UNIT III – INPUT / OUTPUT

Streams classes: Byte – Character - File class - File operations - Console class – Serialization. Multithreading: Java thread model – Creating thread – Creating multi thread - Thread priorities – Synchronization - Inter thread communication.

MULTITHREADING

Thread:

A thread is a single sequential (separate) flow of control within program. Sometimes, it is called an execution context or light weight process.

Multithreading

Multithreading is a conceptual programming concept where a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

Multitasking

Executing several tasks simultaneously is called multi-tasking. There are 2 types of multi-tasking

1. Process-based multitasking
2. Thread-based multi-tasking

1. Process-based multi-tasking

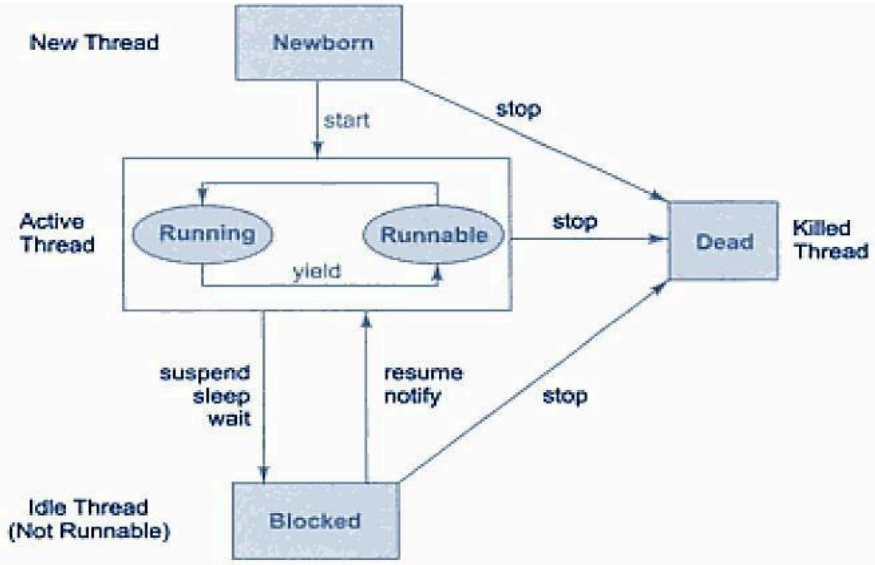
Executing various jobs together where each job is a separate independent operation is called process-based multi-tasking.

2. Thread-based multi-tasking

Executing several tasks simultaneously where each task is a separate independent part of the same program is called Thread-based multitasking and each independent part is called Thread. It is best suitable for the programmatic level. The main goal of multi-tasking is to make or do a better performance of the system by reducing response time.

Life Cycle of Thread

A thread can be in any of the five following states



1.Newborn State:

When a thread object is created a new thread is born and said to be in Newborn state.

2.Runnable State:

If a thread is in this state it means that the thread is ready for execution and waiting for the availability of the processor. If all threads in queue are of same priority then they are given time slots for execution in round robin fashion

3.Running State:

It means that the processor has given its time to the thread for execution. A thread keeps running until the following conditions occur. Thread give up its control on its own and it can happen in the following situations

1. A thread gets suspended using suspend() method which can only be revived with resume() method
2. A thread is made to sleep for a specified period of time using sleep(time) method, where time in milliseconds
3. A thread is made to wait for some event to occur using wait () method. In this case a thread can be scheduled to run again using notify () method.

A thread is pre-empted by a higher priority thread

4. Blocked State:

If a thread is prevented from entering into runnable state and subsequently running state, then a thread is said to be in Blocked state.

5. Dead State:

A runnable thread enters the Dead or terminated state when it completes its task or otherwise

The Main Thread

When we run any java program, the program begins to execute its code starting from the main method. Therefore, the JVM creates a thread to start executing the code present in the main method. This thread is called the main thread. Although the main thread is automatically created, you can control it by obtaining a reference to it by calling currentThread() method. Two important things to know about main thread are,

- It is the thread from which other threads will be produced.
- The main thread must always be the last thread to finish execution.

```
class MainThread
{
	public static void main(String[] args)
	{
		Thread t1=Thread.currentThread();
		t.setName("MainThread");
		System.out.println("Name of thread is "+t1);
	}
}
```

Output:

Name of thread is Thread[MainThread,5,main]

Creation Of Thread

Thread Can Be Implemented In Two Ways

- Implementing Runnable Interface
- Extending Thread Class

1.Create Thread by Implementing Runnable

The easiest way to create a thread is to create a class that implements the Runnable interface. To implement Runnable, a class need only implement a single method called run()

Example:

```
public class ThreadSample implements Runnable
{
	public void run()
	{
		try
		{
			for (int i = 5; i > 0; i--)
			{
				System.out.println("Child Thread" + i);
				Thread.sleep(1000);
```

```
                }
            }
            catch (InterruptedException e)
            {
                System.out.println("Child interrupted");
            }
            System.out.println("Exiting Child Thread");
        }
}
public class MainThread
{
        public static void main(String[] arg)
        {
            ThreadSample d = new ThreadSample();
            Thread s = new Thread(d);
            s.start();
            try
            {
                for (int i = 5; i > 0; i--)
                {
                    System.out.println("Main Thread" + i);
                    Thread.sleep(5000);
                }
            }
            catch (InterruptedException e)
            {
                System.out.println("Main interrupted");
            }
            System.out.println("Exiting Main Thread");
        }
}
```

2. Extending Thread Class

The second way to create a thread is to create a new class that extends Thread, and then to create an instance of that class. The extending class must override the run( ) method, which is the entry point for the new thread. It must also call start( ) to begin execution of the new thread.

Example:

```
public class ThreadSample extends Thread
{
	public void run()
	{
		try
		{
			for (int i = 5; i > 0; i--)
			{
				System.out.println("Child Thread" + i);
				Thread.sleep(1000);
			}
		}
		catch (InterruptedException e)
		{
			System.out.println("Child interrupted");
		}
		System.out.println("Exiting Child Thread");
	}
}
public class MainThread
{
	public static void main(String[] arg)
	{
		ThreadSample d = new ThreadSample();
		d.start();
		try
		{
```

```
                for (int i = 5; i > 0; i--)
                {
                        System.out.println("Main Thread" + i);
                        Thread.sleep(5000);
                }
        }
        catch (InterruptedException e)
        {
                System.out.println("Main interrupted");
        }
        System.out.println("Exiting Main Thread");
    }
}
```

Thread priority:

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

public static int MIN_PRIORITY

public static int NORM_PRIORITY

public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example :

```
public class MyThread1 extends Thread
{
    MyThread1(String s)
    {
        super(s);
        start();
    }
    public void run()
    {
        for(int i=0;i<5;i++)
        {
            Thread cur=Thread.currentThread();
            cur.setPriority(Thread.MAX_PRIORITY);
            int p=cur.getPriority();
```

```
                System.out.println("Thread Name"+Thread.currentThread().
        getName());
                System.out.println("Thread Priority"+cur);
        }
    }
}
class MyThread2 extends Thread
{
    MyThread2(String s)
    {
        super(s);
        start();
    }
    public void run()
    {
        for(int i=0;i<5;i++)
        {
            Thread cur=Thread.currentThread();
            cur.setPriority(Thread.MIN_PRIORITY);
            System.out.println(cur.getPriority());
            int p=cur.getPriority();
            System.out.println("ThreadName"+Thread.currentThread()
        .getName());
            System.out.println("Thread Priority"+cur);
        }
    }
}
public class ThreadPriority
{
    public static void main(String[] args)
    {
        MyThread1 m1=new MyThread1("MyThread1");
        MyThread2 m2=new MyThread2("MyThread2");
    }
}
```

ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY
PALKULAM, KANYAKUMARI
OBSERVE OPTIMIZE OUTSPREAD