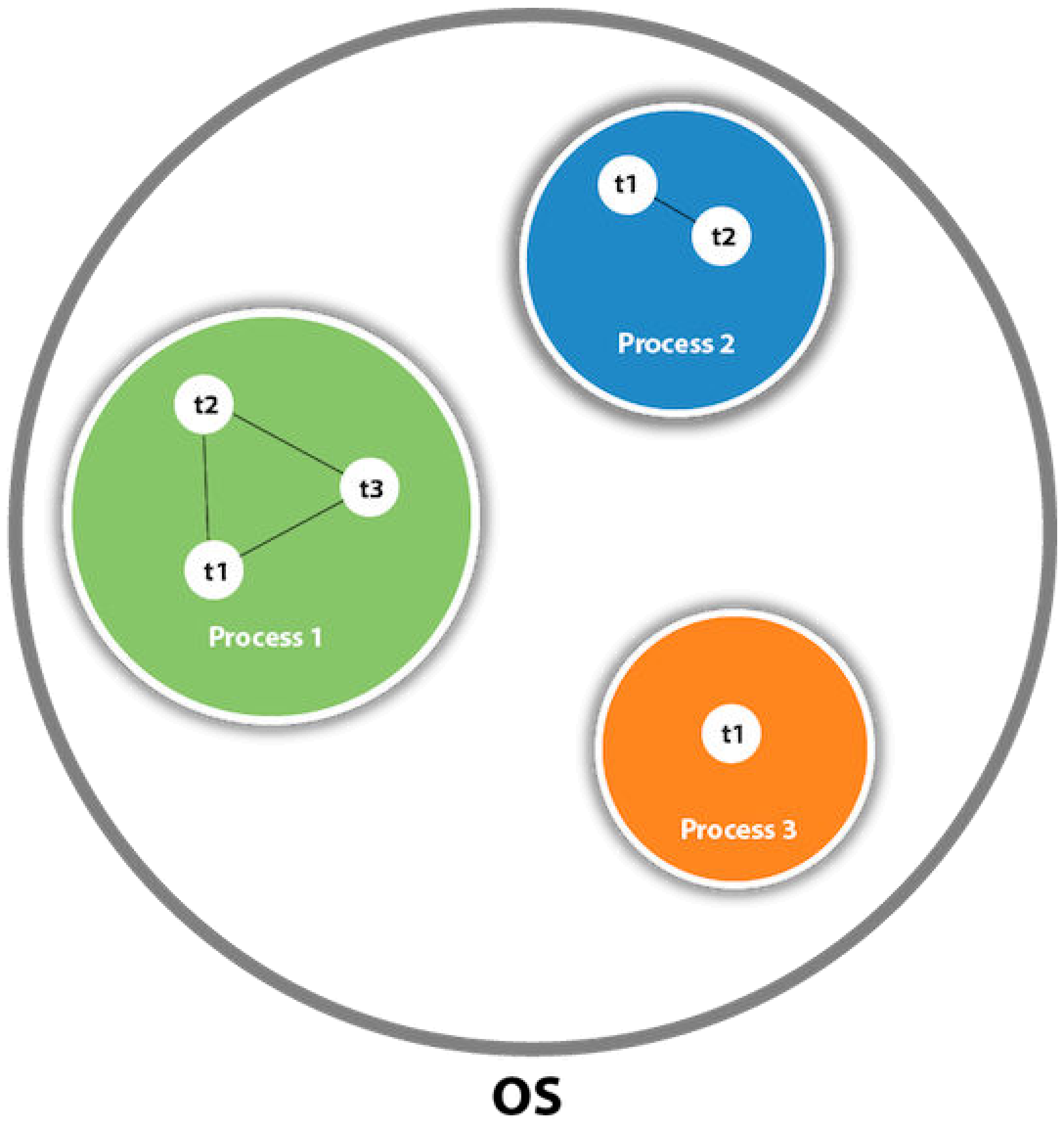
MULTITHREADED PROGRAMMING


t1
t2
Process 2
t2
t3
t1
Process 1
t1
Process 3
OS

Multithreaded Programming

- A multithreaded program contains two or more parts that can run concurrently.
- Each part of a multithreaded program is called a thread.
- Each thread defines a separate path of execution.
- Java provides built in support for multithreaded programming.
- Multithreading is a specialized form of multitasking.

- The two types of multitasking are
    1. Process-based
    2. Thread based

Process-based multitasking is the feature that allows our computer to run two or more programs concurrently.

For ex: It allows us to run the Java compiler at the same time that we are using a text editor.

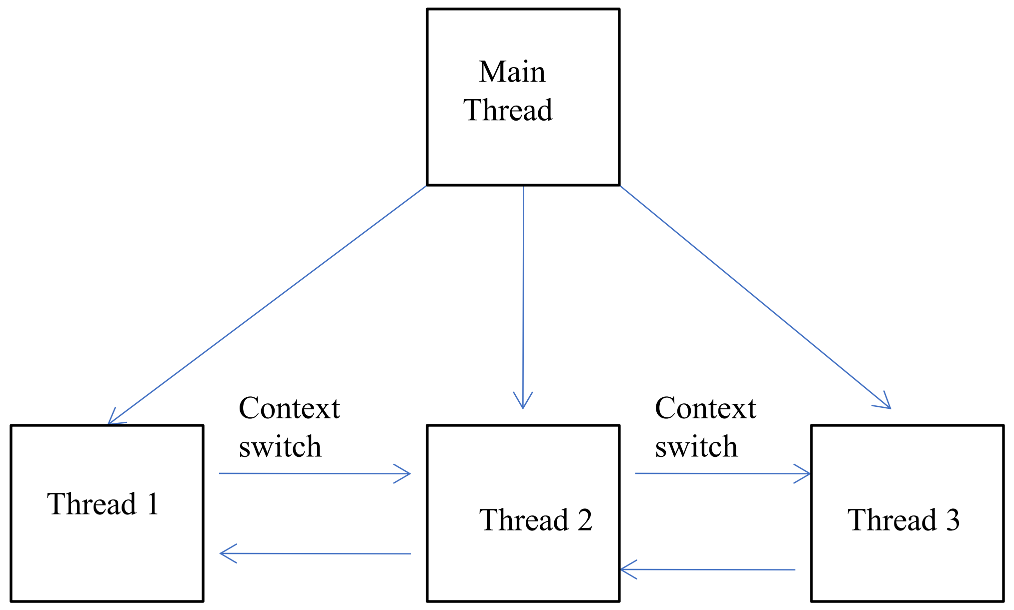
Thread based multitasking:

- Thread is the smallest unit of dispatchable code.
- A single program can perform two or more tasks simultaneously.
- **Ex:** a text editor can format text at the same time that is printing.

Difference between process-based and thread-based multitasking processes:

- Multitasking threads require **less overhead** than multitasking processes.
- Processes are **heavyweight tasks** that require their own **separate address spaces**. But the threads are **lightweight processes** and they **share the same address space.**
- **Context switching** from one process to other process is **costly**. But, context switching from one thread to the next is **low cost**.
- **Interprocess communication is expensive** and limited in process-based multitasking. But, **interthread communication is inexpensive.**
- Process-based multitasking is **not under the control of Java**. But, the multithreaded multitasking is **under the control of Java**.

Multithreaded program



- Multithreading enables us to write very efficient programs that make the maximum use of the CPU, because the idle time can be kept to a minimum.

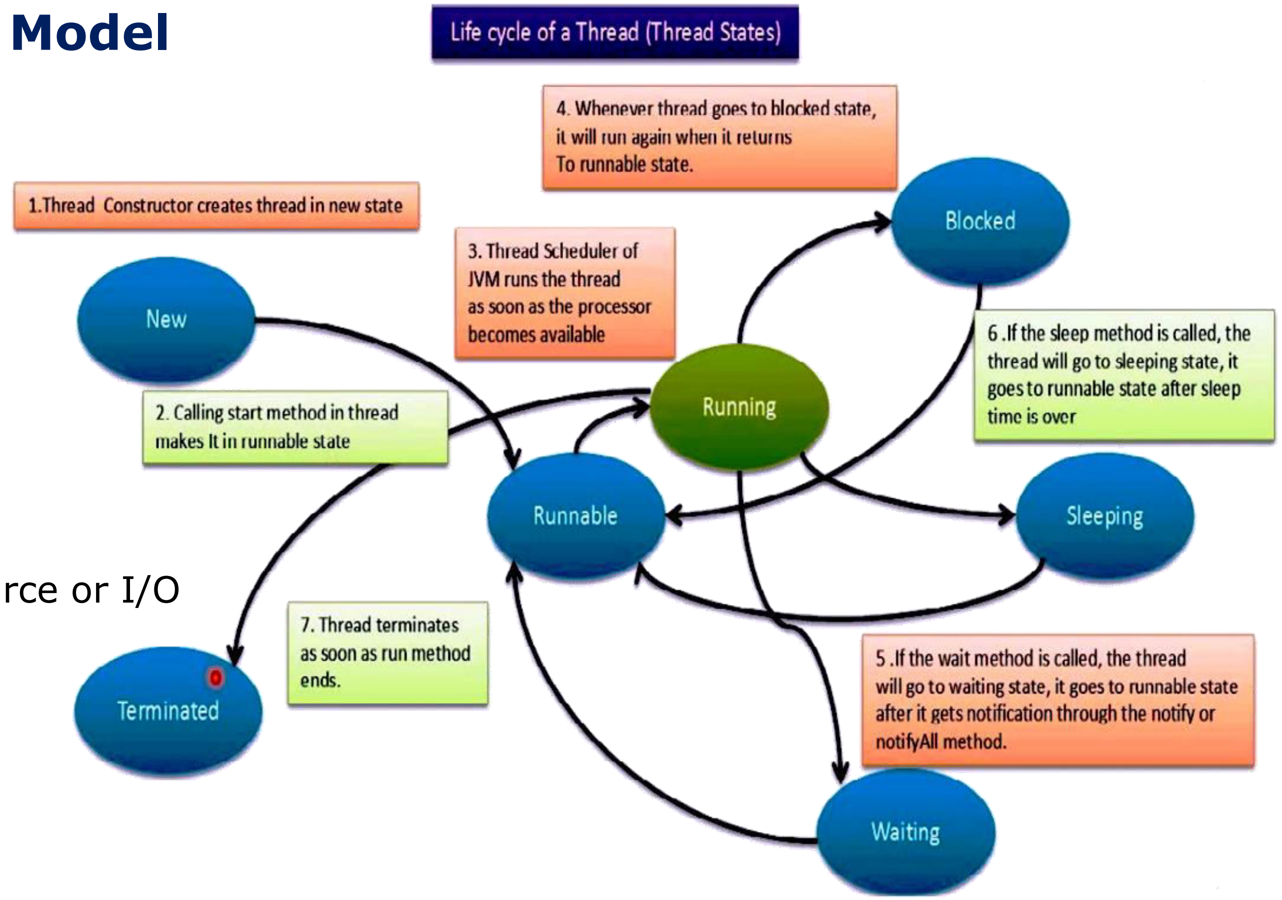
- In a single threaded environment, our program has to wait for each of these tasks to finish before it can proceed to the next one – even though the CPU is sitting idle most of the time.

- Multithreading allows us to gain access to the idle time and put it to good use.

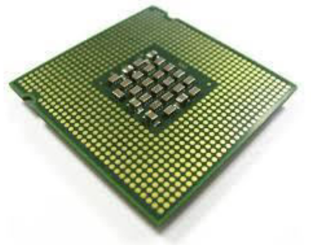
Multithreaded Programming

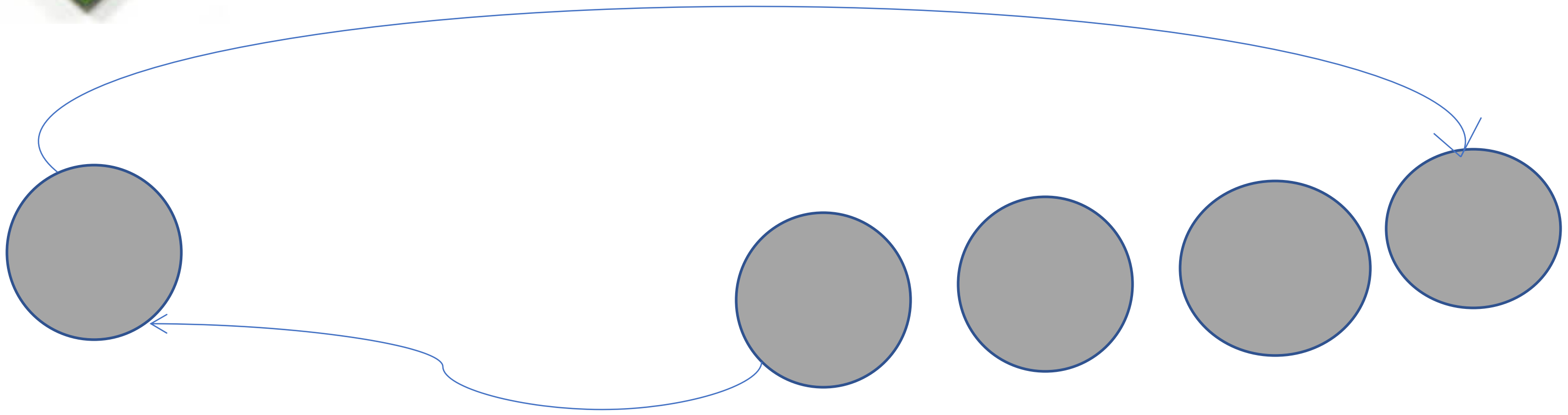
The Java Thread Model

- States of a thread
  - New
  - Ready-to-run
    - waiting for CPU
  - Running
  - Sleeping
  - Suspended
    - can be resumed
  - Blocked
    - waiting for a resource or I/O
  - Terminated


Life cycle of a Thread (Thread States)
4. Whenever thread goes to blocked state, it will run again when it returns To runnable state.
1.Thread Constructor creates thread in new state
3. Thread Scheduler of JVM runs the thread as soon as the processor becomes available
Blocked
New
6 .If the sleep method is called, the thread will go to sleeping state, it goes to runnable state after sleep time is over
Running
2. Calling start method in thread makes It in runnable state
Runnable
Sleeping
7. Thread terminates as soon as run method ends.
5 .If the wait method is called, the thread will go to waiting state, it goes to runnable state after it gets notification through the notify or notifyAll method.
Terminated
Waiting

Runnable state





Example-1: Main thread demo

```java
class CurrentThreadDemo // Controlling the main Thread.
{
    public static void main(String  args[])
    {
        Thread t = Thread.currentThread();
        System.out.println("Current thread: "+t);
        t.setName("MyThread");
        System.out.println("After name change: "+t);
        try
        {
            for( int n = 5 ; n > 0; n-- )
            {
                System.out.println(n);
                Thread.sleep(1000);
            }
        }
        catch( InterruptedException e )
        {   }
    }
}
```

The Main Thread

When a Java program is started the **Main thread runs immediately**. ie, it starts execution.

Importance of Main Thread:

1. It is the thread from which other **“child”** threads will be spawned.

2. Often, it must be the **last thread** to finish execution because it performs various shutdown actions.

The main thread can be controlled through a **Thread** object.

It is done by obtaining a reference to it by calling the method **currentThread()**

The currentThread() is a public static member of thread .

General form :

```
static  Thread  currentThread()
```

It returns a reference to the thread in which it is called.

By using a reference to the main thread, we can control it like any other thread.

```
Current thread: Thread [main, 5, main]
After name change: Thread [My Thread, 5, main]
 5
 4
 3
 2
 1
```

- The first line in the output displays the name of the thread, its priority and the name of its group.
- The second line displays the output after changing the thread name.
- Thread group is a data structure that controls the state of a collection of threads as a whole.
- The sleep() method causes the thread from which it is called to suspend execution for the specified period of milliseconds.

General form:

1. static void sleep(long milliseconds) throws InterruptedException

2. static void sleep(long milliseconds, int nanoseconds) throws InterruptedException

The second form allows us to specify the period in terms of milliseconds and nanoseconds.

We can set the name of a thread by using setName().

General form:

```
final   void   setName(String   threadName)
```

Here threadName specifies the name of the thread.

We can obtain the name of a thread by calling getName().

General form:

```
final   String   getName()
```

The Thread class and the Runnable Interface

Multithreading in Java is facilitated using,

1. **Thread class** and its methods
2. The interface **Runnable**

- To create a new thread our program will have to extend either the **Thread class** or implement the **Runnable interface.**

Extending Thread

- A class that **extends Thread** is another way of **creating a thread** and **creating 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 the **start()** method to **begin the execution** of the **new thread.**

Example-2: Multiple threads demo

```java
class A extends Thread
{
      public void run()
      {
            for( int i = 1;i <= 10 ; i++ )
            {
                  System.out.println("Thread A");

                  try
                  {
                        sleep(1000);
                  }

                  catch( Exception E ) { }
            }

            System.out.println("End of Thread A");
      }
   }
```

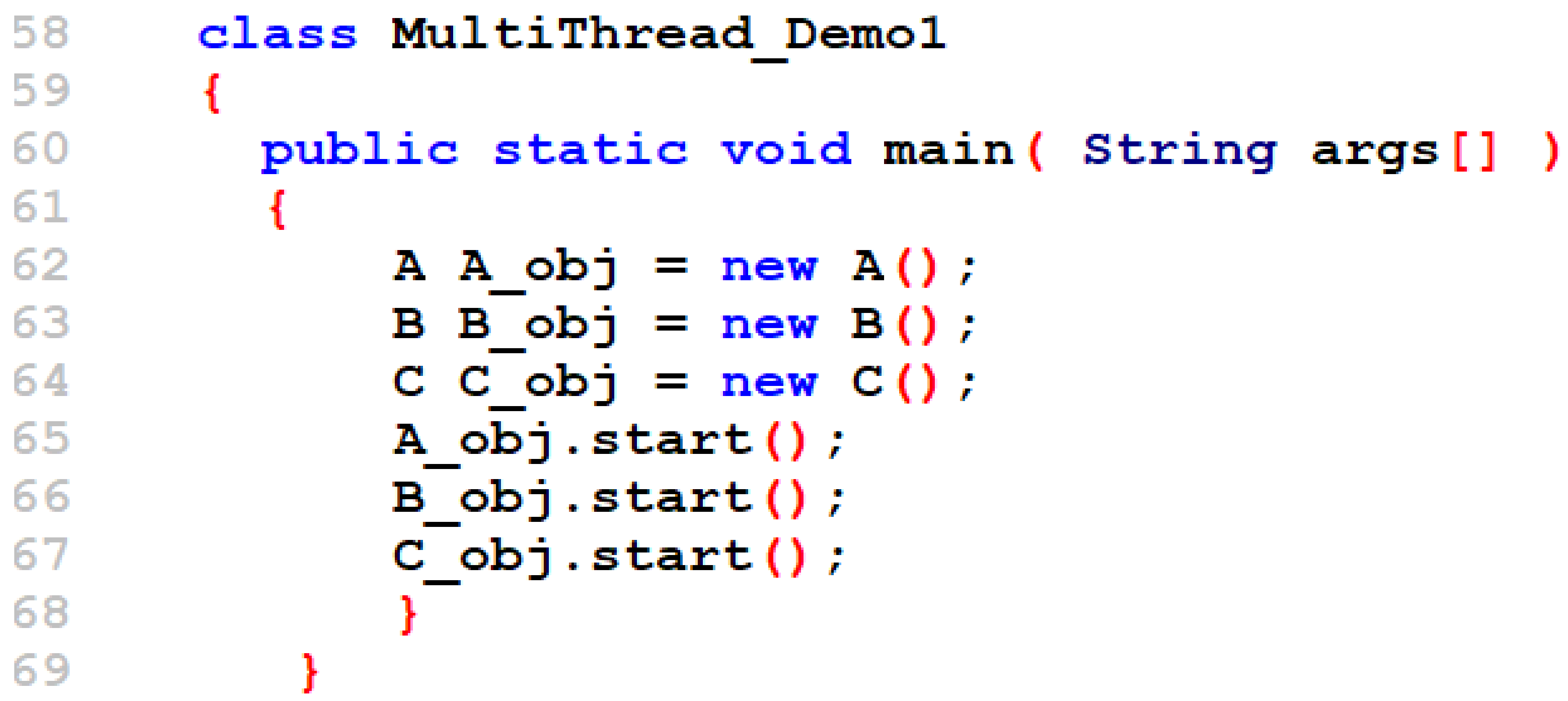
Example-2: Multiple threads demo…

```java
class B extends Thread
{
    public void run()
    {
        for( int i = 1;i <= 10 ; i++ )
        {
            System.out.println("Thread B");

            try
            {
                sleep(1000);
            }

            catch( Exception E ) { }
        }
        System.out.println("End of Thread B");
    }
  }
```

Example-2: Multiple threads demo…

```
class C extends Thread
{
     public void run()
     {
          for( int i = 1;i <= 10 ; i++ )
          {
               System.out.println("Thread C");

               try
               {
                    sleep(1000);
               }

               catch( Exception E ) { }
          }

          System.out.println("End of Thread C");
     }
  }
```

Example-2: Multiple threads demo…



```
class MultiThread_Demo1
{
  public static void main( String args[] )
  {
      A A_obj = new A();
      B B_obj = new B();
      C C_obj = new C();
      A_obj.start();
      B_obj.start();
      C_obj.start();
      }
   }
```

Methods defined by Thread class

Method	Meaning
getName	Obtain a thread's name.
getPriority	Obtain a thread's priority.
isAlive	Determine if a thread is still running.
join	Wait for a thread to terminate.
run	Entry point for the thread.
sleep	Suspend a thread for a period of time.
start	Start a thread by calling its run method.

Question:

- Create 2 threads:
  - T1 to display all the odd numbers upto 10.
  - T2 to display all the even numbers upto 10.

Thread_Demo - 3

```java
class MyThread extends Thread
{
   String thrdName;
   MyThread(String name)
   {
      thrdName = name;
      this.start();
   }

   public void run() // Entry point of thread.
   {
      System.out.println(thrdName + " starting.");
      for(int count=0; count < 5; count++)
      {
          System.out.println("In " + thrdName +
                                   ", count is " + count);
          try
          {  Thread.sleep(2000); }

          catch(InterruptedException exc)
          {  }
      }
      System.out.println(thrdName + " terminating.");
   }
}
```

Thread_Demo – 3…

```java
class MultiThread_Demo2
{
  public static void main(String[] args)
  {
    System.out.println("Main thread starting.");

    MyThread m1 = new MyThread("Child #1");

    for(int i=0; i < 10; i++)
    {
      System.out.println("In main thread..count="+i);

      try
      {  Thread.sleep(2000); }

      catch(InterruptedException exc)
      { }
    }

    System.out.println("Main thread ending.");
  }
}
```

Thread_Demo – 4

```java
class MyThread extends Thread
{
      int Thread_num;
      MyThread( int num )
      {
            Thread_num = num;
            start();
      }
      public void run()
      {
            for( int i = 0 ; i < 5 ; i++ )
            {
                  System.out.println("Thread-"+Thread_num+",i= "+i);

                  try
                  { Thread.sleep(2000); }

                  catch(InterruptedException e)
                  { }
            }
      }
}
```

Thread_Demo – 4…

```
class MultiThread_Demo3
{
    public static void main(String arggs[])
    {
        MyThread T1 = new MyThread( 1 );
        MyThread T2 = new MyThread( 2 );
        MyThread T3 = new MyThread( 3 );

        for(int i = 0 ; i < 5 ; i++ )
        {
            System.out.println(" Main Thread ");
            try
            {
                Thread.sleep( 2000 );
            }
            catch(Exception e)
            { }
        }
    }
}
```

Thread_Demo – 5: isAlive()

```java
class MyThread extends Thread
{
	int Thread_num;
	MyThread( int num )
	{
		Thread_num = num;
		start();
	}
	public void run()
	{ System.out.println("Thread-" +Thread_num ); }
}
class MultiThread_Demo4
{
	public static void main(String arggs[])
	{
		MyThread T1 =new MyThread( 1 );
		MyThread T2 =new MyThread( 2 );
		Thread main_thread = Thread.currentThread();

		System.out.println("T1 Is Alive = "+T1.isAlive());
		System.out.println("T2 Is Alive = "+T2.isAlive());
		System.out.println("main Is Alive =
		                    "+main_thread.isAlive());
	}
}
```

Thread Priorities

- Priority is used to decide when to switch from one running thread to next (context switch)

- setPriority() method: java.lang.Thread.setPriority()

Purpose:	Change thread-priority
Syntax:	`public final void setPriority(int priorityLevel)`
Example:	`t.setPriority(8)`

- getPriority() method: java.lang.Thread.getPriority()

Purpose:	Obtain current thread-priority
Syntax:	`public final int getPriority()`
Example:	`int p = t.getPriority()`

Note: The *priorityLevel* is an integer

- » between MIN_PRIORITY (= 1) and MAX_PRIORITY (= 10)
- » default NORM_PRIORITY = 5
- » all are defined as *static final* variables in Thread class

Thread_Demo – 6: Thread priority

```java
class MyThread extends Thread
{
     int Thread_num;
     MyThread( int num )
     {
          Thread_num = num;
          start();
     }
     public void run()
     {
          System.out.println("Thread-"+Thread_num+",
                                   priority = "+getPriority() );
     }
}
class MultiThread_Demo6
{
     public static void main(String arggs[])
     {
          MyThread T1 =new MyThread( 1 );
          MyThread T2 =new MyThread( 2 );
          Thread main_thread = Thread.currentThread();
          System.out.println("MainThread, priority = "
                                   +main_thread.getPriority());
     }
}
```

Using isAlive() and join()

Two ways to determine whether a thread has finished or not are:

1. By calling **isAlive()** method defined by **Thread** on the thread.

   General form:

   ```
   final boolean isAlive()
   ```

   It **returns true** if the thread upon which it is called is **running,** else **returns false**.

2. By calling the method **join()**

   General form:

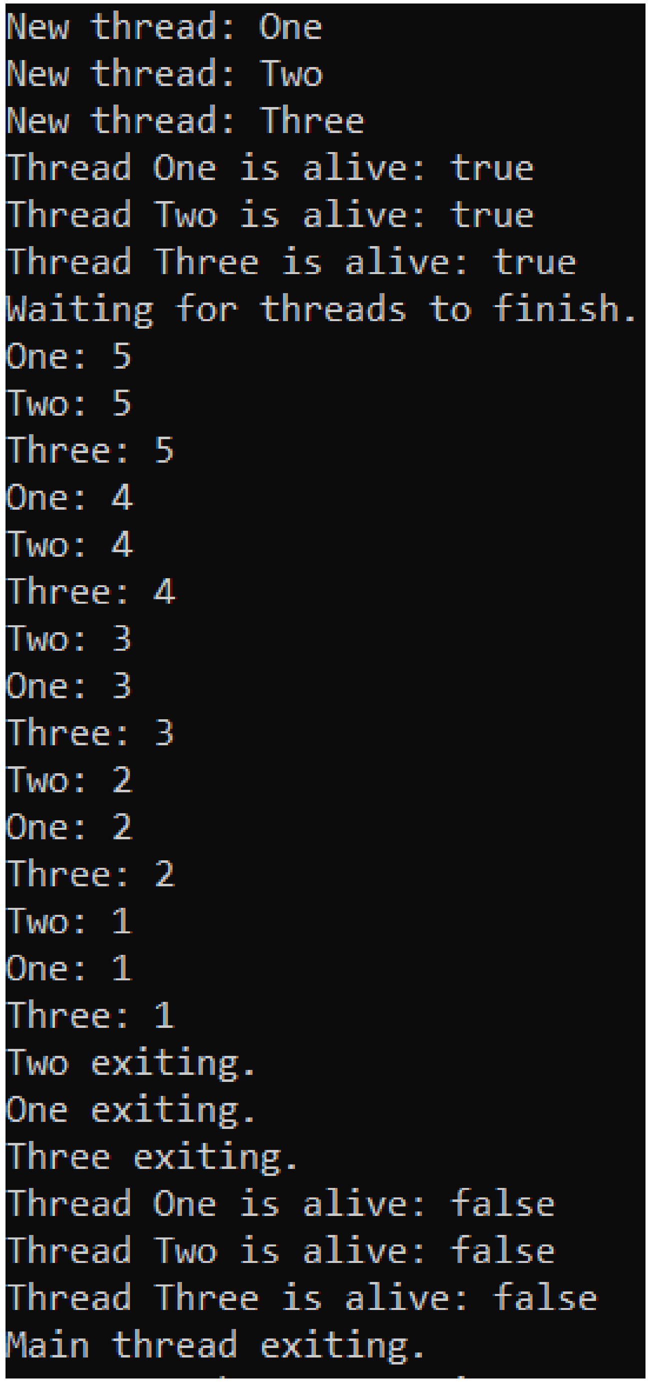
   ```
   final void join() throws InterruptedException
   ```

   This method waits until the thread on which it is called terminates.

Thread_Demo – 7: use of join()

```java
// Using join() to wait for threads to finish.
class NewThread extends Thread
{
   String name; // name of thread
   NewThread(String threadname)
   {
      name = threadname;
      System.out.println("New thread: " +name);
      start(); // Start the thread
   }

   public void run() {
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println(name + ": " + i);
            Thread.sleep(1000);
         }
      } catch (InterruptedException e) {
         System.out.println(name + " interrupted.");
      }
      System.out.println(name + " exiting.");
   }
}
```

```java
class DemoJoin {
  public static void main(String args[]) {
    NewThread ob1 = new NewThread("One");
    NewThread ob2 = new NewThread("Two");
    NewThread ob3 = new NewThread("Three");

    System.out.println("Thread One is alive: "+ ob1.isAlive());
    System.out.println("Thread Two is alive: "+ ob2.isAlive());
    System.out.println("Thread Three is alive: "+ ob3.isAlive());

    try {   // wait for threads to finish
      System.out.println("Waiting for threads to finish.");
      ob1.join();
      ob2.join();
      ob3.join();
    } catch (InterruptedException e) {
      System.out.println("Main thread Interrupted");
    }

    System.out.println("Thread One is alive: "+ ob1.isAlive());
    System.out.println("Thread Two is alive: "+ ob2.isAlive());
    System.out.println("Thread Three is alive: "+ ob3.isAlive());
    System.out.println("Main thread exiting.");
  }
}
```



```
New thread: One
New thread: Two
New thread: Three
Thread One is alive: true
Thread Two is alive: true
Thread Three is alive: true
Waiting for threads to finish.
One: 5
Two: 5
Three: 5
One: 4
Two: 4
Three: 4
Two: 3
One: 3
Three: 3
Two: 2
One: 2
Three: 2
Two: 1
One: 1
Three: 1
Two exiting.
One exiting.
Three exiting.
Thread One is alive: false
Thread Two is alive: false
Thread Three is alive: false
Main thread exiting.
```

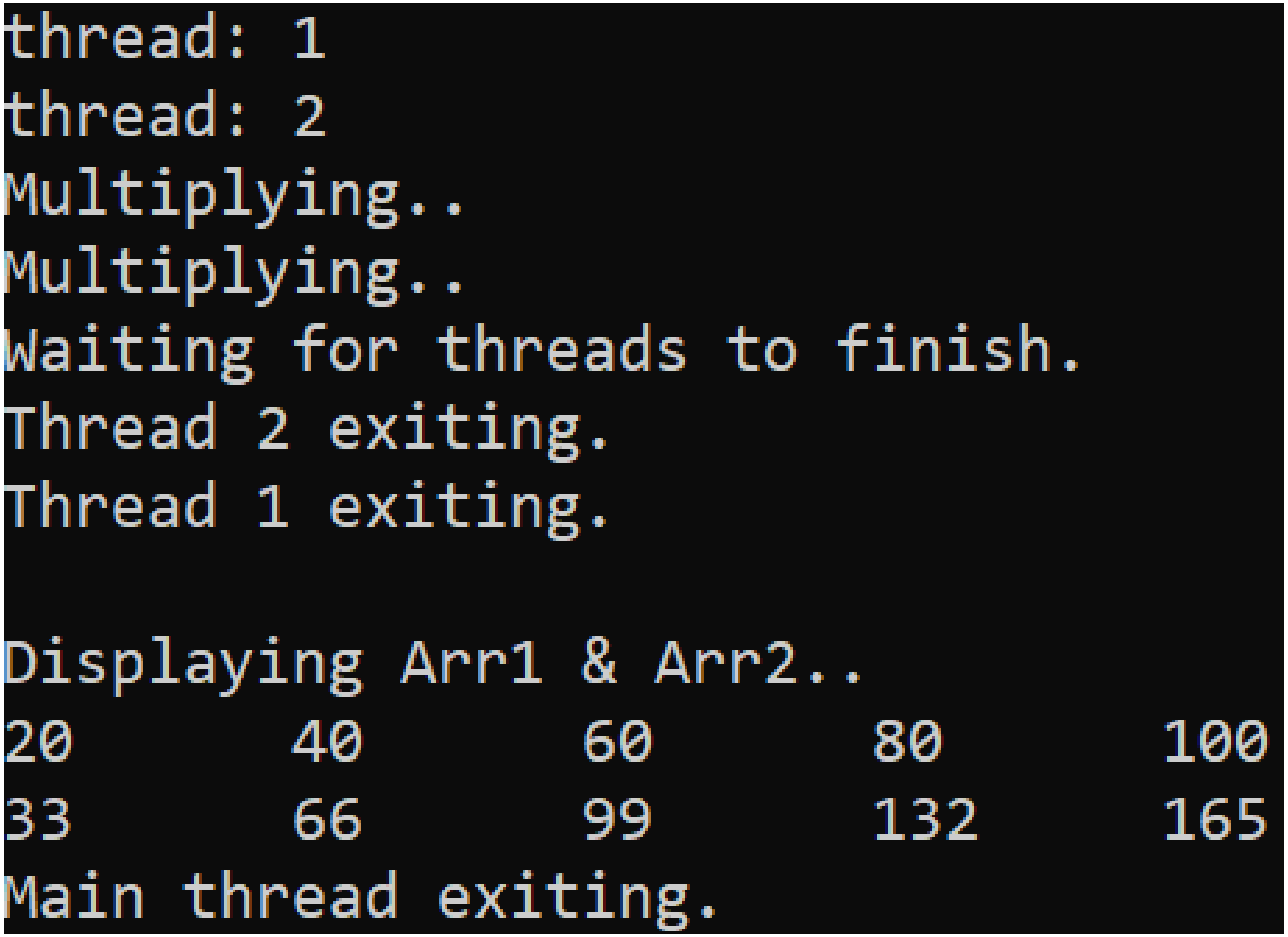
Question:

- Given two integer arrays Arr1 & Arr2, and two values val1 & val2.
- Create two threads T1 and T2:
  - T1: For multiplying all elements of Arr1 by val1
  - T2: For multiplying all elements of Arr2 by val2
- Display the contents of Arr1 and Arr2 in the main thread.

Thread_Demo – 8: Excercise

```java
class NewThread extends Thread
{
  int Thread_Num, val;
  int Arr[];
  NewThread( int no , int A[] , int v )
  {
    Thread_Num = no;
    val = v; Arr = A;
    System.out.println("thread:  " +Thread_Num);
    start();
  }
  public void run()
  {
    multiply();
    System.out.println("Thread "+Thread_Num + " exiting.");
  }
  void multiply()
  {
      System.out.println("Multiplying..");
      for( int i = 0 ; i < Arr.length ; i++ )
        Arr[i] *= val;
  }
}
```

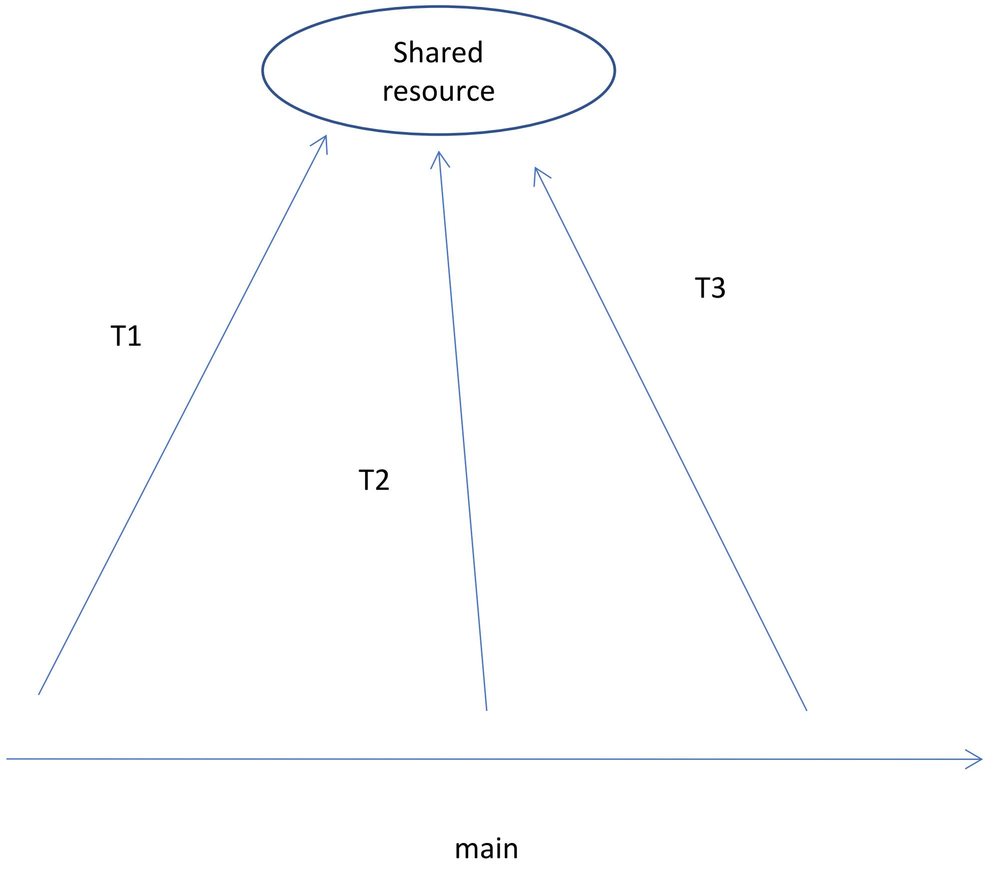
```java
class DemoJoin2 {
  public static void main(String args[]) {

    int Arr1[] = { 10 , 20 , 30 , 40 , 50 };
    int Arr2[] = { 11 , 22 , 33 , 44 , 55 };
    int val1 = 2 , val2 = 3;
    NewThread T1 = new NewThread( 1 , Arr1 , val1 );
    NewThread T2 = new NewThread( 2 , Arr2 , val2 );
    try {   // wait for threads to finish
      System.out.println("Waiting for threads to finish.");
      T1.join();
      T2.join();
     } catch (InterruptedException e) {
      System.out.println("Main thread Interrupted");
    }

    System.out.println("\nDisplaying Arr1 & Arr2..");

    // Display Arr1 and Arr2
    System.out.println("\nMain thread exiting.");
  }
}
```



```
thread: 1
thread: 2
Multiplying..
Multiplying..
Waiting for threads to finish.
Thread 2 exiting.
Thread 1 exiting.

Displaying Arr1 & Arr2..
20      40      60      80      100
33      66      99      132     165
Main thread exiting.
```

Synchronization

- When two or more threads need concurrent access to a shared data resource, they need to take care to only access the data one at a time.
- For example, one thread may try to read a record from a file while another is still writing to the same file. Depending on the situation, we may get strange results.
- Java enables us to overcome this problem using a technique known as synchronization.
- Keyword synchronized helps to solve such problem by keeping a watch on such locations.
- For example the method that will read information from a file and the method that will update the same file may be declared as synchronized.


Shared resource
T1
T2
T3
main

```
synchronized void update()
{


}
```

- When we declare a method synchronized, java creates a “monitor” and hands it over to the thread that calls the method first time.
- As long as the thread holds the monitor, no other thread can enter the synchronized section of the code.

Marking block of code as synchronized

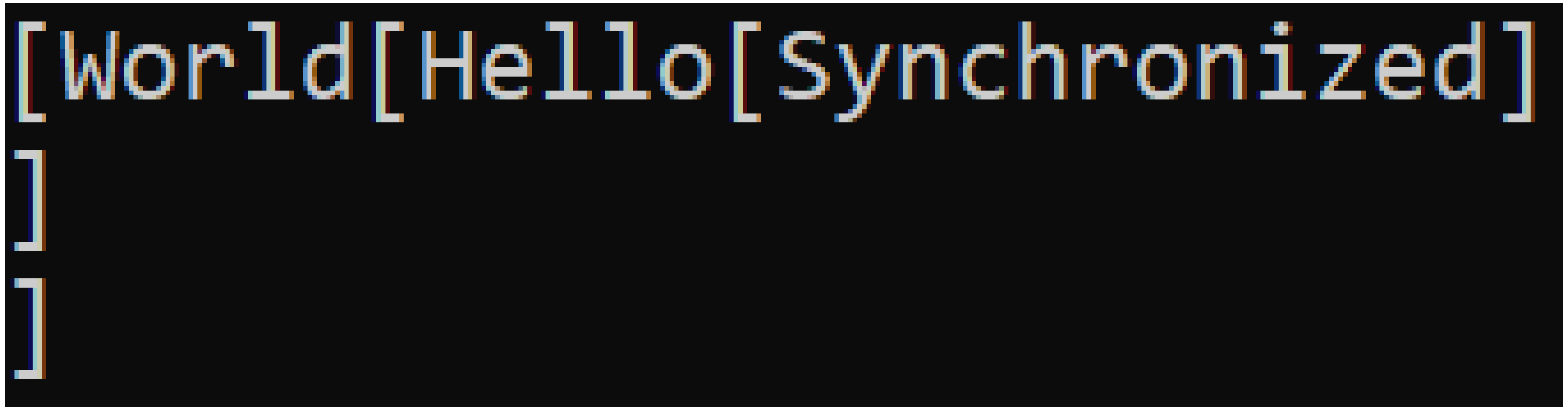
```
 public void run()
{
             // other code
            synchronized(obj)
             {
                        obj.method(msg);
             }
  }
```

Whenever a thread has completed its work of using synchronized method(or block of code), it will handover the monitor to the next thread that is ready to use the same resources.

Thread_Demo – 9:
without synchronization

```
class Callme // This program is not synchronized.
{
  void call(String msg)
  {
    System.out.print("[" + msg);
    try
    { Thread.sleep(1000);   }
    catch(InterruptedException e)
    {     }
    System.out.println("]");
  }
}

class Caller extends Thread {
  String msg;
  Callme target;
  public Caller(Callme targ, String s) {
    target = targ;
    msg = s;
    start();
  }

  public void run()
  {  target.call(msg); }
}
```

```java
class NoSynch
{
  public static void main(String args[])
  {
    Callme target = new Callme();
    Caller ob1 = new Caller(target, "Hello");
    Caller ob2 = new Caller(target, "Synchronized");
    Caller ob3 = new Caller(target, "World");

    // wait for threads to end
    try
    {
      ob1.join();
      ob2.join();
      ob3.join();
    }
    catch(InterruptedException e)
    {  }
  }
}
```

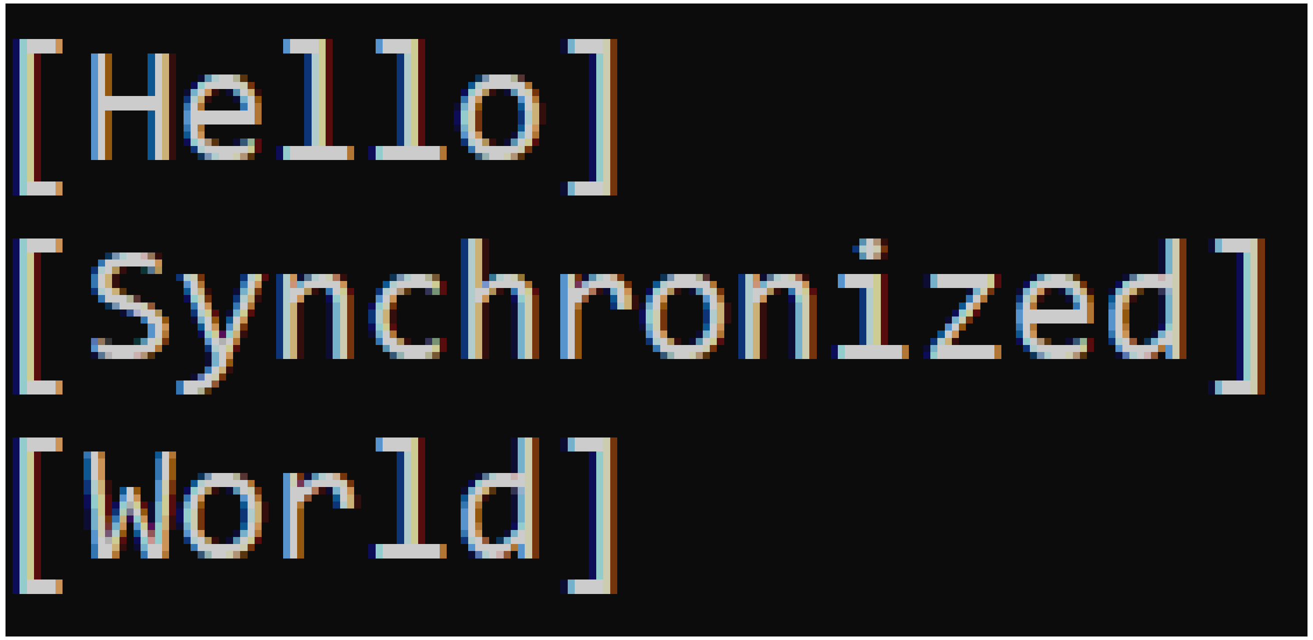


```
[World[Hello[Synchronized]
]
]
```

Thread_Demo – 10 :
synchronization Sol-1

```
class Callme
{
  synchronized void call(String msg)
  {
    System.out.print("[" + msg);
    try
    { Thread.sleep(1000); }
    catch (InterruptedException e)
    { }
    System.out.println("]");
  }
}

class Caller extends Thread
{
  String msg;
  Callme target;
  public Caller(Callme targ, String s)
  {
    target = targ;    msg = s;
    start();
  }
  public void run()
  { target.call(msg); }
}
```

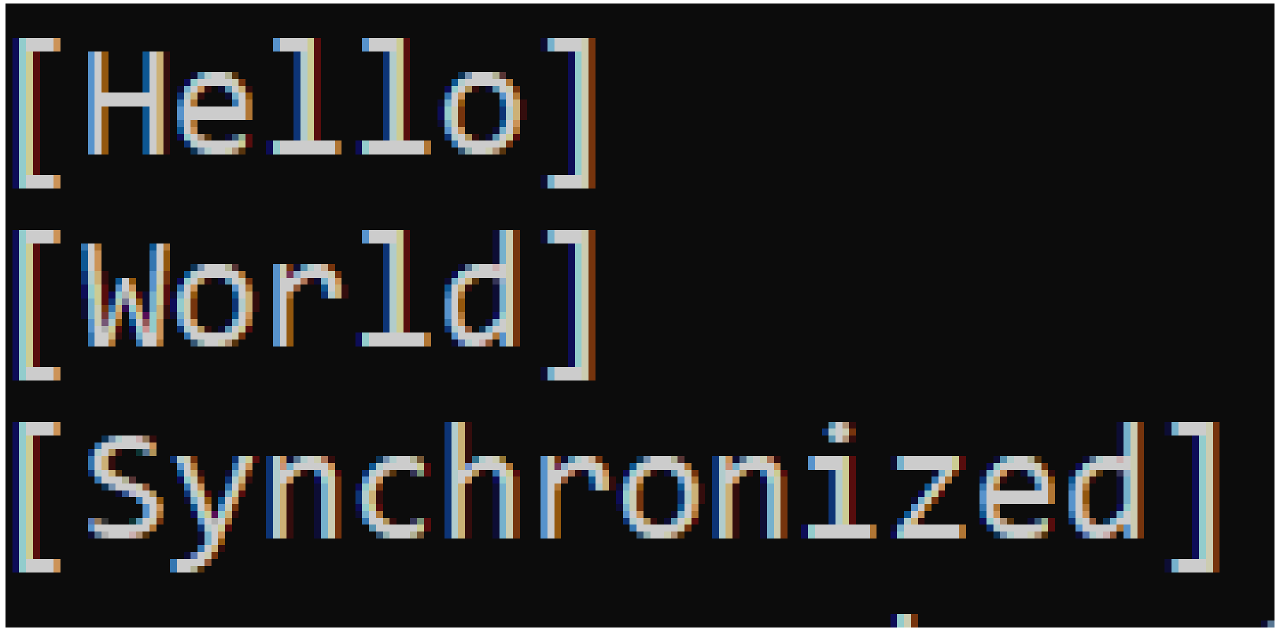
```java
class Synch1
{
  public static void main(String args[])
  {
    Callme target = new Callme();
    Caller ob1 = new Caller(target, "Hello");
    Caller ob2 = new Caller(target, "Synchronized");
    Caller ob3 = new Caller(target, "World");

    // wait for threads to end
    try
    {
      ob1.join();
      ob2.join();
      ob3.join();
    }
    catch(InterruptedException e)
    {   }
  }
}
```


```
[Hello]
[Synchronized]
[World]
```

Thread_Demo – 11 :
synchronization Sol-2

```java
class Callme
{
   void call(String msg)
   {
      System.out.print("[" + msg);
      try
      { Thread.sleep(1000); }
      catch (InterruptedException e)   { }
      System.out.println("]");
   }
}
class Caller extends Thread {
   String msg;
   Callme target;
   public Caller(Callme targ, String s)
   {
      target = targ;     msg = s;
      start();
   }
   public void run()
   {
         synchronized(target)
         { target.call(msg); }
   }
}
```

```java
class Synch2
{
  public static void main(String args[])
  {
    Callme target = new Callme();
    Caller ob1 = new Caller(target, "Hello");
    Caller ob2 = new Caller(target, "Synchronized");
    Caller ob3 = new Caller(target, "World");

    // wait for threads to end
    try
    {
      ob1.join();
      ob2.join();
      ob3.join();
    }
    catch(InterruptedException e)
    {   }
  }
}
```



```
[Hello]
[World]
[Synchronized]
```

Thread_Demo – 12: without synchronization

```java
class SumArray
{
  private int sum;

  int sumArray(int[] nums)
  {
    sum = 0; // reset sum

    for(int i=0; i<nums.length; i++) {
      sum += nums[i];
      System.out.println("Running total for " +
             Thread.currentThread().getName() +
             " is " + sum);
      try {
        Thread.sleep(10); // allow task-switch
      }
      catch(InterruptedException exc) {
        System.out.println("Thread interrupted.");
      }
    }
    return sum;
  }
}
```

```java
class MyThread extends Thread
{
  static SumArray sa = new SumArray();
  int[] a;
  int answer;

 MyThread(String name, int[] nums)
  {
    a = nums;
    setName(name);
    start();
  }

  public void run() {
    int sum;

    System.out.println(getName() + " starting.");

    answer = sa.sumArray(a);
    System.out.println("Sum for " + getName() +
                       " is " + answer);

    System.out.println(getName() + " terminating.");
  }
}
```

```java
class NoSync
{
  public static void main(String[] args)
  {
    int[] a = {1, 2, 3, 4, 5};
    int[] b = {10, 20, 30, 40, 50};


    MyThread mt1 = new MyThread("Child #1", a);
    MyThread mt2 = new MyThread("Child #2", b);

    try
    {
      mt1.join();
      mt2.join();
    }
    catch(InterruptedException exc)
    {  System.out.println("Main thread interrupted."); }
  }
}
```

Thread_Demo – 13: synchronization Sol-1

```java
// Use synchronize to control access.
class SumArray
{
  private int sum;

  synchronized int sumArray(int[] nums)
  {
    sum = 0; // reset sum

    for(int i=0; i<nums.length; i++) {
      sum += nums[i];
      System.out.println("Running total for " +
             Thread.currentThread().getName() +
             " is " + sum);
      try {
        Thread.sleep(10); // allow task-switch
      }
      catch(InterruptedException exc) {
        System.out.println("Thread interrupted.");
      }
    }
    return sum;
  }
}
```

```java
class MyThread extends Thread
{
  static SumArray sa = new SumArray();
  int[] a;
  int answer;

  MyThread(String name, int[] nums)
  {
    a = nums;
    setName(name);
    start();
  }

  public void run()
  {
    int sum;
    System.out.println(getName() + " starting.");

    answer = sa.sumArray(a);
    System.out.println("Sum for " + getName() +
                       " is " + answer);

    System.out.println(getName() + " terminating.");
  }
}
```

```java
class Sync
{
  public static void main(String[] args)
  {
    int[] a = {1, 2, 3, 4, 5};
    int[] b = {10, 20, 30, 40, 50};


    MyThread mt1 = new MyThread("Child #1", a);
    MyThread mt2 = new MyThread("Child #2", b);

    try
    {
      mt1.join();
      mt2.join();
    }
    catch(InterruptedException exc) {
      System.out.println("Main thread interrupted.");
    }
  }
}
```

Thread_Demo – 14: synchronization Sol-2

```java
// Use synchronize to control access.
class SumArray
{
  private int sum;

  int sumArray(int[] nums)
  {
    sum = 0; // reset sum

    for(int i=0; i<nums.length; i++) {
      sum += nums[i];
      System.out.println("Running total for " +
             Thread.currentThread().getName() +
             " is " + sum);
      try {
        Thread.sleep(10); // allow task-switch
      }
      catch(InterruptedException exc) {
        System.out.println("Thread interrupted.");
      }
    }
    return sum;
  }
}
```

```
class MyThread extends Thread
{
  static SumArray sa = new SumArray();
  int[] a;
  int answer;

  MyThread(String name, int[] nums)
  {
    a = nums;
    setName(name);
    start();
  }
  public void run()
  {
    int sum;
    System.out.println(getName() + " starting.");

    synchronized( sa )
    {  answer = sa.sumArray(a); }
    System.out.println("Sum for " + getName() +
                       " is " + answer);

    System.out.println(getName() + " terminating.");
  }
}
```

```java
class Sync2
{
  public static void main(String[] args)
  {
    int[] a = {1, 2, 3, 4, 5};
    int[] b = {10, 20, 30, 40, 50};


    MyThread mt1 = new MyThread("Child #1", a);
    MyThread mt2 = new MyThread("Child #2", b);

    try
    {
      mt1.join();
      mt2.join();
    }
    catch(InterruptedException exc) {
      System.out.println("Main thread interrupted.");
    }
  }
}
```

Exercise:

- Write a multithreaded java program to do the following.
- a. Using first thread generate odd numbers within 100 and store.
- b. Using second thread generate multiples of 5 within 100 and store.
- c. The main thread should display the numbers stored by above threads.

Exercise-2:

- Write a multithreaded java program to do the following.
- a. Using first thread generate odd numbers within 100 and store.
- b. Using second thread generate multiples of 5 within 100 and store.
- c. The main thread should read and display the numbers, which are common.

Runnable interface

Creating a Thread

A thread can be created by **instantiating an object** of type Thread.

The 2 ways defined by Java for the creation of thread are:

1. By **Implementing the Runnable interface.**
2. By **Extending the Thread class.**

Implementing Runnable:

We can construct a thread on any object that implements **Runnable.**

To implement Runnable, a class need to implement a single method called **run().**

General form:

```
public  void  run()
```

- Inside run(), we will define the code that constitutes the new thread.

- The run() can call other methods, use other classes and declare variables like the main thread.
- run() establishes an entry point for another concurrent thread of execution within our program.

After creating a class that implements Runnable ,we can instantiate an object of type Thread from within that class.

Constructors defined by Thread:

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

Here,

- threadOb is an instance of a class that implements Runnable interface. It defines where the execution of the thread will begin.
- threadName is the name of the new thread.

```
class  MyThread  extends Thread
{
     public void run()
     {
                    //statements
     }
}
class prg
{
    public static void main(String args[])
   {
          MyThread   t = new MyThread();
           t.start();
    }
}
```

```
class  MyThread  extends Thread
{
    MyThread()
    {
           start();
     }
     public void run()
     {
                  //statements
     }
}
class prg
{
    public static void main(String args[])
   {
          MyThread   t = new MyThread();
    }
}
```

```
class  MyThread  extends Thread
{
     public void run()
     {
                    // statements
     }
}
class prg
{
    public static void main(String args[])
    {
          MyThread   t = new MyThread();
           t.start();
    }
}
```

```
class  MyThread  implements Runnable
{
     public void run()
     {
                    //statements
     }
}
class prg
{
    public static void main(String args[])
    {
          MyThread   mt = new MyThread();
          Thread     t = new Thread( mt )
           t.start();
    }
}
```

```
class  MyThread  extends Thread
{
     MyThread()
     {
              start();
      }
      public void run()
      {
                       //statements
      }
}
class prg
{
     public static void main(String args[])
    {
            MyThread   t = new MyThread();
     }
}
```

```
class  MyThread  implements Runnable
{
     Thread    t;
     MyThread()
     {        t = new Thread( this )
              t.start();
      }
      public void run()
      {
                       //statements
      }
}
class prg
{     public static void main(String args[])
    {
            MyThread   mt = new MyThread();
     }
}
```

Thread_Demo – 15 : Runnable-1

```
// Create a thread by implementing Runnable.

class MyThread implements Runnable
{
  public void run()
  {
    System.out.println("child-thread is starting.");

      for(int count=0; count < 10; count++)
      {
        System.out.println("In child-thread,count is "+count);

        try
        {
            Thread.sleep(400);
        }

        catch(InterruptedException exc)
        { }
      }

      System.out.println("child-thread is terminating..");
    }
 }
```

```java
class RunnableDemo
{
  public static void main(String[] args)
  {
    // First, construct a MyThread object.
    MyThread mt = new MyThread();

    // Next, construct a thread from that object.
    Thread newThrd = new Thread( mt );

    // Finally, start execution of the thread.
    newThrd.start();
    for(int i=0; i < 5; i++)
    {
        System.out.println("In Main thread..");
        try
        {   Thread.sleep(1000); }

        catch(InterruptedException exc)
        { }
    }
    System.out.println("Main thread ending.");
  }
}
```

Thread_Demo – 16 : Runnable-2

```java
class MyThread implements Runnable
{
  Thread T;
  MyThread()
  {
     T = new Thread( this );
     T.start();
  }
  public void run()
  {
    System.out.println( "child-thread is starting.");

      for(int count=0; count < 10; count++)
      {
        System.out.println("In child-thread, count is " + count);

        try
        { Thread.sleep(500); }

        catch(InterruptedException exc)
        { }
      }
      System.out.println(" child-thread is terminating..");
    }
 }
```

```java
class RunnableDemoImproved
{
  public static void main(String[] args)
  {
    System.out.println("Main thread starting.");

    MyThread mt = new MyThread();

    for( int i = 0; i < 5; i++ )
    {
        System.out.println("In Main thread..");

        try
        {
            Thread.sleep(1000);
        }

        catch(InterruptedException exc)
        { }
    }
    System.out.println("Main thread ending.");
  }
}
```

Exercise-1: Runnable Interface

- Write a multithreaded java program to do the following.
- a. Using first thread generate odd numbers within 100 and store.
- b. Using second thread generate multiples of 5 within 100 and store.
- c. The main thread should display the numbers stored by above threads.

Exercise-2: Runnable Interface

- Write a multithreaded java program to do the following.
- a. Using first thread generate odd numbers within 100 and store.
- b. Using second thread generate multiples of 5 within 100 and store.
- c. The main thread should read and display the numbers, which are common.

Interthread Communication

Methods must be called from inside of synchronized method.

All three are final methods.

1. **wait( )** tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls **notify( )**.

   final void wait( ) throws InterruptedException

2. **notify( )** wakes up the first thread that called wait( ) on the same object.

   final void notify( )

3. **notifyAll( )** wakes up all the threads that called wait( ) on the same object. The highest priority thread will run first.

   final void notifyAll( )