**MULTITHREADING**

 MS WORD : Main Process

Typing : Sub Process

Spell Check : Sub Process

These Sub Process is called threads .

So a thread is a unit of a process . If you break process or differentiate it you will get threads.

Thread.sleep () is a method which is used to suspend our thread.

class Hi

{

    public void show()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hi");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

}

class Hello

{

    public void show()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hello");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

}

public class ThreadDemo {

    public static void main(String[] args) {

        Hi ob1 = new Hi();

        Hello ob2 = new Hello();

        ob1.show();

        ob2.show();

    }

}

o/p :

Hi

Hi

Hi

Hi

Hi

Hello

Hello

Hello

Hello

Hello

so this gives us this output with a gap of half a second . So to execute this process parallely i.e it should print Hi then Hello and so on . As the other method is kept waiting for 2.5s to execute completely . So to do that we will use .

class Hi extends Thread

{

    public void run()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hi");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

}

class Hello extends Thread

{

    public void run()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hello");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

}

public class ThreadDemo {

    public static void main(String[] args) {

        Hi ob1 = new Hi();

        Hello ob2 = new Hello();

        ob1.start();

        ob2.start();

    }

}

here we have made several changes we have used extends Thread in each class so that class inherits the properties of the Thread class . Also we have changed the method name to run as it is compulsory when we use start method in the main method it looks for the run method to execute . And instead of using the method name we are using start as we are using the concept of thread. and start method is available in the thread class.

o/p :

Hi

Hello

Hi

Hello

Hello

Hi

Hi

Hello

Hello

Hi

here we can observe that there is a clash b/w two threads as hello is printed twice because the scheduler selects which process takes less time but here we haven't mentioned the priority so the scheduler by default selects on its own so to avoid this what we can do is . We can add a delay of 10ms to avoid the clash b/w two threads .

class Hi implements Runnable

{

    public void run()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hi");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

}

class Hello implements Runnable

{

    public void run()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hello");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

}

public class ThreadDemo {

    public static void main(String[] args) {

        Hi ob1 = new Hi();

        Hello ob2 = new Hello();

we can also write here

Runnable ob1 = new Hi();

Runnable ob2 = new Hello();

        ob1.start();

   try

       { Thread.sleep(10);

       }

       catch(Exception e)

       {

       }

        ob2.start();

  Thread t1 = new Hi(ob1);

  Thread t2 = new Hello(ob2);

    }

} It will give an error as start method is not available in Runnable class . So we can use Thread object to use start method

Thread t1 = new Thread(); Thread t2 = new Thread(); but it will do nothing as we are not calling the run method of our class it's calling the run method of Thread class. So here we can pass the object to the thread of the Runnable class and it will do the work . It will print hi hello and so on.

So there are two ways to create a thread one by extending the interface or extending the class and one by implementing the interface which is runnable .

So in the above code we have used class Hi and class Hello only once so we ca use anonymous class instead and can reduce few lines of code.

class Hi implements Runnable

{

    public void run()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hi");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

}

class Hello implements Runnable

{

    public void run()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hello");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

}

public class ThreadDemo {

    public static void main(String[] args) {

        Hi ob1 = new Hi();

        Hello ob2 = new Hello();

we can also write here

Runnable ob1 = new Hi();

Runnable ob2 = new Hello();

        ob1.start();

   try

       { Thread.sleep(10);

       }

       catch(Exception e)

       {

       }

        ob2.start();

  Thread t1 = new Hi(ob1);

  Thread t2 = new Hello(ob2);

    }

}

So the above code can be modified and we can remove the two classes Hello and Hi as their only purpose is to implement the Runnable class . So we can use the concept of anonymous class concept here.

public class ThreadDemo {

    public static void main(String[] args) {

Runnable ob1 = new Runnable()

{

 public void run()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hi");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

};

Runnable ob2 = new Runnable()

{

 public void run()

    {

        for(int i = 1 ; i<= 5 ; i++)

        {

        System.out.println("Hello");

       try

       { Thread.sleep(500);

       }

       catch(Exception e)

       {

       }

    }

    }

};

Thread t1 = new Thread(ob1);

Thread t2 = new Thread(ob2);

        t1.start();

   try

       { Thread.sleep(1000);

       }

       catch(Exception e)

       {

       }

       t2.start();

    }

}

We can use lamda expression as well to reduce the number of lines of code and make it more efficient.

public class ThreadDemo {

    public static void main(String[] args) {

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

});

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

});

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

    }

}

join and isAlive method in java

public class ThreadDemo {

    public static void main(String[] args) {

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

});

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

});

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

Sopln("Bye");

    }

}

it's output is

Hi

Bye

Hello

Hi

Hello

Hello

Hi

Hi

Hello

Hello

Hi

This happened because main is also a thread so by the time t2 is going to execute the thread main thread is empty i.e. it has nothing to do so it will print Bye and after that t2 will execute.

So to avoid this and print Bye at the end we will use join which will make the main method to wait .

public class ThreadDemo {

    public static void main(String[] args) throws Exception

//as join may throw exception {

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

});

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

});

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

t1.join();

t2.join(); // as they may throw exception so we will use throw Exception

Sopln("Bye");

    }

}

isAlive is a method by which you can check your thread is in a running state or not. After join method isAlive method returns false as the thread is over after executing join method.

public class ThreadDemo {

    public static void main(String[] args) throws Exception

//as join may throw exception {

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

});

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

});

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

t1.join();

t2.join(); // as they may throw exception so we will use throw Exception

Sopln(t1.isAlive());

Sopln("Bye");

    }

}

o/p

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

false

Bye

public class ThreadDemo {

    public static void main(String[] args) throws Exception

//as join may throw exception

{

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

});

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

});

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

  System.out.println(t1.isAlive());

t1.join();

t2.join(); // as they may throw exception so we will use throw Exception

System.out.println(t1.isAlive());

System.out.println("Bye");

    }

}

o/p :

Hi

true

Hello

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

false

Bye

setName () and getName

setName method is used to assign name to a thread

ex : t1.setName("Hi");

getName method is used to provide the name of the thread

by default the the first thread is named as Thread-0 and second as Thread-1 and so on .

The setName method is also used when we are using multiple threads so it is easy to distinguish which thread is used for which purpose .

public class ThreadDemo {

    public static void main(String[] args) throws Exception

//as join may throw exception

{

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

});

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

});

System.out.println(t1.getName());

System.out.println(t2.getName());

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

t1.join();

t2.join(); // as they may throw exception so we will use throw Exception

System.out.println("Bye");

    }

}

o/p :

Thread-0

Thread-1

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

Bye

public class ThreadDemo {

    public static void main(String[] args) throws Exception

//as join may throw exception

{

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

});

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

});

t1.setName("Hi");

t2.setName("Hello");

System.out.println(t1.getName());

System.out.println(t2.getName());

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

t1.join();

t2.join(); // as they may throw exception so we will use throw Exception

System.out.println("Bye");

    }

}

o/p :

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

Bye

When you create an object of a thread it will ask for two parameters

which are object of runnable and its name so by this we can also assign name to a thread

public class ThreadDemo {

    public static void main(String[] args) throws Exception

//as join may throw exception

{

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

} //till here its object of runnable

,"Hi Thread");

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

} //till this brace its an object of runnable

,"Hello thread");

System.out.println(t1.getName());

System.out.println(t2.getName());

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

t1.join();

t2.join(); // as they may throw exception so we will use throw Exception

System.out.println("Bye");

    }

}

getPriority method gives the priority of the thread

public class ThreadDemo {

    public static void main(String[] args) throws Exception

//as join may throw exception

{

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

} //till here its object of runnable

,"Hi Thread");

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

} //till this brace its an object of runnable

,"Hello thread");

System.out.println(t1.getPriority());

System.out.println(t2.getPriority());

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

t1.join();

t2.join(); // as they may throw exception so we will use throw Exception

System.out.println("Bye");

    }

}

o/p:

5

5

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

Hi

Hello

Bye

The range of priority goes from one to ten ( 1 to 10) 1 means lowest and 10 means highest and 5 means normal priority. So sometimes its not easy to remember the numbers that which number represents which priority .

So we can use Thread.MIN\_PRIORITY for minimum priority and Thread.MAX\_PRIORITY for maximum priority.

public class ThreadDemo {

    public static void main(String[] args) throws Exception

//as join may throw exception

{

Thread t1 = new Thread( () ->

{

for(int i = 1 ; i<= 5 ; i++)

        {

       System.out.println("Hi");

       try

       { Thread.sleep(1000);} catch(Exception e){}

    }

} //till here its object of runnable

,"Hi Thread");

Thread t2 = new Thread( () ->

{

  for(int i = 1 ; i<= 5 ; i++)

        {

         System.out.println("Hello");

       try{ Thread.sleep(1000);} catch(Exception e) {}

    }

} //till this brace its an object of runnable

,"Hello thread");

t1.setPriority(Thread.MIN\_PRIORITY);

t2.setPriority(Thread.MAX\_PRIORITY);

System.out.println(t1.getPriority());

System.out.println(t2.getPriority());

    t1.start();

   try

       { Thread.sleep(10);} catch(Exception e){}

        t2.start();

t1.join();

t2.join(); // as they may throw exception so we will use throw Exception

System.out.println("Bye");

    }

}

o/p:

1

10

Hi

Hello

Hi

Hello

Hello

Hi

Hello

Hi

Hello

Hi

Bye

Synchronized Keyword

class Counter

{

 int count ;

 public void increment()

 {

  count ++;

 }

}

public class SyncDemo

{

 public static void main (String args[] ) throws Exception

 {

  Counter c = new Counter();

  Thread t1 = new Thread(new Runnable()

  {

   public void run()

   {

    for ( int i = 1 ; i <= 1000 ; i++)

    {

     c.increment();

    }

   }

  });

  Thread t2 = new Thread(new Runnable()

  {

   public void run()

   {

    for ( int i = 1 ; i <= 1000 ; i++)

    {

     c.increment();

    }

   }

  });

  t1.start();

  t2.start();

  t1.join();

  t2.join();

  System.out.println("Count = " + c.count);

 }

}

It will generate random o/p such as 2000 , 1658 , 1514 , 1234 . This is happening because t1 and t2 thread are accessing the increment method concurrently i.e. accessing it at the same time. So here what we can do is . If t1 is executing the count method then t2 should not execute it and vice versa. And this can be done by using synchronized keyword in the increment method.

class Counter

{

 int count ;

 public synchronized void increment()

 {

  count ++;

 }

}

public class SyncDemo

{

 public static void main (String args[] ) throws Exception

 {

  Counter c = new Counter();

  Thread t1 = new Thread(new Runnable()

  {

   public void run()

   {

    for ( int i = 1 ; i <= 1000 ; i++)

    {

     c.increment();

    }

   }

  });

  Thread t2 = new Thread(new Runnable()

  {

   public void run()

   {

    for ( int i = 1 ; i <= 1000 ; i++)

    {

     c.increment();

    }

   }

  });

  t1.start();

  t2.start();

  t1.join();

  t2.join();

  System.out.println("Count = " + c.count);

 }

}

so in this case it will not give any random value it will only print the desired result i.e. 2000.

So when you don't synchronize your method then your method or your object counter is not thread safe your class is not thread safe . Which means that multiple thread can access the same method at the same time.

So when we make a method synchronized only one thread can access the method at one time. So when t1 is executing increment method t2 will not interfere and vice versa.