MultiThreading
=>Utilization of CPU time(efficency)

How can we create a thread in java?
Ans. Runnable(I) : void run()
Thread(C)

Which approach is best approach among the 2 mentioned above for creating a Thread?
Ans. using interface approach.

```
Syntax: MyRunnable r = new MyRunnable();
      Thread t = new Thread(r);
           t.start();
```

Which method is responsible for creating,register and performing low level activies for a thread?
Ans. start() -> we say start() as "Heart of MultiThreading".

What is the default name for a thread given by JVM?
Ans. Thread-0.

Who creates main thread and who starts the main thread?
Ans. JVM

When we have multiple threads of the same priority, which thread will start with the execution?
Ans. Depends on the ThreadScheduler.

Can we set the priority and name for the threads?
Ans. Yes using "setPriority(int priority) and int getPriority()".
To get the name of the thread we use "setName(String name), String getName()".

What is the priority range associated with the Thread?
```
Ans. range : 1 to 10
     MIN_RANGE   : 1
     NORMAL_RANGE: 5
     MAX_RANGE   : 10
```

Note:By default what ever property is associated with Parent thread will be shared for Child thread also.

What are the different methods available to prevent the execution of the Thread?
Ans. yield() :: pause the execution, check whether there are any methods of same priority avaialble or not.
join() :: It waits for the other thread to complete the execution.
sleep() :: It wont perform any operation for a particular period of time

Note: join() and sleep() would generate "InterruptedException" which is a fully checkedexception so we need to keep handling code.

What is the usage of interrupt()?
Ans. This method call can be used to interrupt any thread which is waiting state or in sleeping 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 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
```

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

      public static synchronized void displayCharacters(){
            for (int i=65;i<=75 ;i++ )
            {
                  System.out.print((char)i);//A-K
                  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 Test {
      //JVM ---> main thread
      public static void main(String[] args)throws Exception{
            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

```
eg#1.
class Display{
      public void wish(String name){
            ;;;;;;;;;;;;; //l-lakh lines of code

      synchronized(this){
            for (int i=1;i<=10;i++ )
            {
                  System.out.print("Good morning:");
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){}
                  System.out.println(name);
            }
      }
          ;;;;;;;;;;;;;//1-lakh lines of code
      }
}
class MyThread extends Thread{
      Display d;
      String name;

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

      public void run(){
            d.wish(name);
      }
}

class Test {
      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();
      }
}

eg#2.
class Display{
      public void wish(String name){
            ;;;;;;;;;;;;; //l-lakh lines of code

      synchronized(this){
            for (int i=1;i<=10;i++ )
            {
                  System.out.print("Good morning:");
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){}
                  System.out.println(name);
            }
      }
```

```
            ;;;;;;;;;;;;;//1-lakh lines of code
      }
}
class MyThread extends Thread{
      Display d;
      String name;

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

      public void run(){
            d.wish(name);
      }
}
public class Test {
      public static void main(String[] args) {
            Display d1=new Display();
            Display d2=new Display();
            MyThread t1=new MyThread(d1,"dhoni");
            MyThread t2=new MyThread(d2,"yuvi");
            t1.start();
            t2.start();
      }
}
```

Output::Irregular output becoz two object and two threads acting on two different objects

eg#3.

```
class Display{
      public void wish(String name){
            ;;;;;;;;;;;;; //l-lakh lines of code

      synchronized(Display.class){
            for (int i=1;i<=10;i++ )
            {
                  System.out.print("Good morning:");
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){}
                  System.out.println(name);
            }
      }
          ;;;;;;;;;;;;;//1-lakh lines of code
      }
}
class MyThread extends Thread{
      Display d;
      String name;

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

      public void run(){
            d.wish(name);
      }
}
public class Test {
      public static void main(String[] args) {
            Display d1=new Display();
            Display d2=new Display();
            MyThread t1=new MyThread(d1,"dhoni");
            MyThread t2=new MyThread(d2,"yuvi");
            t1.start();
            t2.start();
      }
}
```

Note:: 2 object, 2 thread, but the thread which gets a chance applied class level lock so output
       is regular.

Note:: lock concept applicable only for objects and class types, but not for primitive types. if we try to do it would result in compile time error saying "unexpected type".

```
eg:: int x=10;
     synchronized(x){//CE: unexpected type found:int required:reference
     }
```