ThreadPriorities

- For every Thread in java has some priority.
- The valid range of priority is 1 to 10,it is not 0 to 10.
- if we try to give a different value then it would result in "IllegalArgumentException".
- Thread.MIN_PRIORITY = 1
- Thread.MAX_PRIORITY = 10
- Thread.NORM_PRIORITY = 5
- If both the threads have the same priority then which thread will get a chance as a program 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 would result in **"IllegalArgumentException".**
System.out.println(Thread.currentThread().
setPriority(100);//IllegalArgumentException.

DefaultPriority

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

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

yield()

- It causes the current executing Thread to give a chance for waiting Threads of the same priority.
- If there is no waiting Threads or all waiting Threads have low priority then

the same Thread can continue its execution.

- If all the threads have the same priority and if they are waiting then which thread will get a 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 of "ThreadScheduler" and we can't expect exactly.

public static native void yield()

join()

If the thread has to wait until the other thread finishes its execution then we need to go for join().
if t1 executes t2.join() then t1 should wait till t2 finishes its execution.
t1 will be entered into waiting state until 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 distribution ====> t3.start()====> t2.join()

Waiting of Child Thread until Completing Main Thread

We can make the main thread wait for the child thread as well as we can make the child thread also wait for the main thread.

sleep()

If a thread doesn't want to perform any operation for a particular amount of time then we should go to 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 checked exception so we should compulsorily handle the exception using
try catch or by throws keyword otherwise it would result in a 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 a 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".

synchronization

1. synchronized is a keyword applicable only for methods and blocks
2. if we declare a method/block as synchronized then at a time only one thread can execute that method/ block on that object.
3. The main advantage of synchronized keywords is we can resolve data inconsistency problems.
4. But the main disadvantage of synchronized keyword is it increases waiting time of the Thread and effects performance of the system.
5. Hence if there is no specific requirement then never recommended to use synchronized keyword.
6. Internally synchronization concept is implemented by using lock concept.
7. Every object in java has a unique lock. Whenever we are using synchronized keyword then only the lock concept will come into the picture.
8. If a Thread wants to execute any synchronized method on the given object 1st it has to get the lock of that object. Once a Thread gets the lock of that object then it's allowed to execute any synchronized method on that object. If the synchronized method execution completes then automatically Thread releases lock.
9. While a Thread executing any synchronized method the remaining Threads are not allowed to execute any synchronized method on that object simultaneously. But remaining Threads are allowed to execute any non-synchronized method simultaneously. [lock concept is implemented based on object but not based on method].

Note: Every object will have 2 area[Synchronized area and NonSynchronized area]
Synchronized Area => write the code only to perform update,insert,delete
NonSynchronized Area => write the code only to perform select operation