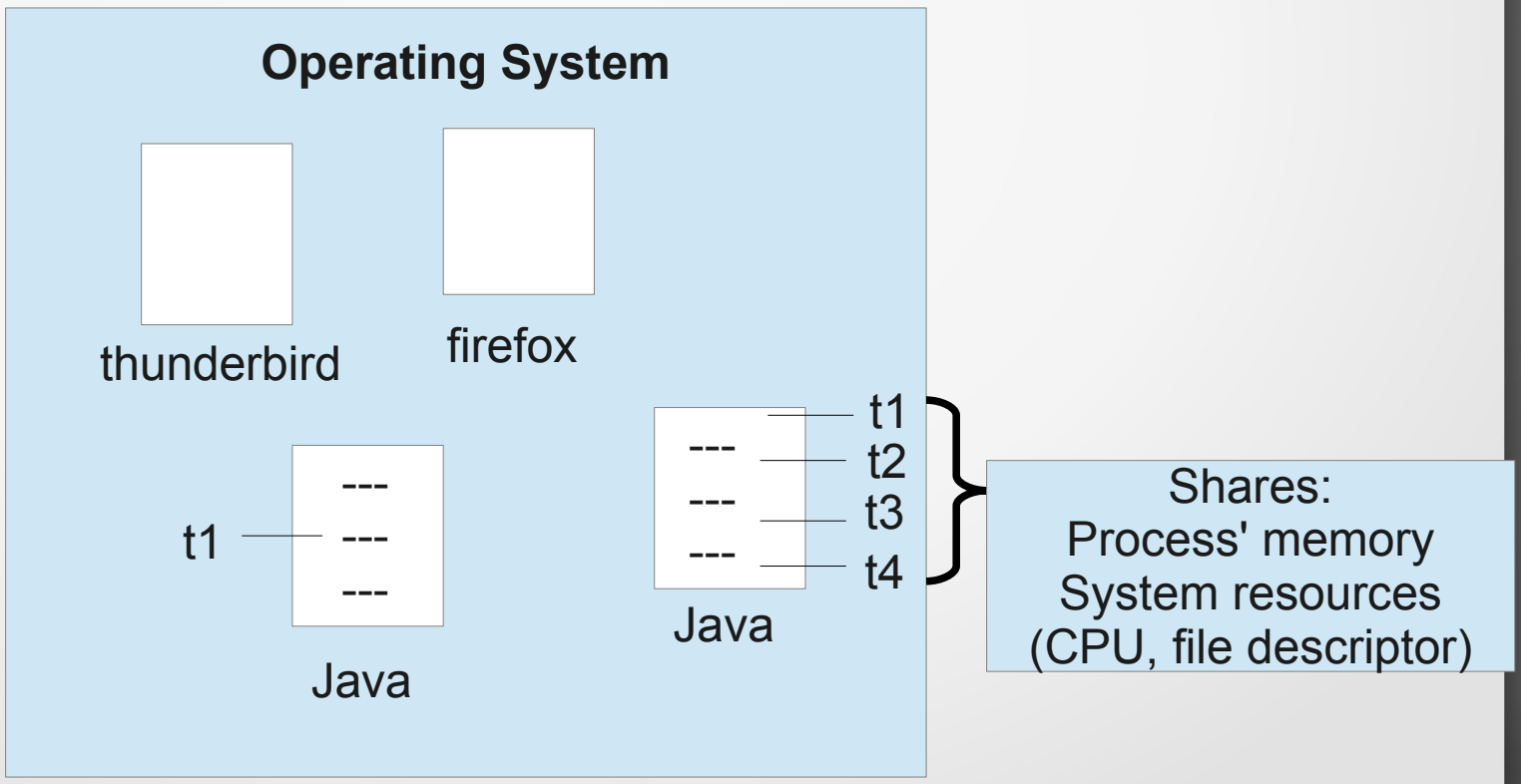
Multithreading Concept

By: R&D Dept.

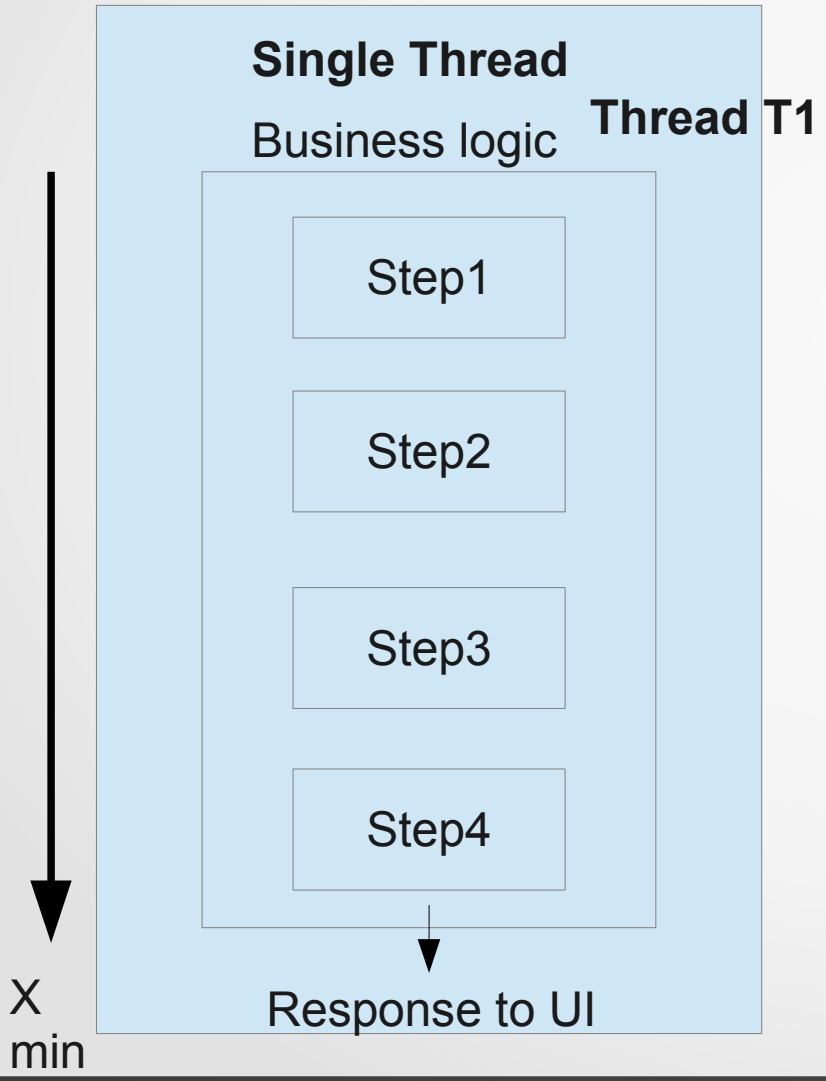
Introduction

- Multithreading is a technique,
  - More than one thread,
  - Could run parallelly with co-ordination,
  - Shares: Process's Memory, System Resources.
  - Helps: to achieve performance & to provide timely response.

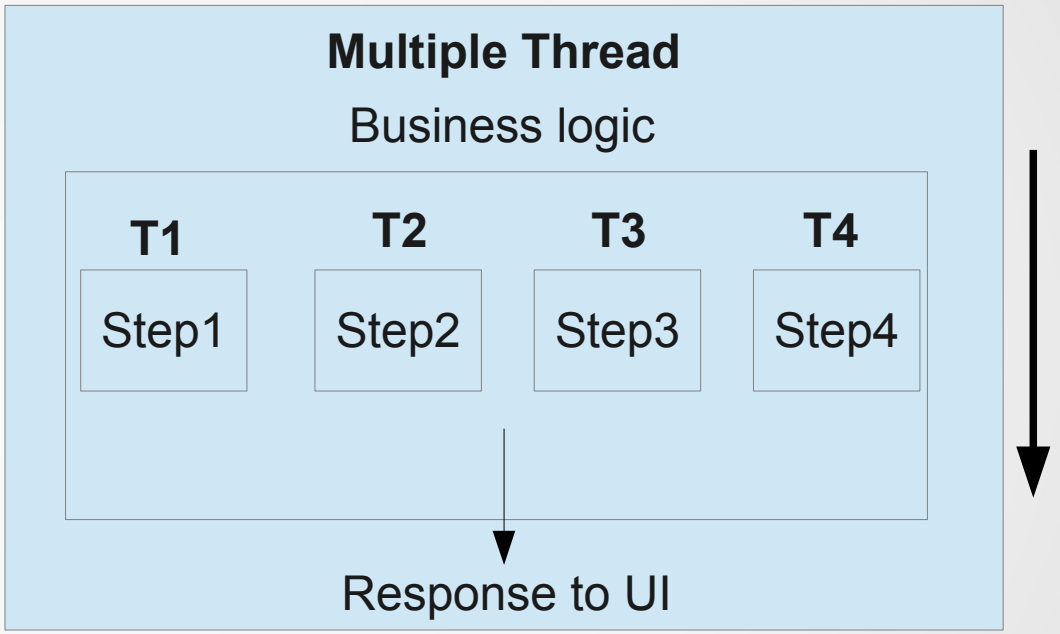


Why to use Multithreading:

Single Thread- One main thread must do everything.



Multiple Thread- Tasks distributed between threads.



approx. 50% faster

Scenarios:

- Processing of large data.
- Produce and consume scenario.
- Fetching market share values of different stocks from multiple sites/resources exactly at the same time to update different broker systems.
- Long Calculations- Could Perform Steps in parallel.

Distributing Logic

- What Is Task?

- **Scenarios** (when to implement multithreading in our environment.)

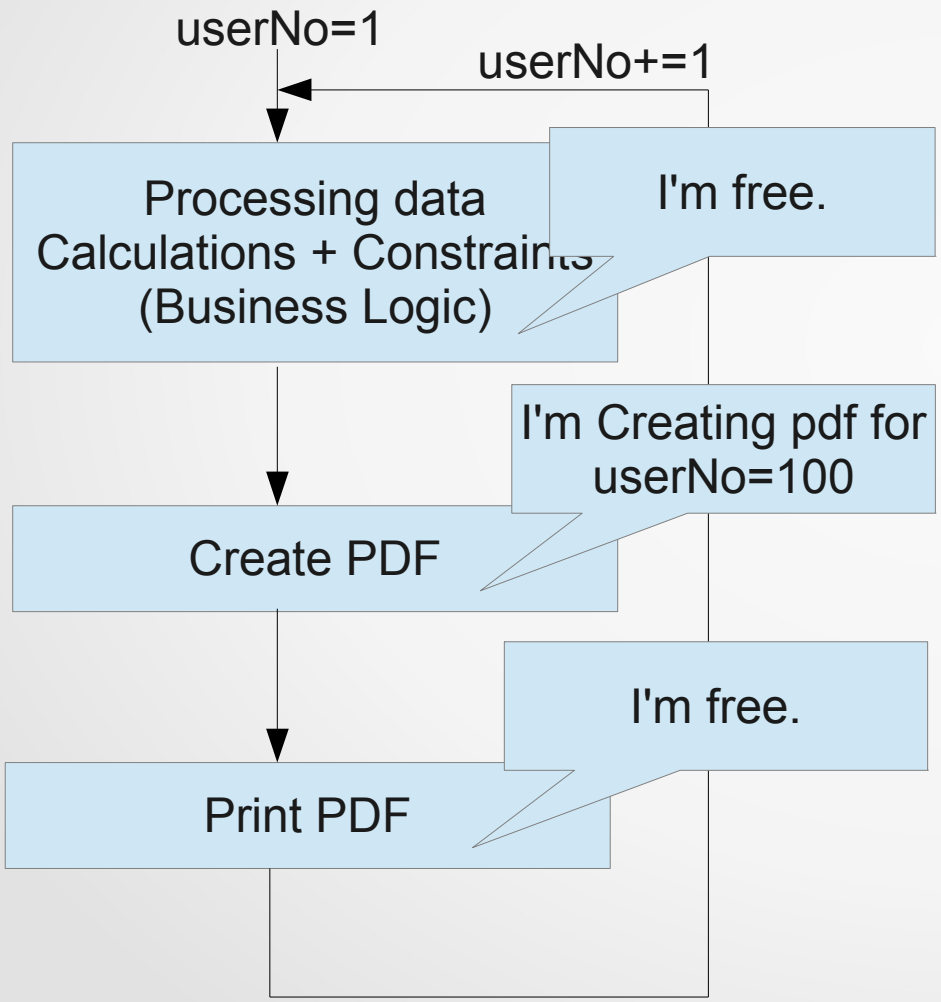
  (I) **Distribute SubTasks** between threads
  When Sub tasks could independently work.

  (II) **Distribute Tasks** between threads
  When Sub tasks are dependent.

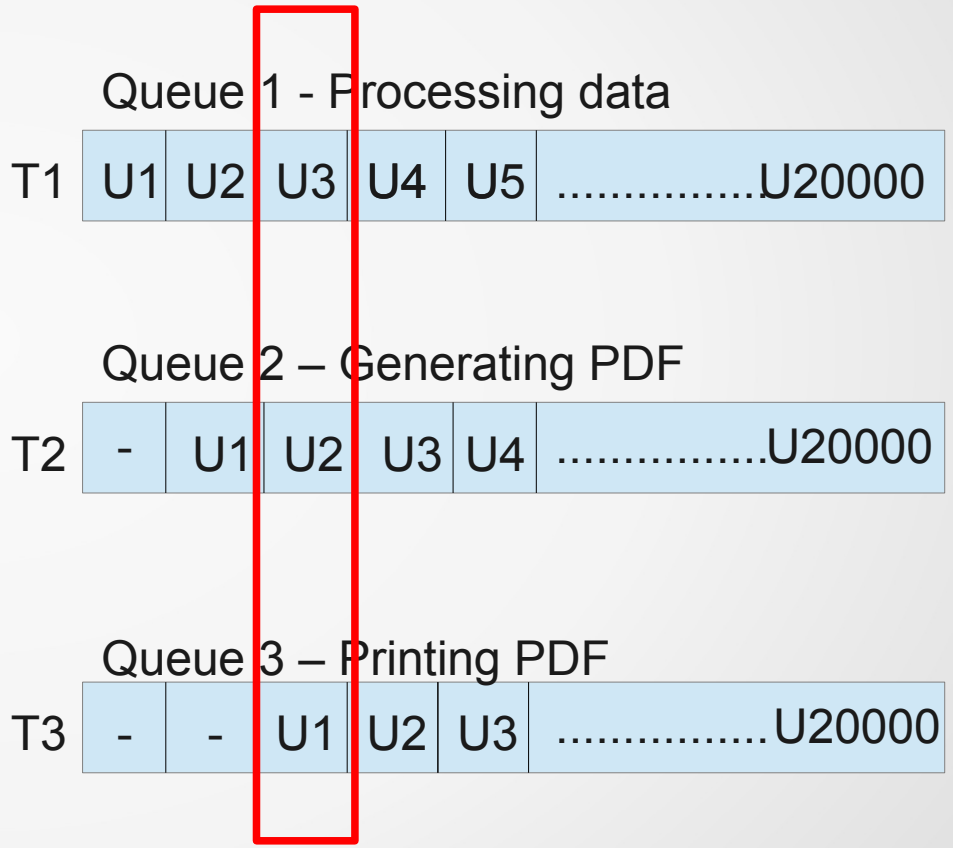
PDF Generation Module (Distributed Subtasks)

20000 Users

Single Thread:



Multi Thread:

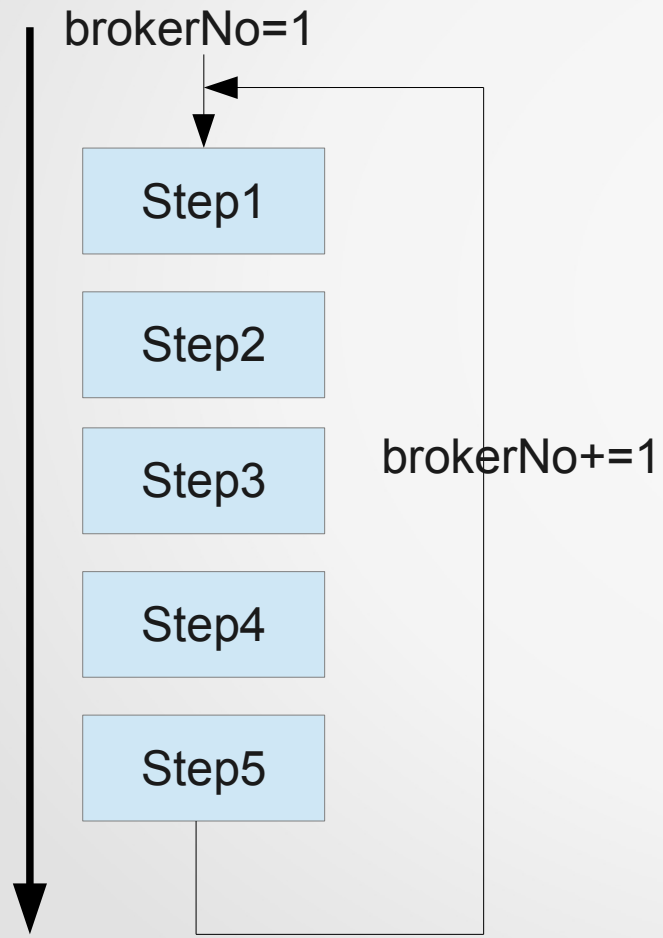


Approx. 40% Performance Gain

Brokerage Trail Bill Processing(Distributed Tasks)

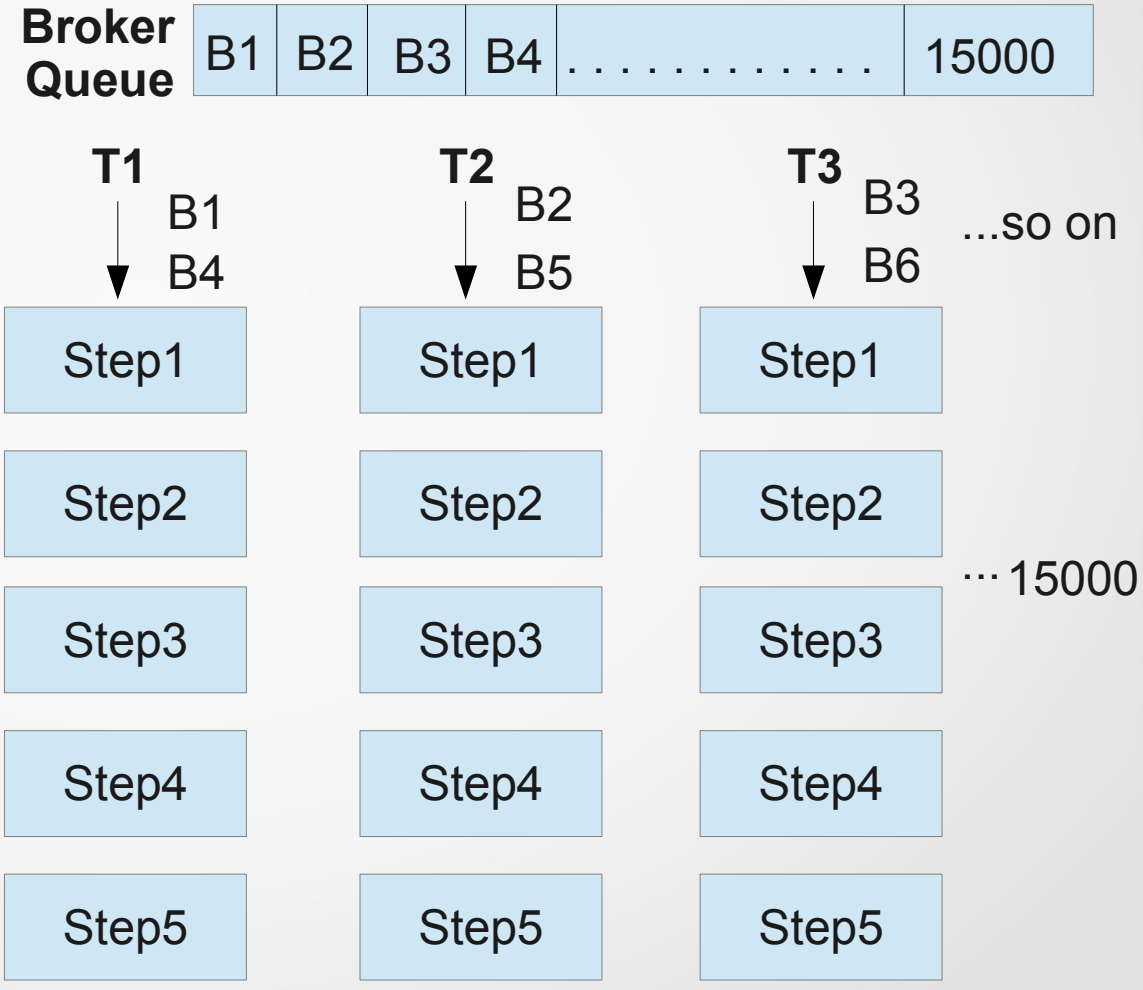
Brokers: approx 15000

Single Thread:



X sec = 1 broker processing

Multi Thread:



X sec = 3 brokers processing (approx.)

Java

MultiThreading

Implementation

- Java is a multithreaded programming language
- To imlement Multithreading, Two ways:

  (I) By Implementing *Runnable* Interface **(II)** By Extending *Thread* Class

Two Ways:

(I) By Implementing *Runnable* Interface

Steps:

A. Implement a *run()* Method of *Runnable* Interface- An Entry point for the Thread

Syntax: *public void run()*

B. Instantiate a Thread Object using the following Constructor

Syntax: *Thread(Runnable* <Thread Obj>*, String* <threadName>*)*

C. Start Thread, It will executes a call to run( ) method

Syntax: *void start( )*

Example: (By Runnable Interface)

```
class RunnableDemo implements Runnable {
        private Thread t;
        private String threadName;

        RunnableDemo(String name) {
                threadName = name;
                System.out.println("Creating " +  threadName );
        }

        public void run(){
                System.out.println("Running " +  threadName );
                try {
                    for(int i = 4; i > 0; i--){
                    System.out.println("Thread: " + threadName + ": " + i);
                    Thread.sleep(50);
                       }
                    }catch(InterruptedException e){....}
                System.out.println(“Thread ”+ threadName + “ exiting”);
        }

        public void start() {
                System.out.println(“Starting: ”+ threadName);
                        if(t==null){
                                t=new Thread(this, threadName);
                                t.start();
                        }
         }
}
```

```
public class TestThread {
    public static void main(String[] args){
        RunnableDemo R1=new
                        RunnableDemo(“Thread1”);
         R1.start();
         RunnableDemo R2=new
                        RunnableDemo(“Thread2”);
         R2.start();
     }
}
```

Output:

```
        Creating Thread1
        Starting Thread1
        Creating Thread2
        Starting Thread2
        Running Thread1
        Thread: Thread1: 4
        Running Thread2
        Thread: Thread2: 4
        Thread: Thread1: 3
        Thread: Thread2: 3
        Thread: Thread1: 2
        Thread: Thread2: 2
        Thread: Thread1: 1
        Thread: Thread2: 1
        Thread Thread1 exiting.
        Thread Thread2 exiting.
```

Implementation

(II) By Extending *Thread* Class

Steps:

A. Override *run()* Method of *Thread* Class

Syntax: *public void run( )*

B. Instantiate a Thread Object using the following Constructor

Syntax: *Thread(Runnable* <Thread Obj>*, String* <threadName>*)*

C. Start Thread, It will executes a call to run( ) method

Syntax: *void start( )*

Example: (By Extending Thread Class)

```
class ThreadDemo extends Thread {
        private Thread t;
        private String threadName;

        ThreadDemo(String name) {
                threadName = name;
                System.out.println("Creating " +  threadName );
        }

       @Override
        public void run(){
                System.out.println("Running " +  threadName );
                try {
                    for(int i = 4; i > 0; i--){
                    System.out.println("Thread: " + threadName + ": " + i);
                    Thread.sleep(50);
                       }
                    }catch(InterruptedException e){....}
                System.out.println("Thread "+ threadName + " exiting");
        }

        public void start() {
                System.out.println("Starting: "+ threadName);
                        if(t==null){
                                t=new Thread(this, threadName);
                                t.start();
                        }
         }
}
```

```
public class TestThread {
    public static void main(String[] args){
        ThreadDemo T1=new
                        ThreadDemo("Thread1");
         T1.start();
         ThreadDemo T2=new
                        ThreadDemo("Thread2");
         T2.start();
     }
}
```

Output:

```
        Creating Thread1
        Starting Thread1
        Creating Thread2
        Starting Thread2
        Running Thread1
        Thread: Thread1: 4
        Running Thread2
        Thread: Thread2: 4
        Thread: Thread1: 3
        Thread: Thread2: 3
        Thread: Thread1: 2
        Thread: Thread2: 2
        Thread: Thread1: 1
        Thread: Thread2: 1
        Thread Thread1 exiting.
        Thread Thread2 exiting.
```

Thread Methods

Thread **Instance Methods:**

1. **public void start()**
   Start the thread in a separate path of execution, and then invokes run() method on same thread object.
2. **public void run()**
   If this thread object was instantiated using separate Runnable Target, the run() method is invoked on that Runnable object.
3. **public void join(long millisec)**
   The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
4. **public final boolean isAlive()**
   Returns true if thread is alive.
5. **public final void setName(String Name) & public final String getName()**
   Set/Get Name of a Thread.

Thread Methods

Thread **Class Methods:**

1. **public static void sleep(long millisec)**
   Causes the currently running thread to block for at least the specified number of millisec.
2. **public static Thread currentThread()**
   Return a reference to the currently running thread, which has invoked this method.
3. **public static void dumpStack()**
   Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

Synchronization

- Synchronization
  - Controls the access of multiple threads to any shared resource.

  Why use Synchronization?
  - To prevent thread interference
  - To prevent consistency problem
  - Two ways to implement it:
    A. By Synchronized Method
    B. By Synchronized Block

Example: (Without Synchronization)

```
Class Table{
        void printTable(int n){
           for(int i=1;i<=5;i++){
                System.out.println(n * i);
                  try{
                      Thread.sleep(400);
                       }catch(Exception e){System.out.println(e);}
                 }
        }
}
Class MyThread1 extends Thread{
        Table t;
        MyThread1(Table t){
                this.t=t;
        }
        public void run(){
                t.printTable(5);
        }
}
Class MyThread2 extends Thread{
        Table t;
        MyThread2(Table t){
                this.t=t;
        }
        public void run(){  t.printTable(100);  }
}
```

```
class Use{
      public static void main(String args[]){
            Table obj = new Table();
            MyThread1 t1=
                        new MyThread1(obj);
            MyThread2 t2=
                        new MyThread2(obj);
            t1.start();
            t2.start();
       }
}
```

Output: 5
100
10
200
15
300
20
400
25
500

Solution1: (By Synchronization Method)

a. by synchronized method

- Declare a method with 'synchronized' keyword
- It Locks an object for any shared resource.

Ex.

```
Class Table{
        synchronized void printTable(int n){
                for(int i=1;i<=5;i++){
                    System.out.println(n*i);
                    try{
                            Thread.sleep(400);
                       }catch(Exception e){System.out.println(e);}
                }
        }
}
```

Output:

```
5
10
15
20
25
100
200
300
400
500
```

Solution2: (By Synchronization Block)

b. by synchronized block

- Need to synchronize a block of statements instead of complete method.
- Scope of synchronized block is smaller than the method.

Syntax:

```
synchronized (object reference expression) {
        //code block
}
```

Ex.

```
class Table{
    void printTable(int n){
        synchronized(this){
            for(int i=1;i<=5;i++){
                System.out.println(n*i);
                try{
                    Thread.sleep(400);
                }catch(Exception e){System.out.println(e);}
            }
        }
    }
}
```

Output:

```
5
10
15
20
25
100
200
300
400
500
```

Current Plan

- We have planned to **Implement** Threading Only,

  When Tasks -

  - Takes more than 2 hours to response. **&**
  - Running as a Cron (Scheduled)

    Or

    Independent Execution from Business Logic (Run.exec())

- **Do Not Implement**

  When Tasks -

  - Is providing Report as a Response even it takes more time

- **Threads**

  - Initially starts with **min 3** threads. Should be increased **upto 10**.
  - No. of threads should be configurable at runtime.

Question?

Thank you