Multi Threading:

1. Multi Threading is the process that performs many operations simultaneously is known as Multi Threading.
2. What ever the operations are performed with the help CPU Scheduler.
3. CPU Scheduler will allocate a particular time period for each and every operation.

Thread:

1. Thread allows a program to divide into two or more operations running at same time.
2. Every thread in Java is created and controlled by the java.lang. Thread class.
3. Thread is a light weight process because whenever we are creating a thread it is not occupying the separate memory it uses the same memory.

Thread Scheduler:

If Multiple threads are trying to execute then which Thread should be executed first it is decided by Thread Scheduler.
Which is a part of JVM.

Which algorithm or behavior followed by Thread Scheduler we cannot expect exactly.
It is a JVM vendor dependent hence in Multithreading Examples we cannot expect exact execution order or exact output.

Life Cycle of Thread:

Once we create 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 the cpu then the Thread will be entered into running state.
Once run() Method completes, then the Thread will entered into dead state.

```
public class Eg1 {

//Main Thread, When Java Program starts up, one thread executes immediately, That is Main Method(Main Method)
public static void main(String[] args) {

//currentThread() is a static method, Returns the Current Executing Thread
Thread thread = Thread.currentThread();
System.out.println(thread); // Thread[main,5,main]
System.out.println(thread.getName()); // main
System.out.println(thread.getClass()); // class java.lang.Thread
System.out.println(thread.getState()); // RUNNABLE

}
}
```

Ways to Create a Thread:

By extending a Thread class.
By implementing Runnable Interface.

Diff between start() method and run() method?

In case of start() method a new thread will be created which is responsible for the execution of run() method.

In case of run() method no new thread will be created. And run() is executed just like a normal method.

```
//Creating a Thread by extending the Thread Class
public class Eg2  extends Thread{

public static void main(String[] args) {

Eg2 eg2 = new Eg2();
eg2.start();  //When ever i call start() Method, run() Method need to be invoked
              //start() Method can be accessed only when we extends from Thread Class

}

public void run() {
System.out.println("Run Method");
}
}
```

```java
//Create a Thread by Implementing Runnable Interface
public class Eg3 implements Runnable {

public static void main(String[] args) {

Eg3 eg3 = new Eg3();

Thread t1 = new Thread(eg3);
t1.start(); // When ever i call start() method run() Method need to be invoked

Thread t2 = new Thread(eg3);
t2.start(); // When ever i call start() method run() Method need to be invoked

Thread t3 = new Thread(eg3);
t3.start(); // When ever i call start() method run() Method need to be invoked
}

// When we implement Runnable Interface it will @override run() Method
@Override
public void run() {

System.out.println("Run Method");
}
}
```

```
Run Method
Run Method
Run Method
```

Getting and Setting Names of Thread

Every Thread in Java has some name.

1. It may be default name provided by JVM or Customized Name provided by the Programmer.

For Example

1. Each Thread has a name like **Thread-0, Thread-1, Thread-2,….so on**
2. Java provides **some methods** to change the **thread name** and they are defined in **java.lang.Thread class**

We have a following methods are useful to set and get the names of Thread.

public String getName(): is used to return the name of a thread.
public void setName(String name): is used to change the name of a thread.

```
//Setting the Custom exception
public class Eg4 extends Thread {

public static void main(String[] args) {

Eg4 e1 = new Eg4();
System.out.println(e1.getName());
Thread.currentThread().setName("Custom Exception or Thread Can be Created here....");//Custom Exception or Thread Can be Created here....
System.out.println(Thread.currentThread().getName());//Thread-0

}
}
```

Get and Set Thread Priority:

public final int getPriority():

java.lang.Thread.**getPriority() method** returns priority of given thread.

public final void setPriority(int newPriority):

java.lang.Thread.**setPriority() method** changes the priority of thread to the value of newPriority.

This Method throws **IllegalArgumentException** if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.

Thread Priority

Whenever we create a **Thread** in Java, it always has some **Priority** assigned to it.

Priority can either be given by **JVM** while creating the **Thread** or it can be given by **Programmer Explicitly.**

Accepted value of **Priority** for a thread is in **range** of 1 to 10.

There are 3 static variables defined in Thread class for priority.

public static int MIN_PRIORITY: This is minimum priority that a thread can have. Value for this is 1.
public static int NORM_PRIORITY: This is default priority of a thread if do not explicitly define it. Value for this is 5.
public static int MAX_PRIORITY: This is maximum priority of a thread. Value for this is 10.

```
public class Eg5 extends Thread {

public static void main(String[] args) {

Eg5 eg1 = new Eg5();
Eg5 eg2 = new Eg5();
Eg5 eg3 = new Eg5();

System.out.println(eg1.getPriority()); // 5
System.out.println(eg2.getPriority()); // 5
System.out.println(eg3.getPriority()); // 5

eg1.setPriority(7);
eg2.setPriority(8);
eg3.setPriority(6);

System.out.println(eg1.getPriority()); // 7
System.out.println(eg2.getPriority()); // 8
System.out.println(eg3.getPriority()); // 6

eg1.setPriority(MAX_PRIORITY);
eg2.setPriority(MIN_PRIORITY);
eg3.setPriority(NORM_PRIORITY);

System.out.println(eg1.getPriority()); // 10
System.out.println(eg2.getPriority()); // 1
System.out.println(eg3.getPriority()); // 5
}
}
```

isAlive() Method

Tests if this **thread is alive**. A thread **is alive if it has been started** and has not yet died.

activeCount() Method

an estimate of the number of active threads in the current thread's thread group

```
public class Eg6 {

public static void main(String[] args) {

Thread t1 = new Thread();
t1.start();
System.out.println(t1.isAlive()); // true

Thread t2 = new Thread();
t2.start();
System.out.println(t2.isAlive()); // true

System.out.println(Thread.activeCount()); //2
}

}
```