Multithreading

NILESH BHANDARE

Agenda

- Multithreading
- Life Cycle of Thread
- Thread Creation
- Thread Priority
- Thread Pooling

What is Multitasking ?

- Process Base
  - Executing several task simultaneously where each task separate independent process
- Thread Base
  - Executing several task simultaneously where each task is separate independent part of a program

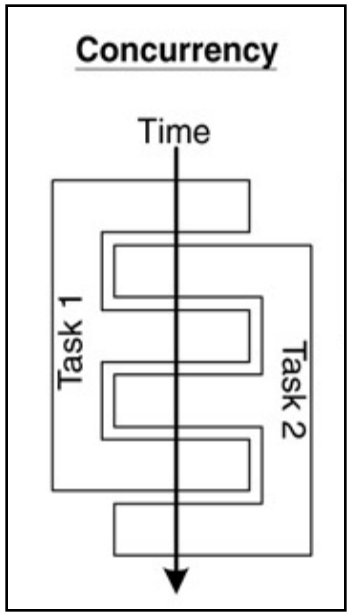
Multithreading

- 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.

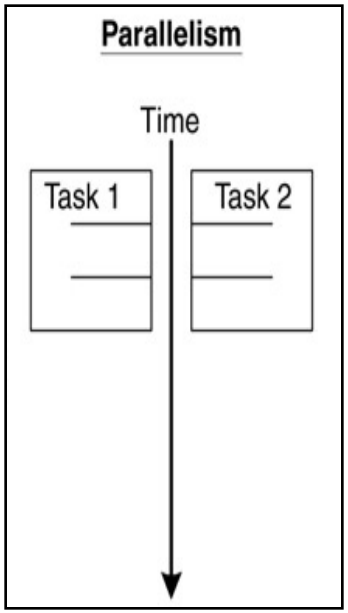
- Java is a multi-threaded programming language

- Multithreading allows an application to have multiple threads of execution running concurrently

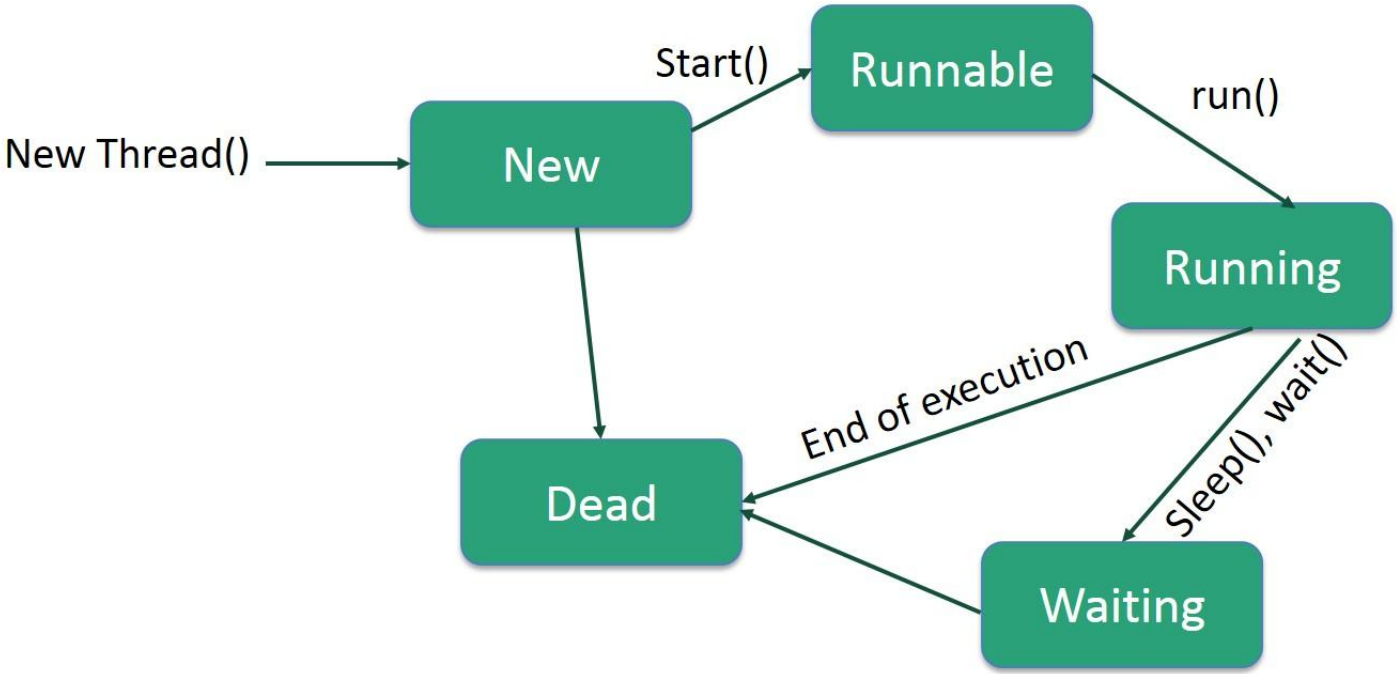
Concurrency and Parallelism



- Concurrent multithreading systems give the appearance of several tasks executing at once, but these tasks are actually split up into chunks that share the processor with chunks from other tasks.
- In parallel systems, two tasks are actually performed simultaneously. Parallelism requires a multi-CPU system.



Thread Lifecycle



How create thread

1. By extending the **java.lang.Thread** class.
2. By implementing the **java.lang.Runnable** interface

- *The run() method is where the action of a thread takes place.*
- *The execution of a thread starts by calling its start() method.*

1. By extending the java.lang.Thread class

```
class Mythread extends Thread
{
public void run()
 {
for(int i=0;i<10;i++)
   {
   System.out.println("Child
   Thread"+i);
   }
 }
}
```

```
public class Extends
{
   public static void
   main(String[] args)
   {
   Mythread t=new Mythread();
   t.start();
   for(int i=0;i<10;i++)
         {
         System.out.println("Main
         Thread"+i);
         }
   }
}
```

Lets try

Case Study

Case	Result
Thread Scheduler	
t.Start() vs t.run()	t.Start – create thread and exc run() t.Run ----not created thread just run() as normal function
If we are not override run() method	
Overloading of run() method	
Overriding of start() method	

2. Implementing the Runnable Interface

- In order to create a new thread we may also provide a class that implements the **java.lang.Runnable** interface.
- Preferred way in case our class has to subclass some other class.
- A Runnable object can be wrapped up into a Thread object:
  - **Thread**(Runnable target)
  - **Thread**(Runnable target, String name)
- The thread's logic is included inside the **run()** method of the **runnable** object.

2. Implementing the Runnable Interface

```
class MyRunnable implements Runnable
{
   public void run()
   {
         for(int i=0;i<5;i++)
         {
         System.out.println("@Child
         Thread"+i);
         }
   }
}
```

```
public class RunnableDemo {
public static void main(String[]
   args)
{
   MyRunnable r=new MyRunnable();
   Thread t= new Thread(r);
   t.start();
   for(int i=0;i<5;i++)
         {
         System.out.println("#Parent
         Thread"+i);
         }
   }
}
```

Lets try

Case Study

```
MyRunnable r=new MyRunnable();
Thread t1= new Thread();
Thread t2= new Thread(r);
```

Case	Result
`t1.start()`	
`t1.run();`	
`t2.start()`	
`t2.run();`	
`r.Start();`	

Get and Set Thread Name

- To get thread ID:
  - `Thread.currentThread().getId()`
- To get thread Name:
  - `Thread.currentThread().getName()`
- To set thread Name:
  - `Thread.currentThread().setName("ThreadName")`
  - getState()

Thread Priority

- Every Thread in java has a priority
- Default Priority generated by JVM
- ranging from 1 to 10 (Low to High)
- Here 3 constants are defined in it namely as follows:
  - public static int NORM_PRIORITY ---5
  - public static int MIN_PRIORITY ---1
  - public static int MAX_PRIORITY ---10
- `public final int getPriority()` – java.lang.Thread.getPriority() method returns priority of given thread.
- `public final void setPriority(int newPriority)` –java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority.

Method to prevent thread from execution

- The **`yield()`** method of thread class causes the currently executing thread object to temporarily pause and allow other threads to execute.
- Syntax-**`public static void`** `yield()`
- If all waiting thread have the low priority or if there is no waiting thread then
  - same thread will continue execution.
- If several thread with same priority then
  - thread scheduler decide which thread get executed.

public static void yield()

```
class Mythread extends Thread
{
public void run()
 {
for(int i=0;i<10;i++)
   {
   Thread.yield();
   System.out.println("Child
   Thread"+i);
   }
 }
}
```

Child thread call yield() method so every time main thread get chance for execution

```
public class Extends
{
   public static void
   main(String[] args)
   {
   Mythread t=new Mythread();
   t.start();
   for(int i=0;i<10;i++)
         {
         System.out.println("Main
         Thread"+i);
         }
   }
}
```

Sleep()

- If a thread do not want to perform any operation for a particular amount of time
- `public static void sleep(long millis)throws Interrupted Exception`
- If any other thread interrupts when the thread is sleeping, then InterruptedException will be thrown
- If the system is busy, then the actual time the thread will sleep will be more as compared to that passed while calling the sleep method and if the system has less load, then the actual sleep time of the thread will be close to that passed while calling sleep() method

https://www.geeksforgeeks.org/thread-sleep-method-in-java-with-examples/

Join() Method

- The join() method permits one thread to wait until the other thread to finish its execution.
  - If t1 executed t2.join() then t1 should go in waiting state till completion of t2.

- **`public final void join()`**

- Every join method throws InterruptedException, which is checked exception hence compulsory we should handle try catch or throws keyword

ThreadPool in Java

- Problem: creates a new thread for every request would spend more time and consume more system resources in creating and destroying threads than processing actual requests

- Solution : **Threadpool**

- A **thread pool** reuses previously created threads to execute current tasks

Cont..

- Java provides the Executor framework which is centered around the Executor interface, its sub-interface **–ExecutorService** and the class-**ThreadPoolExecutor**, which implements both of these interfaces

- `newFixedThreadPool(int)` Creates a fixed size thread pool.

- **Steps to be followed**

  1. Create a task(Runnable Object) to execute ----create jobs runnable
  2. Create Executor Pool using Executors -----pool
  3. Pass tasks to Executor Pool ----assign task to pool
  4. Shutdown the Executor Pool----deallocate

```
package Pool;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class MyRunnablePoolA5 implements Runnable
{
public String name;
public MyRunnablePoolA5 (String temp)//constructor
{
name=temp;
}
public void run()
{
System.out.println("Thread "+Thread.currentThread().getId()+" Started to execute "+name);
try {
Thread.sleep(2000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
System.out.println("Thread "+Thread.currentThread().getId()+" Stopped");
}
}
public class PoolDemoA5 {
public static void main(String[] args)
{
ExecutorService ex = Executors.newFixedThreadPool(2);
for (int i=0;i<10;i++)
{
Runnable r = new MyRunnablePoolA5("MyJob_"+i); //MyJob_0
ex.execute(r);
}
ex.shutdown();
}

}
```

Further study

- Synchronization
- Demon Threads

Thank You