MultiThreading and ConCurrency
++++++++++++++++++++++++++++

Usage of MultiThreading
  a. We don't focus on the output because each threads are executed independetly
     we focus of utilisation of CPU time.

How do we create a Thread in java?
Ans. Using
      1. Thread(C)
      2. Runnable(I)

Which approach is best and why?
Ans. Using Runnable(I),because we can get the facility of inheritance.

When we have multiple threads who would decide which thread would execute?
Ans. ThreadScheduler(It is a program which is a part of JVM)

Which method is responsible for the following things
      a. Register thread to ThreadScheduler
      b. Perforiming low level activities
      c. invoke run()
Ans. start() , since it does many things we say it as "Heart of MultiThreading".

How to create a Thread in java using both the approaches?
Ans. MyRunnable r = new MyRunnable();
      Thread t = new Thread(r);
         t.start();

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

How to prevent the thread execution in java?
Ans. yield() :: pause the exeuction, give chance for other thread of the same
priority.
     join() :: wait till the execution of the other thread.
     sleep() :: thread won't do any operation for a particular period of time.

What is the usage of interrupt() and will the call be wasted it any case?
Ans. To interrupt a particular thread which is in waiting state or sleeping state.
     Upon interruption the thread would generate an Exception called
"InterruptedException".
     interrupt() would wait for the other thread to enter into sleeping state or
waiting state, if the thread
     doens't enter into waiting state or sleeping state then the interrupt() call
will be wasted.

How will you achieve interthread communication in java?
Ans. To threads can interact with eachother using the methods like wait(),notify()
and notifyAll().

What is synchronization in java and synchronization access modifiers can be applied
on what java blocks?
Ans. Synchonization refers to process of applying lock on an object by a thread
such that only one thread can operate
     on a object to avoid "DataInconsistency".
            eg: StringBuffer object.

synchronization can be applied on methods and blocks.

How many types of locks exists in java and using what locks what methods can be exeucted?
Ans. Object level lock => using this lock we can call synchronized instance methods.
Class level lock => using this lock we can call static synchronized methods.

Among synchronized methods and synchronized block, which one would be more efficient?
Ans. synchronized blocks.

Question based on lock
======================
1. If a thread calls wait() immediately it will enter into waiting state without releasing any lock.
Answer : false(it should be released immediately)

2. If a thread calls wait() it releases the lock of that object but may not immediately
Answer: false(it should be released immediately)

3. If a thread calls wait() on any object,it releases all locks acquired by that thread and enters into waiting state
Answer: false(it would release the lock of that particular object)

4. If a thread calls wait() on any object,it immediately releases the lock of that particular object and entered into waiting state
Answer: true

5. If a thread calls notify() on any object,it immediately releases the lock of that particular object
Answer: false(it may or may not release the lock).

6. If a thread calls notify() on any object,it releases the lock of that object but may not immediately.
Answer: true

```
class A
{
	public synchronized void foo(B b)
	{
		System.out.println("Thread1 starts execution of foo() method");
		try
		{
			Thread.sleep(2000);
		}
		catch (InterruptedException ie)
		{
		}
		System.out.println("Thread1 trying to call b.last()");
		b.last();
	}

	public synchronized void last()
	{
		System.out.println("Inside A, this is the last method");
```

```
	}
}
class B
{
	public synchronized void bar(A a)
	{
		System.out.println("Thread2 starts execution of bar() method");
		try
		{
			Thread.sleep(2000);
		}
		catch (InterruptedException ie)
		{
		}
		System.out.println("Thread2 trying to call a.last()");
		a.last();
	}

	public synchronized void last()
	{
		System.out.println("Inside B, this is the last method");
	}

}
public class Test extends Thread{

	//instance area
	 A a=new A();
	 B b=new B();

	//instance method
	 public void m1(){
		this.start();

		a.foo(b);//executed by main thread
	 }

	 @Override
	 public void run(){
		b.bar(a);//executed by child thread
	 }

	//JVM -> main thread
	public static void main(String[] args){
		Test t=new Test();
		t.m1();
	}
}

Output
Thread1 starts execution of foo() method
Thread2 starts execution of bar() method
Thread2 trying to call a.last()
Thread1 trying to call b.last()

Explanation
++++++++++
t1(mainthread)
   => starts foo(),since foo() is synchronized and a part of 'A' class so t1
```

applies lockof(A) and
      starts the execution, while executing it encounters Thread.sleep().so T.S gives chance for t2 thread.
      After getting a chance again by TS, it tries to execute b.last.
      but lock of b is with t2 thread, so t1 enters into waiting state.

t2=> starts bar(),since bar() is synchronized and a part of 'B' class so t2 applies lockof(B) and
     starts the execution, while executing it encounter Thread.sleep(),so TS gives chance again for t1 thread.
     After getting a chance again by TS, it tries to execute a.last()

     but lock of a is with t1 thread, so t2 enters into waiting state.

Since both the threads are in waiting state and it would be waiting for ever,so we say the above pgm would result in "DeadLock".

Daemon Threads
==============
 The thread which is executing in the background is called "DaemonThread".
 eg: AttachListener,SignalDispatcher,GarbageCollector,....

remember the example of movie
   1. producer
   2. director
   3. music director
   4. ....
   5. ....
   6. ....

MainObjective of DaemonThread
  The main objective of DaemonThread, to provide support for Non-Daemon threads(main thread).
eg:: if main threads runs with low memory then jvm will call GarbageCollector thread, to destroy
     the useless objects,so that no of bytes of free memeory will be improved with this free
     memory main thread can continue its execution.

Usually Daemon threads having low priority,but based on our requirement daemon threads can run
with high priority also.
   JVM => creates 2 threads
            a. Daemon Thread(priority=1,priority=10)
            b. main (priority=5)
                  while executing the main code, if there is a shortage of memory then immediately
                  jvm will change the priority of Daemon thread to 10, so Garbage collector
                  activates Daemon thread and it frees the memory after doing it immediately
                  it changes the priority to 1, so main thread it will continue.

How to check whether the Thread is Daemon or not?
  public boolean isDaemon() => To check wheter the thread is "Daemon"
  public void setDaemon(boolean b) throws IllegalThreadStateException
             b=> true,means the thread will become Daemaon,before starting the Thread we need
                 to make the thread as "Daemon" otherwise it would result in

"IllegalThreadStateException".

What is the deafult nature of the Thread?
Ans. By deafult the main thread is "NonDaemon".
for all remaining thread Daemon nature is inherited from Parent to child, that is
if the parent thread is "Daemon" then child thread will become "Daemon" and if the parent
thread is "NonDaemon" then automatically child thread is also "NonDaemon".

Is it possible to change the NonDameon nature of Main Thread?
Ans. Not possible, becoz the main thread starting is not in our hands, it will be started by "JVM".

```
eg::
class MyThread extends Thread{}
public class Test  {
      public static void main(String[] args){
            System.out.println(Thread.currentThread().isDaemon());//false
            Thread.currentThread().setDaemon(true);//RE:IllegalThreadStartException

            MyThread t=new MyThread();
            System.out.println(t.isDaemon());//false
            t.setDaemon(true);
            t.start();
            System.out.println(t.isDaemon());//true
      }
}
```

Note::
Whenever last NonDaemon threads terminates,automatically all Daemon Threads will be terminated
irrespective of their position.
eg:: makeup man in shooting is a DaemonThread
hero is main thread
if hero role is over, then automatically the makeup role is also over automatically.

```
eg::
class MyThread extends Thread{
      public void run(){
            for (int i=1;i<=10 ;i++ ){
                  System.out.println("child thread");
                  try{
                        Thread.sleep(2000);//2sec
                  }
                  catch (InterruptedException e){
                        System.out.println(e);
                  }
            }
      }
}
public class Test  {
      public static void main(String[] args){
            MyThread t=new MyThread();
            t.setDaemon(true);//stmt-1
            t.start();
            System.out.println("end of main thread");
      }
```

```
}
Output:
if we comment stmt-1, then both the threads are NonDaemon threads it would continue
with its
execution.

end of main thread
child thread
child thread
 ...
 ...
 ...

Output
If we remove comment on stmt-1, then main thread is NonDaemon thread where as
userdefined thread is DaemonThread, if the main thread finishes the execution then
automatically the
DaemonThread also will finish the execution.


ThreadGroup
==========
 Based on functionality,we can group the threads into single unit is called
"ThreadGroup".
 ThreadGroup contains a group of threads, in addition to threads the thread group
can also
 contain subthread groups
  ThreadGroup
     t1,t2,t3,t4,.... tn
     tx,ty,tz(subthread group)
     ta,tb,tc(subthread group)
 Advnatage
    We can perform common operations easily(remeber the example of whatsapp group)

EveryThread in java belongs to some ThreadGroup
  eg::
public class Test  {
   public static void main(String[] args){
      System.out.println(Thread.currentThread().getThreadGroup().getName());//main

System.out.println(Thread.currentThread().getThreadGroup().getParent().getName());/
/System
   }
}
 main() is called main thread, main thread belongs to a group called "main".
 for every thread group there would be parent group called "System".

As how for every class there is a parent class called "Object", similarly every
thread in java
belongs to some Thread group,every Threadgroup in java is the child group of System
group directly
or indirectly.

System group contains several SystemLevelThreads
    a. Finalizer(GarbageCollector)
    b. ReferenceHandler
    c. SignalDispatchter
    d. AttachListener
           ....
```

```
          ....
    z. mainthreadgroup
           a. Thread-0
         b. Thread-1
            ....
            ....
           z. subthreadgroup
                 Thread-0
             Thread-1
                   .....
```

To create ThreadGroup in java, we need use ThreadGroup class constructor
a. ThreadGroup is class present in "java.lang" pacakge and it is the direct child class of
   "Object".

Constructors
  ThreadGroup t=new ThreadGroup(String name);
     eg:: ThreadGroup g1=new ThreadGroup("firstGroup");