case5:Overloading of run() method
we can overload run() method but Thread class start() will always call run() with zero argument.
if we overload run method with arguments, then we need to explicitly call argument based run method and it will be executed just like
normal method.
eg::

```
class MyThread extends Thread{
	public void run(){
		System.out.println("no arg method");
	}
	public void run(int i){
		System.out.println("zero arg method");
	}
}
class ThreadDemo{
	public static void main(String... args){
		MyThread t=new MyThread();
		t.start();
	}
}
```

Output:: NO arg method.

Case6::Overriding of start() method
If we override start() then our start() method will be executed just like normal method,but no new Thread will be
created and no new Thread will be started.

eg#1.

```
class MyThread extends Thread{
	public void run(){
		System.out.println("no arg method");
	}
	public void start(){
		System.out.println("start arg method");
	}
}
class ThreadDemo{
	public static void main(String... args){
		MyThread t=new MyThread();
		t.start();
	}
}
```

Output:: start arg method
It is never recomended to override start() method.

case7::

```
class MyThread extends Thread{
	public void run(){
		System.out.println("run method");
	}
	public void start(){
		System.out.println("start  method");
	}
}
class ThreadDemo{
	public static void main(String... args){
		MyThread t=new MyThread();
```

```
            t.start();
            System.out.println("Main method");
      }
}
Output::
MainThread
  a. Main method
  b. start method.

eg#2.
class MyThread extends Thread{
      public void start(){
            super.start();
            System.out.println("start  method");
      }
      public void run(){
            System.out.println("run method");
      }

}
class ThreadDemo{
      public static void main(String... args){
            MyThread t=new MyThread();
            t.start();
            System.out.println("Main method");
      }
}
Output::
MainThread
  a. Main method
  b. start method
UserDefinedThread
  a. run method

case8:: Life cycle of a Thread
MyThread t=new MyThread(); // Thread is in born state
      t.start();  //Thread is in ready/runnable state
if Thread scheduler allocates CPU time then we say thread entered into Running
state.
if run() is completed by thread then we say thread entered into dead state.

=> Once we created a Thread object then the Thread is said to be in new state or
born state.
=> Once we call start() method then the Thread will be entered into Ready or
Runnable state.
=> If Thread Scheduler allocates CPU then the Thread will be entered into running
state.
=> Once run() method completes then the Thread will entered into dead state.

case9::
After starting the Thread, we are not supposed to start the same Thread again,then
we say Thread
is in "IllegalThreadStateException".
 MyThread t=new MyThread(); // Thread is in born state
      t.start();  //Thread is in ready state
       ....
       ....
      t.start(); //IllegalThreadStateException
```

```
Creation of Thread using Runnable interface
===================================
1. Creating a Thread using java.lang.Thread class
            a. use start() from Thread class
            b. override run() and define the job of the Thread.

2. Creation of a Thread requirement to SUNMS is an SRS
            interface Runnable{
                        void run();
            }
            class Thread implements Runnable{ // Adapter class
                  public void start(){
                        1. Register the thread with ThreadScheduler
                        2. All other mandatory low level activities(memory level)
                        3. invoke or call run() method
                  }
                  public void run(){
                        //job for a thread
                  }
            }

shortcuts of eclipse
ctrl+shift+T => To open a defnition of any class
ctrl + o => To list all the methods of the class

Note:
  public java.lang.Thread();
                              |=> thread class start(), followed by thread class
run()
  public java.lang.Thread(java.lang.Runnable);
                              |=> thread class start(), followed by implementation
class of Runnable run()


Defining a Thread by implementing Runnable Interface
=================================================
public interface Runnable{
    public abstract void run();
}
public class Thread implements Runnable{
      public void start(){
            1. register Thread with ThreadScheduler
            2. All other mandatory low level activites
            3. invoke run()
      }
      public void run(){
            //empty implementation
      }
}

eg::1
 class MyRunnable implements Runnable{
      @Override
      public void run(){
            for(int i=1;i<=10;i++)
                  System.out.println("child thread");
      }
 }
public class ThreadDemo{
```

```
      public static void main(String... args){
            MyRunnable r=new MyRunnable();
            Thread t=new Thread(r);//call MyRunnable run()
            t.start();
            for(int i=1;i<=10;i++)
                  System.out.println("main thread");
      }
}
Output::
 MainThread
   a. main thread
       ....
       ....
 ChildThread
   a. child thread
      ...
      ...
        ...

Case study
=========
 MyRunnable r=new MyRunnable();
 Thread t1=new Thread();
 Thread t2=new Thread(r);

case1: t1.start()
                  A new thread will be created,which is responsible for executing
Thread class run()

output
mainthread
            main thread
            main thread
            main thread
            main thread
            main thread

case2: t2.start()
                  A new thread will be created,which is responsible for executing
MyRunnable run()

output
mainthread
            main thread
            main thread
            main thread
            main thread
            main thread

userdefinedthread
      child thread
      child thread
      child thread
      child thread
      child thread

case3: t1.run()
                 No new thread will be created,but Thread class run() will be
executed just like normal method call.
```

```
output
mainthread
            main thread
            main thread
            main thread
            main thread
            main thread
```

case4: t2.run()
No new thread will be created,but MyRunnable class run() will be executed just like normal method call.

```
output
mainthread
       child thread
      child thread
      child thread
      child thread
      child thread
      main thread
      main thread
      main thread
      main thread
      main thread
```

case5: r.start()
It results in CompileTime Error

case6. r.run()
No new thread will be created,but MyRunnable class run() will be executed just like normal method call.

```
output
mainthread
       child thread
      child thread
      child thread
      child thread
      child thread
      main thread
      main thread
      main thread
      main thread
      main thread
```

```
MyRunnable r=new MyRunnable();
 Thread t1=new Thread();
 Thread t2=new Thread(r);
```

case1: t1.start()
case2: t2.start()
case3: t2.run()
case4: t1.run()
case5: r.start()
case6: r.run()

In which of the above cases a new Thread will be created which is responsible for the execution of MyRunnable run() method ?

```
      t2.start();

In which of the above cases a new Thread will be created ?
      t1.start();
      t2.start();

In which of the above cases MyRunnable class run() will be executed?
      t2.start();
      t2.run();
      r.run();

Different approach for creating a Thread?
 A. extending Thread class
 B. implementing Runnable interface

Which approach is the best approach?
 a. implements Runnable interface is recomended becoz our class can extend other
class through
     which inheritance benift can brought in to our class.
     Internally performance and memory level is also good when we work with
interface.
 b. if we work with extends feature then we will miss out inheritance benefit becoz
already our
     class has inherited the feature from "Thread class", so we normally don't
prefere
     extends approach rather implements approach is used in real time for working
with  "MultiThreading".


Various Constructors available in Thread class
==============================================
a. Thread t=new Thread()
b. Thread t=new Thread(Runnable r)
c. Thread t=new Thread(String name)
d. Thread t=new Thread(Runnable r,String name)
e. Thread t=new Thread(ThreadGroup g, String name);
f. Thread t=new Thread(ThreadGroup g, Runnable r);
g. Thread t=new Thread(ThreadGroup g, Runnable r,String name);
h. Thread t=new Thread(ThreadGroup g, Runnable r,String name,long stackSize);


Alternate approach to define a Thread(not recomended)
=====================================================
class MyThread extends Thread{
      public void run(){
            System.out.println("child thread");
      }
}
class ThreadDemo {
      public static void main(String... args){
            MyThread t=new MyThread();
            Thread t1=new Thread(t);
            t1.start();
            System.out.println("main thread");
      }
}

Output::2 threads are created
 MainThread
```

```
    main thread
 ChildThread
    child thread

internally related
==================
Runnable
  ^
  |
Thread
  ^
  |
MyThread

Names of the Thread
===================
 Internally for every thread,there would be a name for the thread.
   a. name given by jvm
   b. name given by the user.

eg::
class MyThread extends Thread{

}
public class TestApp{
      public static void main(String... args){
            System.out.println(Thread.currentThread().getName());//main

            MyThread t=new MyThread();
            t.start();
            System.out.println(t.getName());//Thread-0

            Thread.currentThread().setName("Yash");//Yash
            System.out.println(Thread.currentThread().getName());//Yash

            System.out.println(10/0);
               //Exception in thread "yash" java.lang.ArithmeticException:/by zero
                                 TestApp.main()
      }
}
=> It is also possible to change the name of the Thread using setName().
=> It is possible to get the name of the Thread using getName().

methods
  public final String getName();
  public final void setName(String name);

eg#2.
class MyThread extends Thread{

      @Override
      public void run(){
           System.out.println("run() executed by Thread  ::
"+Thread.currentThread().getName());
      }
}
public class TestApp{
      public static void main(String... args){
            MyThread t=new MyThread();
```

```
            t.start();
             System.out.println("main() executed by Thread ::
"+Thread.currentThread().getName());
      }
}
Output:: run()  executed  by  Thread:: Thread-0
                 main() executed  by  Thread:: main
```