International Journal of Innovative Research in Information Security (IJIRIS) ISSN: 2349-7017(O)
Volume 1 Issue 4 (October 2014) ISSN: 2349-7009(P)
www.ijiris.com

Mathematical foundations of Multithreaded programming concepts in Java language

Manoj Kumar Srivastav
Indira Gandhi National Open University
St. Xavier's College (Autonomous), Kolkata,
Pin Code-700016, India

Dr. Asoke Nath
Associate Professor, Department of Computer Science, St. Xavier's College (Autonomous),
Kolkata, Pin code-700016, India

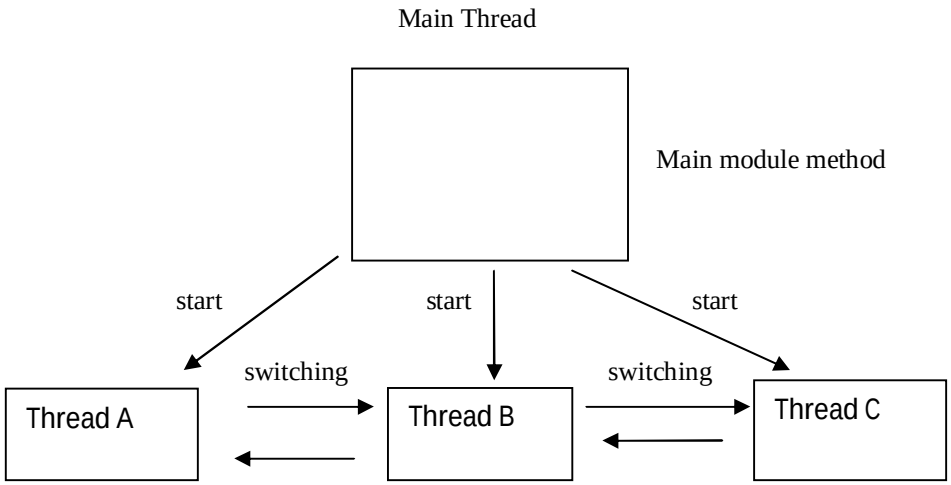
Abstract-The mathematical description of object oriented language has already been developed by Srivastav et al. The authors have already published papers where they have shown that it is possible to describe programming language such as C, Java using simple mathematical sets and relations. The authors have established that it is possible to describe object oriented language like Java and its various aspects such as object, class, inheritance using simple mathematical models. In the present study the authors have proposed the mathematical modeling of Multi threaded programming in java language. The authors have tried to explore the single threaded program and as well as multi threaded program using simple mathematical modeling. The same idea may be applied to C# language also

Keyword---object oriented, modeling, multithreaded, java, C#

I. INTRODUCTION

The term 'thread' plays an important role in object oriented programming in Java. Actually a thread is 'a single sequential flow of control within a program.' The concept of single thread is quite simple to understand and it becomes somewhat complex when there are multithreads running simultaneously, each performing different tasks within a single program. This can be enabled by multithreading where we can write program containing multiple path of execution, running concurrently, within a single program .In other words we can say that " a single program having multiple threads , executing concurrently, can be termed as multithreaded program."

Definition of Multithreading : Multithreading is a conceptual programming paradigm where a program (process) into two or more subprogram (processes), which can be implemented at the same time in parallel. For example, one subprogram can display animation on the screen while another may build the next animation to be displayed. This is something similar to dividing tasks into subtasks and assigning them.



It is important to remember that 'thread running in parallel' does not really mean that they actually run at the same time. Since all the threads are running on a single processor, the flow of execution is shared between the threads.

Multithreading and memory space: Multithreading is a actually a form of multitasking. Multitasking can either be process-based or thread-based. If we assume some programs as processes, the process-based multitasking is nothing but execution of more than one program concurrently.
On the other hand, thread-based multitasking is executing a program having more than one thread, performing different tasks simultaneously. Processes are heavyweight tasks, while threads are lightweight's tasks. In process-based multitasking, different processes are actually different programs, thus **they share different address spaces.** The context switching of CPU from one process to another requires more overhead as different spaces are involved in the same. On the contrary, the thread-based multitasking, different threads are part of the same program, thus **they share the same address space** and context switching of CPU occurs within the program, i.e within the same address space.

Multitasking:

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:
(i)Process-based Multitasking(Multiprocessing)
(ii)Thread-based Multitasking(Multithreading)

(i)Process-based Multitasking (Multiprocessing)

- Each process have its own address in memory i.e. each process allocates separate memory area.
- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

(ii) Thread-based Multitasking (Multithreading):

Threads share the same address space.
Thread is lightweight.
Cost of communication between the thread is low.
Note: At least one process is required for each thread.

II. MATHEMATICAL DESCRIPTION OF MULTITHREADING

Mathematical Description of class and object: Object-Oriented programming language is based on the following properties:

(i) Class and Object
(ii) Data Abstraction and Encapsulation
(iii) Inheritance
(iv) Polymorphism.

Class is a collection of objects of similar type.

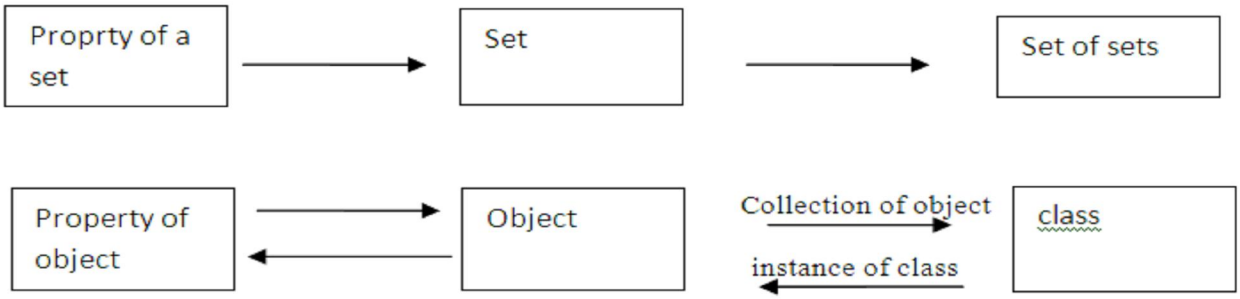
Set: A set is well- defined collection of distinct objects of our perception or our thought ,to be conceived as a whole.

Set of sets: we have defined a set as a collection of its elements. If the elements be set themselves, then we have a family of sets, or set of sets. For example, the collection of all subsets of a non-empty set S is a set of sets. This set is said to be the power set of S and is denoted by P(S). *Class may be defined as follows: -*
In set theory and its application throughout mathematics, a class is collection of sets where each set has some common property. The members of a class are countable.
So therefore, class={ sets : where all members have some common property}
The basic form of a class in object –oriented programming is
class classname

{ parameter declaration;
Method declaration;
}

Object: The members of a class are called an object. Since the members of a class is a set .Therefore it should possesses some property. Objects are real world entity such as pen, chair, tables,etc.

Similar comparision between set theory and object oriented language :



Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. Any entity that has state and behavior is known as an object. For example: chair, pen, table, etc. It can be physical and logical. Collection of objects is called class. It is a logical entity. Any programming problem is analyzed in terms of objects and the nature of communication between them. **An object takes up space in the memory and has an associated address like structure in C.**

Thus, class is a set of object and object is a set having some common feature. We can write the class by the symbol , ρ, τ, σ , ……etc and object by X,Y, Z……. etc.

MEMORY SPACE :An object is in Java is essentially a block of memory that contains space to store all instance variables. i.e each object of a class contain an address in memory space. Memory space for an object={address of the instance variables: variables are members of class}

Creating an object is also referred to as instantiating an object. Objects in Java are created using the new operator .The new operator creates an object of the specified class and returns a reference to that object. Therefore the address of the new object is changed.

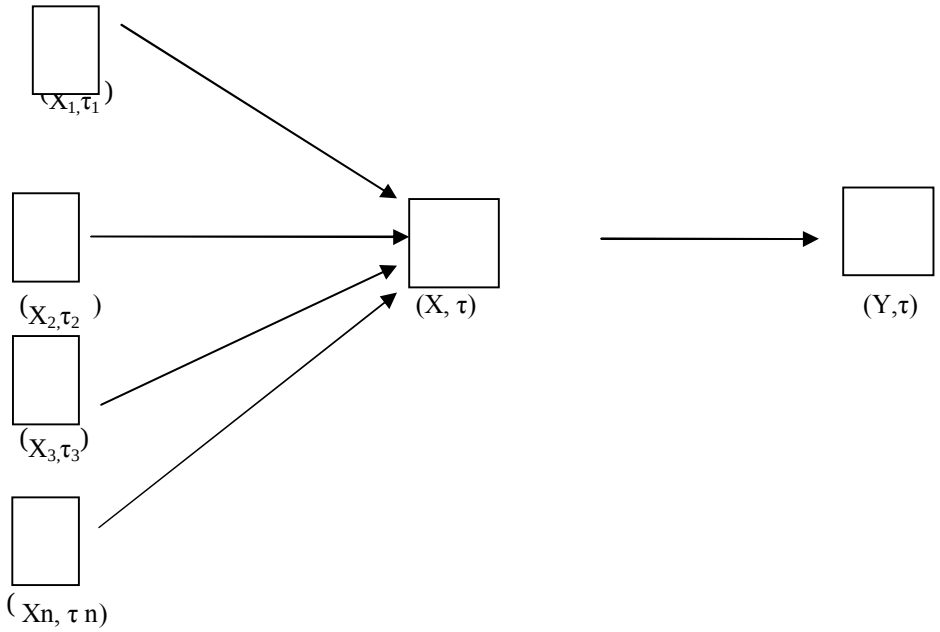
Initial object has an address	New object have new address

Therefore we define a mapping f:$(X,\tau) \rightarrow (Y,\sigma)$ such that f(address of object)=address of new object[Here X and Y are object set and τ are set of sets with respect to object X and σ are set of sets with respect to object Y. Actually τ represents classes for domain and σ represents class **for range in the above function and f represents main function.**

Mathematical Explanation of Multithreading:

A unique property of Java is its support for multithreading .That is, Java enables us to use multiple flows of control in developing program. Each flow of control may be thought of as a separate tiny program (or module) known as thread that runs in parallel to others.

Let $(X_{1,}\tau_1)$, $(X_{2,}\tau_2)$, $(X_{3,}\tau_3)$, …….. $(X_{n,}\ \tau_n)$ be representing the n-threads and may be running parallel and are subprogram of main threads/program (X,τ). Let (Y,τ) be representing new threads(objects).



Mathematically, the function can occurs as follows:

Let f1 : $(X_1,\tau_1) \longrightarrow (X,\tau)$,

f2 : $(X_2,\tau_2) \longrightarrow (X,\tau)$,

f3 : $(X_3,\tau_3) \longrightarrow (X,\tau)$,

…….

fn : $(Xn,\tau_n) \longrightarrow (X,\tau)$, and finally, the function f :$(X, \tau) \longrightarrow (Y ,\tau)$,i.e the address of the thread (X_1,τ_1) , (X_2,τ_2), (X_3,τ_3) …… (Xn ,τ_n) mapping to the thread (X ,τ) and finally the address of the thread (X ,τ) maps to new object/thread (Y,τ).Thus we can say that f(f i) : $(X ,\tau) \longrightarrow (Y ,\tau)$,

i.e . f (f i((Address of domain))=address of range ,for i=1,2,3,……,n.

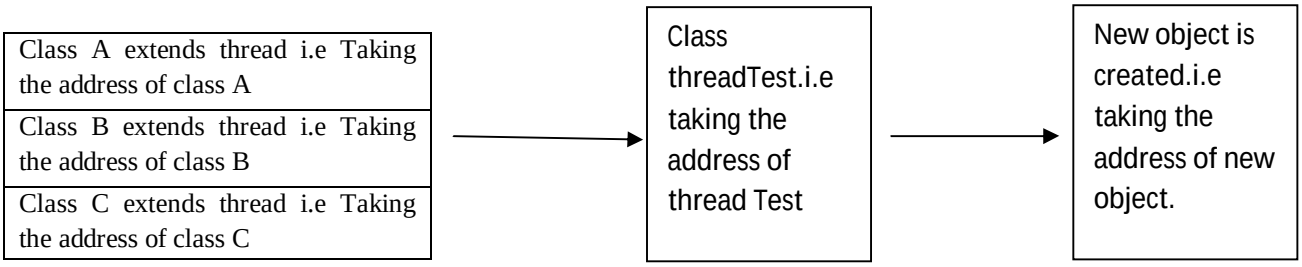
Example :Illustrate with an example on multithreading in Java.

```
Class A extends Thread
{
public void run()
{
for(int i=1; i<=5; i++)
{
System.out.println("\t From Thread A :   = " +i);
}
System.out.println(" Exit from  A");
}
}
Class B extends Thread
{
public void run()
{
for(int i=1; i<=5; i++)
{
System.out.println("\t From Thread A :   = " +i);
}
```



```
System.out.println(" Exit from  A");
}
}
Class C extends Thread
{
public void run()
{
for(int i=1; i<=5; i++)
{
System.out.println("\t From Thread A :   = " +i);
}
System.out.println(" Exit from  A");
}
}
Class ThreadTest
{
public static void main(String args[])
{
new A().start();
new B().start();
new C().start();
}
}
```

Let us explain the example mathematically:



III MATHEMATICAL PROPERTIES OF THE MULTITHREADING

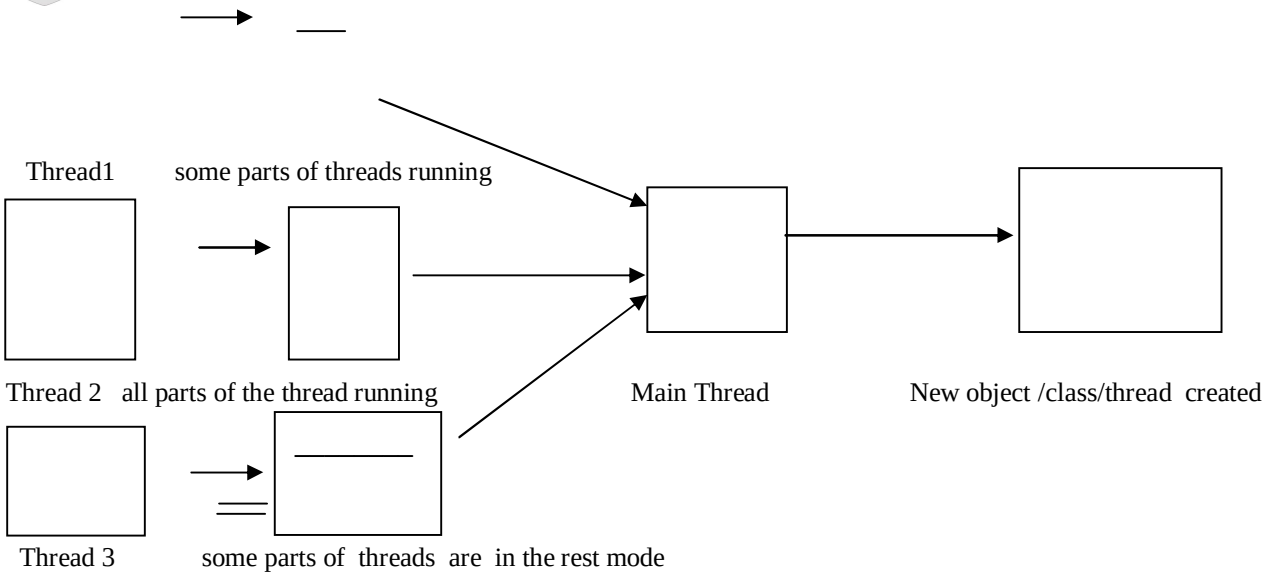
Properties(i): The space (X_1,τ_1) , (X_2,τ_2), (X_3,τ_3) …… (Xn ,τ_n) are disconnected space.All the threads are disconnected.

[Definition of disconnected space.Then a subset E of X is said to be disconnected iff there exist two non-empty separated set A and B such that E=A ∪ B.

Separated Set : :Let (X, τ) be a topological space. Two non empty subset A and B of X are said to be separated if and only if $A \cap B^c = \emptyset$ and $A^C \cap B = \emptyset$]

Properties (ii) :Two sets are separated if and only if they are disjoint and neither of them contains limit point of the other. Similarly all threads runs separately and output given by each threads does not depends on any threads.

Properties (iii) Subset of separated sets are separated. The thing can be illustrated by the following diagram for the multithreading.




```
//:class anthread _2: Write a program in java to implement
    //multithreaded program : Use yield(), sleep(), stop() method
    import java.io.*;
    class thread_A extends Thread
    {
        public void run()
        {
        int i,s,n;
        try
        {
        System.out.println("\nEntered in thread_A");
        n=4;
        s=0;
            for(i=1;i<=n;i++)
            {
            s=s+i;
            System.out.println("i="+i+" s="+s);
                if(i==3)
                {
            System.out.println("\nthread_A goes to sleep mode");
                sleep(1000);
            System.out.println("\nOut of sleep mode in thread_A");
                }
            }
         }
         catch(Exception e){}
         }
    }

    class thread_B extends Thread
    {
         public void run()
         {
         int j,p,m;
         System.out.println("\nEntered in thread_B");
         m=4;
```

```
        p=1;
            for(j=1;j<=m;j++)
            {
            p=p*j;
            System.out.println("j="+j+" p="+p);
                if(j==3)
                {
            System.out.println("\nthread_A goes to yield() mode");
                yield();
            System.out.println("\nOut of yield() mode in thread_B");
                }
            }
        }
}

class thread_C extends Thread
{
        public void run()
        {
        int k,s1,d,k1;
        System.out.println("\nEntered in thread_C");
        k=1234;
        k1=k;
        s1=0;
            while(k1 != 0)
            {
            d=k1%10;
            s1=s1+d;
            System.out.println("k1="+k1+" d="+d);
                if(k1<100)
                {
            System.out.println("\nthread_C goes to stop() mode");
                stop();
            System.out.println("\nOut of stop() mode in thread_C");
                }
            k1=k1/10;
            }
        }
}

class anthread_2
{
      public static void main(String args[])throws IOException
      {
      thread_A A=new thread_A();
      thread_B B=new thread_B();
      thread_C C=new thread_C();
      System.out.println("Calling thread_A");
      A.start();
      System.out.println("Calling thread_B");
      B.start();
      System.out.println("Calling thread_C");
      C.start();
      System.out.println("\nEnd of main thread");
      }
}
```


Remarks : There may be the situation in which some threads are running and some are in the stop mode or some parts of a thread are working..But in all the situation each threads are separated.

class thread_A extends Thread	sleep() mode may be working in some parts here.
class thread_B extends Thread	yield() mode may be working on the some stage of thread
class thread_C extends Thread	stop() mode is working on the some stage of thread

Thread Priority : In Java, each thread is assigned a priority, which affects the order in which it is scheduled for running. Mathematically, thread priority are the set which assigned with some indexed number and having some fixed probability value in all the situation.

```
//anthread_3.java: Write a program in java to implement
//multithreaded program: Use setPriority(), sleep(), stop()
//method

 import java.io.*;
class thread_A extends Thread
{
      public void run()
      {
      int i,s,n;
      try
      {
      System.out.println("\nEntered in thread_A");
      n=4;
      s=0;
            for(i=1;i<=n;i++)
            {
            s=s+i;
            System.out.println("i="+i+" s="+s);
                  if(i==3)
                  {
            System.out.println("\nthread_A goes to sleep mode");
                  sleep(1000);
            System.out.println("\nOut of sleep mode in thread_A");
                  }
            }
       }
       catch(Exception e){}
       }
}

class thread_B extends Thread
{
       public void run()
       {
       int j,p,m;
       System.out.println("\nEntered in thread_B");
       m=4;
       p=1;
            for(j=1;j<=m;j++)
            {
```



```
                p=p*j;
                System.out.println("j="+j+" p="+p);
                    if(j==3)
                    {
                System.out.println("\nthread_A goes to yield() mode");
                    yield();
                System.out.println("\nOut of yield() mode in thread_B");
                    }
                }
            }
        }

    class thread_C extends Thread
    {
            public void run()
            {
            int k,s1,d,k1;
            System.out.println("\nEntered in thread_C");
            k=1234;
            k1=k;
            s1=0;
                while(k1 != 0)
                {
                d=k1%10;
                s1=s1+d;
                System.out.println("k1="+k1+" d="+d);
                    if(k1<100)
                    {
                System.out.println("\nthread_C goes to stop() mode");
                    stop();
                System.out.println("\nOut of stop() mode in thread_C");
                    }
                k1=k1/10;
                }
            }
    }
}
    }

    class anthread_3
    {
        public static void main(String args[])throws IOException
        {
        thread_A A=new thread_A();
        thread_B B=new thread_B();
        thread_C C=new thread_C();
        System.out.println("Calling thread_A");
        A.setPriority(Thread.MAX_PRIORITY);
        B.setPriority(Thread.MIN_PRIORITY);
        C.setPriority(Thread.NORM_PRIORITY);
        System.out.println("Calling thread_C");
        C.start();
        System.out.println("Calling thread_B");
        B.start();
        System.out.println("Calling thread_A");
        A.start();
```



```
        System.out.println("\nEnd of main thread");
        }
    }
```

Remarks :

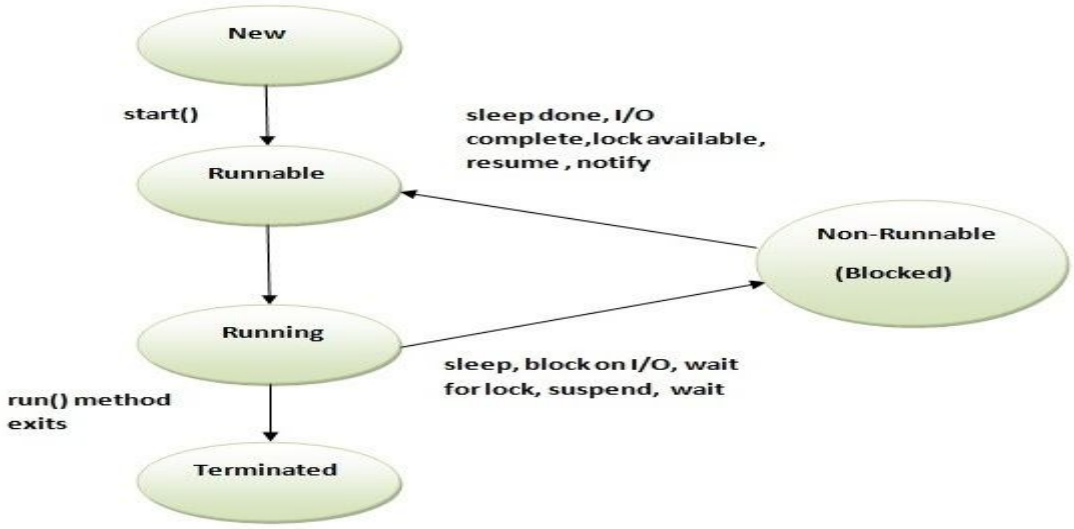
Threads	Probability
A.setPriority(Thread.MAX_PRIORITY);	Thread has highest value of probability of running
B.setPriority(Thread.MIN_PRIORITY);	Thread has lowest value of probability of running
C.setPriority(Thread.NORM_PRIORITY);	Thread has the value of probability that lies between highest value and lowest value of probability of running of threads.

Definition of Probability of an Event: If there are n elementary events associated with a random experiment and m of them are favourable to an event A, then the probability of happening or occurrence of A is denoted by P(A) and is defined in the ratio m/n. Thus,P(A)=m/n, clearly, $0 \le m \le n$.
Therefore , $0 \le m/n \le 1$
$0 \le P(A) \le 1$. If P(A)=1, then A is called certain events and A is called impossible events.
The value of thread priority lies between 1-10,The Min_Priority,Norm _Priority and Max_Priority have distinct value and it are respectively 1,5 and 10 .

IV .LIFE CYCLE OF A THREAD (THREAD STATES)

A thread can be in one of the five states in the thread. According to sun, there is only 4 states new, runnable, non-runnable and terminated. There is no running state. But for better understanding the threads, we are explaining it in the 5 states. The life cycle of the thread is controlled by JVM. The thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



1)New :
The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

International Journal of Innovative Research in Information Security (IJIRIS) ISSN: 2349-7017(O)
Volume 1 Issue 4 (October 2014) ISSN: 2349-7009(P)
www.ijiris.com

2)Runnable
The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
3)Running
The thread is in running state if the thread scheduler has selected it.
4)Non-Runnable (Blocked)
This is the state when the thread is still alive, but is currently not eligible to run.
5)Terminated
A thread is in terminated or dead state when its run() method exits.

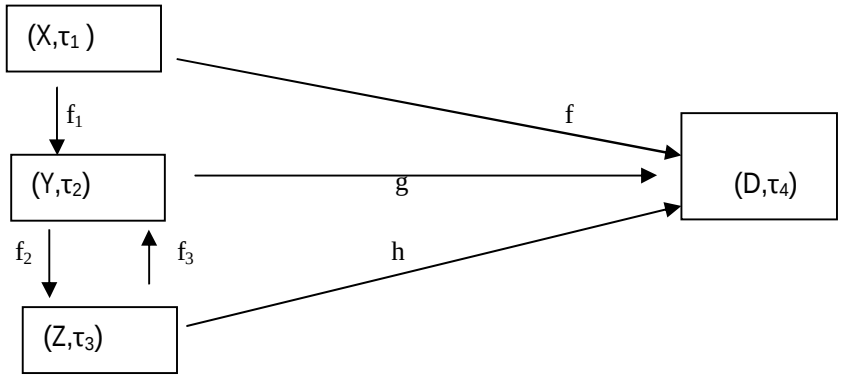
***Mathematical Description Of State Transition diagram of a thread* :**

Let (X,τ_1) denotes the newborn thread where X is object and τ_1 denotes the class (or thread) with respect to object X.

Let (Y,τ_2) denotes the runnable/running thread where Y is object and τ_2 denotes the class (or thread) with respect to object Y

Let (Z,τ_3) denotes the Blocked thread where Z is object and $\tau_3$denotes the class (or thread) with respect to object Z

Let (D,τ_4) denotes the dead/ killed thread where D is object and $\tau_4$denotes the class (or thread) with respect to object D



Note : Let us assume the following mapping occurs in the different memory spaces.

Here f1 :$(X,\tau_1) \longrightarrow (Y,\tau_2)$
f2 :$(Y,\tau_2) \longrightarrow (Z,\tau_3)$
f3 :$(Z,\tau_3) \longrightarrow (Y,\tau_2)$
f :$(X,\tau_1) \longrightarrow (D,\tau_4)$
g : $(Y,\tau_2) \longrightarrow (D,\tau_4)$
h :$(Z,\tau_3) \longrightarrow (D,\tau_4)$

Case 1:Newborn state to Dead state: Here ,we can define the function f :$(X,\tau_1) \longrightarrow (D,\tau_4)$ Such that f(address value of newborn thread)=address value of killed thread. Actually ,we may use stop() as a function in the stage.

Case 2:Newborn state to Dead state via running and runnable state: Here we can define the function f1 :$(X,\tau_1) \longrightarrow (Y,\tau_2)$ and g: $(Y,\tau_2) \longrightarrow (D,\tau_4)$ such that g(f) : :$(X,\tau_1) \longrightarrow (D,\tau_4)$
i.e. g(f(address value of new born state))= address value of dead state.

Case 3: Newborn state to Dead state via running , runnable state and Blocked state::In this stage, we can define the composite function as follows :

$h(f_2(f_1))$:$(X,\tau_1) \longrightarrow (D,\tau_4)$
$h(f_2(f_1$ (address value of new born state)))= address value of dead state
or
$g(f_3(f_2(f_1)))$: $(X,\tau_1) \longrightarrow (D,\tau_4)$

$g(f_3(f_2(f_1$ (address value of new born state))))= address value of dead state.

V. CONCLUSION

Multithreading is a process where we can make the correlation between the multithreading and disconnected space of a Topological scope. The embedding of the thread in to the main threads will be helpful to justify the memory space of different threads. Life Cycle of a thread can also be described mathematically and it also obeys the properties of disconnected space.

REFERENCES

[1] Manoj Kumar Srivastav , Asoke Nath Mathematical modeling of various statements of C-type Language, , International Journal of Advanced Computer Research(IJACR), Vol-3,Number-1, Issue-13, Page:79-87 Dec(2013).
[2] Manoj Kumar Srivastava, Asoke Nath Mathematical Description of variables, pointers, structures, Unions used in C-type language, , Joiurnal of Global Research Computer Science, Vol-5, No-2, Page:24-29, Feb(2014)
[3] Manoj Kumar Srivastav, Asoke Nath, A Mathematical Modeling of Object Oriented Programming Language : a case study on Java programming language, Current Trends in Technology and Science(CTTS) Vol-3, Issue-3, Page 134-141,(2014).
[4] Manoj Kumar Srivastav, Asoke Nath, Analysis of Compilation Errors, Runtime Errors, Reliability and Validity of a Program: A Case Study on C-language : IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 3, May 2014. Page-281-288.
[5] E Balaguruswamy- Programming with Java-TataMcGrawHill EductionP rivate Limited.,2011
[6] Sachin Malhotra, Sourabh Choudhary- Programming in Java- OXFORD University Press,2012
[7] Herbert Schildt, Java TM 2: The Complete Reference, TataMcGraw-Hill Publishing Company Limited,2001
[8] Satish Jain,Vineeta Pillai,Kratika, Introduction to Object Oriented Programming through Java,BPB Publications,2011
[9] S.K.Mapa, Real Analysis , Asoke Prakasan 1998
[10] S.K.Mapa Higher Algebra, -Sarat Book Distribution,2000
[11] K.D.Joshi-Topology.New Age International Publiocation.
[12] J.N Sharma, "Topology", Krishna Prakashan Media(p)Ltd., Meerut, (2000)
[13]R.S.aggarwal-Topology-S.chand publication.
[14]James R Munkers-Topology- Pearson Education Asia, 2001
[15] www.javatpoint.com
[16] R.D.Sharma. Mathematics class xi-. Danapat Rai Publications(P) LTD.

