Java Programming (TCS408)

Module 3

Multithreading

Objectives

After completing this section, students will be able to
▪ Understand Threads in Java and their use
▪ Create multithreaded programs
▪ Use synchronized methods or blocks to synchronize threads

Contents

Multitasking/Multithreading

- Multitasking is when multiple processes share common processing resources such as a CPU.

Example: When working in MS word, at the same time we are playing sings also at the same time. (2 task at the same time)

- Multi-threading extends the idea of multitasking into applications where we can subdivide specific operations within a single application into individual threads.

Example: When typing in MS word at the same time it is suggesting for spelling check. (only one process different subprocesses are there)

- These subprocesses are known as thread.

What is a Thread ?

- Threads allows a program to operate more efficiently by doing multiple things at the same time.

- Threads can be used to perform complicated tasks in the background without interrupting the main program.

- A thread is a lightweight sub-process, the smallest unit of processing.

- By default there is one thread in java program, i.e, main.

Application of thread

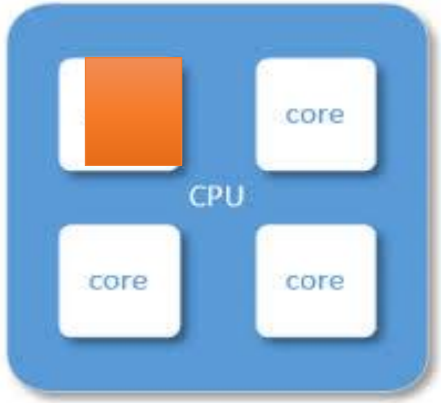
- ❑Ex: in a Web browser we may do the following tasks at the same time:
  - 1. scroll a page
  - 2. download an applet or image
  - 3. play sound
  - 4 print a page

- ❑Example: When multiple clients try to access the servlet, then multiple threads are created for that servlet and each client request is sent via a different thread.

- ❑In Games: when both the players can act at the same time.

Need of Thread

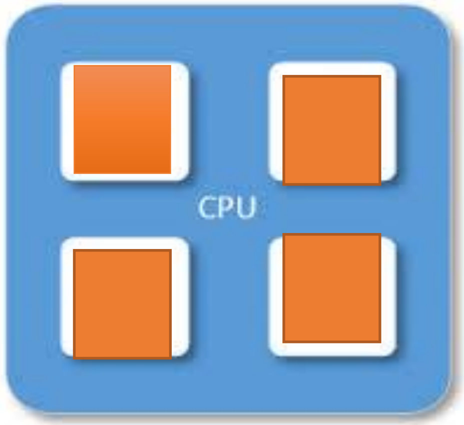
- Now a days, multicore processor are used. For example: we are using quadcore processor and there is an array of 500 elements, and we want to multiply each element by 2. Suppose it is taking 8seconds. If we are using one thread(main-t1) for this task it will occupy only one core of the quadcore processor.



Quad-core CPU

- If we want to use all four cores of the processor then we have create total four threads(t1,t2,t3,t4), so that processor can be utilized Completely and processing time can be reduced.

- These threads will work in parallel and each thread will take (8/4=2seconds) to complete the task.



Quad-core CPU

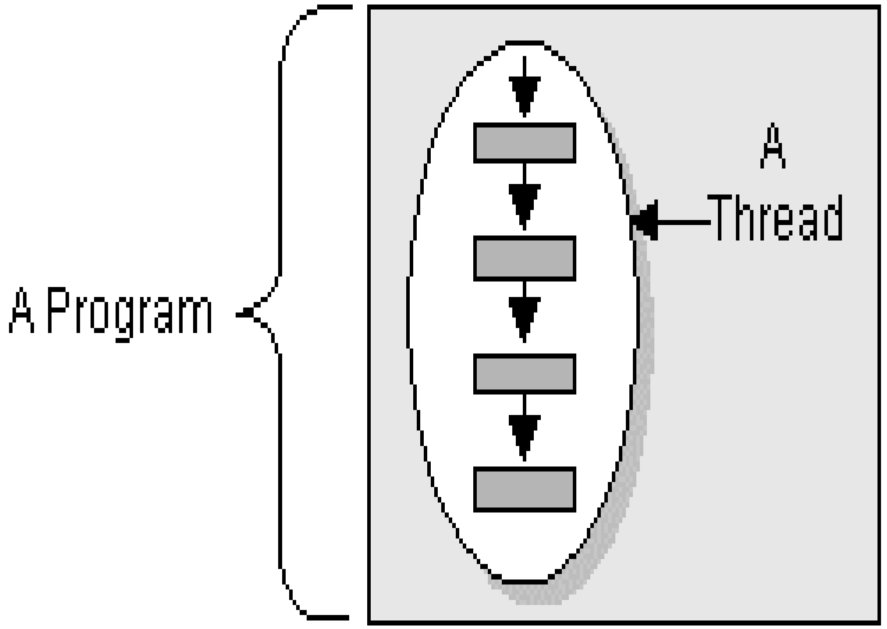
What is a Thread ?

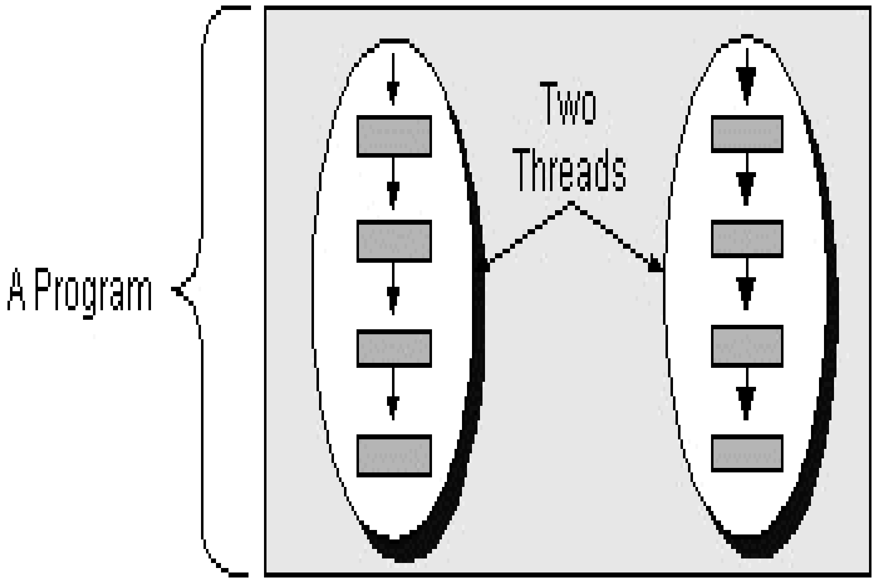
- Multiprocessing and multithreading, both are used to achieve multitasking.
- Multithreading in Java is a process of executing multiple threads simultaneously.
- Multithreading has advantages than multiprocessing because threads use a shared memory area.
- They don't allocate separate memory area so saves memory and switching between the threads takes less time than process.

- Java is a *multi-threaded programming language* which means we can develop multi-threaded program using Java.

- A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

Single-Threaded vs Multithreaded Programs





```
{ A(); A1();  A2();   A3();
  B1();  B2();  }
```

```
{  A();
   newThreads {
     { A1(); A2(); A3() };
     {B1(); B2() }
   }
}
```

Define and Launch a Java Thread

- Each Java Run time thread is encapsulated in a java.lang.Thread instance.

- Two ways to define a thread:

  1. Extend the Thread class
  2. Implement the Runnable interface

The major difference is that when a class extends the Thread class, we cannot extend any other class, but by implementing the Runnable interface, it is possible to extend from another class as well.

class MyClass extends OtherClass implements Runnable

Define and Launch a Java Thread

Steps for extending the Thread class:

1. Subclass the Thread class;

2. Override the default Thread method run(), which is the entry point of the thread, like the main(String[]) method in a java program.

Define and Launch a Java Thread

Implement the Runnable interface

```
        package java.lang;
        public class Main implements Runnable
{
             public void run() ;
}
```

Define a Thread

```
// Example:
public class Main extends Thread
{
     public void run( )
      {  // run() is to a thread what main() is to a java program
        for (int b = 1; b <=5 ; b++)
     {
       System.out.println(b);
     }
     }
     }
```

Define a Thread

- Implement the Runnable interface if you need a parent class:

```
public class Main implements Runnable
{
    public void run( )
     {
       for (int b = 1; b <=5; b++)
    {
    System.out.println(b);
    }
      }
    }
```

Example: extends Thread class

```
class ABC extends Thread
{
   public void run( )
   {
      for(int i=1;i<=3;i++)
      {
    System.out.println("ABC Thread");
      }}}
class XYZ extends Thread{

   public void run()
   {
      for(int i=1;i<=3;i++)
      {
System.out.println("XYZ Thread");
      }}}
public class Main
{
public static void main (String[] args)
{
  ABC obj1=new ABC();
  XYZ obj2=new XYZ();
  obj1.start();
  obj2.start();
}
}
```

O/P: any thread can execute first.

```
ABC    or       XYZ
ABC                   XYZ
ABC                   XYZ
XYZ                   ABC
XYZ                   ABC
XYZ                   ABC
```

Example: implements Runnable Interface

If the class implements the Runnable interface, the thread can be run by passing an instance of the class to a Thread object's constructor and then calling the thread's start() method.

```
class ABC implements Runnable
{
   public void run()
   {
      for(int i=1;i<=3;i++)
      {
         System.out.println("ABC Thread");
      }}}
```

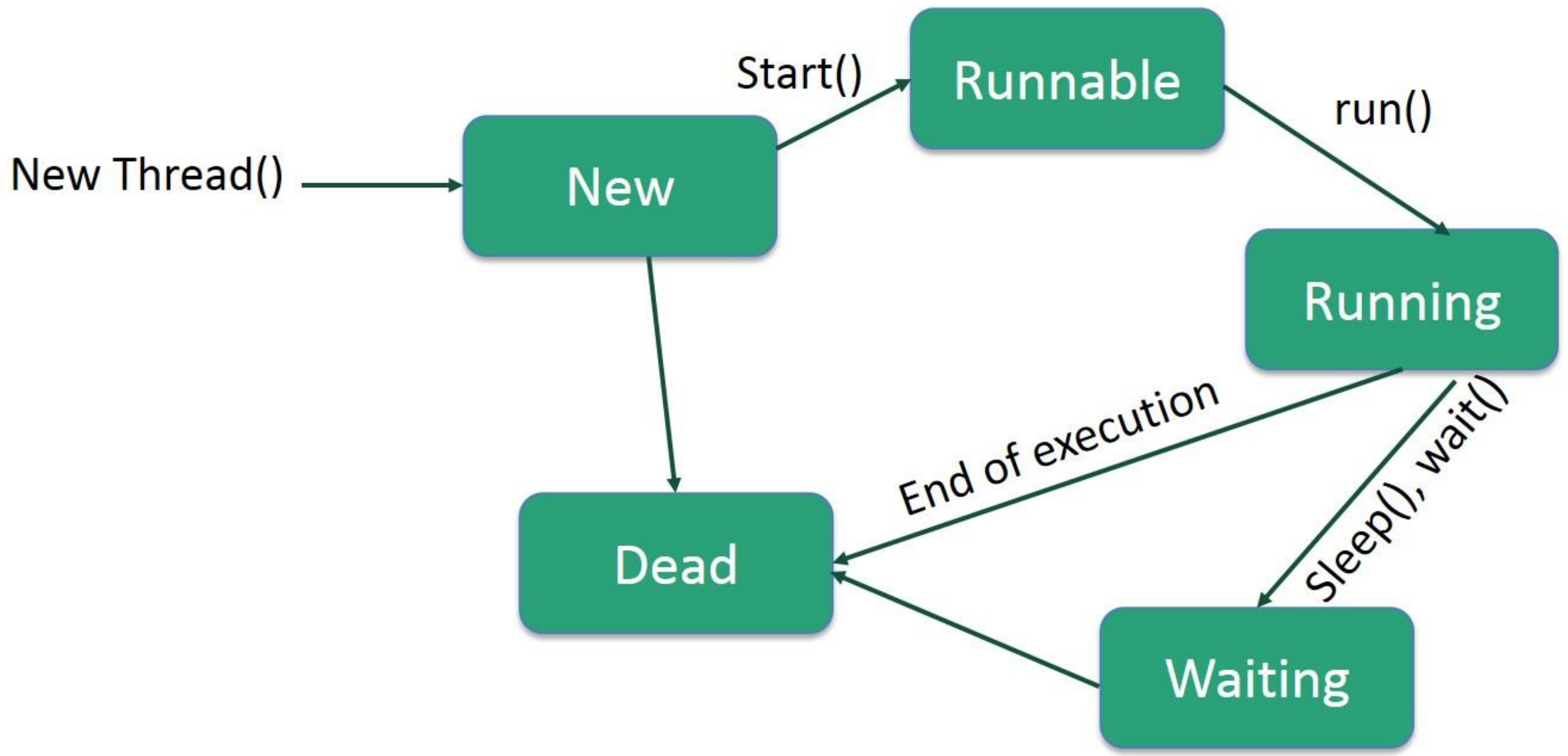
```
class XYZ implements Runnable
{

   public void run()
   {
      for(int i=1;i<=3;i++)
      {
         System.out.println("XYZ Thread");
      }
   }
}
```

```
public class Main
{
public static void main (String[] args)
{
   ABC obj1=new ABC();
   XYZ obj2=new XYZ();
   Thread t=new Thread(obj1);   //Thread object constructor
     t.start();
   Thread t1=new Thread(obj2);
   t1.start();

}
```

Life Cycle of a Thread



Blocked: when it is prevented from entering into the runnable state.

Dead: when thread ends its life cycle. It can also be killed as soon as it is born or while running or even when not in runnable condition.

Thread Method

public void run( ): used to perform action on thread.

public void start( ): to start execution of thread. JVM calls run( )method on the thread.

public void sleep(long milliseconds): Causes the currently executing thread to sleep for the specified number of milliseconds.

public void join(): waits for a thread to die.

public void join(long miliseconds): waits for a thread to die for the specified miliseconds.

public void yield(): causes the currently executing thread object to temporarily pause and allow other threads to execute.

Concurrency Methods

public void suspend(): is used to suspend the thread(running to waiting state)

public void resume(): is used to resume the suspended thread(waiting to running state)

public void stop(): is used to stop the thread.

Thread Method

- getName(): reruns name of current thread.
- setName(): to change name of current thread at runtime.
- getPriority(): returns priority of current thread.
- setPriority(): to set priority of a thread.
- isAlive(): to check a particular thread is in runnable condition or not. It returns Boolean value.

Thread.sleep( )

- interacts with the thread scheduler to put the current thread in wait state for specified period of time.
- Once the wait time is over, thread state is changed to runnable state and wait for the CPU for further execution.
- The actual time that current thread sleep depends on the thread scheduler (part of operating system).

```
public class Main {
public static void main(String[] args) throws InterruptedException {
        long start = System.currentTimeMillis();
        System.out.println("Thread Example");
        Thread.sleep(2000);
        System.out.println("Thread sleep time="+(System.currentTimeMillis()-start)); } }
}
```

WAP to display name of current thread and change another name of current thread.

```
public class Myclass extends Thread
{
public static void main (String[] args)
{
  Thread t= new Thread();
   System.out.println("Current Thread Name="+ t.getName());
   t.setName("My thread");
    System.out.println("New name="+t);
    }
}
```

```
current Thread Name=Thread-0
New Name=Thread[My thread,5,main]
```

priority

thread group