Lecture 3 - Java Multi-Threading

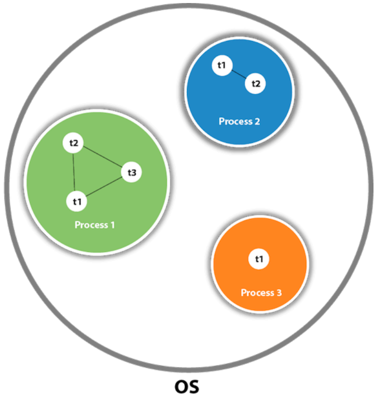
Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

CSC-1004: Computational Laboratory Using Java
Course Page: [Click]

Multitasking in Java

Multitasking is a process of executing multiple tasks simultaneously.



Multitasking can be achieved in two ways:

Multitasking in Java

Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. It allocates a separate memory area.
- A process is heavy-weight.
- Cost of communication between the process is high. Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

Multitasking in Java

Thread-based Multitasking (Multithreading)

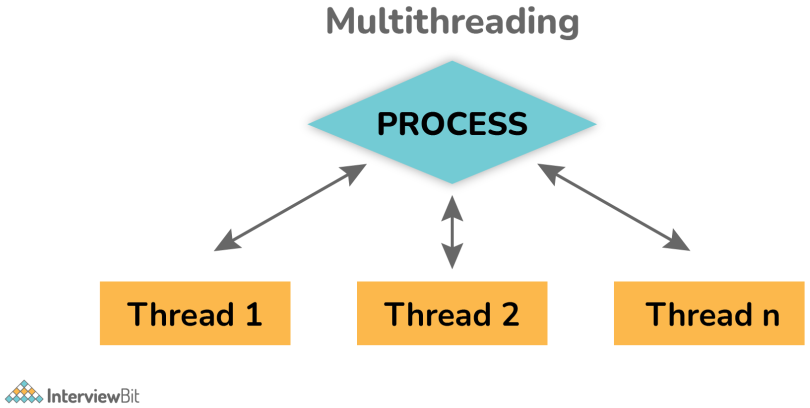
What is Thread in java?

- A thread is a lightweight subprocess, the smallest unit of processing.
- Threads are independent. If there is an exception occurs in one thread, it doesn't affect other threads. It uses a shared memory area.
- Threads use a **shared memory** area. They don't allocate separate memory areas so save memory, and context-switching between the threads takes less time than the process.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.



Multithreading in Java

Advantages of Java Multithreading.

- It doesn't block the user because threads are independent and you can perform multiple operations at the same time.
- You can perform many operations together, so it saves time.
- Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. **New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. **New:**
2. **Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is runnable, and the other is running.
    - 2.1 *Runnable*: A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. The thread scheduler provides the thread time to run.
    - 2.2 *Running*: When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. **New:**
2. **Active:**
3. **Blocked / Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state. For example, a thread (A) may want to print some data from the printer. However, at the same time, the other thread (B) is using the printer to print some data. Therefore, thread A has to wait for thread B to use the printer. Thus, thread A is in the blocked state.

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. **New:**
2. **Active:**
3. **Blocked / Waiting:**
4. **Timed Waiting:** Sometimes, waiting leads to starvation. For example, a thread (A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B. A real example of timed waiting is when we invoke the sleep(#Time) method on a specific thread.

Life cycle of a Thread (Thread States)

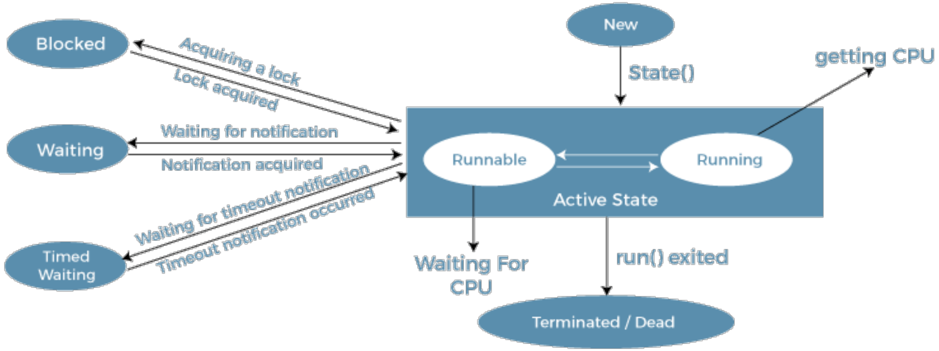
In Java, a thread always exists in any one of the following states. These states are:

1. **New:**
2. **Active:**
3. **Blocked / Waiting:**
4. **Timed Waiting:**
5. **Terminated:** A thread reaches the termination state because of the following reasons:1) When a thread has finished its job, then it exists or terminates normally. 2) Abnormal termination: It occurs when some unusual events such as an unhandled exception or segmentation fault.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Life cycle of a Thread (Thread States)

The following diagram shows the different states involved in the life cycle of a thread.



Life Cycle of a Thread

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 1: Create a thread in Java by implementing a Runnable interface. For example:

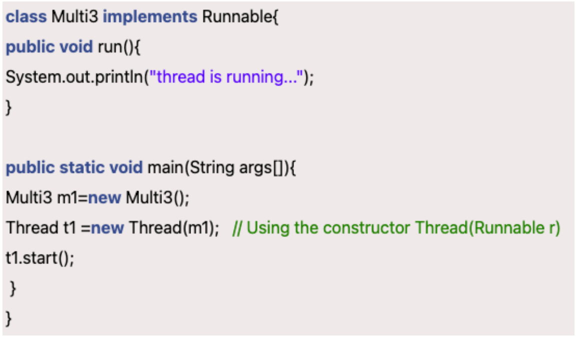
```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 1: Create a thread in Java by implementing a Runnable interface. For example:



```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

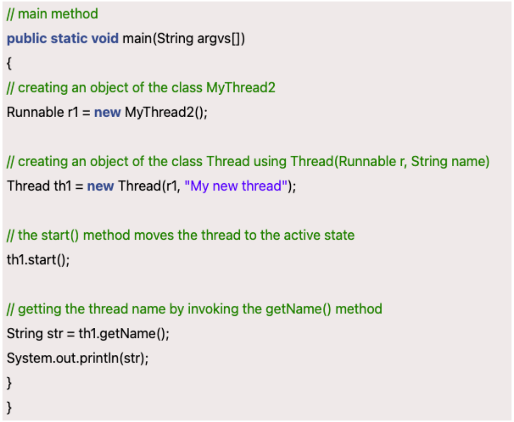
Output:

```
thread is running...
```

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 2: Create a thread in Java by using the Thread Class: Thread(Runnable r, String name). For example:

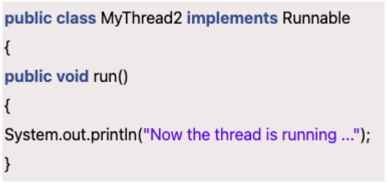


```
// main method
public static void main(String argvs[])
{
// creating an object of the class MyThread2
Runnable r1 = new MyThread2();

// creating an object of the class Thread using Thread(Runnable r, String name)
Thread th1 = new Thread(r1, "My new thread");

// the start() method moves the thread to the active state
th1.start();

// getting the thread name by invoking the getName() method
String str = th1.getName();
System.out.println(str);
}
}
```



```
public class MyThread2 implements Runnable
{
public void run()
{
System.out.println("Now the thread is running ...");
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 2: Create a thread in Java by using the Thread Class: Thread(Runnable r, String name). For example:

```
// main method
public static void main(String argvs[])
{
// creating an object of the class MyThread2
Runnable r1 = new MyThread2();

// creating an object of the class Thread using Thread(Runnable r, String name)
Thread th1 = new Thread(r1, "My new thread");

// the start() method moves the thread to the active state
th1.start();

// getting the thread name by invoking the getName() method
String str = th1.getName();
System.out.println(str);
}
}
```

Output:

My new thread

Now the thread is running ...

```
public class MyThread2 implements Runnable
{
public void run()
{
System.out.println("Now the thread is running ...");
}
```

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Question and Answering (Q&A)

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen