ThreadPriorities
==========
 For every Thread in java has some priority.
 valid range of priority is 1 to 10,it is not 0 to 10.
 if we try to give a differnt value the it would result in
"IllegalArgumentException".
 Thread.MIN_PRIORITY = 1
 Thread.MAX_PRIORITY = 10
 Thread.NORM_PRIORITY = 5
 Thread class does not have priorities is Thread.LOW_PRIORITY,Thread.HIGH_PRIORITY.
 Thread scheduler allocates cpu time based on "Priority".
 If both the threads have the same priority then which thread will get a chance as
a pgm we can't
 predict becoz it is vendor dependent.
 We can set and get priority values of the thread using the following methods
   a. public final void setPriority(int priorityNumber)
   b. public final int getPriority()
 The allowed priorityNumber is from 1 to 10,if we try to give other values it woudl
result in
 "IllegalArgumentException".

System.out.println(Thread.currentThread().setPriority(100);//IllegalArgumentExcepti
on.

DefaultPriority
==========
The default priority for only main thread is "5",where as for other threads
priority will be
inherited from parent to child.
Parent Thread priority will be given as Child Thread Priority.

eg#1.
```
class MyThread extends Thread{}
public class TestApp{
      public static void main(String... args){
            System.out.println(Thread.currentThread().getPriority());//5
            Thread.currentThread().setPriority(7);
            MyThread t= new MyThread();
            System.out.println(Thread.currentThread().getPriority());//7
      }
}
```

reference
=========
```
Thread
  ^
  |extends
  |
MyThread
```

MyThread is creating by "mainThread", so priority of "mainThread" will be shared
as a priority for "MyThread".

eg#2.
```
class MyThread extends Thread{
      @Override
      public void run(){
            for (int i=1;i<=5 ;i++ ){
                  System.out.println("child thread");
```

```
            }
      }
}
public class TestApp{
      public static void main(String... args){

            MyThread t= new MyThread();
            t.setPriority(7);//line -1
            t.start();
            for (int i=1; i<=5; i++){
                  System.out.println("main thread");
            }

      }
}
```
Since priority of child thread is more than main thread, jvm will execute child thread first
whereas for the parent thread priority is 5 so it will get last chance.
if we comment line-1, then we can't predict the order of execution becoz both the threads have same priority.
  Some platform won't provide proper support for Thread priorities.
   eg:: windows7,windows10,...

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

yield() => It causes to pause current executing Thread for giving chance for waiting Threads of
                same priority.
            If there is no waiting Threads or all waiting Threads have low priority then
            same Thread can continue its execution.
            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.
            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.
            public static native void 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 than
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
            }

      }
}

output:
2 threads(Main,child thread)
main thread
      ;;;;
      ;;;;
childthread
```

We will get no output, as resulted in deadlock.

```
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
```