

# Java2 Lab 16

## (Multithread)

### [Experimental Objective]

Master the basic use of multi-threading

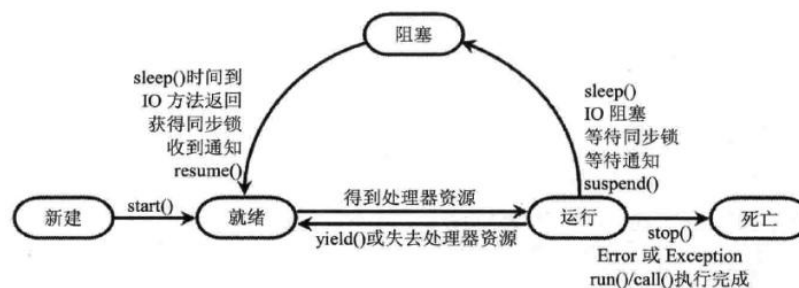
### [Multithread]

Multithreading is the simultaneous execution of multiple threads in a single program.

Thread life cycle:

There are 5 states, New, Runnable, Running, Blocked, and Dead.

启动线程使用 `start()` 方法，而不是 `run()` 方法！永远不要调用线程对象的 `run()` 方法！调用 `start()` 方法来启动线程，系统会把该 `run()` 方法当成线程执行体来处理；但如果直接调用线程对象的 `run()` 方法，则 `run()` 方法立即就会被执行，而且在 `run()` 方法返回之前其他线程无法并发执行——也就是说，如果直接调用线程对象的 `run()` 方法，系统把线程对象当成一个普通对象，而 `run()` 方法也是一个普通方法，而不是线程执行体。



### 【Case 1 multi-threaded creation】

#### (1) The method of inheriting the Thread class creates a thread as follows:

1. Define a subclass that inherits from the Thread class, and override the `run()` method, `run()`, which is the specific task or function to be run by the new thread in the future.
2. instantiate (new) the subclass just defined
3. Run the start method of this new object. It's important to remember that it's the start method, and only then will a new thread be started. If you run the run method, it is still a simple single-threaded execution.

`package` multithread;

```

public class CreateThread1 extends Thread {
    public void run() {
        for (int i = 0; i <= 100; i++) {
            System.out.println(getName());
        }
    }
}
  
```

```

    }
}

public static void main(String[] args) {
    for (int i = 0; i < 100; i++) {
        if (i % 10 == 0) {
            new CreateThread1().start();
            new CreateThread1().start();
        }
    }
}
}

```

You can see that the two threads created are running at the same time:



```

<terminated> CreateThread1 [Java Application] C:\Program Files\Java\jre1.8.0_181\bin
Thread-0
Thread-0
Thread-0
Thread-0
Thread-5
Thread-5
Thread-6
Thread-6

```

## (2) Implement the Runnable interface and create the thread as follows:

- 1, define a class, this class needs to implement the Runnable interface, still need to rewrite the run method in the interface in this class, like method 1, this run method is also the future thread executive
  - 2, instantiate (new) the class just defined
  - 3, instantiate (new) a Thread class, and use A as the target, run the start method
- package** multithread;

```

public class CreateThread2 implements Runnable {
    public void run() {
        for (int i = 0; i <= 100; i++) {
            System.out.println(Thread.currentThread().getName());
        }
    }

    public static void main(String[] args)
    {
        for(int i=0;i<100;i++)
        {
            if(i%10==0)
            {
                CreateThread2 st=new CreateThread2();
                new Thread(st,"name1").start();
                new Thread(st,"name2").start();
            }
        }
    }
}

```

**【Exercise 1 Multithreaded creation】**

Please use these two methods to create 2 threads. The thread Thread 1 is used to create the thread 1 to print the number 1-52, and the thread 2 is created using the Runnable interface method to print the letter A-Z. (order is not required for the time being)

**【Case 2 Synchronization and mutual exclusion】****(1) synchronized:**

In the following example, if multiple processes need to use the same resource, simultaneous access will occur:

```
package multithread;

public class SynchronizedTest2 implements Runnable {
    public void run() {
        this.UsingResource();
    }

    public static void main(String[] args) {
        SynchronizedTest2 st = new SynchronizedTest2();
        new Thread(st, "name1").start();
        new Thread(st, "name2").start();
    }

    private void UsingResource() {
        cnt++;
        System.out.println(Thread.currentThread().getName() + " is the" +
cnt + "th using resource");
    }

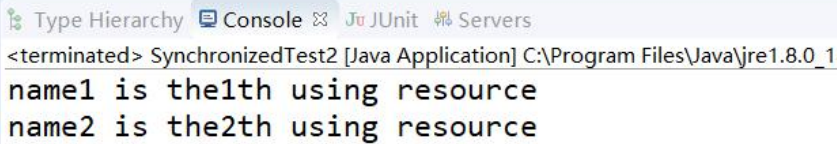
    int cnt = 0;
}
```

It turns out that two threads are using resources at the same time(UsingResource()):

```
<terminated> SynchronizedTest2 [Java Application] C:\Program Files\Java\jre
name2 is the2th using resource
name1 is the2th using resource
```

If you add a synchronized modifier to the resources you need to access, you can see the effect:

```
private synchronized void UsingResource() {
    cnt++;
    System.out.println(Thread.currentThread().getName() + " is the" +
cnt + "th using resource");
}
```



```
<terminated> SynchronizedTest2 [Java Application] C:\Program Files\Java\jre1.8.0_1
name1 is the1th using resource
name2 is the2th using resource
```

## (2) lock:

The use of lock to lock a variable or member function of a mutually exclusive access can also achieve the purpose of mutual exclusion access.

**package** multithread;

**import** java.util.concurrent.locks.ReentrantLock;

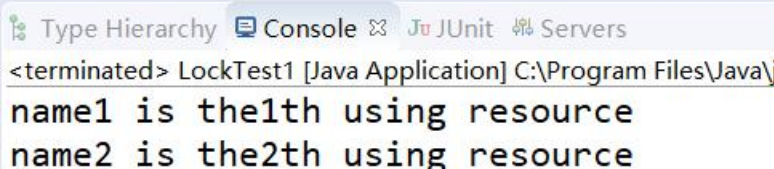
```
public class LockTest1 implements Runnable {
    public void run() {
        this.UsingResource();
    }

    public static void main(String[] args) {
        LockTest1 st = new LockTest1();
        new Thread(st, "name1").start();
        new Thread(st, "name2").start();
    }

    private void UsingResource() {
        lock.lock();
        cnt++;
        System.out.println(Thread.currentThread().getName() + " is the" +
cnt + "th using resource");
        lock.unlock();
    }

    int cnt = 0;
    ReentrantLock lock = new ReentrantLock();
}
```

It can be seen that when accessing resources, it is also possible to protect resources from being accessed by multiple threads by locking and unlocking.



```
<terminated> LockTest1 [Java Application] C:\Program Files\Java\
name1 is the1th using resource
name2 is the2th using resource
```

## (3) wait and notify

The wait() method causes the thread to enter a wait state, while notify() can wake the wait state. Such a synchronization mechanism is well suited to the producer, consumer model: the consumer consumes a resource and the producer produces the resource. When the resource is missing, the consumer calls the wait() method to self-block, waiting for the producer's production; after the producer finishes production, call notify/notifyAll() to wake up the consumer for consumption.

```
public class WaitNotifyTest {
```

```

public static void main(String[] args) {
    final Object object = new Object();
    Thread t1 = new Thread() {
        public void run()
        {
            synchronized (object) {
                System.out.println("T1 start!");
                try {
                    object.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("T1 end!");
            }
        }
    };
    Thread t2 = new Thread() {
        public void run()
        {
            synchronized (object) {
                System.out.println("T2 start!");
                object.notify();
                System.out.println("T2 end!");
            }
        }
    };

    t1.start();
    t2.start();
}

```

### 【Exercise 2 Synchronization and Mutual Exclusion】

Please make appropriate modifications based on the output 1-52 and A-Z threads in Exercise 1. Add synchronization and mutual exclusion, and the output is required to be "12A34B56C...5152Z".