# Thread Group:

- Java allows to group multiple threads in to a single object. So that, we can trigger all the threads by a single method call.
- In java thread grouping is done using **ThreadGroup** class

```
1  package com.pack1;
2
3  public class ClassA extends Thread
4  {
5      @Override
6      public void run()
7      {
8          System.out.println(Thread.currentThread().getName());
9      }
10     public static void main(String[] args)
11     {
12         ClassA aobj=new ClassA();
13
14         Thread t1=new Thread(aobj);
15         t1.setName("First");
16         t1.start();
17     }
18 }
```

```
First
```

```
1  package com.pack1;
2
3  public class ClassA extends Thread
4  {
5      @Override
6      public void run()
7      {
8          System.out.println(Thread.currentThread().getName());
9      }
10     public static void main(String[] args)
11     {
12         ClassA aobj=new ClassA();
13
14         Thread t1=new Thread(aobj,"First");
15
16         t1.start();
17     }
18 }
```

```
First
```

```
    {
        System.out.println(Thread.currentThread().getName()
    }
    public static void main(String[] args)
    {
        ThreadGroupTest robj = new ThreadGroupTest(); // Cl

        ThreadGroup tg = new ThreadGroup("Parent ThreadGrou

        //tg.setMaxPriority(10);

        Thread t1 = new Thread(tg, robj, "one");
        t1.start();

        Thread t2 = new Thread(tg, robj, "two");
        t2.start();

        Thread t3 = new Thread(tg, robj, "three");
        t3.start();

        //System.out.println("Active Threads : "+tg.activeC
        System.out.println("Thread Group Name: " + tg.getNa
    }
```

```
Thread Group Name: Parent ThreadGroup
two  5
one  5
three  5
```

```
    public static void main(String[] args)
    {
        ThreadGroupTest robj = new ThreadGroupTest(); // Cla

        ThreadGroup tg = new ThreadGroup("Parent ThreadGroup

        //tg.setMaxPriority(10);

        Thread t1 = new Thread(tg, robj, "one");
        t1.start();

        Thread t2 = new Thread(tg, robj, "two");
        t2.start();

        Thread t3 = new Thread(tg, robj, "three");
        t3.start();

        System.out.println("Active Threads : "+tg.activeCour
        System.out.println("Thread Group Name: " + tg.getNam
    }
}
```

```
one  5
three  5
two  5
Active Threads : 3
Thread Group Name: Parent ThreadGroup
```

```
    {
        System.out.println(Thread.currentThread().getName()+"  "+Thread
    }
    public static void main(String[] args)
    {
        ThreadGroupTest robj = new ThreadGroupTest(); // Class Object

        ThreadGroup tg = new ThreadGroup("Parent ThreadGroup");

        tg.setMaxPriority(10);

        Thread t1 = new Thread(tg, robj, "one");
        t1.start();

        Thread t2 = new Thread(tg, robj, "two");
        t2.start();
        System.out.println("Active Threads : "+tg.activeCount());

        Thread t3 = new Thread(tg, robj, "three");
        t3.start();


        System.out.println("Thread Group Name: " + tg.getName());
```

```
<terminated> ThreadGroupTest [Java Application] C:\Program Files\Java
Active Threads : 2
Thread Group Name: Parent Threa
two  5
one  5
three  5
```

```java
public class ThreadGroupTest implements Runnable
{
    @Override
    public void run()
    {
        System.out.println(Thread.currentThread().getName()+"  "+Thread.currentThread().getPriority());
    }
    public static void main(String[] args)
    {
        ThreadGroupTest robj = new ThreadGroupTest(); // Class Object

        ThreadGroup tg = new ThreadGroup("Parent ThreadGroup");

        tg.setMaxPriority(10);

        Thread t1 = new Thread(tg, robj, "one");
        t1.start();

        Thread t2 = new Thread(tg, robj, "two");
        t2.start();
        System.out.println("Active Threads : "+tg.activeCount());

        Thread t3 = new Thread(tg, robj, "three");
        t3.start();

        System.out.println("Thread Group Name: " + tg.getName());
    }
}
```

## Thread Pool:

- Thread pool represents a group of worker threads that are waiting for the task and reuse number of times.
- Thread pool contain a group of fixed size threads.
- A thread from the thread pool is assigned a task by the service provider.
- After completion of the task, thread is contained in the thread pool again.

- At any point, at most threads will be in actively processing the tasks.
- If additional tasks are submitted when all threads are active, they will wait in the queue until a thread is available.
- If any thread terminates due to a failure during execution prior to shutdown, a new thread will take its place to execute subsequent tasks.
- The threads in the pool will **exist** until it is explicitly shutdown.



```
1  package com.pack1;
2
3  import java.util.concurrent.ExecutorService;
4  import java.util.concurrent.Executors;
5
6  public class ThreadPoolDemo extends Thread
7  {
8      private String msg;
9
10     public ThreadPoolDemo(String s)
11     {
12         this.msg = s;
13     }
14     @Override
15     synchronized public void run()
16     {
17         System.out.println(Thread.currentThread().getName() + " (Begining) message = " + msg);
18         processMessage();
19         System.out.println(Thread.currentThread().getName() + " (Ending)");
20         processMessage();
21     }
22     private void processMessage()
23     {
```

```
24          try
25          {
26              Thread.sleep(10000);
27          }
28          catch (InterruptedException e)
29          {
30              e.printStackTrace();
31          }
32      }
33      public static void main(String[] args)
34      {
35          ExecutorService executor = Executors.newFixedThreadPool(3);
36
37          for(int i = 1; i <= 5; i++) // 5 iterations ====> 5 tasks
38          {
39              ThreadPoolDemo tpdobj=new ThreadPoolDemo(""+i);
40              Thread t=new Thread(tpdobj);
41              executor.execute(t);
42          }
43          executor.shutdown();
44          while (!executor.isTerminated())   // !(true)===> false
45          {
46
47            }
48            System.out.println("Finished all threads");
49      }
50  }
```

# ThreadGroup Vs ThreadPool

| Feature | ThreadGroup | ThreadPool (Executor Framework) |
|---|---|---|
| **Purpose** | Organize threads into groups for easier management | Manage a pool of worker threads for executing tasks |
| **Functionality** | Manage threads as a unit (interrupt, set priorities) | Efficiently manage and reuse threads for task execution |
| **Package** | java.lang | java.util.concurrent |
| **Thread Reuse** | Does not provide thread reuse | Provides thread reuse to minimize overhead |