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
       .....
       .....
```

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

Note::using lambda expression

```
Runnable r = ()-> {
                                         for (int i = 1;i<=5 ; i++)
                                         {
                                               System.out.println("child thread");
                                         }
                                     };
Thread t = new Thread(r);
t.start();
```

using annonmyous inner class
=========================

```
new Thread(new Runnable(){
                  @Override
                  public void run(){
                              for (int i = 1;i<=5 ;i++ )
                              {
                                          System.out.println("child thread");
                              }
                        }
                  }
      ).start();
```

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 keyword 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.

```
class X{
   synchronized void m1(){}
   synchronized void m2(){}
   void m3(){}
}
```

KeyPoints
=========

1. if t1 thread invokes m1() then on the Object X lock will applied.
2. if t2 thread invokes m2() then m2() can't be called because lock of X object is with m1.
3. if t3 thread invokes m3() then execution will happen becoz m3() is non-synchronized.

Lock concept is applied at the Object level not at the method level.

7. Every object in java has a unique lock. Whenever we are using synchronized keyword then only 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 got the lock of that object then it's allow 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 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

```
class ReservationApp{
      checkAvailablity(){
            //perform read operation
        }
      synchronized bookTicket(){
            //peform update operation
      }
}
```

eg#1.

```
class Display{
      public void wish(String name){
            for (int i=1;i<=10 ;i++ )
            {
                  System.out.print("Good Morning: ");
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){

                  }
                  System.out.println(name);
            }
      }
}

class MyThread extends Thread{

      Display d;
      String name;

      MyThread(Display d,String name){
            this.d=d;
            this.name=name;
      }

      @Override
      public void run(){
            d.wish(name);
      }
}
public class Test3 {
      public static void main(String... args){
            Display d=new Display();
            MyThread t1= new MyThread(d,"dhoni");
            MyThread t2= new MyThread(d,"yuvi");
            t1.start();
            t2.start();
      }
}
```

Ouput:: As noticed below the output is irregular becoz at a time on a resource called wish()
2 threads are acting simulataneously.

3 Threads
 a. Main Thread
 b. Child Thread-1
 c. Child Thread-2

```
GoodMorning :GoodMorning : ..
....
....
....
....
....
```

eg#2.

```
class Display{
      public synchronized void wish(String name){
            for (int i=1;i<=10 ;i++ )
            {
                  System.out.print("Good Morning: ");
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){

                  }
                  System.out.println(name);
            }
      }
}

class MyThread extends Thread{

      Display d;
      String name;

      MyThread(Display d,String name){
            this.d=d;
            this.name=name;
      }

      @Override
      public void run(){
                  d.wish(name);
      }
}
public class Test3 {
      public static void main(String... args)throws InterruptedException{
            Display d=new Display();
            MyThread t1= new MyThread(d,"dhoni");
            MyThread t2= new MyThread(d,"yuvi");
            t1.start();
            t2.start();
      }
}
Ouput::
3 Threads
 a. Main Thread
 b. Child Thread-1
      GoodMorning:dhoni
      GoodMorning:dhoni
      .....
      .....
      .....
 c. Child Thread-2
      GoodMorning:yuvi
      GoodMorning:yuvi
      ....
      ....
      ....
```

Note::

As noticed above there are 2 threads which are trying to operate on single object called "Display" we need synchronization to resolve the problem of "Datainconsistency".

casestudy::

```
 Display d1=new Display();
 Display d2=new Display();
 MyThread t1=new MyThread(d1,"yuvraj");
 MyThread t2=new MyThread(d2,"dhoni");
  t1.start();
  t2.start();
```

In the above case we get irregular output, because two different object and since

the method
is synchronized lock is applied w.r.t object and both the threads will start simulataneously on different java objects
due to which the output is "irregular".

Conclusion :
If multiple threads are operating on multiple objects then there is no impact of Syncronization.
If multiple threads are operating on same java objects then syncronized concept is required(applicable).

classlevel lock
===============

1. Every class in java has a unique level lock.
2. If a thread wants to execute static synchronized method then the thread requires "class level lock".
3. While a Thread executing any static synchronized method the remaining Threads are not allow to execute any static synchronized method of that class simultaneously.
4. But remaining Threads are allowed to execute normal synchronized methods, normal static methods, and normal instance methods simultaneously.
5. Class level lock and object lock both are different and there is no relationship between these two.

eg::

```
class X{
      static synchronized m1(){}//class level lock
        static synchronized m2(){}
             static m3(){}//no lock required
             synchronized m4(){}//object level lock
            m5(){}//no lock required
}
 t1=> m1() => class level lock applied and chance is given
 t2=> m2() => enter into waiting state
 t3=> m3() => gets a chance for execution without any lock
 t4=> m4() => object level lock applied and chance is given
 t5=> m5() => gets a chance for execution without any lock
```

eg#1.

```
class Display{
      public synchronized void displayNumbers(){
            for (int i=1;i<=10 ;i++ )
            {
                  System.out.print(i);
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){

                  }

            }
      }
      public synchronized void displayCharacters(){
            for (int i=65;i<=75 ;i++ )
            {
                  System.out.print((char)i);
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){

                  }

            }
      }
}

class MyThread1 extends Thread{

      Display d;
      MyThread1(Display d){
            this.d=d;
      }
      @Override
      public void run(){
            d.displayNumbers();
      }
}
class MyThread2 extends Thread{
      Display d;
      MyThread2(Display d){
            this.d=d;
      }
      @Override
      public void run(){
            d.displayCharacters();
      }
}
public class Test3 {
      public static void main(String... args){
            Display d1=new Display();
            MyThread1 t1= new MyThread1(d1);
            MyThread2 t2= new MyThread2(d1);
            t1.start();
            t2.start();
      }
}
```

```
Output::
3 Threads
  a.MainThread
  b.userdefinedThread
      displayCharacters()
  c.userdefinedThread
      displayNumbers()
```

Synchronized block
================

```
 synchronized void m1(){
      ...
      ...
      ...
      ...
      ...
      ======
      ======
      ======
      ======

      ...
      ...
      ...
      ...
      ...


}
```

if few lines of code is required to get synchronized then it is not recomeneded to make method only as synchronized.
If we do this then for threads performance will be low, to resolve this problem we use "synchronized block",
due to synchronized block performance will be improved.

Case Study
==========

If a thread got a lock of current object, then it is allowed to execute that block
a.

```
synchronized(this){
      .....
      .....
      .....
}
```

To get a lock of particular object:: B
b.

```
synchronized(B){
      .....
      .....
      .....
}
```

If a thread got a lock of particular object B, then it is allowed to execute that block.

c. To get class level lock we have to declare synchronized block as follow

```
 synchronized(Display.class){
      ....
      ....
      ....
 }
```

If a thread gets class level lock, then it is allowed to execute that block