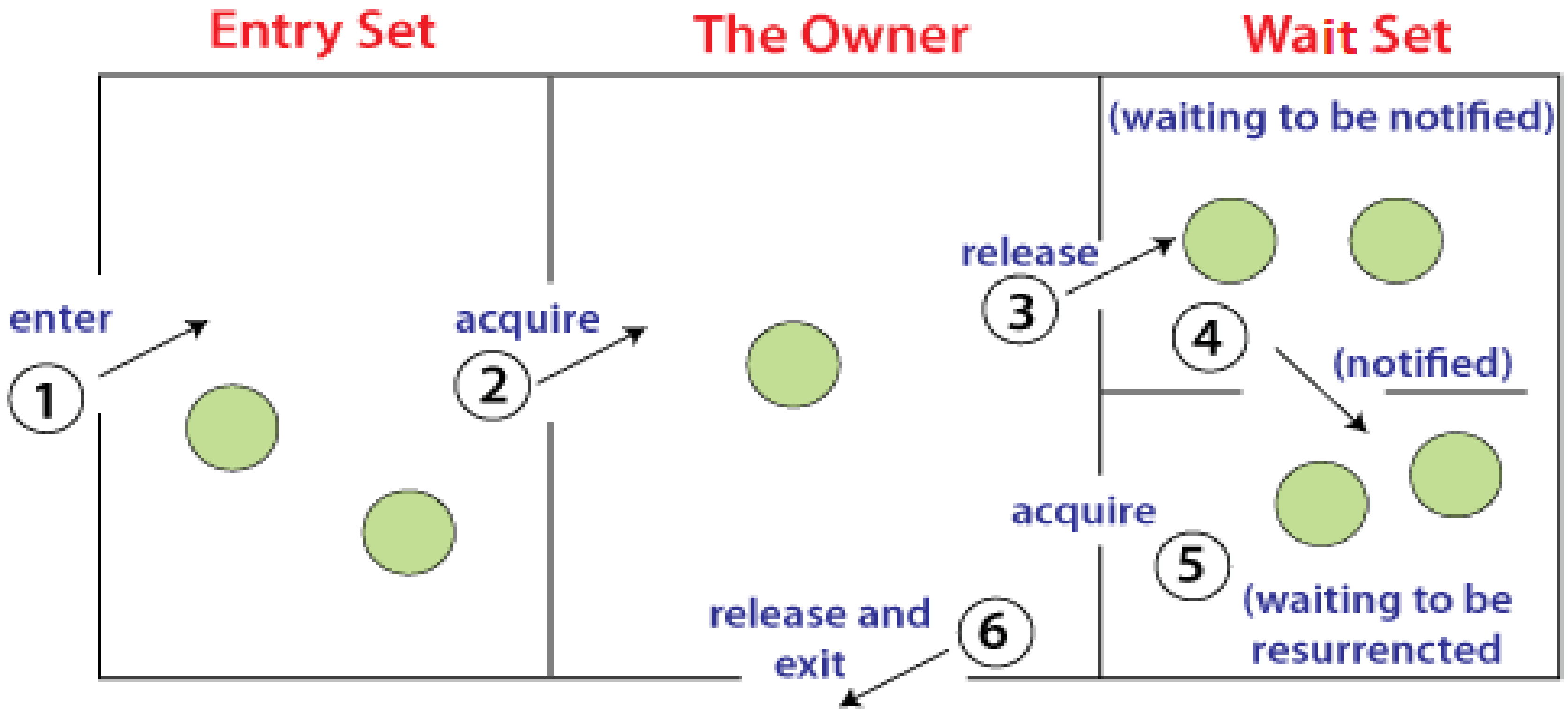
3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

Syntax:

public final void notifyAll()

Understanding the process of inter-thread communication



process of inter-thread communication

1. Threads enter to acquire lock.
2. Lock is acquired by one thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.