In Java, a Semaphore is a synchronization primitive that allows you to control access to a shared resource by a fixed number of threads. It is part of the java.util.concurrent package and is used to manage concurrent access to resources in multi-threaded applications.

A Semaphore maintains a set of permits, which represent the number of concurrent threads allowed to access the shared resource. It has two main methods:

acquire(): This method is used by a thread to acquire a permit from the Semaphore. If there is a permit available, the thread will acquire it and proceed. If no permits are available, the thread will be blocked until a permit becomes available.

release(): This method is used by a thread to release a permit back to the Semaphore. After releasing a permit, it becomes available for other threads to acquire.

Here's a simple example to demonstrate how to use a Semaphore in Java:

java

Copy code

import java.util.concurrent.Semaphore;

public class SemaphoreExample {

public static void main(String[] args) {

int numberOfPermits = 3; // Number of permits available in the semaphore

Semaphore semaphore = new Semaphore(numberOfPermits);

// Create and start multiple threads

for (int i = 1; i <= 5; i++) {

Thread thread = new Thread(new Task(semaphore, i));

thread.start();

}

}

static class Task implements Runnable {

private final Semaphore semaphore;

private final int taskId;

public Task(Semaphore semaphore, int taskId) {

this.semaphore = semaphore;

this.taskId = taskId;

}

@Override

public void run() {

try {

System.out.println("Task " + taskId + " is trying to acquire a permit.");

semaphore.acquire();

System.out.println("Task " + taskId + " has acquired a permit and is using the resource.");

Thread.sleep(2000); // Simulate resource usage

System.out.println("Task " + taskId + " has released the permit and finished.");

semaphore.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

In this example, we create a Semaphore with three permits. Five threads are created and started to execute a Task class. Each task tries to acquire a permit from the Semaphore. If there are available permits, the task acquires one, and after a simulated resource usage (sleeping for 2 seconds), it releases the permit back to the Semaphore. If there are no permits available, the task waits until a permit becomes available.

Semaphore is a useful tool for scenarios where you want to limit concurrent access to a shared resource, such as connections to a database, read/write access to a file, or access to a pool of resources.