# MODULE 4

## CHAPTER 2 MULTITHREADED PROGRAMMING

Prepared By Mr. EBIN PM, AP, IESCE 1

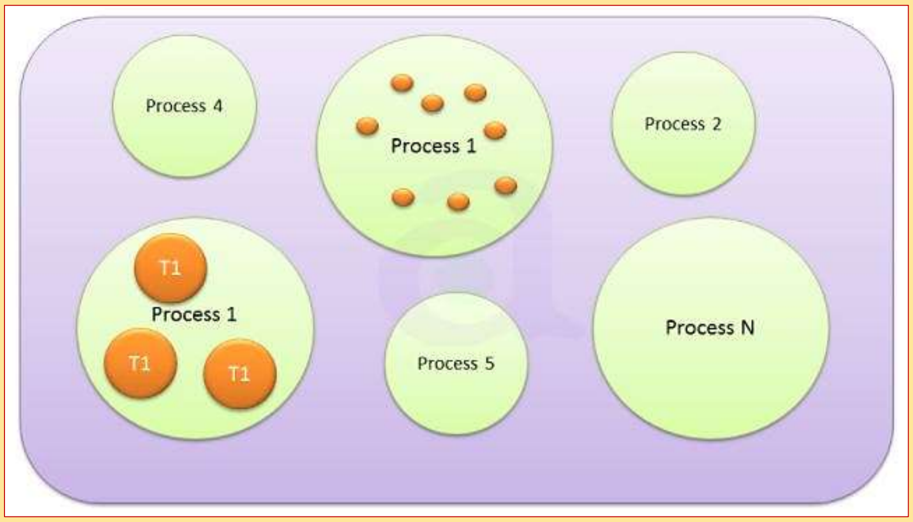
## THREAD

- 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.
- Each part of such program is called a thread. So, threads are light-weight processes within a process.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 2

- Multiprocessing and multithreading, both are used to achieve multitasking But we use multithreading than multiprocessing because threads share a common 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..
- A thread is a lightweight sub process, a smallest unit of processing.
- It is a separate path of execution.
- They are independent, if there occurs exception in one thread, it doesn't affect other threads.

➢ *At least one process is required for each thread.*



## ❖ Advantages of Java Multithreading

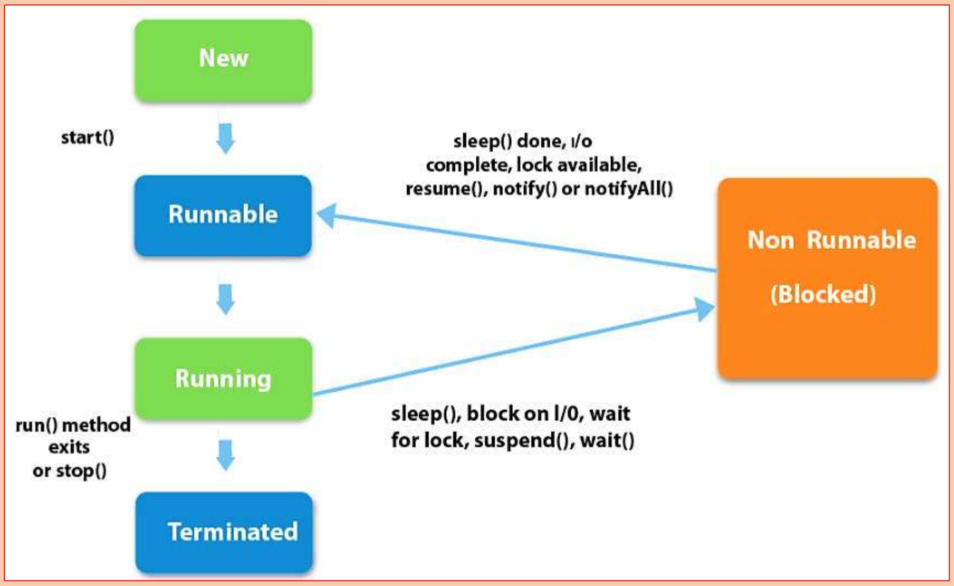
- It doesn't block the user because threads are independent and you can perform multiple operations at same time.
- You can perform many operations together so it saves time.
- Threads are independent so it doesn't affect other threads if exception occur in a single thread.
- Note: At a time one thread is executed only.

## LIFE CYCLE OF THREAD

- A thread can be in one of the **five states.**
- According to sun, there is only 4 states in thread life cycle in java new, runnable, non-runnable and terminated.
- There is no running state. But for better understanding the threads, we can explain it in the 5 states.

❖ New
❖ Runnable
❖ Running
❖ Non-Runnable (Blocked)
❖ Terminated

## Life Cycle


New
start()
Runnable
Running
Terminated
run() method exits or stop()
Non Runnable (Blocked)
sleep() done, I/o complete, lock available, resume(), notify() or notifyAll()
sleep(), block on I/0, wait for lock, suspend(), wait()

- **New** - The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
- **Runnable** - The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
- **Running** - The thread is in running state if the thread scheduler has selected it.
- **Non-Runnable (Blocked)** - This is the state when the thread is still alive, but is currently not eligible to run.
- **Terminated** - A thread is in terminated or dead state when its run() method exits.

- ➢A Running Thread transit to one of the non-runnable states, depending upon the circumstances.
- **Sleeping:** The Thread sleeps for the specified amount of time.
- **Blocked for I/O:** The Thread waits for a blocking operation to complete.
- **Blocked for join completion:** The Thread waits for completion of another Thread.
- **Waiting for notification:** The Thread waits for notification another Thread.
- **Blocked for lock acquisition:** The Thread waits to acquire the lock of an object.
- ➢JVM executes the Thread, based on their priority and scheduling.

## CREATING THREAD

➢There are two ways to create a thread:

- By extending **Thread class**
- By implementing **Runnable interface**.

**❖Extending Thread class:**

- Thread class provide constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.

**Commonly used Constructors of Thread class:**

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

❖**Thread Methods -** Following is the list of important methods available in the Thread class.

- **public void run()** : is used to perform action for a thread.
- **public void start()** : starts the execution of the thread. JVM calls the run() method on the thread.
- **public void sleep(long miliseconds)** : Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
- **public void join()** : waits for a thread to die.
- **public int getPriority()** : returns the priority of the thread.
- **public int setPriority(int priority)** : changes the priority of the thread.

- **public String getName():** returns the name of the thread.
- **public Thread currentThread()** : returns the reference of currently executing thread.
- **public int getId()** : returns the id of the thread.
- **public Thread.State getState()** : returns the state of the thread.
- **public boolean isAlive()** : tests if the thread is alive.
- **public void suspend()** : is used to suspend the thread(depricated).
- **public void resume()** : is used to resume the suspended thread
- **public void stop()** : is used to stop the thread(depricated).
- **public boolean isDaemon()** : tests if the thread is a daemon thread.

## ❖Thread.start() & Thread.run()

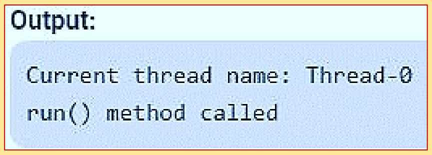
➢In Java's multi-threading concept, **start()** and **run()** are the two most important methods.

- When a program calls the start() method, a new thread is created and then the run() method is executed.
- But if we directly call the run() method then no new thread will be created and run() method will be executed as a normal method call on the current calling thread itself and no multi-threading will take place.

➢Let us understand it with an example:

```
class MyThread extends Thread {
  public void run()
  {
    System.out.println("Current thread name: "
              + Thread.currentThread().getName());
    System.out.println("run() method called");
  }
}
class Xyz {
  public static void main(String[] args)
  {
    MyThread t = new MyThread();
    t.start();
  }
}
```

**Output**



Output:

```
Current thread name: Thread-0
run() method called
```

## start ()

➢when we call the start() method of our thread class instance, a new thread is created with default name Thread-0 and then run() method is called and everything inside it is executed on the newly created thread.

## run ()

➢when we called the *run()* method of our MyThread class, no new thread is created and the *run()* method is executed on the current thread i.e. *main* thread. Hence, no multi-threading took place. The *run()* method is called as a normal function call.

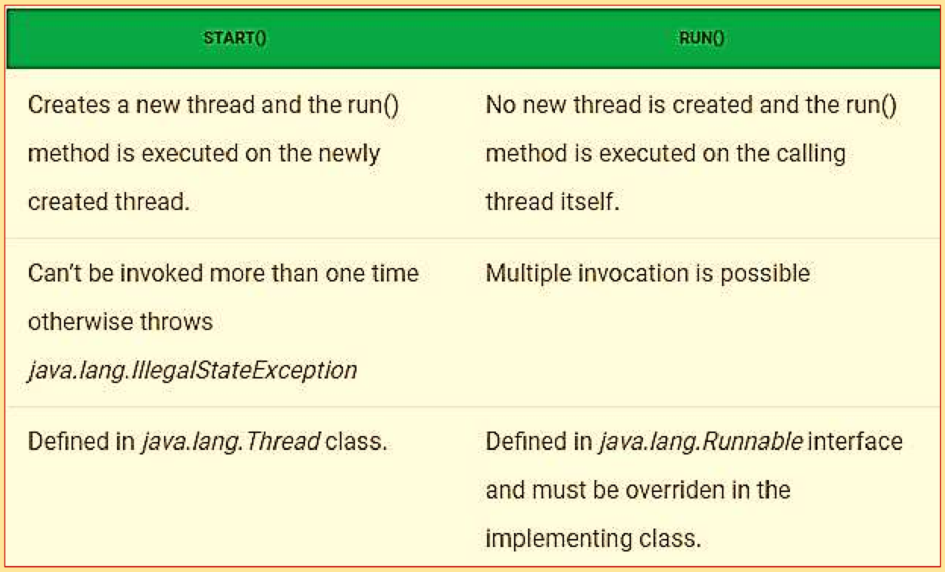
## Let us try to call run() method directly instead of start() method:

```
class MyThread extends Thread {
   public void run()
   {
      System.out.println("Current thread name: "
                             + Thread.currentThread().getName());
      System.out.println("run() method called");
   }
}
class  Xyz {
   public static void main(String[] args)
   {
      MyThread t = new MyThread();
      t.run();
   }
}
```

**Output**

```
Current thread name: main
run() method called
```

## Difference



| START() | RUN() |
|---|---|
| Creates a new thread and the run() method is executed on the newly created thread. | No new thread is created and the run() method is executed on the calling thread itself. |
| Can't be invoked more than one time otherwise throws *java.lang.IllegalStateException* | Multiple invocation is possible |
| Defined in *java.lang.Thread* class. | Defined in *java.lang.Runnable* interface and must be overriden in the implementing class. |

## ❖Implementing Runnable interface:

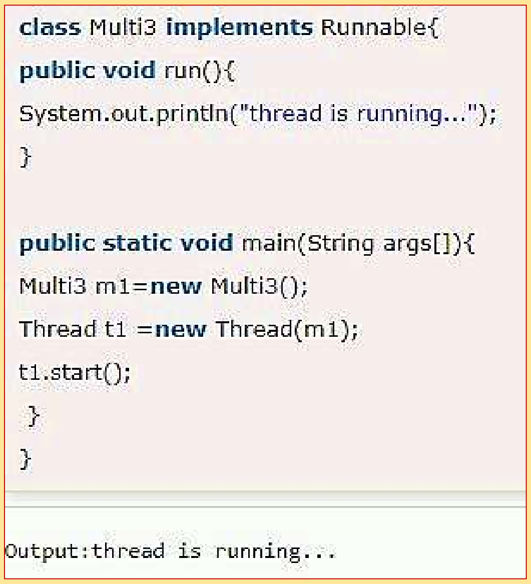
- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.
- Runnable interface have only one method named **run().**

  **public void run():** is used to perform action for a thread.

## ❖ Steps to create a new Thread using Runnable :

- Create a Runnable implementer and implement run() method.
- Instantiate Thread class and pass the implementer to the Thread, Thread has a constructor which accepts Runnable instance.
- Invoke start() of Thread instance, start internally calls run() of the implementer. Invoking start(), creates a new Thread which executes the code written in run().

## ❖Thread Example by implementing Runnable interface



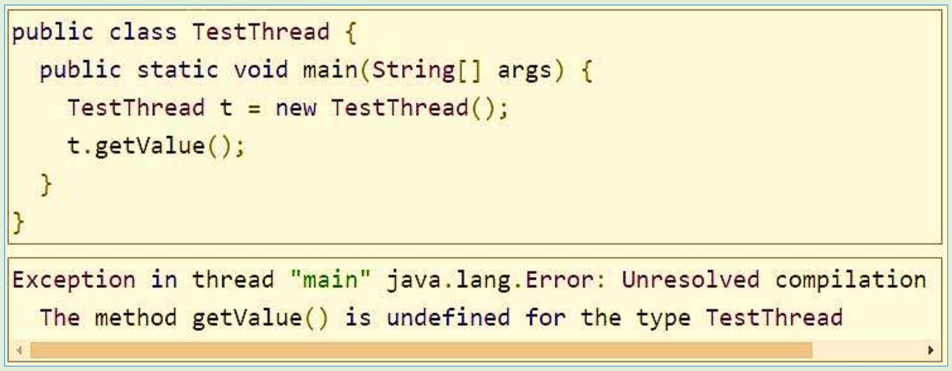
```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

```
Output:thread is running...
```

## MAIN THREAD

- Every java program has a main method. The main method is the entry point to execute the program.
- So, when the JVM starts the execution of a program, it creates a thread to run it and that thread is known as the main thread.
- Each program must contain at least one thread whether we are creating any thread or not.
- The JVM provides a default thread in each program.
- A program can't run without a thread, so it requires at least one thread, and that thread is known as the main thread.

- If you ever tried to run a Java program with compilation errors you would have seen the mentioning of main thread. Here is a simple Java program that tries to call the non-existent getValue() method.



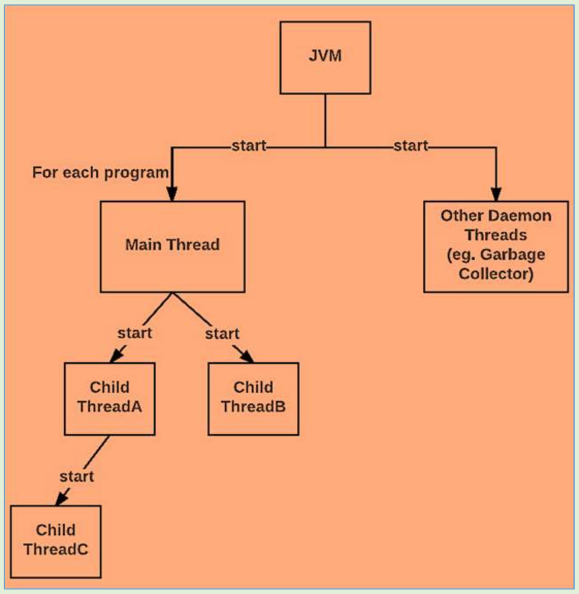
```
public class TestThread {
  public static void main(String[] args) {
    TestThread t = new TestThread();
    t.getValue();
  }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation
  The method getValue() is undefined for the type TestThread
```

- As you can see in the error when the program is executed, main thread starts running and that has encountered a compilation problem.

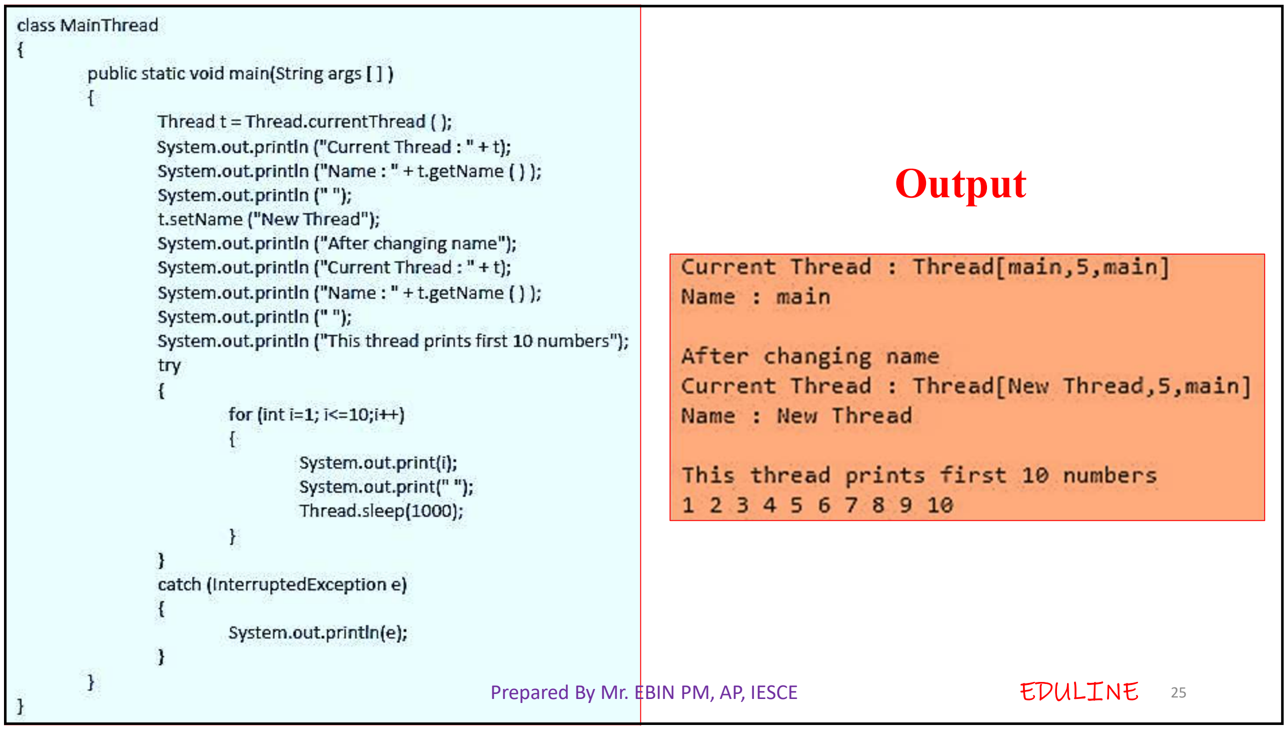
## ❖Properties

- It is the thread from which other "child" threads will be spawned.
- Often, it must be the last thread to finish execution because it performs various shutdown actions



## ❖How to control Main thread

- The main thread is created automatically when our program is started.
- To control it we must obtain a reference to it.
- This can be done by calling the method **currentThread( )** which is present in Thread class.
- This method returns a reference to the thread on which it is called.
- The default priority of Main thread is 5 and for all remaining user threads priority will be inherited from parent to child.

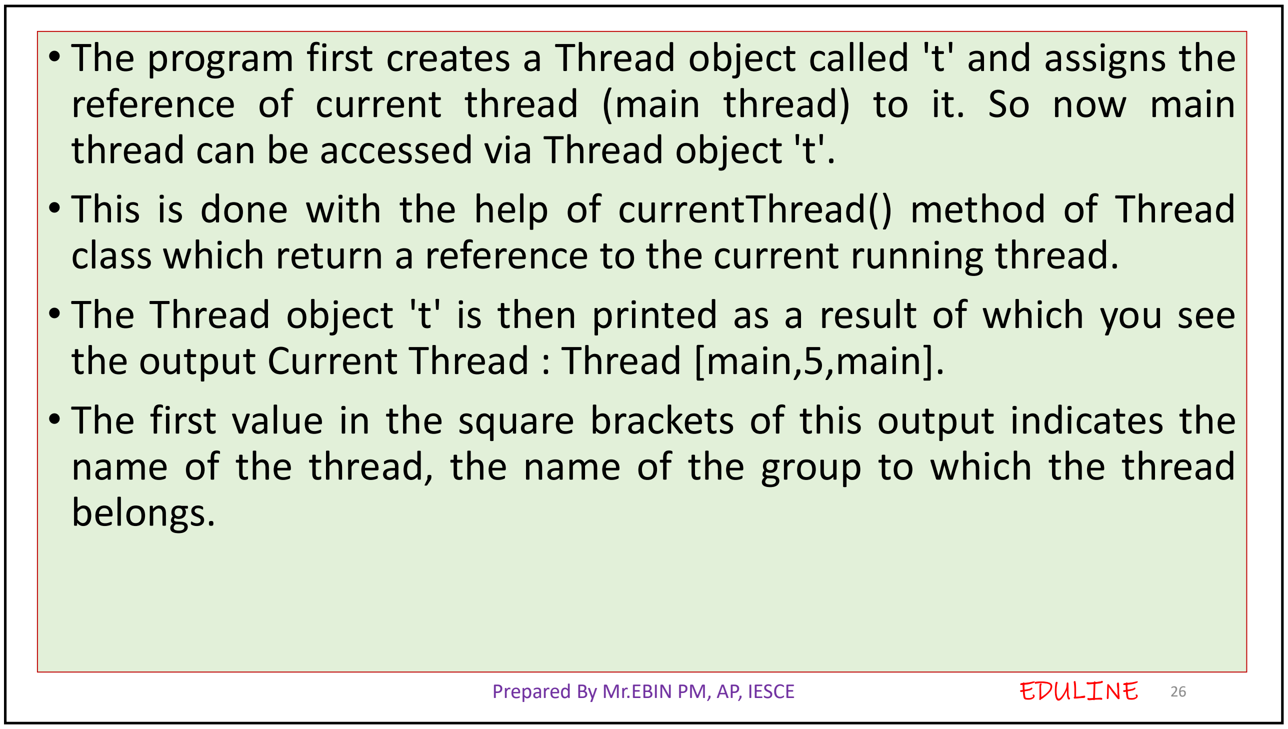


```
class MainThread
{
        public static void main(String args [ ] )
        {
                Thread t = Thread.currentThread ( );
                System.out.println ("Current Thread : " + t);
                System.out.println ("Name : " + t.getName ( ) );
                System.out.println (" ");
                t.setName ("New Thread");
                System.out.println ("After changing name");
                System.out.println ("Current Thread : " + t);
                System.out.println ("Name : " + t.getName ( ) );
                System.out.println (" ");
                System.out.println ("This thread prints first 10 numbers");
                try
                {
                        for (int i=1; i<=10;i++)
                        {
                                System.out.print(i);
                                System.out.print(" ");
                                Thread.sleep(1000);
                        }
                }
                catch (InterruptedException e)
                {
                        System.out.println(e);
                }
        }
}
```

**Output**

```
Current Thread : Thread[main,5,main]
Name : main

After changing name
Current Thread : Thread[New Thread,5,main]
Name : New Thread

This thread prints first 10 numbers
1 2 3 4 5 6 7 8 9 10
```

Prepared By Mr. EBIN PM, AP, IESCE EDULINE 25



- The program first creates a Thread object called 't' and assigns the reference of current thread (main thread) to it. So now main thread can be accessed via Thread object 't'.
- This is done with the help of currentThread() method of Thread class which return a reference to the current running thread.
- The Thread object 't' is then printed as a result of which you see the output Current Thread : Thread [main,5,main].
- The first value in the square brackets of this output indicates the name of the thread, the name of the group to which the thread belongs.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 26

- The program then prints the name of the thread with the help of getName() method.
- The name of the thread is changed with the help of setName() method.
- The thread and thread name is then again printed.
- Then the thread performs the operation of printing first 10 numbers.
- When you run the program you will see that the system wait for sometime after printing each number.
- This is caused by the statement Thread.sleep (1000).

## CREATING MULTIPLE THREADS

```
class ThreadA extends Thread
 {
   public void run()
   {
       for (int i=1;i<=5;i++)
         {
            System.out.println("ThreadA  i ="+(-1*i));
         }
        System.out.println("Exiting ThreadA");
   }
 }
class ThreadB extends Thread
 {
   public void run()
   {
       for (int j=1;j<=5;j++)
         {
            System.out.println("ThreadB  j ="+(2*j));
         }
        System.out.println("Exiting ThreadB");
   }
 }
```

```
class ThreadC extends Thread
 {
   public void run()
   {
       for (int k=1;k<=5;k++)
         {
            System.out.println("ThreadC  k ="+(2*(k-1)));
         }
        System.out.println("Exiting ThreadC");
   }
 }
class MultiThreadDemo
{
   public static void main (String args [])
   {
       ThreadA t1 = new ThreadA();
       ThreadB t2 = new ThreadB();
       ThreadC t3 = new ThreadC();
       t1.start();
       t2.start();
       t3.start();
   }
}
```

**Output**

```
ThreadA  i = -1
ThreadB  j = 2
ThreadC  k =1
ThreadA  i = -2
ThreadB  j = 4
ThreadC  k =3
ThreadA  i = -3
ThreadB  j = 6
ThreadC  k =5
ThreadA  i = -4
ThreadB  j = 8
ThreadC  k =7
ThreadA  i = -5
ThreadB  j = 10
ThreadC  k =9
Exiting ThreadA
Exiting ThreadB
Exiting ThreadC
```

## 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.

Following is the general form of the synchronized statement :

**Syntax**

```
synchronized(object identifier) {
        // Access shared variables and other shared resources
    }
```

**❖Understanding the problem without Synchronization**

- In this example, we are not using synchronization and creating multiple threads that are accessing display method and produce the random output.

```
class First {
  public void display(String msg)
   {
      System.out.print ("["+msg);
      try {
         Thread.sleep(1000);
        }
      catch(InterruptedException e) {
          e.printStackTrace();
        }
    System.out.println ("]");
  }
}
class Second extends Thread {
  String msg;
  First fobj;
  Second (First fp,String str) {
    fobj = fp;
    msg = str;
    start();
  }
  public void run() {
    fobj.display(msg);
  }
}
```

```
public class Syncro
{
  public static void main (String[] args)
  {
    First fnew = new First();
    Second ss = new Second(fnew, "welcome");
    Second ss1= new Second(fnew,"new");
    Second ss2 = new Second(fnew, "programmer");
  }
}
```

In the above program, object fnew of class First is shared by all the three running threads(ss, ss1 and ss2) to call the shared method(void display). Hence the result is nonsynchronized and such situation is called Race condition

OUTPUT:

```
[welcome [ new [ programmer]
]
]
```

## ❖Synchronized Keyword

- To synchronize above program, we must synchronize access to the shared display() method, making it available to only one thread at a time. This is done by using keyword **synchronized** with display() method.
- With a synchronized method, the lock is obtained for the duration of the entire method.
- So if you want to lock the whole object, use a synchronized method

**synchronized void display (String msg)**

Example : implementation of synchronized method

```
class First
{
 synchronized public void display(String msg) {
   System.out.print ("["+msg);
   try {
     Thread.sleep(1000);
        }
    catch(InterruptedException e) {
     e.printStackTrace();
    }
   System.out.println ("]");
 }
}
class Second extends Thread {
 String msg;
 First fobj;
 Second (First fp,String str) {
   fobj = fp;
   msg = str;
   start();
 }
 public void run() {
   fobj.display(msg);
 }
}
```

```
public class MyThread
{
 public static void main (String[] args)
 {
   First fnew = new First();
   Second ss = new Second(fnew, "welcome");
   Second ss1= new Second(fnew,"new");
   Second ss2 = new Second(fnew, "programmer");
 }
}
```

OUTPUT:

```
[welcome]
[programmer]
[new]
```

## ❖ Using Synchronized block

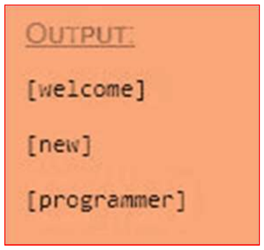
- If we want to synchronize access to an object of a class or only a part of a method to be synchronized then we can use synchronized block for it.
- It is capable to make any part of the object and method synchronized.
- With synchronized blocks we can specify exactly when the lock is needed. If you want to keep other parts of the object accessible to other threads, use synchronized block.

### Example

- In this example, we are using synchronized block that will make the display method available for single thread at a time.

```
class First {
  public void display(String msg) {
    System.out.print ("["+msg);
    try  {
      Thread.sleep(1000);
    }
    catch(InterruptedException e) {
      e.printStackTrace();
    }
    System.out.println ("]");
  }
}
class Second extends Thread {
  String msg;
  First fobj;
  Second (First fp,String str)  {
    fobj = fp;
    msg = str;
    start();
  }
  public void run()  {
    synchronized(fobj)    //Synchronized block
    {
      fobj.display(msg);
    }
  }
}
```

```
public class MyThread
{
  public static void main (String[] args)
  {
    First fnew = new First();
    Second ss = new Second(fnew, "welcome");
    Second ss1= new Second (fnew,"new");
    Second ss2 = new Second(fnew, "programmer");
  }
}
```



OUTPUT:

```
[welcome]

[new]

[programmer]
```

## Which is more preferred - Synchronized method or Synchronized block?

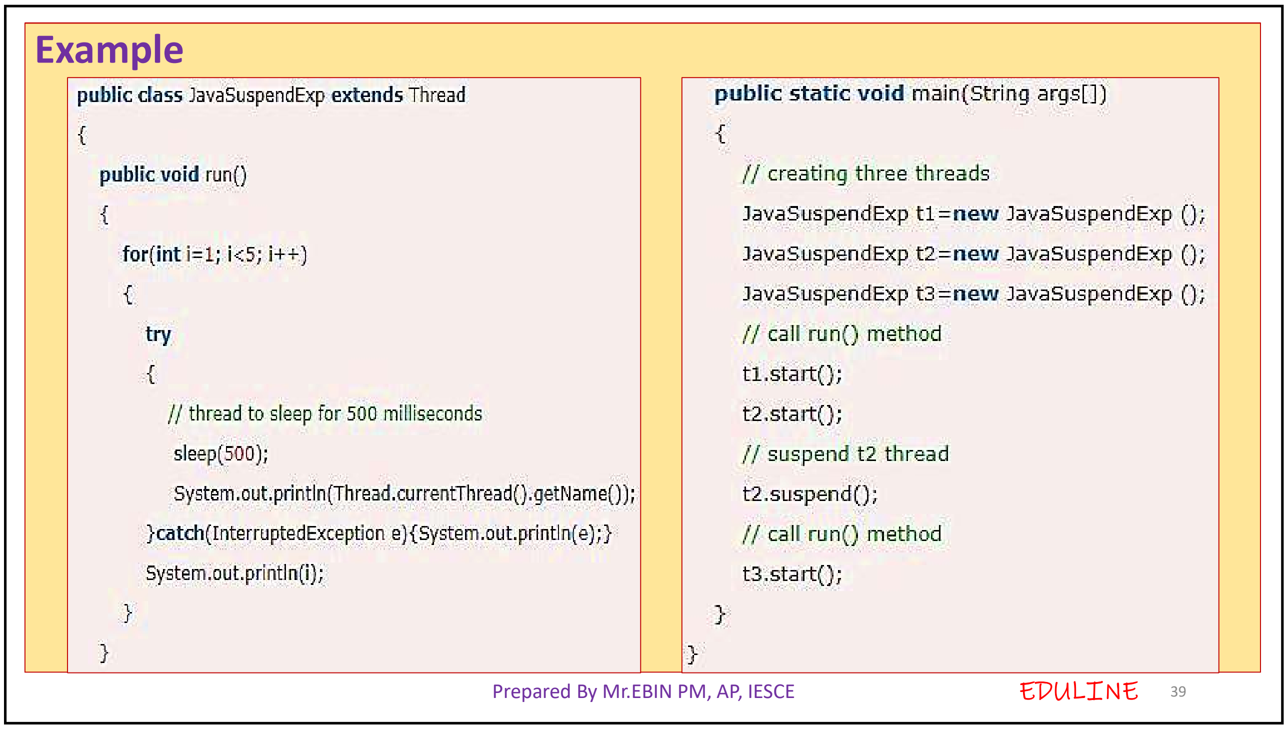
- In Java, synchronized keyword causes a performance cost.
- A synchronized method in Java is very slow and can degrade performance.
- So we must use synchronization keyword in java when it is necessary else, we should use Java synchronized block that is used for synchronizing critical section only.

## Thread suspend() method

- The suspend() method of thread class puts the thread from running to waiting state.
- This method is used if you want to stop the thread execution and start it again when a certain event occurs.
- This method allows a thread to temporarily cease execution.
- The suspended thread can be resumed using the resume() method.

**Syntax**

**public final void suspend()**

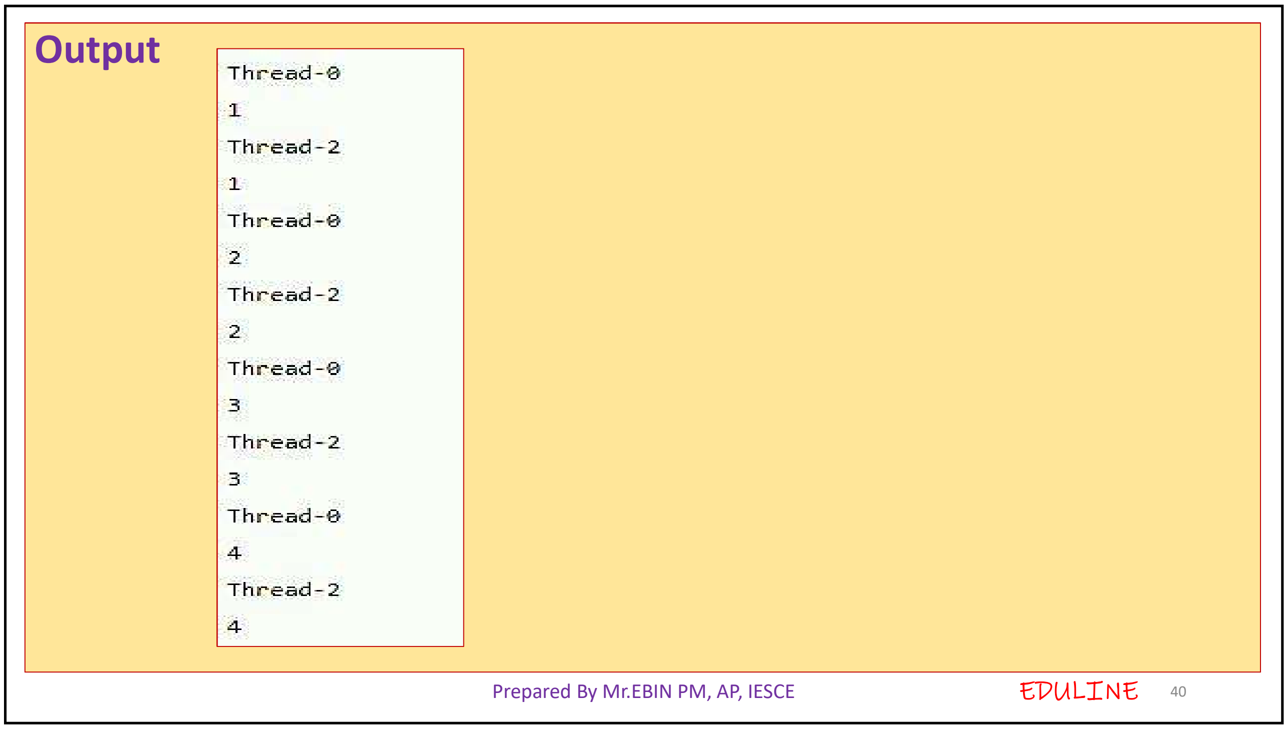


## Example

```
public class JavaSuspendExp extends Thread
{
   public void run()
   {
      for(int i=1; i<5; i++)
      {
         try
         {
            // thread to sleep for 500 milliseconds
             sleep(500);
             System.out.println(Thread.currentThread().getName());
         }catch(InterruptedException e){System.out.println(e);}
         System.out.println(i);
      }
   }
```

```
   public static void main(String args[])
   {
      // creating three threads
      JavaSuspendExp t1=new JavaSuspendExp ();
      JavaSuspendExp t2=new JavaSuspendExp ();
      JavaSuspendExp t3=new JavaSuspendExp ();
      // call run() method
      t1.start();
      t2.start();
      // suspend t2 thread
      t2.suspend();
      // call run() method
      t3.start();
   }
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 39



## Output

```
Thread-0
1
Thread-2
1
Thread-0
2
Thread-2
2
Thread-0
3
Thread-2
3
Thread-0
4
Thread-2
4
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 40

## Thread resume() method

- The resume() method of thread class is only used with suspend() method.
- This method is used to resume a thread which was suspended using suspend() method.
- This method allows the suspended thread to start again.

**Syntax**

**public final void resume()**

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 41

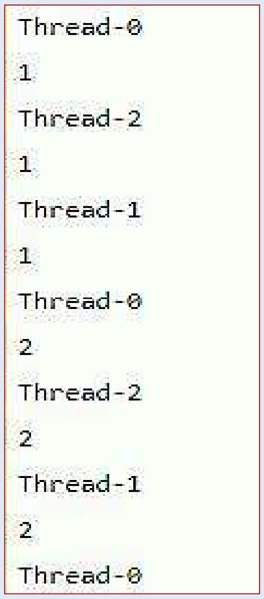
**Example**

```
public class JavaResumeExp extends Thread
{
   public void run()
   {
      for(int i=1; i<5; i++)
      {
         try
         {
            // thread to sleep for 500 milliseconds
             sleep(500);
             System.out.println(Thread.currentThread().getName());
         }catch(InterruptedException e){System.out.println(e);}
         System.out.println(i);
      }
   }
```

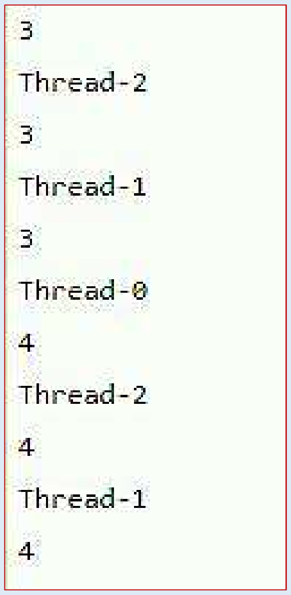
```
   public static void main(String args[])
   {
      // creating three threads
      JavaResumeExp t1=new JavaResumeExp ();
      JavaResumeExp t2=new JavaResumeExp ();
      JavaResumeExp t3=new JavaResumeExp ();
      // call run() method
      t1.start();
      t2.start();
      t2.suspend(); // suspend t2 thread
      // call run() method
      t3.start();
      t2.resume(); // resume t2 thread
   }
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 42

## Output



```
Thread-0
1
Thread-2
1
Thread-1
1
Thread-0
2
Thread-2
2
Thread-1
2
Thread-0
```



```
3
Thread-2
3
Thread-1
3
Thread-0
4
Thread-2
4
Thread-1
4
```

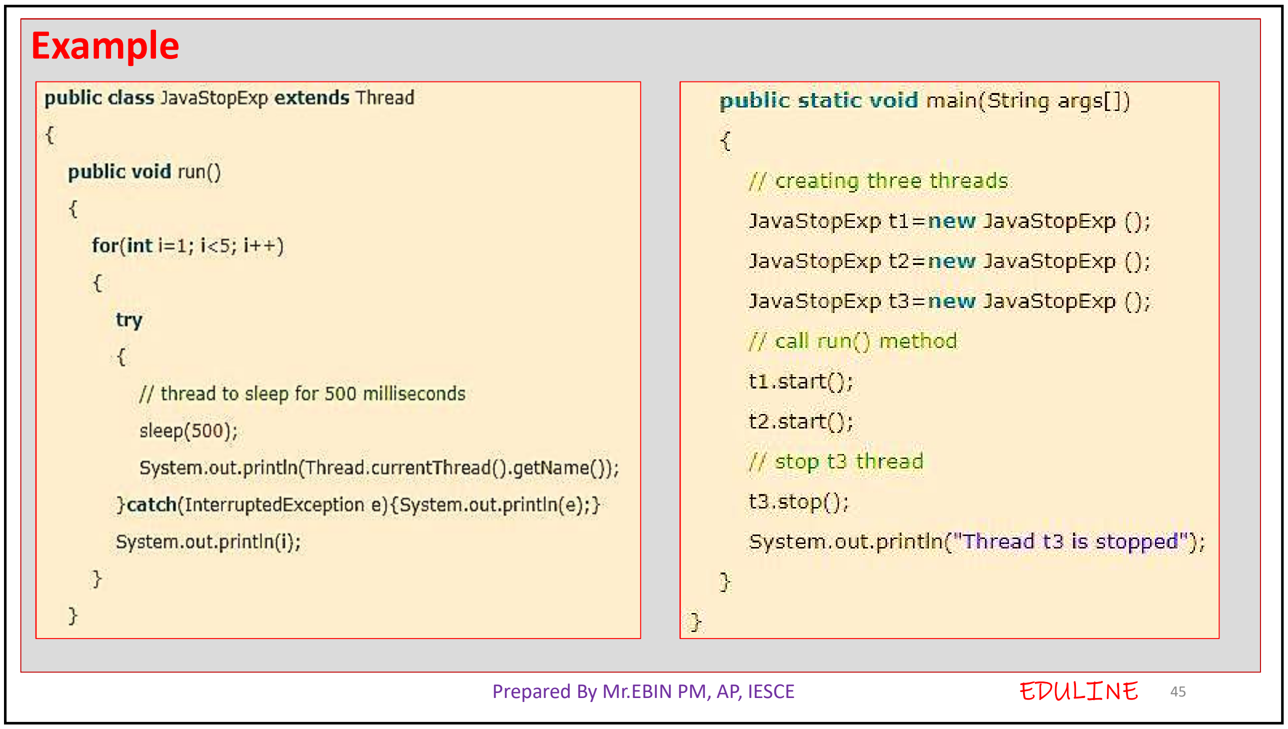
## Thread stop() method

- The stop() method of thread class terminates the thread execution.
- Once a thread is stopped, it cannot be restarted by start() method.

**Syntax**

**public final void stop()**

**public final void stop(Throwable obj)**



## Example

```java
public class JavaStopExp extends Thread
{
   public void run()
   {
      for(int i=1; i<5; i++)
      {
         try
         {
            // thread to sleep for 500 milliseconds
            sleep(500);
            System.out.println(Thread.currentThread().getName());
         }catch(InterruptedException e){System.out.println(e);}
         System.out.println(i);
      }
   }
```

```java
    public static void main(String args[])
    {
       // creating three threads
       JavaStopExp t1=new JavaStopExp ();
       JavaStopExp t2=new JavaStopExp ();
       JavaStopExp t3=new JavaStopExp ();
       // call run() method
       t1.start();
       t2.start();
       // stop t3 thread
       t3.stop();
       System.out.println("Thread t3 is stopped");
    }
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 45