Multithreading

The ability of executing several programs simultaneously is known as multitasking.In system terminology it is called as multithreading where a program(process) is divided into two or more subprograms(processes) which can be implemented at the same time in parallel.

A unique property of JAVA is its support for multithreading.JAVA allows use of multiple flows of control in developing programs.Each flow of control is called as a thread.Threads run in parallel.Multithreading is a powerful programming tool.It allows the user to do multiple things at the same time.User can divide a long program into threads and execute them in parallel.

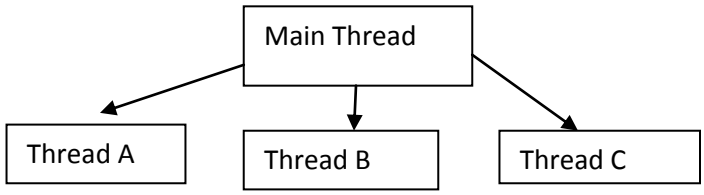


Fig -1 Multithreaded Program

How to create Thread

Threads are implemented in the form of objects that contain a method called run().

The run() method is the heart of any thread.Its the method where the thread behavior is written.

```
public void run()
{
//statements
}
```

The run () method should be invoked by an object of the concerned thread.

A new thread can be created by using two methods

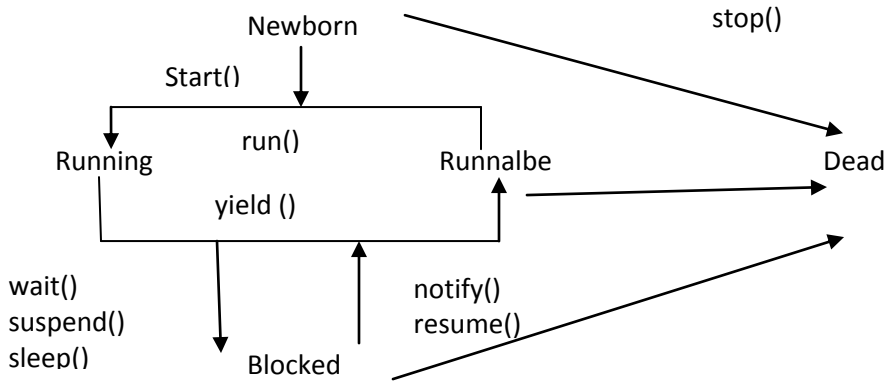
I. By creating a thread class - Define a class that extends Thread class and override the method run().
II. By converting a class to a thread – Define a class that implements Runnable interface.The Runnable interface has only one method run().This method need to be defined in the class which implements this interface.

Life cycle of Thread

Following are the states of thread life cycle

1. **Newborn state** – when we create a thread a thread is born which is not scheduled for running.We can schedule it for running by using strat() method.We can kill it using stop() method.
2. **Runnable state** – Runnable means the thread is ready for execution and is waiting for the availability of the processor.If all the thread waiting in queue for the processor are of same priority then they will be given a time slot for execution in round robin fashion.We can use yield() method to relinquish control to another thread.
3. **Running state** – Running state means that the processor has given its time to the thread for its execution.The thread will run until it relinquish control on its own or it is preempted by a higher priority thread.A running thread may relinquish its control in one of the situations as suspend(),sleep(time),wait().
4. **Blocked state** – A thread is said to be blocked when it I prevented from entering into the runnable state and subsequently the running state.This happens when the thread is suspended,sleeping or waiting in order to satisfy certain requirements.
5. **Dead state** – A running thread ends its life when it has completed its task in run() method and it will go to dead state.We can also kill it by sending the stop message to it at any state to put the thread in dead state.

Following diagram explains transition of thread states



Declaration of a Thread

Declaring a class

```
class mythread extends Thread
{
//statements
}
```

Implementing run() method

The run() method will be inherited in mythread class.We need to override run() method

```
public void run()
{
//statements
}
```

How to start a new thread

```
mythread t= new mythread();
t.strat();
```

e.g How to used threads – Program to Print table of 2 and 7 using threads

```
class  Two extends Thread
{
public void run()
{
System.out.println("Table of 2 is");
for(int i=1;i<10;i++)
{
System.out.println(" "+i*2);
}
System.out.println("Exit from thread Two");
}
class Seven extends Thread
{
public void run()
{
```

```
System.out.println(“Table of 7 is”);
for(int i=1;i<10;i++)
{
System.out.println(“ “+i*7);
}
System.out.println(“Exit from thread Seven”);
}
class Theradtest
{
public static void main(Str ing args[])
{
Two t=new Two();
Seven s=new Seven();
t.start();
s.start();
}
}
```

Use of stop(),sleep() methods in thread

```
class  Two extends Thread
{
public void run()
{
System.out.println(“Table of 2 is”);
for(int i=1;i<10;i++)
{

System.out.println(“ “+i*2);
If(i = = 5)
stop();
}
System.out.println(“Exit from thread Two”);
}
class Seven extends Thread
{
public void run()
{
System.out.println(“Table of 7 is”);
for(int i=1;i<10;i++)
{
System.out.println(“ “+i*7);
if( i == 2)
try
```

```
{
sleep(500);
}
catch(Exception e)
{
 }
}
System.out.println(“Exit from thread Seven”);
}
class Theradtest
{
public static void main(Str ing args[])
{
Two t=new Two();
Seven s=new Seven();
t.start();
s.start();
}
}
```

Thread Priority

Each thread can be assigned a priority.A priority can be set by using setPriority(int number) function as

Threadname.setPriority(intnumber);

The integernumber is an value to which the thread priority is set.The Thread class defines several priority constants as

MIN_PRIORITY = 1

NORM_PRIORITY = 5

MAX_PRIORITY = 10

Intnumber can take any value as given above or any value between 1 to 10.

How to use priorities in Threads

```
class  Two extends Thread
{
public void run()
{
System.out.println(“Table of 2 is”);
for(int i=1;i<10;i++)
{
System.out.println(“ “+i*2);
```

```
}
System.out.println("Exit from thread Two");
}
class Seven extends Thread
{
public void run()
{
System.out.println("Table of 7 is");
for(int i=1;i<10;i++)
{
System.out.println(" "+i*7);
}
System.out.println("Exit from thread Seven");
}
class Theradtest
{
public static void main(Str ing args[])
{
Two t=new Two();
Seven s=new Seven();
t.setPriority(Thread.MIN_PRIORITY);
s.setPriority(5);
t.start();
s.start();
}
}
```

SYNCHRONIZATION

When two threads try to use data and methods outside themselves during this situation they compete for the same resources and may lead to serious problems.

e.g. one thread may try to read a record from a file while another is still writing to the same file.Depending on the situation we may get strange results.To avoid this JAVA uses a technique known as synchronization.

The keyword synchronized helps to solve such problems e.g. the method that will read information from a file and the method that will update the same file may be declared as synchronized.

e.g.

Producer Consumer Program which explains synchronization

```
class Room {
  public  int contents;
  public  boolean available = false;
  public synchronized int get() {
    while (available == false) {
      try {
        wait();
      }
      catch (InterruptedException e) {
      }
    }
    available = false;
    notifyAll();
    return contents;
  }
  public synchronized void put(int value) {
    while (available == true) {
      try {
        wait();
      }
      catch (InterruptedException e) {
      }
    }
    contents = value;
    available = true;
    notifyAll();
  }
}

class Consumer extends Thread {
  public Room r;
  public  int number;
  public Consumer(Room  r, int number) {
    r = r;
    this.number = number;
  }
  public void run() {
    int value = 0;
      for (int i = 1; i < 5; i++) {
        value = r.get();
        System.out.println("Consumer #" + this.number + " got: " + value);
```

```
}
}
}

class Producer extends Thread {
private Room r;
private int number;

public Producer(Room  r, int number) {
r = r;
this.number = number;
}

public void run() {
for (int i = 1; i < 5; i++) {
r.put(i);
System.out.println("Producer #" + this.number + " put: " + i);
}
}
}
public class ProducerConsumer {
  public static void main(String arg[]) {
    Room  r = new Room();
    Producer p = new Producer(r, 1);
    Consumer c = new Consumer(r, 1);
    p.start();
    c.start();
  }
}
```

OUTPUT :-

```
Producer #1 put: 1
Consumer #1 got: 1
Producer #1 put: 2
Consumer #1 got: 2
Producer #1 put: 3
Consumer #1 got: 3
Producer #1 put: 4
Consumer #1 got: 4
Producer #1 put: 5
Consumer #1 got: 5
```