# MULTI THREADING

## MultiThreading

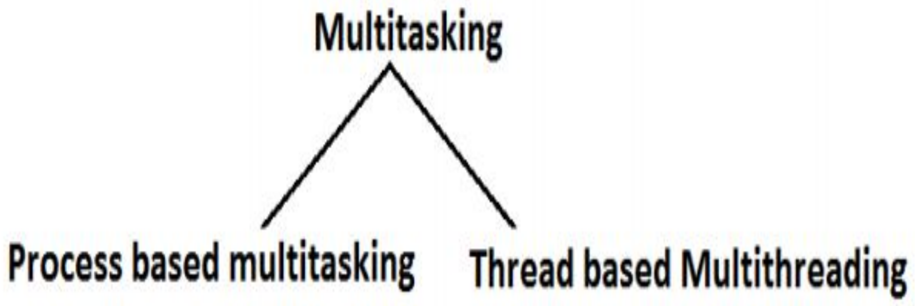
1) Introduction.
2) The ways to define instantiate and start a new Thread.
3) Getting and setting name of a Thread.
4) Thread priorities.

## Multitasking

**Multitasking:** Executing several tasks simultaneously is the concept of multitasking. There are two types of multitasking's.

1) **Process based multitasking.**
2) **Thread based multitasking.**

**Diagram:**



## Example

Example:

- While typing a java program in the editor we can able to listen mp3 audio songs at the same time we can download a file from the net all these tasks are independent of each other and executing simultaneously and hence it is Process based multitasking.

- This type of multitasking is best suitable at “os level”.

## process based multitasking

- Executing several tasks simultaneously where each task is a separate independent process such type of multitasking is called process based multitasking

## Thread based multitasking

- **Thread based multitasking**: 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 a “Thread”.
- This type of multitasking is best suitable for “programatic level”.

## Thread Based Multitasking

- When compared with “C++”, developing multithreading examples is very easy in java because java provides in built support for multithreading through a rich API (Thread, Runnable….etc).
- In multithreading on 10% of the work the programmer is required to do and 90% of the work will be down by java API.

## Thread Based Multitasking

- The main important application areas of multithreading are:

  1) To implement multimedia graphics.

  2) To develop animations.

  3) To develop video games etc.

  4)To  Develop Web and application server
- Whether it is process based or Thread based the main objective of multitasking is to improve performance of the system by reducing response time and to improve performance

## The ways to define instantiate and start a new Thread:

- We can define a Thread in the following 2 ways.

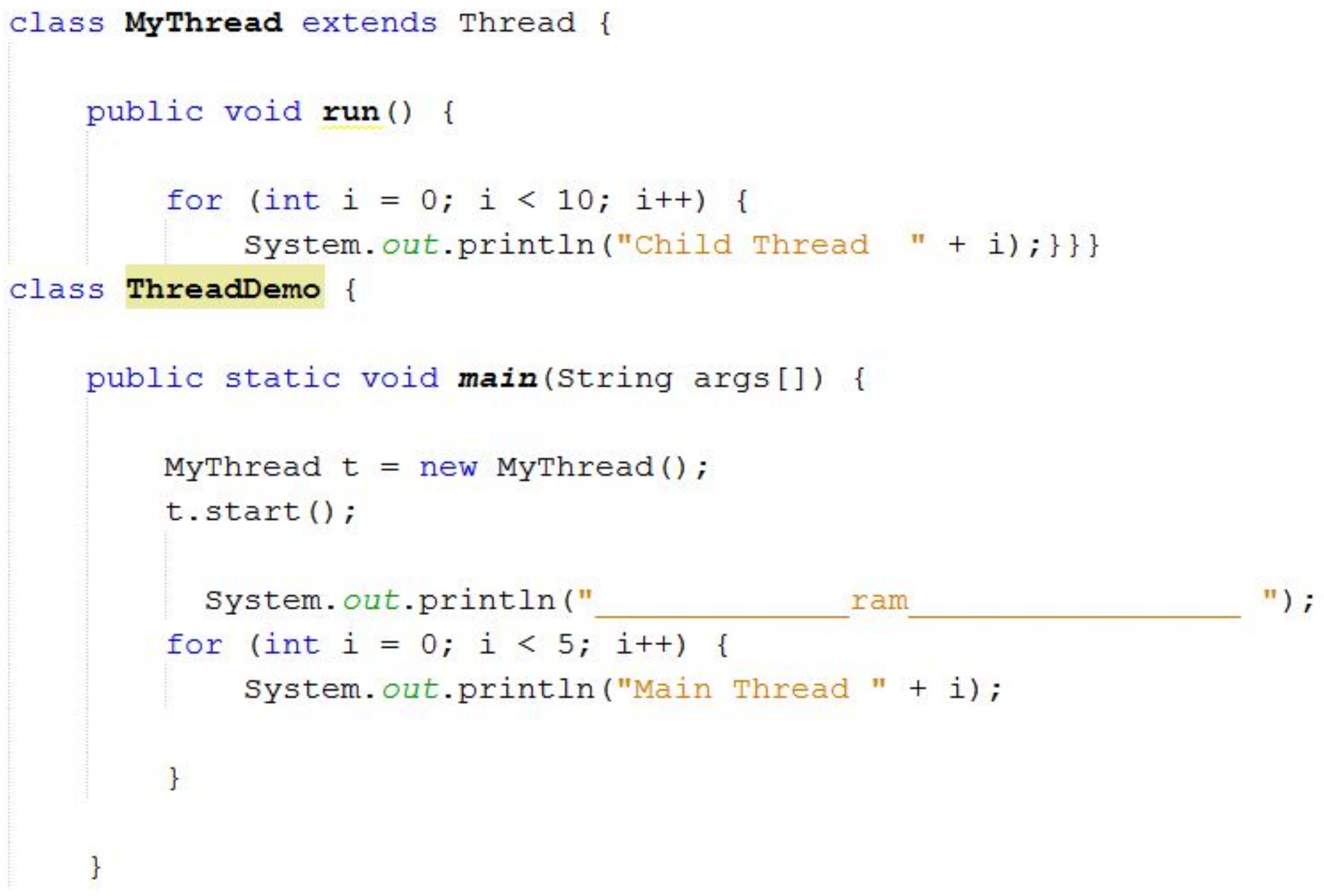
1. By extending Thread class.
2. By implementing Runnable interface.

## Example

- Defining a Thread by extending “Thread class”:

```
Class Test Extends Thread{
//override Run Method


}
```

# Example



```
class MyThread extends Thread {

    public void run() {

        for (int i = 0; i < 10; i++) {
            System.out.println("Child Thread  " + i);}}}
class ThreadDemo {

    public static void main(String args[]) {

        MyThread t = new MyThread();
        t.start();

          System.out.println("_____________ram_________________ ");
        for (int i = 0; i < 5; i++) {
            System.out.println("Main Thread " + i);

        }

    }
```

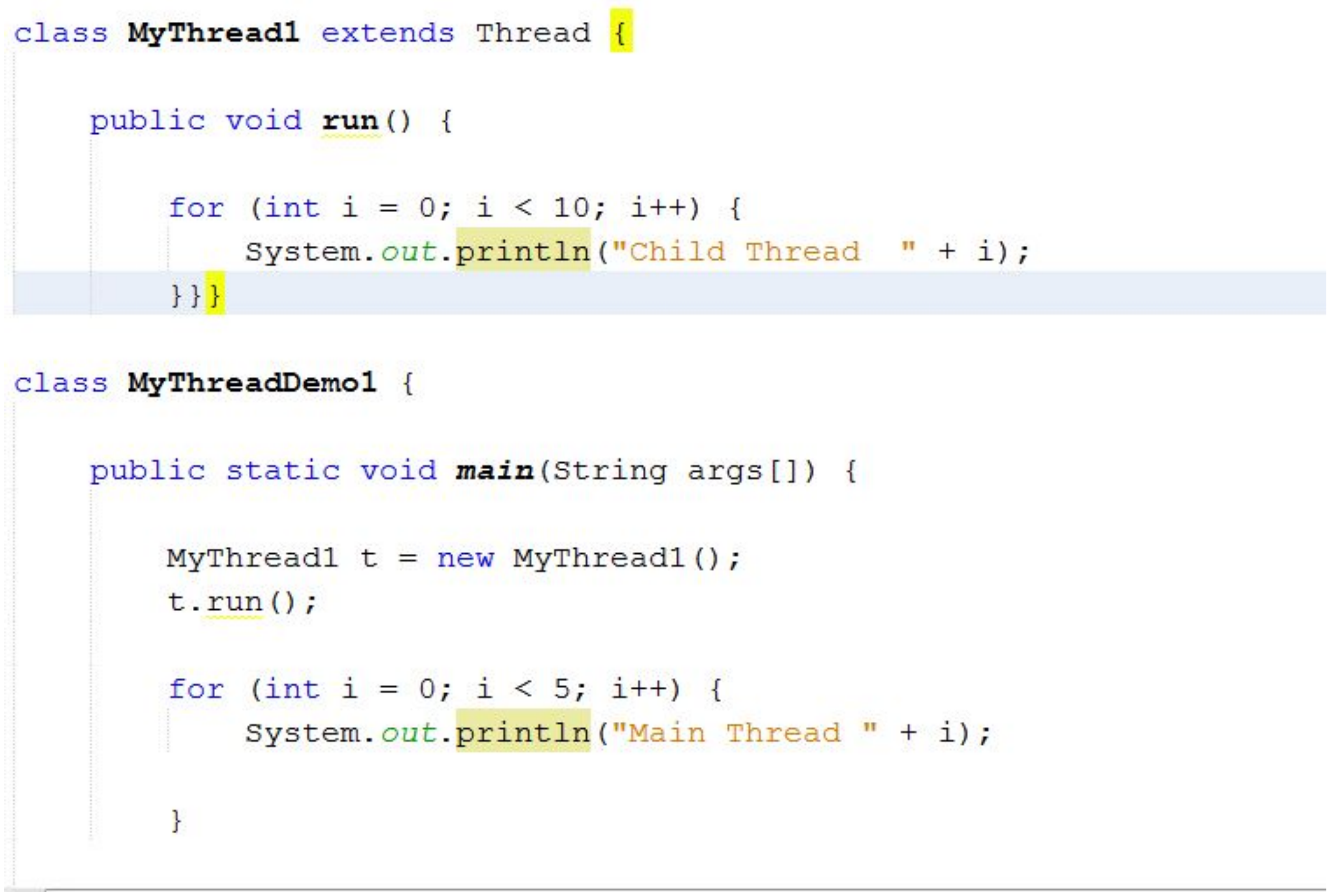
## Case 1: Thread Scheduler

- If multiple Threads are waiting to execute then which Thread will execute 1st is decided by “Thread Scheduler” which is part of JVM.

## Difference between t.start() and t.run() methods.

- In the case of t.start() a new Thread will be created which is responsible for the execution of run() method. But in the case of t.run() no new Thread will be created and run() method will be executed just like a normal method by the main Thread. In the above program if we are replacing t.start() with t.run() the following is the output.

# Example



```java
class MyThread1 extends Thread {

    public void run() {

        for (int i = 0; i < 10; i++) {
            System.out.println("Child Thread  " + i);
        }}}

class MyThreadDemo1 {

    public static void main(String args[]) {

        MyThread1 t = new MyThread1();
        t.run();

        for (int i = 0; i < 5; i++) {
            System.out.println("Main Thread " + i);

        }
```

## importance of Thread class start() method.

- For every Thread the required mandatory activities like registering the Thread with Thread Scheduler will takes care by Thread class start() method and programmer is responsible just to define the job of the Thread inside run() method. That is start() method acts as best assistant to the programmer.

## Example

start() {

1. Register Thread with Thread Scheduler

2. All other mandatory low level activities.

3. Invoke or calling run() method.

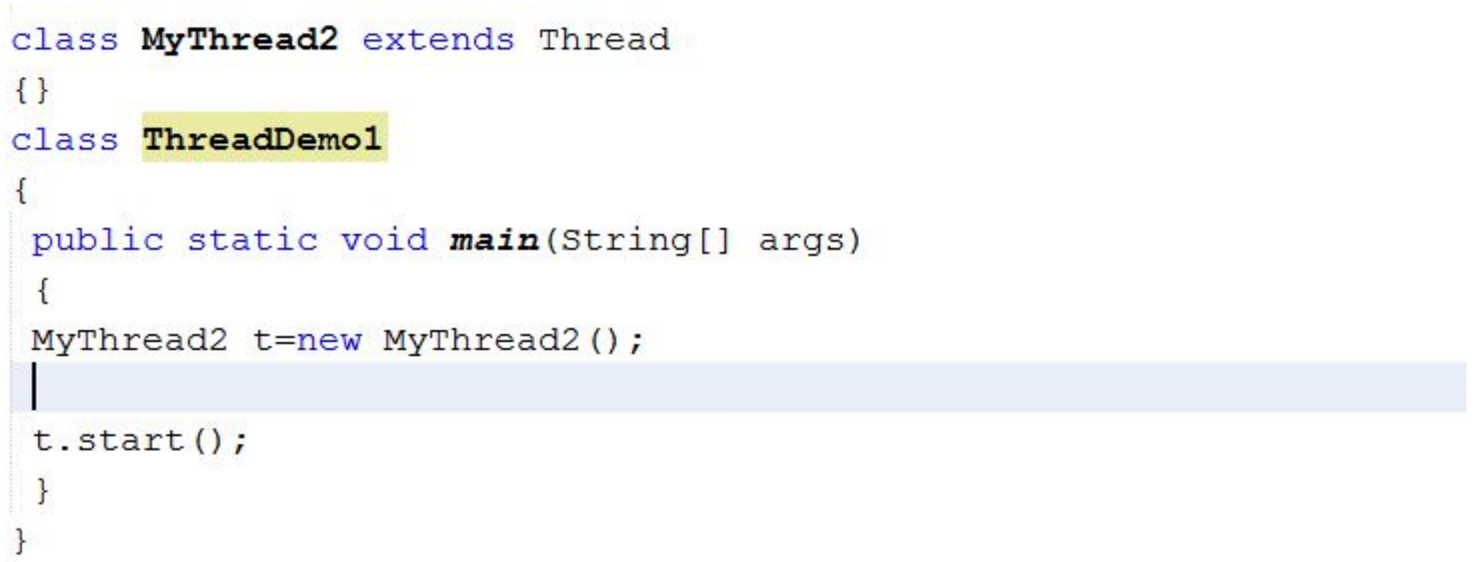
}

- We can conclude that without executing Thread class start() method there is no chance of starting a new Thread in java.

## If we are not overriding run() method:

- If we are not overriding run() method then Thread class run() method will be executed which has empty implementation and hence we won’t get any output.

# Example



```
class MyThread2 extends Thread
{}
class ThreadDemo1
{
 public static void main(String[] args)
 {
 MyThread2 t=new MyThread2();

 t.start();
 }
}
```

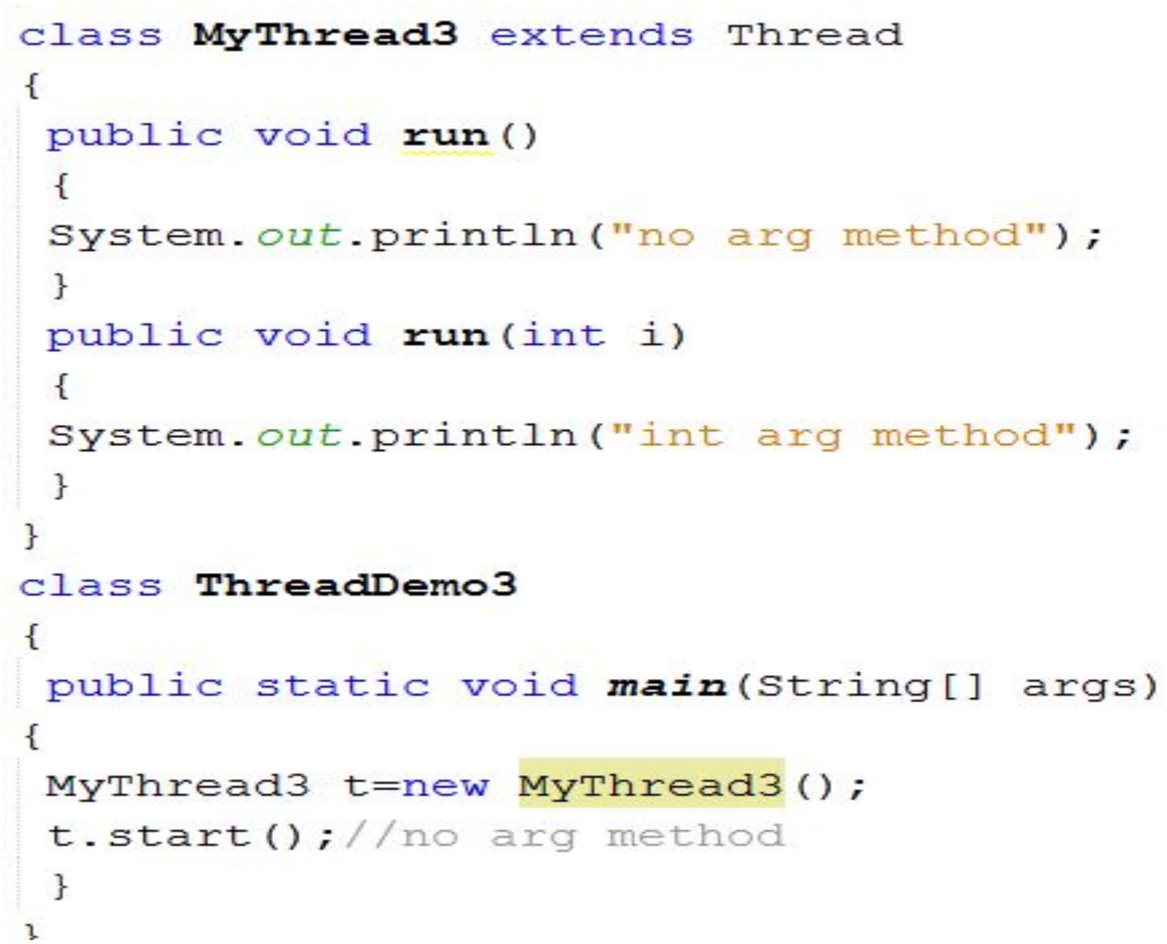
## conclusion

- It is highly recommended to override run() method. Otherwise don’t go for multithreading concept

## Overriding of run() method

- We can overload run() method but Thread class start() method always invokes no argument run() method the other overload run() methods we have to call explicitly then only it will be executed just like normal method.

# Example

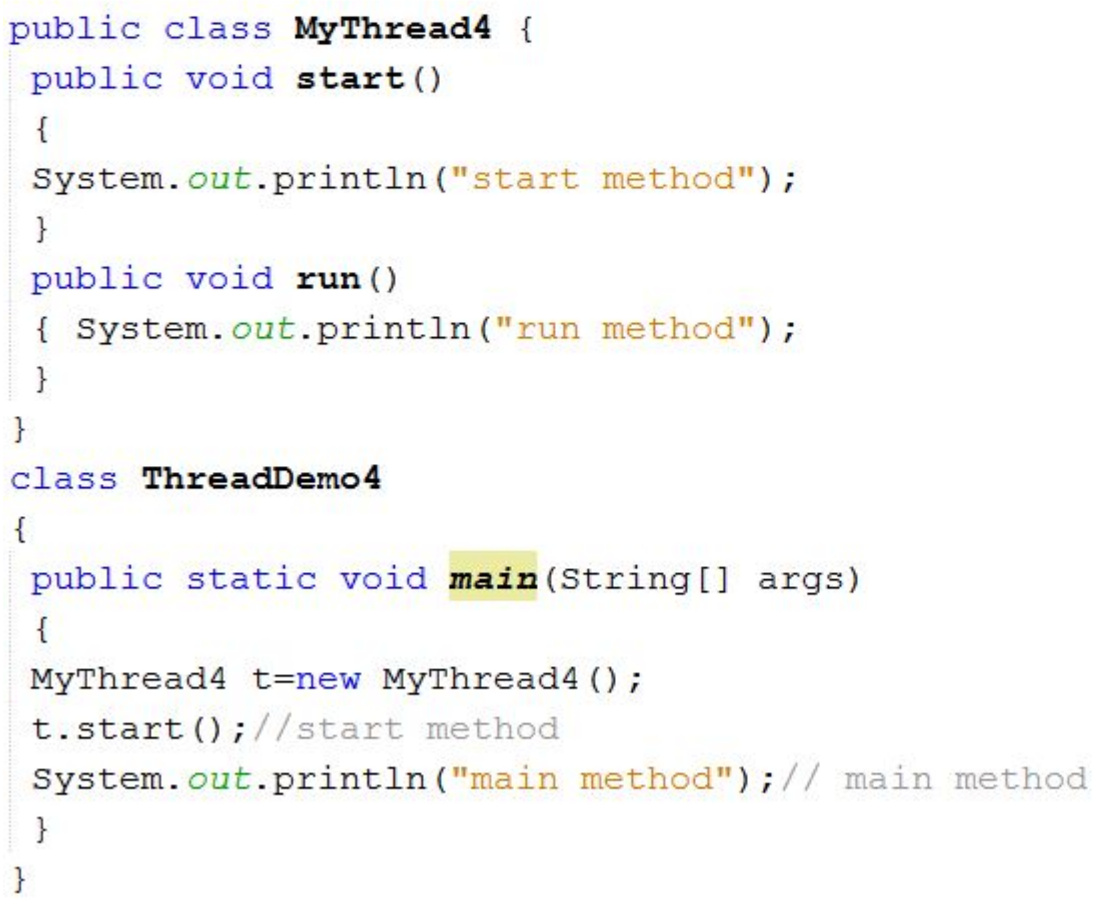


```
class MyThread3 extends Thread
{
 public void run()
 {
 System.out.println("no arg method");
 }
 public void run(int i)
 {
 System.out.println("int arg method");
 }
}
class ThreadDemo3
{
 public static void main(String[] args)
{
 MyThread3 t=new MyThread3();
 t.start();//no arg method
 }
}
```

## overriding of start() method:

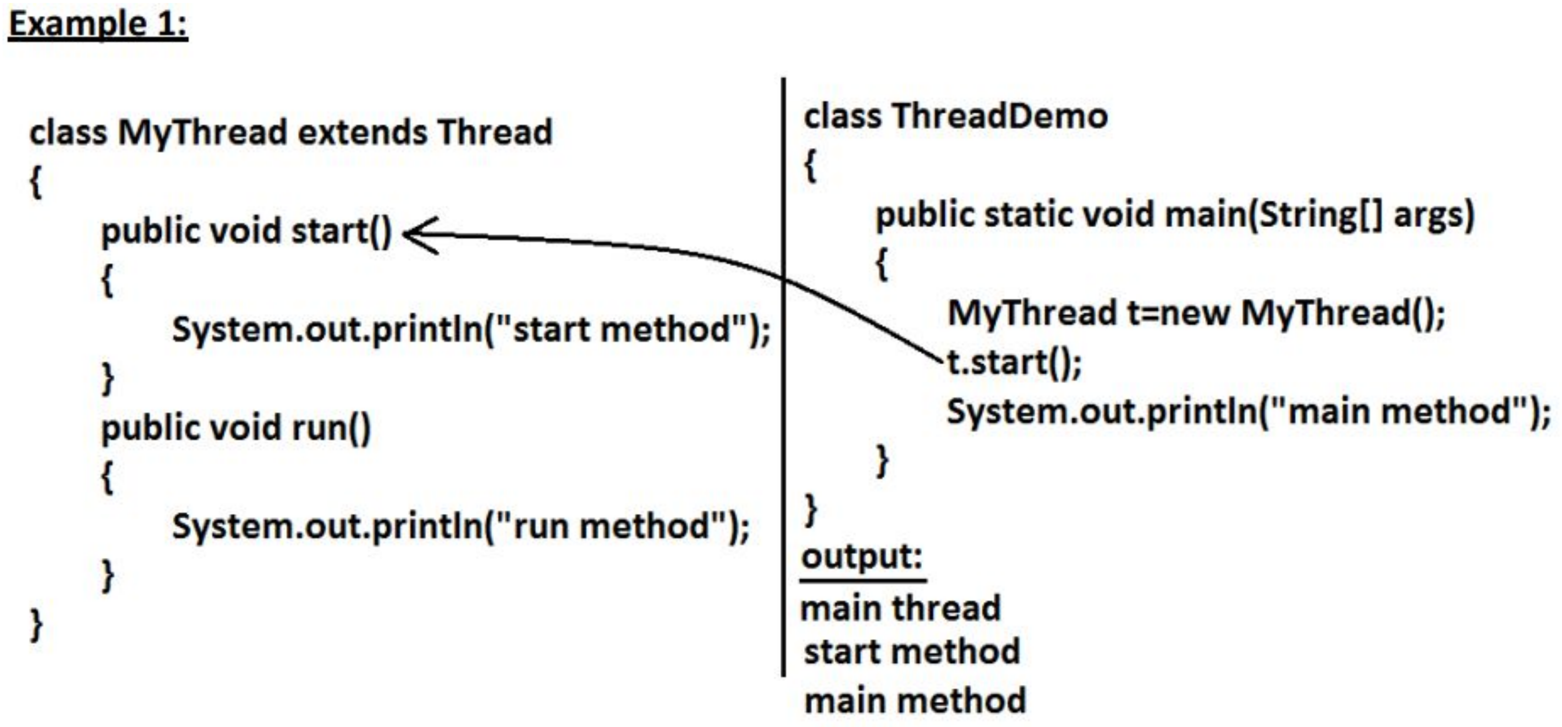
- If we override start() method then our start() method will be executed just like a normal method call and no new Thread will be started.

# Example



```
public class MyThread4 {
 public void start()
 {
 System.out.println("start method");
 }
 public void run()
 { System.out.println("run method");
 }
}
class ThreadDemo4
{
 public static void main(String[] args)
 {
 MyThread4 t=new MyThread4();
 t.start();//start method
 System.out.println("main method");// main method
 }
}
```

## Entire output produced by only main Thread.



**Example 1:**

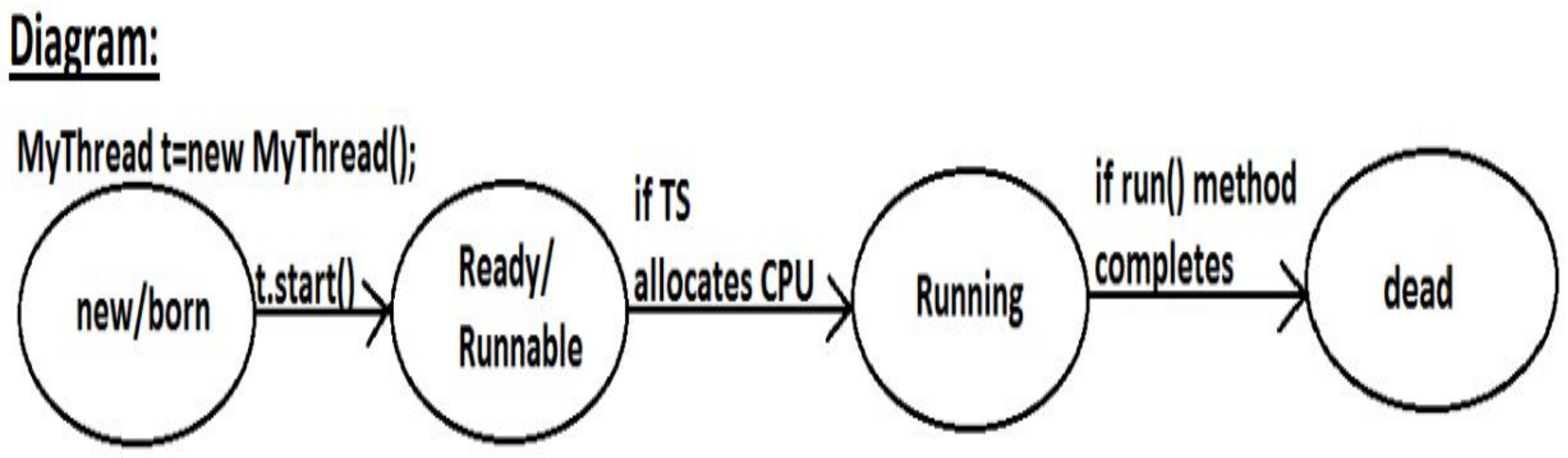
```
class MyThread extends Thread
{
    public void start()
    {
        System.out.println("start method");
    }
    public void run()
    {
        System.out.println("run method");
    }
}
```

```
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        t.start();
        System.out.println("main method");
    }
}
```

**output:**

```
main thread
start method
main method
```

# life cycle of the Thread:



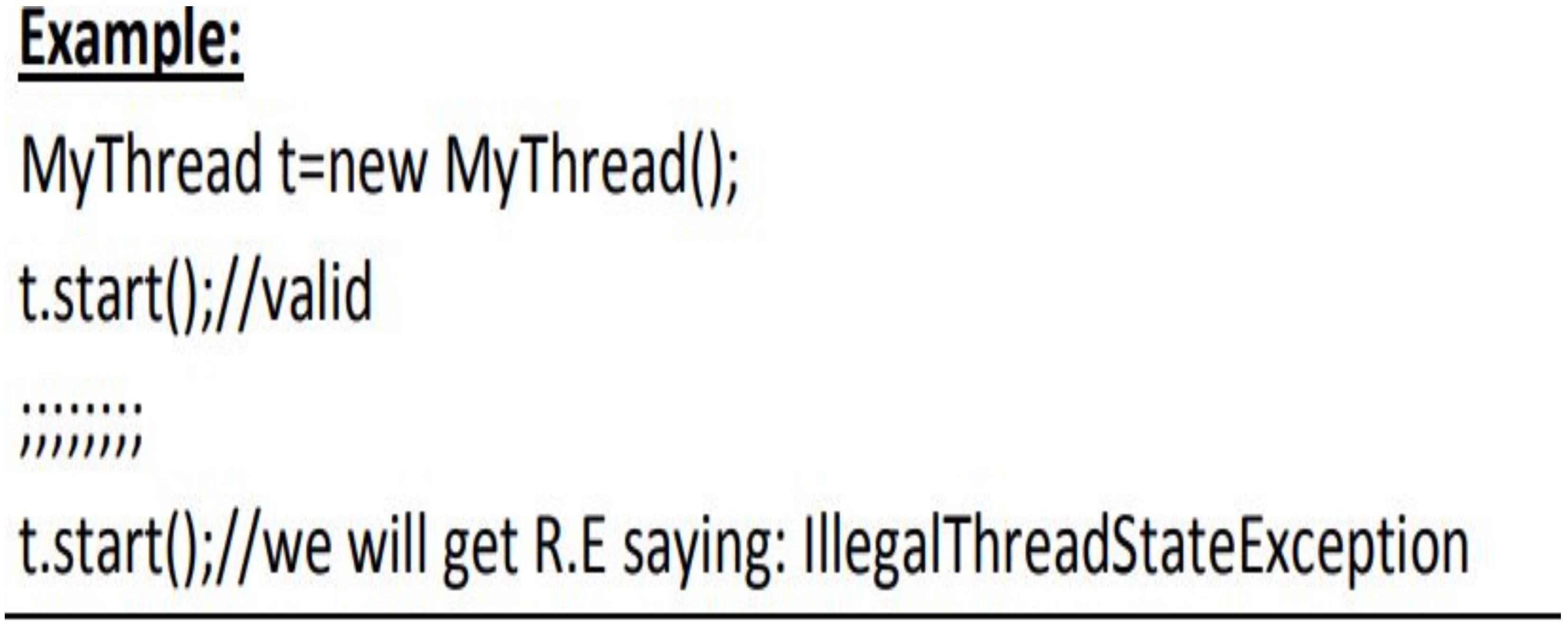
## Life cycle of Thread

- Once we created a Thread object then the Thread is said to be in new state or born state.
- Once we call start() method then the Thread will be entered into Ready or Runnable state.
- If Thread Scheduler allocates CPU then the Thread will be entered into running state.
- Once run() method completes then the Thread will entered into dead state.

## conclusion

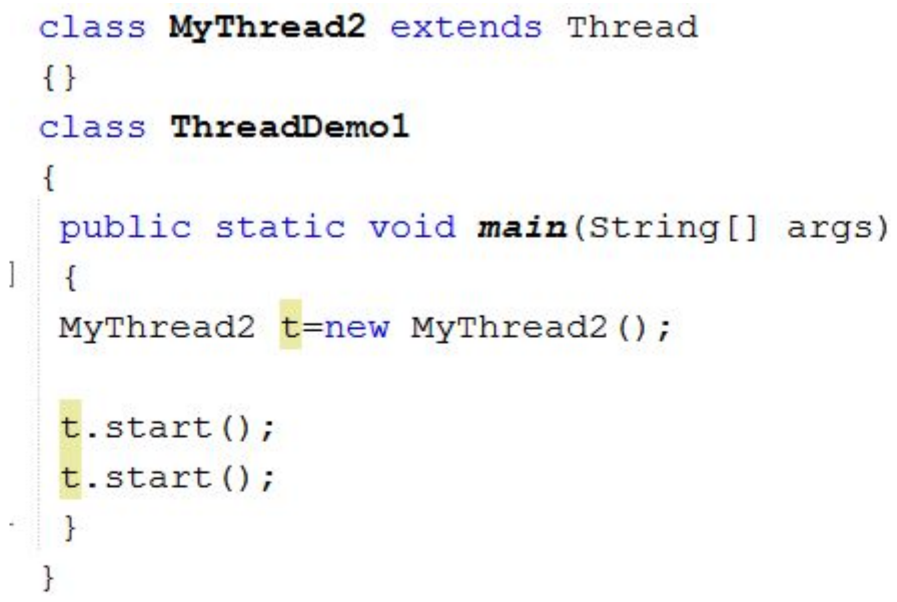
- After starting a Thread we are not allowed to restart the same Thread once again otherwise we will get runtime exception saying "IllegalThreadStateException".

# Example



**Example:**

```
MyThread t=new MyThread();
t.start();//valid
........
,,,,,,,,
t.start();//we will get R.E saying: IllegalThreadStateException
```



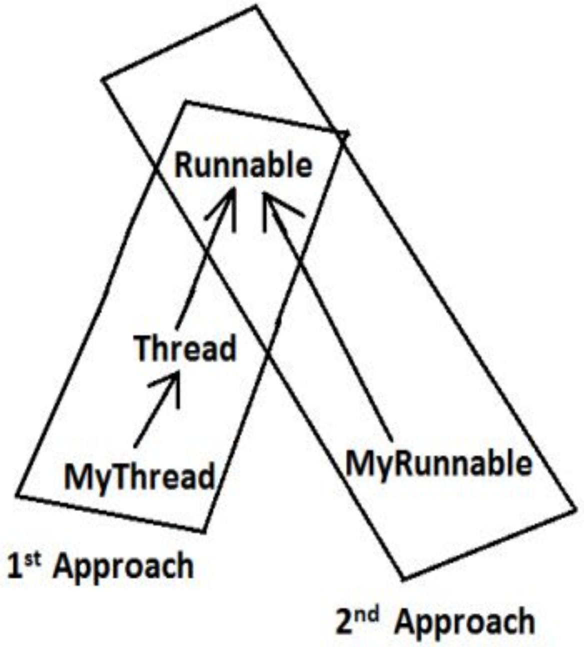
```
class MyThread2 extends Thread
{}
class ThreadDemo1
{
 public static void main(String[] args)
 {
 MyThread2 t=new MyThread2();

 t.start();
 t.start();
 }
}
```

## Defining a Thread by implementing Runnable interface:

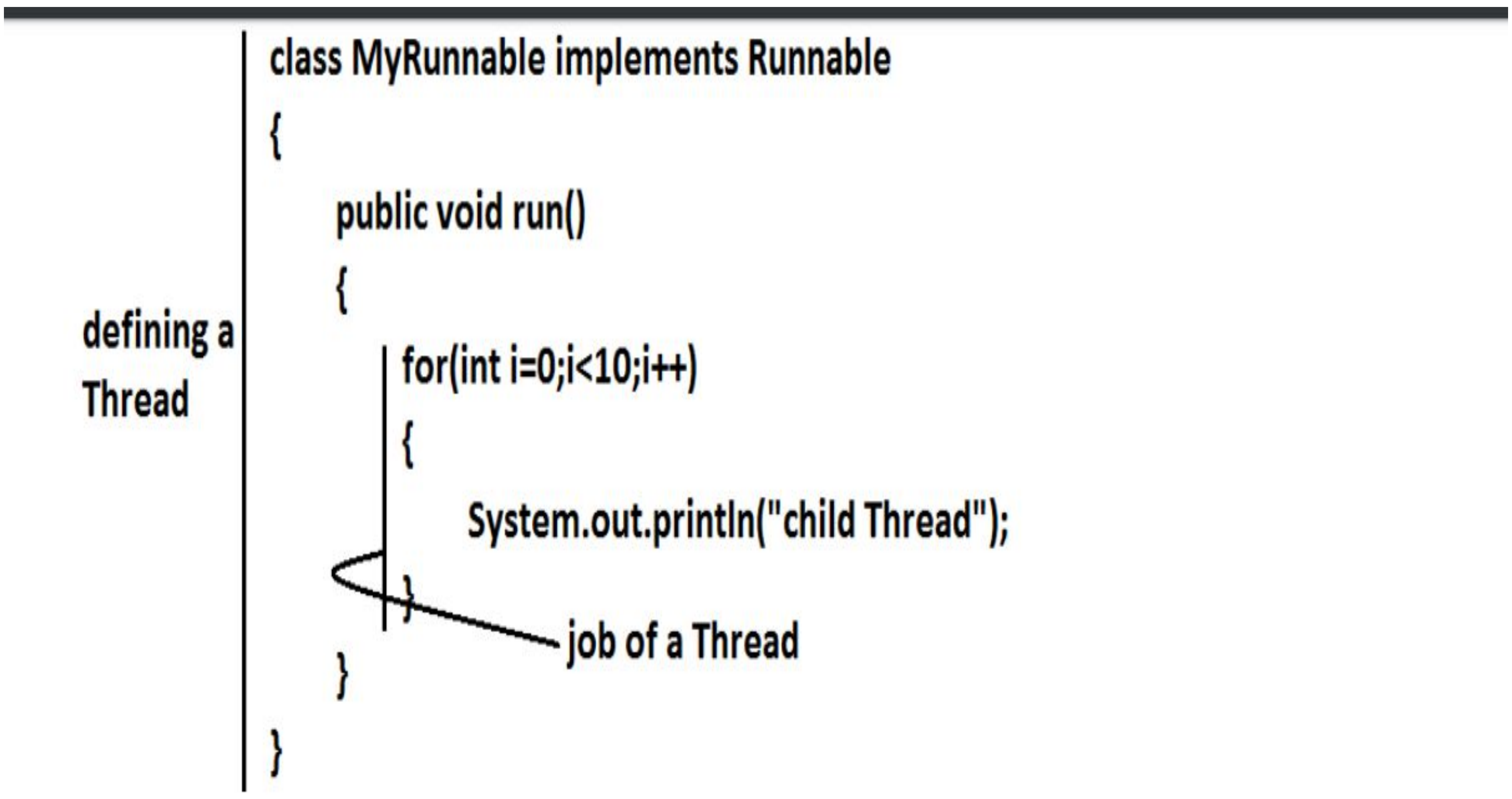
- We can define a Thread even by implementing Runnable interface also. Runnable interface present in java.lang.pkg and contains only one method run().

# Defining a Thread by implementing Runnable interface:

**Diagram:**



# Example



```
class MyRunnable implements Runnable
{
        public void run()
        {
                for(int i=0;i<10;i++)
                {
                        System.out.println("child Thread");
                }
        }
}
```

defining a Thread
job of a Thread

## Example

```
class ThreadDemo
{
        public static void main(String[] args)
        {
                MyRunnable r=new MyRunnable();
                Thread t=new Thread(r);//here r is a Target Runnable
                t.start();
        ⟶
                for(int i=0;i<10;i++)
                {
                        System.out.println("main thread");
                }
        }
```

## conclusion

- We can’t expect exact output but there are several possible outputs.
- **Case study**:

```
MyRunnable r=new MyRunnable();
Thread t1=new Thread();
Thread t2=new Thread(r);
```

## Case 1: t1.start():

- A new Thread will be created which is responsible for the execution of Thread class run()method.

Output:

```
main thread
main thread
 main thread
main thread
main thread
```

## Case 2: t1.run():

- No new Thread will be created but Thread class run() method will be executed just like a normal method call

  Output:

```
main thread
main thread
main thread
main thread
main thread.
```

## Case 3: t2.start():

- New Thread will be created which is responsible for the execution of MyRunnable run() method.
- **Case 4: t2.run():**
- No new Thread will be created and MyRunnable run() method will be executed just like a normal method call.

## Case 5: r.start():

- We will get compile time error saying start()method is not available in MyRunnable class.
- **Case 6: r.run():**
- No new Thread will be created and MyRunnable class run() method will be executed just like a normal method call.

## Best approach to define a Thread:

- Among the 2 ways of defining a Thread, implements Runnable approach is always recommended.
- In the 1st approach our class should always extends Thread class there is no chance of extending any other class hence we are missing the benefits of inheritance.

## Approcah

- But in the 2nd approach while implementing Runnable interface we can extend some other class also. Hence implements Runnable mechanism is recommended to define a Thread.

## Thread class constructors:

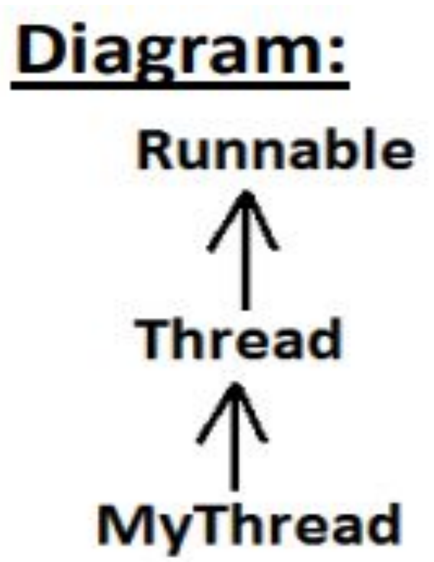
1) Thread t=new Thread();

2) Thread t=new Thread(Runnable r);

3) Thread t=new Thread(String name);

4) Thread t=new Thread(Runnable r,String name);

## Thread



## Getting and setting name of a Thread:

- Every Thread in java has some name it may be provided explicitly by the programmer or automatically generated by JVM.
- Thread class defines the following methods to get and set name of a Thread.

## Methods:

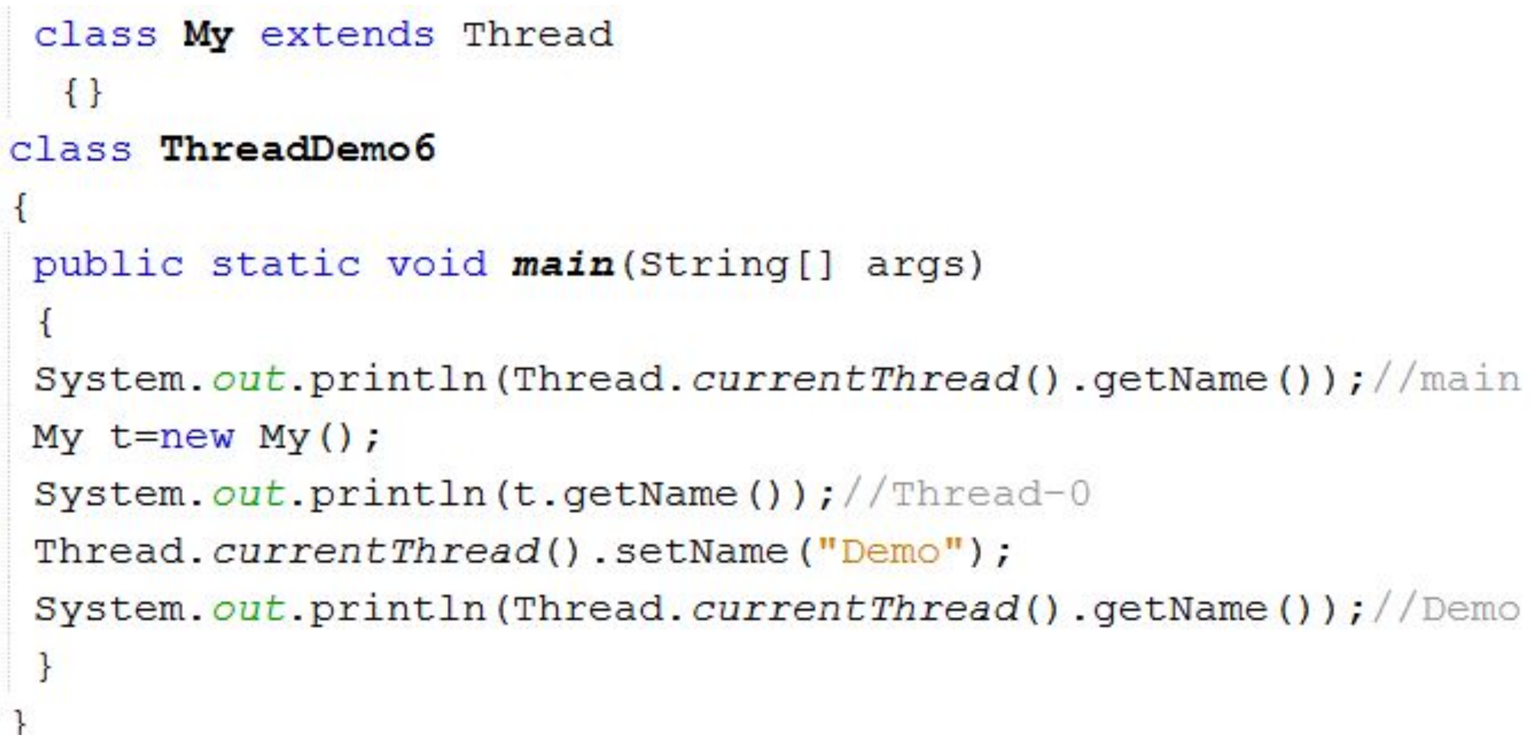
**1) public final String getName()**

**2) public final void setName(String name)**

**Note:-**

We can get current executing Thread object reference by using Thread.currentThread() method.

# Example



```
 class My extends Thread
  {}
class ThreadDemo6
{
 public static void main(String[] args)
 {
 System.out.println(Thread.currentThread().getName());//main
 My t=new My();
 System.out.println(t.getName());//Thread-0
 Thread.currentThread().setName("Demo");
 System.out.println(Thread.currentThread().getName());//Demo
 }
}
```