Creating Threads in Java

↳ By Extending Thread class

1) create a class by extending thread class
2) Provide definition for the thread by overriding run() method
3) create object for the class which extends the thread class
4) call the overrided run method (of thread class) using start() method

```
import java.lang.*;
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0; i<10; i++)
        {
            S.O.P("child Thread");
        }
    }
}
class Test
{
    public static void main(String args[])
    {
        MyThread t = new MyThread();
        t.start();
        for(int i=0; i<10; i++)
        {
            S.O.P("Main Thread");
        }
    }
}
```

o/p

child Thread } 10 times
:
Main Thread } 10 times
:

↳ By implementing Runnable Interface

1) create a class by implementing runnable interface
2) provide definition for thread by implementing run() method
3) create object for the class which implements Runnable interface
4) create an object for Thread class & pass runnable interface object as the argument
5) call the start method then the run method will be executed

```
import java.lang.*;
class MyThread implements Runnable
{
    public void run()   // abstract method in Runnable
    {
        for(int i=0; i<10; i++)
        {
            S.O.P("child Thread");
        }
    }
}
class Test
{
    public static void main(String args[])
    {
        MyThread t1 = new MyThread();
        Thread t2 = new Thread(t1);
        t2.start();   // start() internally calls run() of the implementer
        for(int i=0; i<10; i++)
        {
            // invoking start() create new Thread that executes the code written in run()
            S.O.P("Main Thread");
        }
    }
}
```

o/p

```
child Thread
child Thread
Main Thread
Main Thread
child Thread
:
```

creating multiple Threads by extending 'Thread' class

(Thread scheduler of JVM decide the order of execution of the thread)

① creating multiple Threads using multiple classes which extends Thread class
(for each thread we are creating seperate class)

3 Threads in the pgm
↓
t_1: multiplying the numbers 1 to 5 by −1
t_2: multiplying the numbers 1 to 5 by 2
t_3: printing first 5 odd numbers.

Multiple Thread Class Demo

```
class ThreadA extends Thread
{
  public void run()
  {
    for(int i=1; i<=5; i++)
    {
      System.out.println("ThreadA : "+(-1*i));
    }
  }
}
class ThreadB extends Thread
{
  public void run()
  {
    for(int j=1; j<=5; j++)
    {
      System.out.println("ThreadB : "+(2*j));
    }
  }
}
class ThreadC extends Thread
{
  public void run()
  {
    for(int k=1; k<=5; k++)
    {
      System.out.println("ThreadC : "+((k*2)-1));
    }
  }
}
class MultipleThreadClassDemo
{
  public static void main(String args[])
  {
    ThreadA t1 = new ThreadA();
    ThreadB t2 = new ThreadB();
    ThreadC t3 = new ThreadC();
    t1.start();
    t2.start();
    t3.start();
  }
}
```

o/p

```
ThreadA : -1
ThreadA : -2
ThreadB : 2
ThreadC : 1
ThreadC : 3
```

k
1×2 = 2 = 1
2×2 = 4 = 3
3×2 = 6 = 5

② creating multiple threads using single class that extends thread class

MultipleThreadDemo.java

```
class MyThread extends Thread
{
    String name;
    MyThread(String n)
    {
        name=n;
    }
    public void run()
    {
        for (int i=1; i<=5; i++)
        {
            System.out.println("name:"+i);
        }
    }
}
class MultipleThreadDemo
{
    public static void main(String args[])
    {
        MyThread t1=new MyThread("ThreadA");
        MyThread t2=new MyThread("ThreadB");
        t1.start();
        t2.start();
    }
}
```

o/p (vary on each time)

```
ThreadA : 1
ThreadB : 1
ThreadA : 2
ThreadB : 2
1
2
```

Creating multiple Threads using by Implementing Runnable Interface

① Create Multiple Threads using multiple classes that implements Runnable Interface

```
class ThreadA implements Runnable
{
  public void run()
   {
     for (int i=1; i<=5; i++)
       { System.out.println("ThreadA :"+(-1*i));
   }
   System.out.println("Exiting ThreadA");
  }
}
class ThreadB implements Runnable.
{ public void run()
   {
     for (int i=1, j<=5; j++)
       {
         System.out.println("ThreadB :" + (2*j));
   }
   System.out.println("Exiting ThreadB");
}
class MultipleThreads
{ public static void main(String args[])
   {
     ThreadA obj1 = new ThreadA();
     Thread t1 = new Thread(obj1);
     ThreadB obj2 = new ThreadB();
     Thread t2 = new Thread(obj2);
     t1.start();
     t2.start();
   }
}
```

op

```
ThreadA = -1
ThreadB = 2
ThreadA = -2
ThreadB = 3
ThreadB = 4
ThreadB = 6
ThreadB = 8
ThreadB = 10
Exiting ThreadB
ThreadA = -4
ThreadA = -5
Exiting ThreadA
```

② creating multiple thread using single class that implements Runnable Interface

```
class MyThread implements Runable
{
    String name;
    MyThread(String n)
    {
        name = n;
    }
    public void run()
    {
        for(i=1; i<=5; i++)
        { System.out.println(name+":"+i);
        }
    }
}
class MultipleThreadDemo
{
    public static void main(String args[])
    {
        MyThread obj1 = new MyThread("ThreadA");
        Thread t1 = new Thread(obj1);
        MyThread obj2 = new MyThread("ThreadB");
        Thread t2 = new Thread(obj2);
        t1.start();
        t2.start();
    }
}
```

o/p

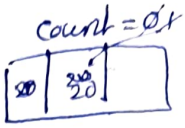
```
ThreadA:1
ThreadA:2
ThreadB:1
ThreadA:3
ThreadB:2
```

Thread Synchronization in Java

① using synchronization method

```
class Buffer
{
  static int count=7;
  Synchronized void computeItemCount(String name)
  {
    if(name=="produces")
    {
      int s1 = count;
      s1 = s1+1;
      count = s1;
      System.out.println("Produces: count="+count);
    }
    else
    {
      int s2 = count;
      s2 = s2-1;
      count = s2;
      System.out.println("consumes: count="+count);
    }
  }
}
class MyThreadd implements Runnable
{
  Buffer buffer;
  String name;
  MyThreadd(Buffer buffer, String name)
  {
    this.buffer = buffer;
    this.name = name;
  }
  public void run()
  {
    buffer.computeItemCount(this.name);
  }
}
```

Produces Thread — Consumes Thread



Shared Buffer

executes Threads in concurrent manner

suppose order

P_1 · 1

P_2 · 2

C_1

C_2

$P_1 \rightarrow R_1 = 7$

$P_2 \rightarrow R_1 = 8$

$C_1 \rightarrow R_2 = 7$

$C_2 \rightarrow R_2 = 6$

$P_3 \rightarrow count = 7$

$C_3 \rightarrow count = 6$

after

initial items in buffer

count = 7 8 7

Produces

1) $R_1 = count$
2) $R_1 = R_1 + 1$
3) $count = R_1$

Internally perform operation using registers

Consumes

1) $R_2 = count$
2) $R_2 = R_2 - 1$
3) $count = R_2$

cant do synchronization on predefined method

run() is predefined method, variable, class

only on user defined method, block

i) Synchronize

ii) Synchronize

inst

P

data Inconsistency / Race condition

(since multiple threads accessing same shared Variable/data)

when multiple threads accessing same shared data then 1 thread is accessing shared data then we should not allow other thread to access the shared data

eg: while produces thread accessing shared buffer then not allow consumes thread to access the shared buffer & vice versa.

Then can solve Inconsistency prblm

```
public class ThreadSynchronizationDemo
{
    public static void main(String args[])
    {
        Buffer buffer = new Buffer();

        MyThreadd produces = new MyThreadd(buffer, "produces");
        Thread t1 = new Thread(produces);

        MyThreadd consumes = new MyThreadd(buffer, "consumes");
        Thread t2 = new Thread(consumes);

        t1.start();
        t2.start();
    }
}
```

o/p1

Produces : count = 8

Consumes : count = 7

o/p2

Consumes : count = 6

Produces : count = 7

, class

method, block.

i) synchronized method — lock applied on the method. lock 1 – locked, lock 0 – unlocked. when 1 thread accessing it, none other can access it.

ii) Synchronized block.

instead of performing synchronization on entire method, performing on block

② Using synchronized block / Synchronized statement

```
class Buffer
{
    static int count=7;
    void computeItemCount(String name)
    {
        if(name == "produces")
        {
            int s1 = count;
            s1 = s1+1;
            count = s1;
            System.out.println("produces : count="+count);
        }
        else
        {
            int s2 = count;
            s2 = s2-1;
            count = s2;
            System.out.println("consumes : count=" +count);
        }
    }
}
class MyThreadd implements Runnable
{
    Buffer buffer;
    String name;
    MyThreadd(Buffer buffer, String name)
    {
        this.buffer = buffer;
        this.name = name;
    }
    public void run()
    {
        Synchronized(this)
        {
            buffer.computeItemCount(this.name);
        }
    }
}
```

```
public class ThreadSynchronizationDemo
{
    public static void main(String args[])
    {
        Buffer buffer = new Buffer();
        MyThreadd produces = new MyThread
                (buffer, "produce");
        Thread t1 = new Thread(produces);
        MyThreadd consumes = new MyThreadd
                (buffer, "consumes");
        Thread t2 = new Thread(consumes);

        t1.start();
        t2.start();
    }
```

o/p1

produces : count = 8
consumes : count = 7

o/p2

consumes : count=6
produces : count=7

```
class Buffer
{
   static int count=7;
   void computeItemCount(String name)
   {
      Synchronized(this)
      {
         if(name == "produces")
         {  int s1 = count;
            s1 = s1+1;
            count = s1;
            System.out.println("produces: count=" + count);
         }
         else
         {  int s2= count;
            s2 = s2-1;
            count = s2;
            System.out.println("consumes: count=" + count);
         }
      }
   }
}
class MyThreadd implements Runnable
{ Buffer buffer;
  String name;
  MyThreadd(Buffer buffer, String name)
  { this.buffer = buffer;
    this.name = name;
  }
  public void run()
  { buffer.computeItemCount(this.name);
  }
}
```

```
public class ThreadSynchronizationDemo
{
 public static void main(String args[])
 {
 Buffer buffer = new Buffer();
 MyThreadd produce = new MyThread
        (buffer, "produces");
 Thread t1 = new Thread(produce);
 MyThreadd consumes = new MyThreadd
        (buffer, "consumes");

   t1.start();
   t2.start();
 }
```

o/p1

Produces: count = 8

Consumes: count = 7

o/p2

consumes: count = 6

Produce: count = 7