linkedin id :: https://www.linkedin.com/in/nitin-manjunath-110169136/

MultiThreading
  => To utitlize the CPU Time effectively.

How to create Threads in Java?
 a. Runnable(I)
 b. Thread(c)

When we have 2 ways to create a thread, which approach is good in programming?
Ans. Using an interface called "Runnable".

Note:
1.   when we have multiple threads, which threads will get a chance for execution?
Ans. It will be decided based on the priority.
     If multiple threads have the same priority then ThreadScheduler will decide which thread to execute.

2. When we create a thread(user-defined), what is the default name given to the thread by JVM?
Ans.Thread-0

3. What is the priority given for the userdefined thread as default by JVM?
Ans. Same as that of the Parent thread(main thread) which is "5".

4. What is the priority of a Thread?
Ans. priority range is "1-10". 1 -> low priority , 5-> average priority, 10-> High priority.

We can prevent Threads from Execution
 a. yield()
 b. sleep()
 c. join()

+++++++
yield()
+++++++
1. It causes to pause current executing Thread for giving chance for waiting Threads ofsame priority.
2. If there is no waiting Threads or all waiting Threads have low priority then same Thread can continue its execution.
3. If all the threads have same priority and if they are waiting then which thread will get chance we can't expect, it depends on
    ThreadScheduler.
4. The Thread which is yielded, when it will get the chance once again depends on the mercy on "ThreadScheduler" and
    we can't expect exactly.
5. public static native void yield()

state diagram of yield()
++++++++++++++++++++++++

```
MyThread t= new MyThread() //new state or born state
    t.start() // enter into ready state/runnable state
 if ThreadScheduler allocates processor then enters into running state.
    a. if running Thread calls yield() then it enters into runnable state.
```

 if run() is finished with execution then it enters into dead state.

eg#1.

```
class MyThread extends Thread{

      @Override
      public void run(){
            for (int i=1;i<=5 ;i++ ){
                  System.out.println("child thread");
                  Thread.yield();//line-1
            }
      }

}
public class TestApp{
      public static void main(String... args){
            MyThread t= new MyThread();
            t.start();
            for (int i=1;i<=5 ;i++ ){
                  System.out.println("Parent Thread");
            }
      }
}
```

Note:: If we comment line-1, then we can't expect the output becoz both the threads have same priority then which thread the ThreadScheduler will schedule is not in the hands of programmer but if we don't comment line-1, then there is a possibility of main thread getting more no of times, so main thread execution is faster then child thread will get chance.

Note: Some platforms wont provide proper support for yield(),becuase it is getting the execution code from other
      language prefereably from 'C'.

b. join()
       If the thread has to wait untill the other thread finishes its execution then we need to go for join().
       if t1 executes t2.join() then t1 should should wait till t2 finishes its execution.
       t1 will be entered into waiting state untill t2 completes, once t2 completes then t1 can continue with its execution.

eg#1.

```
venue fixing               =====> t1.start()
wedding card printing      =====> t2.start()=====> t1.join()
wedding card distrubution =====> t3.start()=====> t2.join()
```

Prototype of join()
===================

```
public final void join() throws InterruptedException
public final void join(long ms)throws InterruptedException
public final void join(long ms,int ns)throws InterruptedException
```

Note: While one thread is in waiting state and if one more thread interupts then it would result
      in "InteruptedException".InteruptedException is checkedException which should always be
      handled.

```
Thread t =new Thread();//new/born state
```

```
  t.start();//ready/runnable state
-> If T.S allocates cpu time then Thread enters into running state
-> If currently executing Thread invokes t.join()/t.join(1000),t.join(1000,100),
then it
   would enter into waiting state.
-> If the thread finishes the execution/time expires/interupted then it would come
back to
   ready state/runnable state.
-> If run() is completed then it would enter into dead state.

eg#1.
class MyThread extends Thread{
      @Override
      public void run(){
            for (int i=1;i<=10 ;i++ ){
                  System.out.println("Sita Thread");
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){
                  }
            }
      }
}
public class Test3 {
      public static void main(String... args)throws InterruptedException{
            MyThread t=new MyThread();
            t.start();
            t.join(10000);//line-n1
            for (int i=1;i<=10;i++ ){
                  System.out.println("rama thread");
            }
      }
}

=> If line-n1 is commented then we can't predict the output becoz it is the duty of
the T.S to       assign C.P.U time
=> If line-n1 is not commented, then rama thread(main thread) will enter into
waiting state till
   sita thread(child thread) finishes its execution.
Output
 2 Threads
 a. Child Thread
      sita thread
      sita thread
       ....
 b. Main Thread
      rama thread
      rama thread
      ....

Waiting of Child Thread untill Completing Main Thread
=====================================================
we can make main thread to wait for child thread as well as we can make child
thread also to wait for main thread.

eg#1.
class MyThread extends Thread{
      static Thread mt;
      @Override
      public void run(){
                  try{
                        mt.join();
                  }
                  catch (InterruptedException e){
                  }

                  for (int i=1;i<=10 ;i++ ){
                        System.out.println("child thread");
                  }

      }
}
public class Test3 {
      public static void main(String... args)throws InterruptedException{
            MyThread.mt=Thread.currentThread();

            MyThread t=new MyThread();
            t.start();

            for (int i=1;i<=10;i++ ){
                  System.out.println("main thread");
                  Thread.sleep(2000);//20sec sleep
            }

      }
}
Output
 2 Threads(MainThread,ChildThread)
MainThread
 a. main thread
     ....
     ....
ChildThread
 a. child thread
     ....
     ....


eg#2.
class MyThread extends Thread{
      static Thread mt;

      @Override
      public void run(){
                  try{
                        mt.join();
                  }
                  catch (InterruptedException e){
                  }

                  for (int i=1;i<=10 ;i++ ){
                        System.out.println("child thread");
                  }

      }
}
```

```
public class Test3 {
      public static void main(String... args)throws InterruptedException{
            MyThread.mt=Thread.currentThread();

            MyThread t=new MyThread();
            t.start();
            t.join();

            for (int i=1;i<=10;i++ ){
                  System.out.println("main thread");
                  Thread.sleep(2000);//20sec sleep
            }

      }
}
```

Note::
If both the threads invoke t.join(),mt.join() then the program would result in "deadlock".

eg#3.

```
public class Test3 {
      public static void main(String... args)throws InterruptedException{
            Thread.currentThread().join();
      }
}
```

Output:: Deadlock, becoz main thread is waiting for the main thread itself.

sleep()
======
 If a thread dont' want to perform any operation for a particular amount of time then we
 should go for sleep().

Signature

```
  public static native void sleep(long ms) throws InterruptedException
  public static void sleep(long ms,int ns) throws InterruptedException
```

every sleep method throws InterruptedException, which is a checkedexception so we should compulsorily handle the exception using
try catch or by throws keyword otherwise it would result in compile time error.

```
Thread t=new Thread(); //new or born state
  t.start() // ready/runnable state
```

=> If T.S allocates cpu time then it would enter into running state.
=> If run() completes then it would enter into dead state.
=> If running thread invokes sleep(1000)/sleep(1000,100) then it would enter into Sleeping state
=> If time expires/ if sleeping thread got interrupted then thread would come back to
    "ready/runnable state".

eg#1.

```
public class SlideRotator {
      public static void main(String... args)throws InterruptedException{
            for (int i=1;i<=10 ;i++ ){
                  System.out.println("Slide: "+i);
                  Thread.sleep(5000);
            }
      }
}
Output::
Slide:: 1
Slide:: 2
Slide:: 3
Slide:: 4
Slide:: 5
Slide:: 6
Slide:: 7
Slide:: 8
Slide:: 9
Slide:: 10

Interupting a Thread
====================
 public void interrupt()
      => If thread is in sleeping state or in waiting state we can interupt a
thread.
eg#1.
class MyThread extends Thread{
      @Override
      public void run(){
            try{
                  for (int i=1;i<=10;i++ ){
                        System.out.println("I am lazy thread");
                        Thread.sleep(2000);
                  }
            }
            catch (InterruptedException e){
                  System.out.println("I got interrupted");
            }
      }
}
public class Test3 {
      public static void main(String... args)throws InterruptedException{

            MyThread t=new MyThread();
            t.start();

            t.interrupt();//line-n1
            System.out.println("End of Main...");

      }
}

Scenario:: If a comment line-n1
 2 thread
a. Main Thread
      End of Main...
b. Child Thread
     I am lazy thread
       .....
       .....

Scneario:: If t.interrupt() then
```

```
 2 thread
a. Main Thread
      main thread
b. Child Thread
     I am lazy thread
     I got interrupted


eg#2.
class MyThread extends Thread{

      @Override
      public void run(){
            for (int i=1;i<=10000 ;i++ ){
                  System.out.println("I am lazy thread : "+i);
            }
            System.out.println("I am entering into sleeping state");
            try
            {
                  Thread.sleep(2000);
            }
            catch (InterruptedException ie){
                  ie.printStackTrace();
            }
      }
}
public class TestApp {
      public static void main(String[] args)throws InterruptedException {

            MyThread t=new MyThread();
            t.start();

            t.interrupt();//line-n1
            System.out.println("main thread");

      }
}

line-n1 is commented then no problem
line-n1 is not commented, then interrupt() will wait till the Thread enters into
waiting state/sleeping state.


Note::
 If thread is interrupting another thread, but target thread is not in waiting
state/sleeping  state then there would be no exception.
 interrupt() call be waiting till the target thread enters into waiting
state/sleeping state so this call wont be wasted.
 once the target thread enters into waiting state/sleeping state then interrupt()
will interrupt and it causes the exception.
 interrupt() call will be wasted only if the Thread does not enters into waiting
state/sleeping  state.


yield() join() sleep()
================================
1) Purpose
      yield()
            To pause current executing Thread for giving the chance of remaining
```

waiting Threads of same priority.

join()

If a Thread wants to wait until completing some other Thread then we should go for join.

sleep()

If a Thread don't want to perform any operation for a particular amount of time then we should go for sleep() method.

2) Is it static

yield() yes
join() no
sleep() yes

3) Is it final?

yield() no
join() yes
sleep() no

4) Is it overloaded?

yield() no
join() yes
sleep() yes

5) Is it throws IE?

yield() no
join() yes
sleep() yes

6) Is it native method? //logic of those methods would come from language like 'c' not from java.

yield() yes
join() no
sleep()
sleep(long ms) -->native
sleep(long ms,int ns) -->non-native