Threads in Java

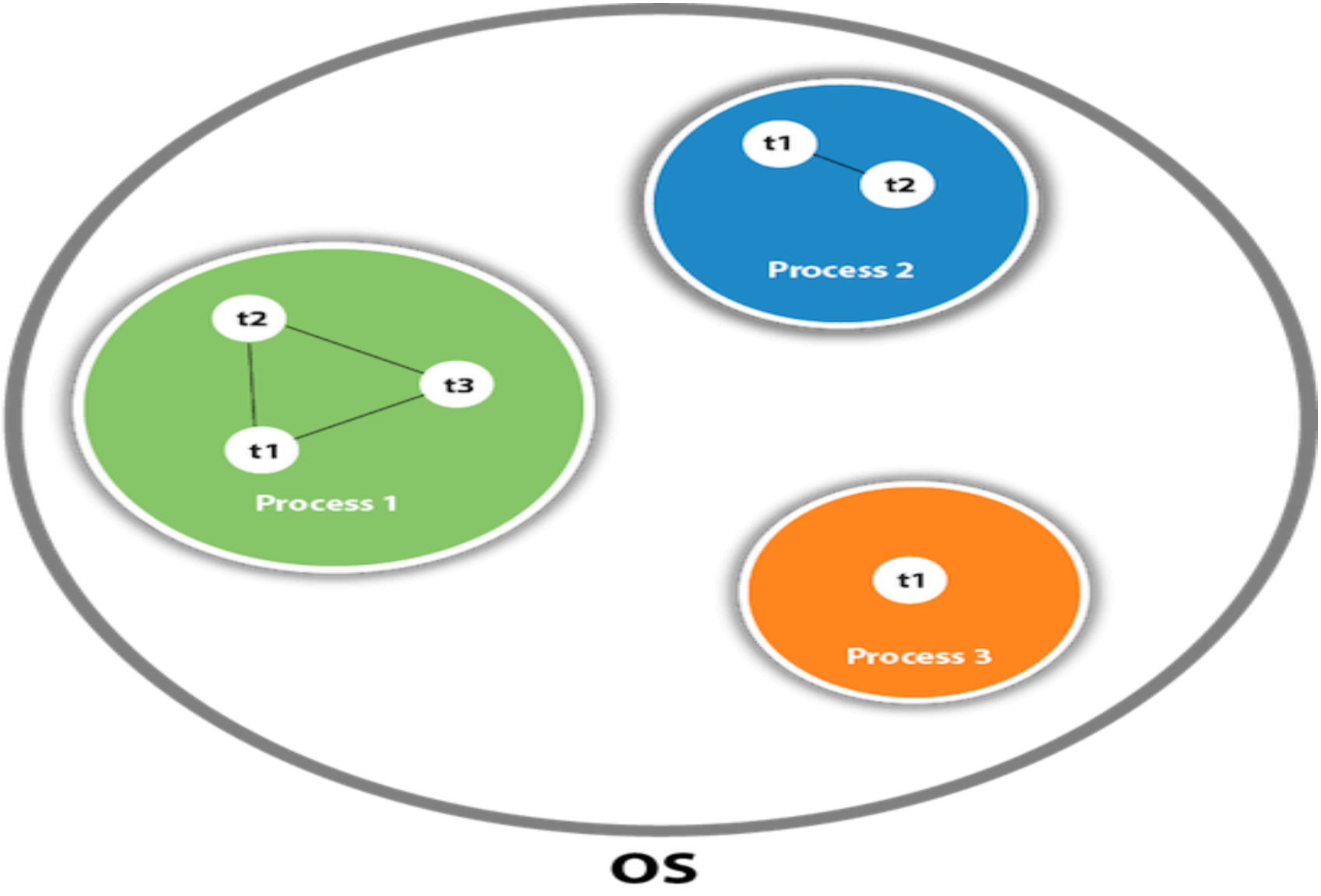
Multitasking and Multithreading

- Multitasking refers to a computer's ability to perform multiple jobs concurrently
  - more than one program are running concurrently, e.g., UNIX
- A thread is a single sequence of execution within a program
- Multithreading refers to multiple threads of control within a single program
  - each program can run multiple threads of control within it

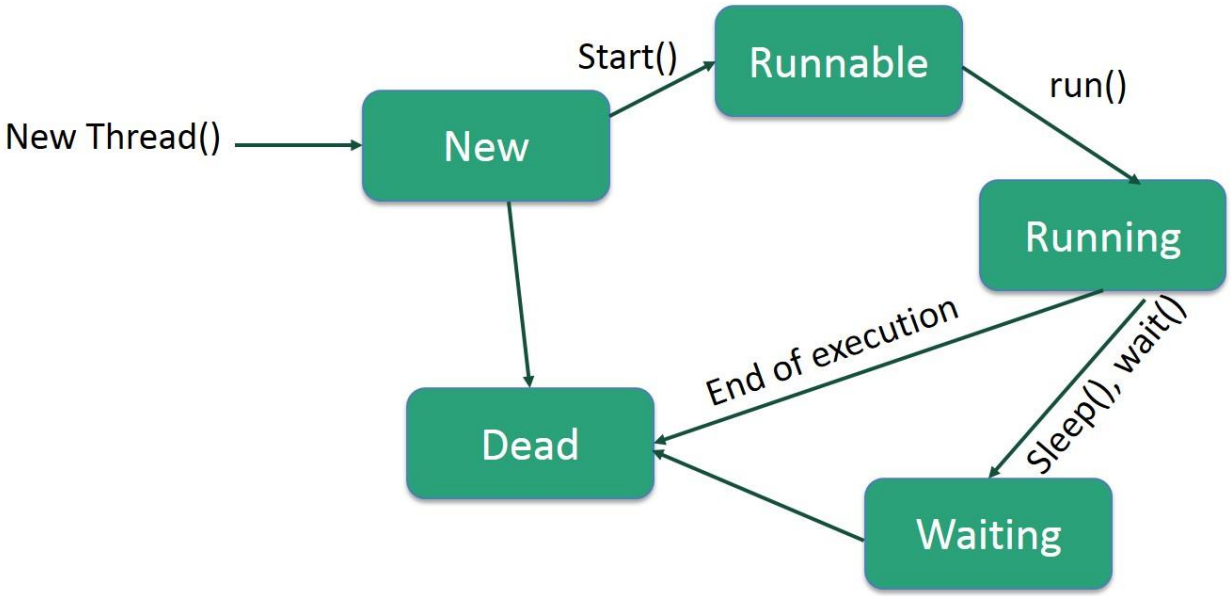
Multithreading in Java

- **Multithreading in Java** is a process of executing multiple threads simultaneously.
- A thread is a lightweight sub-process, the smallest unit of processing.
- Multiprocessing and multithreading, both are used to achieve multitasking.
- However, we use multithreading than multiprocessing because threads use a shared memory area.
- Java Multithreading is mostly used in games, animation, etc.

Cont..



Life Cycle of a thread



Java Thread class

- Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Java Thread Methods

S.N.	Modifier and Type	Method	Description
1)	void	start()	It is used to start the execution of the thread.
2)	void	run()	It is used to do an action for a thread.
3)	static void	sleep()	It sleeps a thread for the specified amount of time.
4)	static Thread	currentThread()	It returns a reference to the currently executing thread object.
5)	void	join()	It waits for a thread to die.
6)	int	getPriority()	It returns the priority of the thread.
7)	void	setPriority()	It changes the priority of the thread.
8)	String	getName()	It returns the name of the thread.
9)	void	setName()	It changes the name of the thread.
10)	long	getId()	It returns the id of the thread.

Cont..

Creating Threads

- There are two ways to create our own **Thread** object
  1. Subclassing the **Thread** class and instantiating a new object of that class
  2. Implementing the **Runnable** interface
- In both cases the **run()** method should be implemented

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.

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().

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

Starting a thread:

start() method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

Example

1) Java Thread Example by extending Thread class

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

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

Example

2) Java 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...
```

Sleep method in java

- The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
- **Syntax of sleep() method in java**

- The Thread class provides two methods for sleeping a thread:
- public static void sleep(long miliseconds)throws InterruptedException

- public static void sleep(long miliseconds, int nanos)throws InterruptedException

Example of sleep method in java

```
class TestSleepMethod1 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();

  t1.start();
  t2.start();
 }
}
```

Extending Thread Class

```
class NewThread extends Thread {
NewThread() {
      super("Demo Thread");
       System.out.println("Child thread: " + this);
      start();
   }
public void run() {
   try {
      for(int i = 5; i > 0; i--) {
      System.out.println("Child Thread: " + i);
        Thread.sleep(100);
      }
   } catch (InterruptedException e) {
System.out.println("Child interrupted.");
      }
System.out.println("Exiting child thread.");
}
}
```

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

NewThread t1=new NewThread(); // create a new
thread
   try {
      for(int i = 5; i > 0; i--) {
System.out.println("Main Thread: " + i);
Thread.sleep(200);
      }
   } catch (InterruptedException e) {
System.out.println("Main thread interrupted.");
}
System.out.println("Main thread exiting.");
}
}
```

OUTPUT

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

Priority of a Thread (Thread Priority):

- Each thread have **a priority**.
- Priorities are represented by a number between 1 and 10.

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Thread Priority Example

```
classthreadp implements Runnable {
   long count=0;
   private volatile boolean running=true;
   Thread  t;
   public threadp(int p) {
     t=new Thread(this,"Tno."+p);
     t.setPriority(p);
   }
   public void run()  {
            System.out.println("execution of thread  " + t.getName() );
            while(running)
                        count++;
            System.out.println("thread " + t.getName() +  "count " + count);
   }
   void startthread() throws InterruptedException  {
              System.out.println("thread priority is " + t.getPriority());
               t.start();
   }
```

```
void stopthread() {
      running=false;
    }
}
public class threadpriority {
   public static void main(String a[]) throws InterruptedException
{
                Thread.currentThread().setPriority(5);
                threadp t1,t2;
                t1=new threadp(Thread.MAX_PRIORITY);
                t2=new threadp(5);

                t2.startthread();
                t1.startthread();
                 Thread.sleep(100);
                  t1.stopthread();
                  t2.stopthread();

                  System.out.println("high priority: " + t1.count);
                  System.out.println("low priority: " + t2.count );
    }
}
```

OUTPUT

1)

```
thread priority is 10
thread priority is 5
execution of thread  Tno.10
execution of thread  Tno.5
high priority: 52273434
thread Tno.5count 51755147
thread Tno.10count 52273434
low priority: 51755147
the difference 518287
```

join() method

- The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

public void join()throws InterruptedException

public void join(long milliseconds)throws InterruptedException

Example

```
class threadjoin implements Runnable
{
   Thread t;
   public threadjoin() {
            t= new Thread(this);
    }
   public void run() {
            System.out.println("r1");
            try{
                        t.sleep(200);
              }
            catch(InterruptedException e) {
            System.out.println("thrread
interrupted");          }
            System.out.println("r2");
    }
}
```

```
public class Thread3 {
public static void main(String[] args) {
threadjoin t1=new threadjoin();
threadjoin t2=new threadjoin();
try{
        t1.t.start();
        t2.t.start();
        if(t1.t.isAlive())
            t1.t.join();
        if(t2.t.isAlive())
            t2.t.join();
}catch(InterruptedException e){
            System.out.println("interrupted");
}
System.out.println("t1-->"+  t1.t.isAlive());
System.out.println("t2-->"+  t2.t.isAlive());
    }
}
```

Do it Yourself

Create a thread which takes input an array and return the sum of array. Write a program to find sum of a 2D array using this thread class. Demonstrate the working of the threads in main().

```
class sumthread implements Runnable{
   int[] a;
   int total;
   Thread t;
   sumthread(int[] ob)
   {
      t=new Thread(this);
      a=ob;
   }
   @Override
   public void run()    {
         total=0;
         for(int i=0;i<a.length;i++)          {
            total+= a[i];
         }
         System.out.println(t.getName()+ " completed");
   }
}
public class threaddemo3 {
   public static void main(String ar[]) throws
InterruptedException {
      int[][] a={{1,2,3},{2,3,4},{4,5,6}};
      sumthread t1=new sumthread(a[0]);
      sumthread t2=new sumthread(a[1]);
      sumthread t3=new sumthread(a[2]);
      t1.t.start();
      t2.t.start();
      t3.t.start();
      if(t1.t.isAlive()){
         System.out.println("waiting for thread0 to join");
         t1.t.join();
         System.out.println("thread 0 joined");
      }
      if(t2.t.isAlive()){
         System.out.println("waiting for thread1 to join");
         t2.t.join();
         System.out.println("thread 1 joined");
      }
      if(t3.t.isAlive())       {
         System.out.println("waiting for thread2 to join");
         t3.t.join();
         System.out.println("thread 2 joined");
      }
      int sum=t1.total +t2.total+t3.total;
      System.out.println("sum of matrix is" + sum);
   }
}
```

Synchronize Keyword

Output
[Hello [World]]

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

class caller implements Runnable{
   String s;
callme ob;
   Thread t;
caller(callme ob,String s)   {
this.ob=ob;
this.s=s;
      t=new Thread(this);
      t.start();
   }
   @Override
       public void run()   {
       ob.call(s);
      }    }
public class threadsync {
   public static void main(String a[]){
          callme c=new callme();
          caller t1=new caller(c, "hello");
          caller t2=new caller(c, "world");
   }
}
```

Synchronize Keyword

Output
[Hello] [World]

```
class callme
{
  synchronize void call(String s) {
  try{
        System.out.print("["+s);
        Thread.sleep(1000);
        System.out.println("]");
}catch(InterruptedException e){
        System.out.println("interrupted");
   }   }  }

class caller implements Runnable{
   String s;
callme ob;
   Thread t;
caller(callme ob,String s)    {
this.ob=ob;
this.s=s;
      t=new Thread(this);
      t.start();
   }
   @Override
       public void run()    {
       ob.call(s);
      }     }
public class threadsync {
   public static void main(String a[]){
          callme c=new callme();
          caller t1=new caller(c, "hello");
          caller t2=new caller(c, "world");
   }
}
```

Synchronize Keyword

Output
[Hello] [World]

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

class caller implements Runnable{
    String s;
callme ob;
    Thread t;
caller(callme ob,String s)    {
this.ob=ob;
this.s=s;
        t=new Thread(this);
        t.start();
    }
    @Override
        public void run()    {
        Synchronize(ob)
        {
        ob.call(s);
        } }     }
public class threadsync {
    public static void main(String a[]){
            callme c=new callme();
            caller t1=new caller(c, "hello");
            caller t2=new caller(c, "world");
    }
}
```

Producer Consumer

```
import java.util.*;
public class producerconsumer1 {
   public static void main(String[] args) throws InterruptedException {

ArrayList<Integer> queue=new ArrayList<>();
      int size=5;
      consumer c=new consumer(queue, size);
      Thread pthread=new Thread(new producer(queue, size), "producer");
      Thread cthread=new Thread(c, "consumer");

      pthread.start();
      Thread.currentThread().sleep(10);
      cthread.start();
   }
}
```

```
class producer implements Runnable{
ArrayList<Integer> queue=new
ArrayList<>();
final int size;
   public producer(ArrayList<Integer>
queue, int size)
   {
       this.queue=queue;
       this.size=size;
   }
   @Override
   public void run() {
      for(int i=0;i<7;i++){
         if(queue.size()==size) {
            synchronized(queue) {
                  System.out.println("queue
                      is full producer is
                      waiting" );
                  try {
                     queue.wait();
                  }
catch (InterruptedException ex) {
   System.out.println("interrupted");
    }
     }
  }
  synchronized(queue)
  {
     System.out.println("produced "
                + i +       queue.add(i));
     queue.notifyAll();
  }
   }
   }
   }
```

```
class consumer implements Runnable{
ArrayList<Integer> queue;
final int size;
boolean flag=true;

   public consumer(ArrayList<Integer> queue, int
size) {
      this.queue=queue;
      this.size=size;
   }

   @Override
   public void run() {
    while(true){

      while(queue.isEmpty())
      {
         synchronized(queue){
         try {
            queue.wait();
            System.out.println("out of waiting”);
} catch (InterruptedException ex) {
System.out.println("interrupted");
         }
         }
      }
         synchronized(queue)
         {
            queue.notifyAll();
            int k=queue.remove(0);
            System.out.println("consumed " + k);
         }
      }}   }
```