Parul® University

Object Oriented Programming with JAVA

Ms.Vaishalee Joishar,Cyber Secutiy Trainer

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-9

Multi Threading

Multithreading

- Multithreading in Java is a process of *executing multiple threads* simultaneously.
- A thread is a *lightweight sub-process*, the smallest unit of processing.
- Multiprocessing and multithreading, both are used to achieve multitasking.
- Threads use a *shared memory area*. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- Java Multithreading is mostly used in *games, animation,* etc.

Cont..

Advantages of Java Multithreading

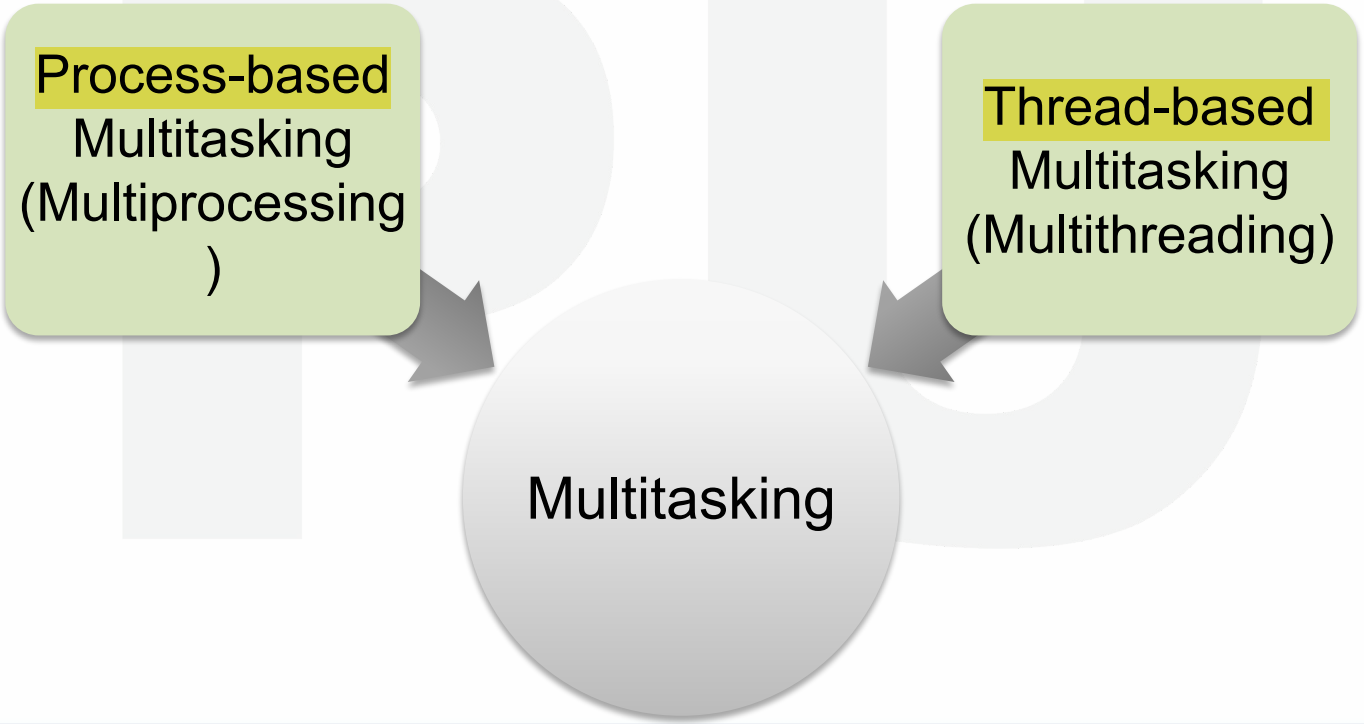
1) It doesn't block the user because threads are independent and you can perform multiple operations at the same time.

2) You can perform many operations together, so it saves time.

3) Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

Multitasking

- Multitasking is a process of executing multiple tasks simultaneously.
- We use multitasking to utilize the CPU.
- Multitasking can be achieved in two ways:



Cont..

1. **Process-based Multitasking (Multiprocessing)**
   - Each process has an address in memory. In other words, each process allocates a separate memory area.
   - A process is heavyweight.
   - 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.

2. **Thread-based Multitasking (Multithreading)**
   - Threads share the same address space.
   - A thread is lightweight.
   - Cost of communication between the thread is low.

Procce Based VS. Thread Based

Process Based Multi-tasking	Thread Based Multi-tasking
➢ A process is essence of program that is executing which running parallel	➢ Thread is a such part of multithreaded program which defines separate path for execution
➢ It allows you to run java compiler and text editor at a same time.	➢ It doesn't support java compiler and text editor run simultaneously. Both have performed different tasks.
➢ Process multitasking has more overhead than thread multitasking.	➢ Thread multitasking has less overhead than process multitasking.
➢ Here, it is unable to gain access over idle time of CPU.	➢ It allows taking gain access over idle time taken by CPU.
➢ It is not under control of java.	➢ It is totally under control of java.
➢ Process based multitasking is comparatively heavyweight process comparing to thread based multitasking.	➢ Thread based multitasking is known to be lighter eight process.
➢ Inter-process communication is expensive and limited.	➢ Inter-thread communication is inexpensive and context switching from one thread to other.
➢ It has slower data rate multitasking.	It has faster data rate multithreading or tasting.

LifeCycle of Thread

A thread can always in any one of the five states. It can move from one state to other via variety of ways as shown in the fig.

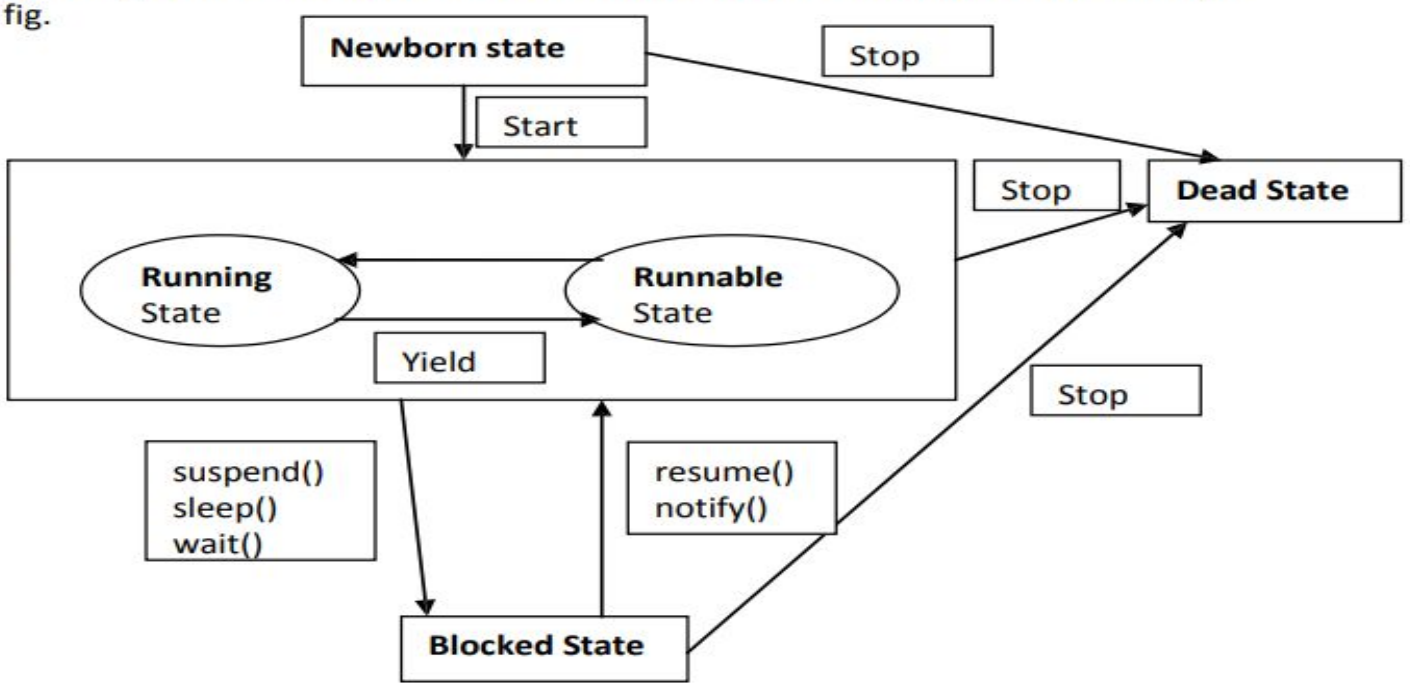


Fig: Life Cycle of Thread

Cont...

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.

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.

I. **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. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.

II. **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.

Cont...

3. Blocked or 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.

4.Terminated: A thread reaches the termination state because of the following reasons:

Thread Methods

Sl No	Method Name	Description
1	run()	Used to write the body of the thread
2	start()	Used to start the thread
3	sleep()	Used to make the thread sleep for milliseconds
4	suspend()	Used to suspend a running thread
5	wait()	Waits until further ordered
6	yield	Used to give control to other thread before it turn comes
7	Stop()	Used to stop the thread
8	resume()	Used to start the blocked thread
9	notify()	Used to notify the waiting thread
10	notifyAll()	Used to notify theall waiting threads

Creating a Thread in Java

There are two ways to create a Thread

- extending the Thread class
- implementing the Runnable interface

1) Extending Thread Class

- One way to create a thread is to create a new class that extends Thread, and then to create an instance of that class.
- The extending class must override the run( ) method, which is the entry point for the new thread.
- It must also call start( ) to begin execution of the new thread.

```
class NewThread extends Thread {
    NewThread() {
        super("Demo Thread");
        System.out.println("Child thread: " + this);
        start(); // Start the thread
    }
    public void run() {
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
    }}
```

```
class ExtendThread {
    public static void main(String args[]) {
        new NewThread(); // create a new thread
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Main Thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread
        interrupted.");
        }
        System.out.println("Main thread exiting.");
    }
}
```

2) Implementing Runnable Interface

- To implement thread using Runnable interface, Runnable interface needs to be implemented by the class.

```
class NewThread implements Runnable
```

- Class which implements Runnable interface should override the run() method which containts the logic of the thread.

```
public void run( )
```

- Instance of Thread class is created using following constructor.

```
Thread(Runnable threadOb, String threadName);
```

- Here threadOb is an instance of a class that implements the Runnable interface and the name of the new thread is specified by threadName.
- start() method of Thread class will invoke the run() method.

Suspending, Blocking and Stopping Threads

- Whenever we want stop a thread we can stop from running using "**stop()**" method of thread class. It's general form will be as follows:
- **Thread.stop();** This method causes a thread to move from running to dead state. A thread will also move to dead state automatically when it reaches the end of its method.
- Blocking Thread - A thread can be temporarily suspended or blocked from entering into the runnable and running state by using the following methods:
- **sleep()** —blocked for specified time.
- **suspend()** ----blocked until further orders.
- **wait()** --blocked until certain condition occurs.
- These methods cause the thread to go into the blocked state. The thread will return to the runnable state when the specified time is elapsed in the case of **sleep(),** the **resume()** method is invoked in the case of **suspend()**, and the **notify()** method is called in the case of **wait().**

Thread Synchronization

- When we start *two or more threads* within a program, there may be a situation when multiple threads try to *access the same resource* and finally they can produce unforeseen result due to concurrency issues.

- For example, if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file.

- So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time.

- Java programming language provides a very handy way of creating threads and synchronizing their task by using *synchronized methods & synchronized blocks.*

Problem without synchronization (Example)

```
class Table {
  void printTable(int n) {
    for (int i = 1; i <= 5; i++) {
      System.out.print(n * i + " ");
      try {
        Thread.sleep(400);
      } catch (Exception e) {
        System.out.println(e);
      }
    }
  }
}
```

```
class MyThread1 extends Thread {
  Table t;
  MyThread1(Table t) {
    this.t = t;
  }
  public void run() {
    t.printTable(5);
  }
}
```

```
C:\WINDOWS\system32\cmd.exe
D:\DegreeDemo\PPTDemo>javac TestSynchronization.java
D:\DegreeDemo\PPTDemo>java TestSynchronization
5 100 200 10 300 15 400 20 500 25
```

```
class MyThread2 extends Thread {
  Table t;
  MyThread2(Table t) {
    this.t = t;
  }
  public void run() {
    t.printTable(100);
  }
}
```

```
public class TestSynchronization {
  public static void main(String args[]){
    Table obj = new Table();
    MyThread1 t1 = new MyThread1(obj);
    MyThread2 t2 = new MyThread2(obj);
    t1.start();
    t2.start();
  }
}
```

Solution with synchronized method

```
class Table {
 synchronized void printTable(int n) {
    for (int i = 1; i <= 5; i++) {
      System.out.print(n * i + " ");
      try {
        Thread.sleep(400);
      } catch (Exception e) {
        System.out.println(e);
      }
    }
  }
}
```

```
class MyThread1 extends Thread {
  Table t;
  MyThread1(Table t) {
    this.t = t;
  }
  public void run() {
    t.printTable(5);
  }
}
```

```
class MyThread2 extends Thread {
  Table t;
  MyThread2(Table t) {
    this.t = t;
  }
  public void run() {
    t.printTable(100);
  }
}
```

```
public class TestSynchronization {
  public static void main(String args[]){
    Table obj = new Table();
    MyThread1 t1 = new MyThread1(obj);
    MyThread2 t2 = new MyThread2(obj);
    t1.start();
    t2.start();
  }
}
```

```
C:\WINDOWS\system32\cmd.exe
D:\DegreeDemo\PPTDemo>javac TestSynchronization.java
D:\DegreeDemo\PPTDemo>java TestSynchronization
5 10 15 20 25 100 200 300 400 500
```

Solution with synchronized blocks

```
class Table {
  void printTable(int n) {
    for (int i = 1; i <= 5; i++) {
      System.out.print(n * i + " ");
      try {
        Thread.sleep(400);
      } catch (Exception e) {
        System.out.println(e);
      }
    }
  }
}
```

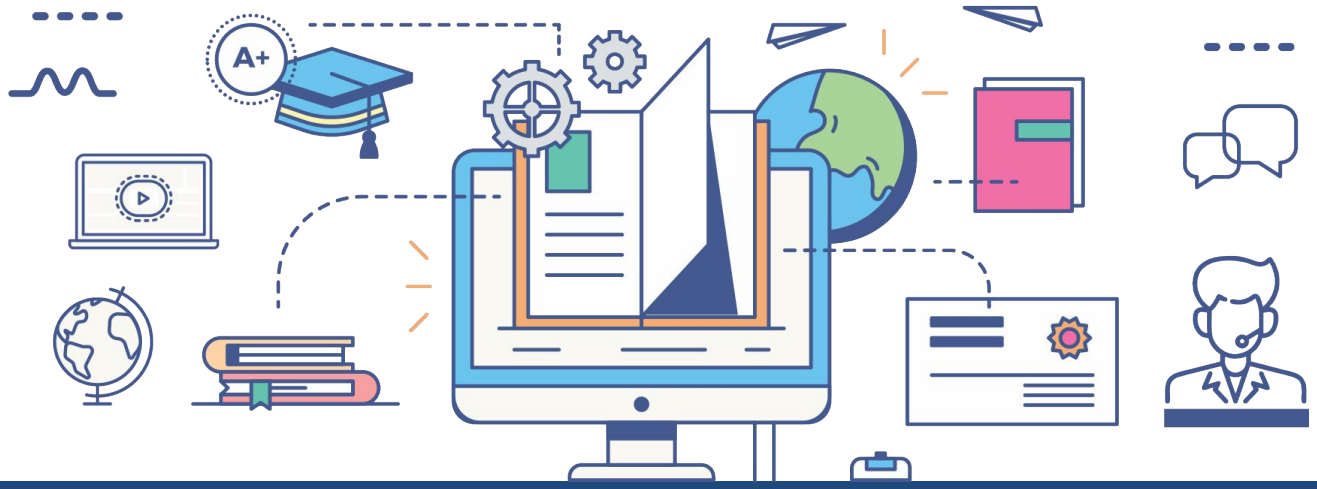
```
class MyThread1 extends Thread {
  Table t;
  MyThread1(Table t) {
    this.t = t;
  }
  public void run() {
    synchronized (t) {
      t.printTable(5);
    }
  }
```

```
class MyThread2 extends Thread {
  Table t;
  MyThread1(Table t) {
    this.t = t;
  }
  public void run() {
    synchronized (t) {
      t.printTable(100);
    }
  }
}
```

```
public class TestSynchronization {
  public static void main(String args[]){
    Table obj = new Table();
    MyThread1 t1 = new MyThread1(obj);
    MyThread2 t2 = new MyThread2(obj);
    t1.start();
    t2.start();
  }
}
```

```
C:\WINDOWS\system32\cmd.exe
D:\DegreeDemo\PPTDemo>javac TestSynchronization.java
D:\DegreeDemo\PPTDemo>java TestSynchronization
5 10 15 20 25 100 200 300 400 500
```

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in