Multi-Threading

- Multi-tasking is the process of executing more than one task simultaneously .
- Multi-tasking is classified in to 2 types
- Process Based multi-Tasking
  - Process is nothing but a program which is under execution
  - Process based multi-tasking is the process of executing more than one program which is existed in the different location of the "RAM" memory
  - Process based multi-tasking is heavy weight .i.e. these are taking more resources and more process time
- Thread based Multi-Tasking
  - Thread is nothing but smallest part of the process
  - Thread Based Multi Tasking is the process of executing more then on functionality of the process simultaneously
  - Thread Based Multi-Tasking is light weight , These are taking less resources and less processing time

```
class Sample {
  void m1( )  //1 sec
   {.......}

  void m2( )  //2 sec
   {......}

  void m3( )  //2 sec
   {......}

 public static void main(String args[ ])
  {
     Sample s=new Sample( );
              s.m1( );
              s.m2( ) ;
              s.m3( ) ;
  }
}
```

If we execute the above program then execution will be the sequential. In order to make all the functionalities simultaneously then we have re-engineer the above program using multi-Threading

If we develop the program using multi-Threading then complete program will be taken by thread scheduler for scheduling .

Scheduling is the process in which specific time is allocated for every functionality where the control remains in the particular functionality till the particular period of time. Once the time is lapsed then control will jumps to another functionality with different time slice, But we are not guarantee in execution order . The Order my change.

In-Order to achieve "Multi-Thread" in Java we have to Approaches

- By using java.lang.Thread©
- By using java.lang.Runnable(i)

java.lang.Thread

predefined constant variable of Thread class
public final static int NORM_PRIORITY (5)
public final static int MIN_PRIORITY (1)
public final static int MAX_PRIORITY (10)

Constructor:

Thread()

Eg: Thread t=new Thread();
* it is default constructor of Thread class.
* It will create a thread class object with their default names [Thread-0 | Thread-1 | Thread-2.....]

Thread(String):

Thread t=new Thread("Shashi");
> It will create thread object with specified name

Thread(Runnable)

Thread t=new Thread(st);
> It is a copy constructor it will create thread object by taking the properties of Runnable Interface Object

Methods :

java.lang.Thread

public void run( ){ } :
* Whatever the logic you want execute as a thread then corrospoding logic must be define in the "run()" of the Thread class

* run ( ) of Thread class should not be called explicitly rather "run( )" of thread class will be executed "start( )" of java.lang.Thread

Example:

```
import java.lang.Thread;
class Cat extends Thread
{
        @Override
        public void run()
        { for(int i=1;i<=10;i++)
                { System.out.println("Cat ... : "+i); }
        }
}

class Rat extends Thread
{  @Override
        public void run()
        { for(int i=20;i<=30;i++)
                { System.out.println("Rat ... : "+i); }
        }
}

class ThreadDemo {
        public static void main(String args[ ]){

                Cat c=new Cat( );
                Rat r=new Rat( );
                  c.start();
                        r.start();
                for(int i=40;i<=50;i++)
                  {System.out.println("Main ... : "+i); }

        }
}
```

Note : we should not call "run()" of Thread class .By chance if we call "run()" of Thread then execution will be sequence

java.lang.Thread

public String getName();

- It will return name of the Thread Object

public void setName(String):

- Used to Set the name of an existed Thread

Example:

```
import java.lang.Thread;
class Child extends Thread
{
        @Override
        public void run()
        { System.out.println("Child-Thread ..."); }
}

class ThreadNames
{
        public static void main(String args[])
        {
                Child c=new Child();  //Child <-- Thread Object
        String tname=c.getName();
                System.out.println("Thread Name is : "+tname);

                    c.setName("Child");
                   tname=c.getName();
                   System.out.println("Thread Name is : "+tname);
        }
}
```

What is default thread in Java ?

main thread is default thread and it is auto executed Thread.

Inorder to get the Object of current working Thread then we have to

java.lang.Thread

public static Thread currentThread( );

Example:

```
import java.lang.Thread;
class ThreadNames2{
        public static void main(String args[]){
                Thread t=Thread.currentThread();
                String tname=t.getName();
                System.out.println("Name of CWT : "+tname);
        }
}
```

- Whenever we define any thread by default every thread is created with "NORM_PRIORITY".Thus we are not guarantee in execution order of the threads
- Based On Our Application requirements we can also set Priority of thread by using " public void setPriority(int newPriority)"

java.lang.Thread

Public void setPriority(int newPriority);

- **If you want get Priority of thread then we have use:**

public int getPriority( )

```
import java.lang.Thread;

class Cat extends Thread
{
        @Override
        public void run()
        {}
}
```

```
class Rat extends Thread
{
        @Override
        public void run()
        {}
}

class ThreadPriorityDemo
{
        public static void main(String args[])
        {
                Cat c=new Cat();
                Rat r=new Rat( );

                int cp=c.getPriority();  // 5
                int rp=r.getPriority(); //5

                System.out.println("Priority of Cat : "+cp);
                System.out.println("Priority of Rat : "+rp);

                Thread cwt=Thread.currentThread();
                int mp=cwt.getPriority(); //5
                System.out.println("Priority of Main : "+mp);

                r.setPriority(Thread.MAX_PRIORITY);
                                // public static final int MAX_PRIORITY [10]
                rp=r.getPriority();
                System.out.println("Priority of Rat : "+rp);
        }
}
```


Note: the Thread which is having MAX_PRIORITY will be executed First. The Thread Which is having MIN_PRIORITY then that thread will be executed at last and the Threads which are having same priorities will be execute simultaneously.

Note: Based on our application requirements we can also delay the execution of thread based on the particular period of type. Then we have to use the following overloaded methods

//java.lang.Thread
//public static void sleep(long ms) throws InterruptedException
//public static void sleep(long ms,int ns) throws InterruptedException

```
import java.lang.Thread;

class Child extends Thread
{
	@Override
	public void run()
	{
		try{ sleep(20000,40); }
		catch(InterruptedException ie)
			{ ie.printStackTrace(); }

		for(int i=1; i<=10; i++)
		{ System.out.println("Child - Thread ... : "+i); }
	}
}

class Parent
{
	public static void main(String args[ ])
	{
		Child c=new Child();
			c.start();

		for(int i=20; i<=30; i++)
		  {System.out.println("Main - Thread ... : "+i); }
	}
}
```

Note: If Required we can also interrupt the Thread which are in sleep by using

```
//java.lang.Thread
  //public  void interrupt( ) throws InterruptedException

import java.lang.Thread;

class Child extends Thread
{
        @Override
        public void run()
        {
                try{
                        System.out.println
                                        ("Child-Thread going for Sleep for 10Sec");
                                sleep(10000);
                         } //try
                catch(InterruptedException ie)
                {
                        System.out.println("Oh..!!! I Got Interrupted ....  :(   ");
                }//catch
        }//run
}//class

class MainThread {
        public static void main(String args[ ])  throws InterruptedException
        {
                Child c=new Child();
                        c.start();

     System.out.println("Main Thread ..");
      Thread.sleep(5000);  //5sec
       c.interrupt();

        }
}
```

Note: Based the application requirements we also delay the execution of the thread till another thread execution process is "completed". Using "join()"

java.lang.Thread
public static void sleep(long ms) throws InterruptedException
public static void sleep(long ms,int ns) throws InterruptedException
public void join( ) throws InterruptedException
public void join(long ms) throws InterruptedException
public void join(long ms,int ns) throws InterruptedException

Example:

```
import java.lang.Thread;
class Child extends Thread
{
   @Override
        public void run()
        {
                for(int i=1;i<=10;i++)
                {
                        try{ sleep(2000); }
                        catch(InterruptedException j){ }
                       System.out.println("Child - Thread ... : "+i);
                }//for
        }//run
}//class

class JoinDemo {
        public static void main(String args[ ]) throws InterruptedException
        {
                Child c=new Child();
                        c.start();
                                c.join();

                for(int i=20;i<=30;i++)
```

```
                { System.out.println("Main Thread ... : "+i);  }
        }
}
```

What is synchronization ?

- It is the process of restrict more than one thread to perform the operations on the same functionality simultaneously
- We can achieve thread safety by using "synchronized" keyword
- Advantages are avoiding data inconsistency
- Disadvantages are delaying the execution of the program

Example:

```
import java.lang.Thread;
class ATM
{
        public synchronized  void wd(String s)
        {
                for(int i=1; i<=10;i++)
                {
                        try{ Thread.sleep(1000); }
                        catch(InterruptedException ie)
                          { ie.printStackTrace(); }
                        System.out.println("WD by Mr|Mrs ... : "+s);
                }
        }
}

class Thread1 extends Thread
{
        ATM a;
        Thread1(ATM a){ this.a=a; }

        @Override
        public void run()
        { a.wd("Sai"); }
}
```

```
class Thread2 extends Thread
{
        ATM a;
        Thread2(ATM a)
          { this.a=a; }

        @Override
        public void run()
        { a.wd("Balu"); }
}

class MainThread
{
        public static void main(String args[ ])
        {
                ATM a=new ATM();
                Thread1 t1=new Thread1(a);
                Thread2 t2=new Thread2(a);
                        t1.start();
                                        t2.start();
        }
}
```

Example application using java.lang.Runnable(i)

- Runnable(i) predefined interface to achieve multi-threading
- While working multi-Threading then we must override "public abstract void run( )" of Runnable interface

```
import java.lang.Runnable;
import java.lang.Thread;
class SeetaThread implements Runnable
{
        @Override
        public void run()
        { for(int i=1;i<=10;i++)
                { System.out.println("Seeta-Thread ..."+i);}
        }
}
class RunnableDemo
{
        public static void main(String args[ ])
        {
                 SeetaThread st=new SeetaThread();
                // Thread t=new Thread( Runnable );
                 Thread t=new Thread(st);
                          t.start();
                 for(int i=20;i<=30;i++)
                 { System.out.println("Main-Thread ... "+i); }
        }
}
```