# MultiThreading(V.V.IMPORTANT)

# Q)What is the difference between Process, Procedure and Processor?

# Ans:

**Process is a flow of execution to perform a particular task and this flow of execution is not visible to us but we eventually see the effect of this flow of execution. Called as a PROCESS.**

**Procedure is a set of instructions or number of instructions or block of instructions representing a particular task is called as PROCEDURE.**

**Processor is an H/W component/device to generate no. of processes inorder to execute applications so hardware device is called as PROCESSOR.**

**At starting point of the computers we have Single Process Mechanism or Single Tasking to execute applications.**

**In Single Process Mechanism, System is able to allow only one task at a time to load into the memory even our system main memory is capable to manage all the tasks.**

**In Single Process mechanism, System is able to allow only one process to execute all the tasks which are available in our application, it will follow sequential kind of execution, it will increase application execution time and it will reduce application performance.**

**To overcome the above problems we have to use Multi Process Mechanism or multi tasking. Multi tasking system is able to load more than one task at a time in main memory and it allows more than one process to execute application, it will follow parallel execution , it will reduce application execution time and will improve application performance.**

**To execute applications by using Multi Tasking or Multi Process Mechanism we have to use the following components.**

* Main Memory: To load all the tasks.
* Process Waiting Queue: To keep track of all processes
* Process Context Block: to manage status of all the processes execution.
* Process Scheduler: It will take process from Process Waiting Queue and it will assign time stamps to each and every process inorder to execute.

**In the above multi tasking system, controlling is swiched from one process context to another process context called as "Context Switching".**

There are two types of Context Switchings:

1. **Heavy Weight Context Switching:**

**It is the context switching between two heavy weight components, it will take more memory and more execution time , it will reduce application performance.**

**EX:** Context switching between two Processes.

1. **Light Weight Context Switching:**

**It is the context switching between two lite weight components, it will take less memory and less execution time and it will improve application performance. EX: Context switching between two threads.**

**Q)What is the difference between Process and Thread?**

**Ans:**

Process is heavy weight , to handle it, system has to consume more memory and more execution time, it will reduce application performance.

Thread is lite weight, to handle it, system has to consume less memory and less execution time, it will improve application performance.

There are two thread models to execute applications:

1. Single Thread Model
2. Multi Thread Model
3. **Single Thread Model:**

**It will allow only one thread to execute application, it will follow sequential exexcution, it will increase application execution time and will reduce application performance.**

1. **Multi Thread Model:**

**It will allow more than one thread to execute application, it will follow parallel execution[all at a time will be started] , it will reduce application execution time and will improve application performance.**

**Java is following Multi-Thread Model to execute applications and it will provide very good environment to create and execute more than one thread at a time.**

**In java applications, to create Threads JAVA has provided following predefined library in the form of java.lang package.**

**Q) What is thread and in how many ways we are able to create threads in java applications?**

**Ans:**

**Thread is a flow of execution to perfrom a particulaer task.**

**As per the predefined library provided by JAVA , there are two ways to create threads in java applications.**

1. **Extending Thread class:**

**In this approach , we have to declare a class , it must be extended from java.lang.Thread class.**

**class MyThread extends Thread**

**{**

**--implementation----**

**}**

1. **Implementing Runnable interface:**

**In this approach, we have to declare a class, it must implement java.lang.Runnable interface.**

**class MyThread implements Runnable**

**{**

**---implementation----**

**}**

**Threads Design in Java:**

There are two approaches to create threads in java applications:

1. Extending Thread class
2. Implementing Runnable interface
3. **Extending Thread class:**
4. **Declare a user defined class.**
5. **Extend java.lang.Thread class to user defined class**
6. **Override Thread class run() method in user defined thread class with the implementation[application logic] repersenting a particular task which we want to perform/execute by creating a thread.**
7. **In main class, in main() method, create object for user defined class.**
8. **Create a new Thread and access run() method on it by calling start() method on thread.**

6) **Access Thread class provided start() method on user defined thread class object reference variable.**

**Note: Thread is a flow of execution to perform a particular task and keep that task in run() method.**

**The main intention of start() method is to create new thread and to access run() method by passing the generated thread.**

**Note: In JAVA/J2EE applications, only start() method from Thread class can create new thead and access run() method.**

**public void start()**

**EX:**

class MyThread extends Thread

{

public void run()

{

for(int i=0;i<10;i++)

{

System.out.println("User Thread :"+i);

}

}

}

class Test

{

public static void main(String[] args)

{

MyThread mt=new MyThread();

mt.start();

for(int i=0;i<10;i++)

{

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

}

}

}

1. **How many threads in above program?**

**Ans)**

**2 theads are running in the above program:**

1. **main thread: JVM is executing main() method by creating a thread and that JVM created thread is called as main thread. main() method must be executed by a thread[called main thread].**
2. **User thread: USER thread is executing run() method. Both are running parallely so mixed output.**

**Note: Internally thread scheduler is giving time stamps and as per the time stamps only the execution is going on.**

Q) **In java applications, to create threads we have already first approach[Extending Thread class] then what is the requirement to go for Second Approach[ implementing Runnable interface]?**

**Ans:**

In java applications, to create threads if we use first approach then we have to declare a user defined class and it must be extended from java.lang.Thread class, in this context, it is not possible to extend other classes , if we extend any other class like Frame, along with Thread class then it will represent Multiple Inheritance, it is not possible in java .

class MyClass extends Frame, Thread{

---

}

To overcome the above problem, we have to use second approach to create thread , that is, implementing Runnable interface.

class MyClass extends Frame implements Runnable{

----

}

1. **Implementing Runnable interface:**
2. **Declare a user defined class.**
3. **Implement java.lang.Runnable interface.**
4. **Provide implementation part[application logic] in run() method which we want to execute by creating a thread.**
5. **In main class, in main() method, create a thread and access user defined thread class run() method.**

To perform the above step we have to use the following cases:

class MyThread implements Runnable{

public void run(){

for(int i=0;i<10;i++)

{

System.out.println(“USER THREAD :”+i);

}

}

}

**case-1**:

**MyThread mt=new MyThread();**

**mt.start();**

**Status**: Compilation Error.

**Reason:** start() method was not declared in MyThread class and

in its super class java.lang.Objectclass, start() method is existed in java.lang.Thread class.

**Case-2**:

**MyThread mt=new MyThread();**

**mt.run();**

**Status: No Compilation Error, but, only main thread access MyThread class run() method like a normal java method, no multi threading environment,No user defined thread is created, only sequential execution is going on.**

**Case-3:**

**MyThread mt=new MyThread();**

**Thread t=new Thread();**

**t.start();**

**Status:** No Compilation Error, start() method creates new thread and it access Thread class run()method, not MyThread class run() method.

**Case-4:**

**MyThread mt=new MyThread();**

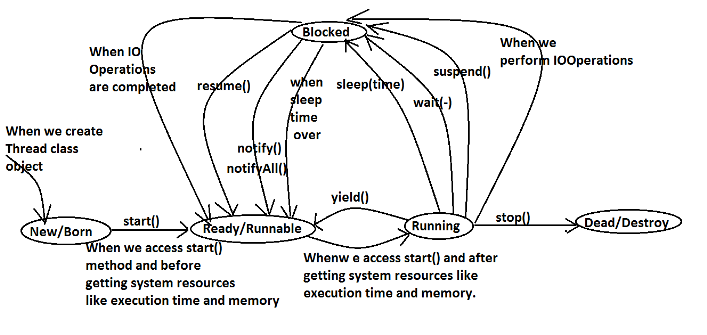
**Thread t=new Thread(mt);**

**t.start();**

**Status:** No Compilation Error, start() method creates new thread and it will bypass new thread toMyThread class run() method and now output will be a mixed output means parallel execution is going on.

# **Thread Lifecycle:**

**The collective information of a thread from its starting point to ending point is called as "Thread Life Cycle".**

****

**In java applications, Threads are able to have the following states as**

**part of their lifecycle:-**

1. **New/Born State:**

**When we create Thread class object in java applications then Thread will come to New/Born state.**

1. **Ready/Runnable State**:

**When we access start() method Thread Scheduler has to assign system resources like memory and time, here, before assigninig system resources and after calling start() method is called as Ready/Runnable state.**

1. **Running State:**

**In java applications, after calling start() method and after getting system resources like memory and execution time is called as "Running State".**

**NOTE: We can send a thread from Running state to Runnable state directly by accessing yield() method , but, it is not supported by Windows operating system, because, it will peform its functinoality on the basis of Threads priority**

**values, and priority based operations are not supported by windows operating system.**

1. **Dead/Destroy State:**

**In java applications, when we access stop() method over Running thread then that thread will come to Dead/Destroy state.**

1. **Blocked State:**

**In java applications, we are able to keep a thread in Blocked state from Running state in the following situations.**

1. **When we access sleep(--) method with a particular sleep time.**
2. **When we access wait() method.**
3. **When we access suspend() method.**
4. **When we perform IO Operations.**

**In java applications, we are able to bring a thread from Blocked state to Ready / Runnable state in the following situations.**

1. **When sleep time is over.**
2. **If any other thread access notify() / notifyAll() methods.**
3. **If any other thread access resume() method.**
4. **When IO Operations are completed.**