

# **MULTITHREADING & SYNCHRONIZATION**

## **TUTORIAL 1**

# Process Vs Thread

## ▶ Processes :

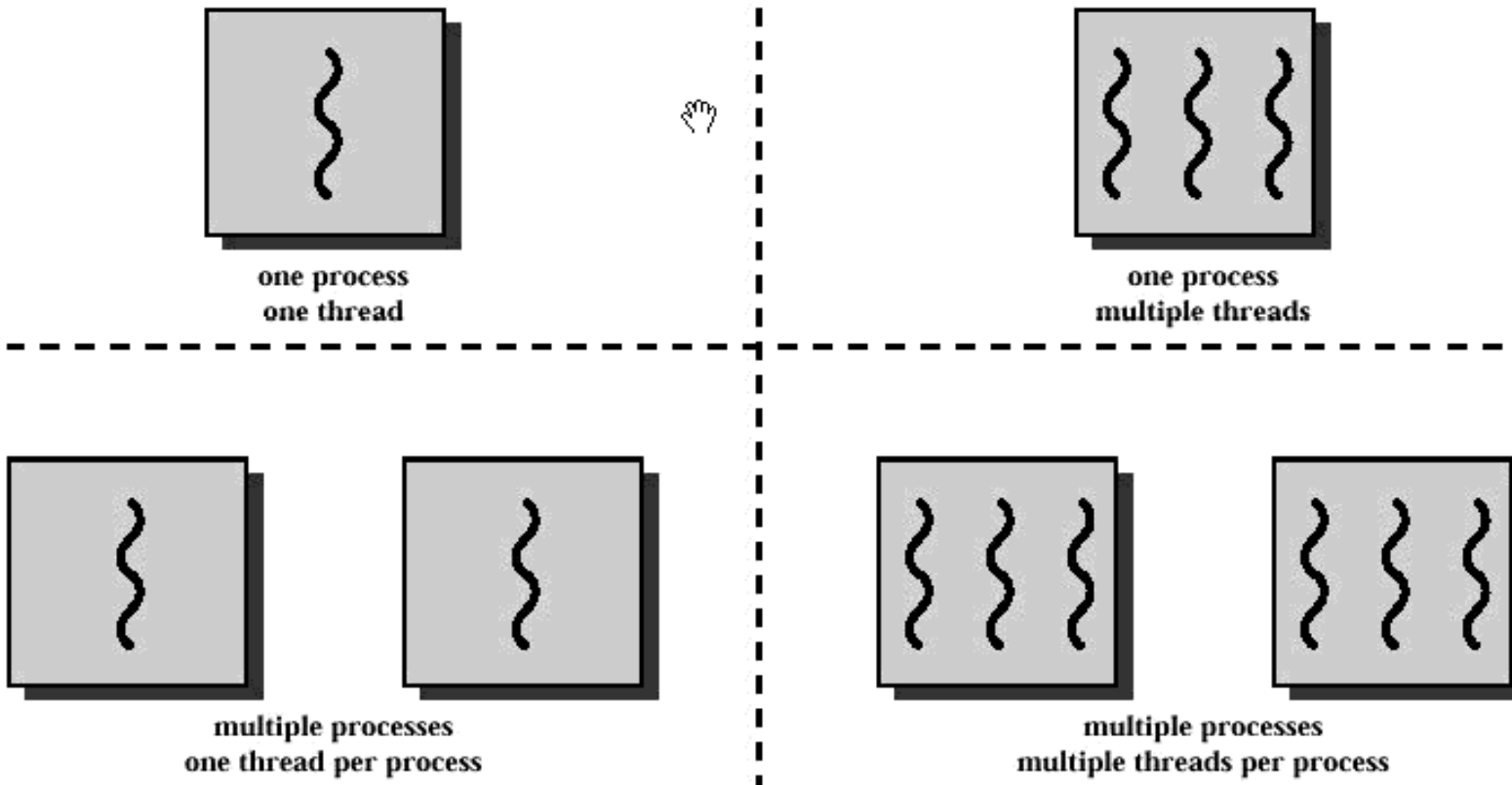
program in execution  
synonymous with Program or Application

## ▶ Threads :

component of process  
it's lightweight processes  
many threads can exist within the same process

# Process Vs Thread (Cont.)

---



# Multi Threading in General

- ▶ Multitasking is sharing of computer resources among number of processes or execution of different processes at the same time like working on Eclipse while chrome is running **whereas**
- ▶ Multithreading is execution of number of different tasks within a same process like in MSWord check your sentence and word while you are writing simultaneously.
- ▶ They are allocated CPU time via context switching in such a way that they appear to be executing in parallel(except when there are multiple cores to deal with them separately)

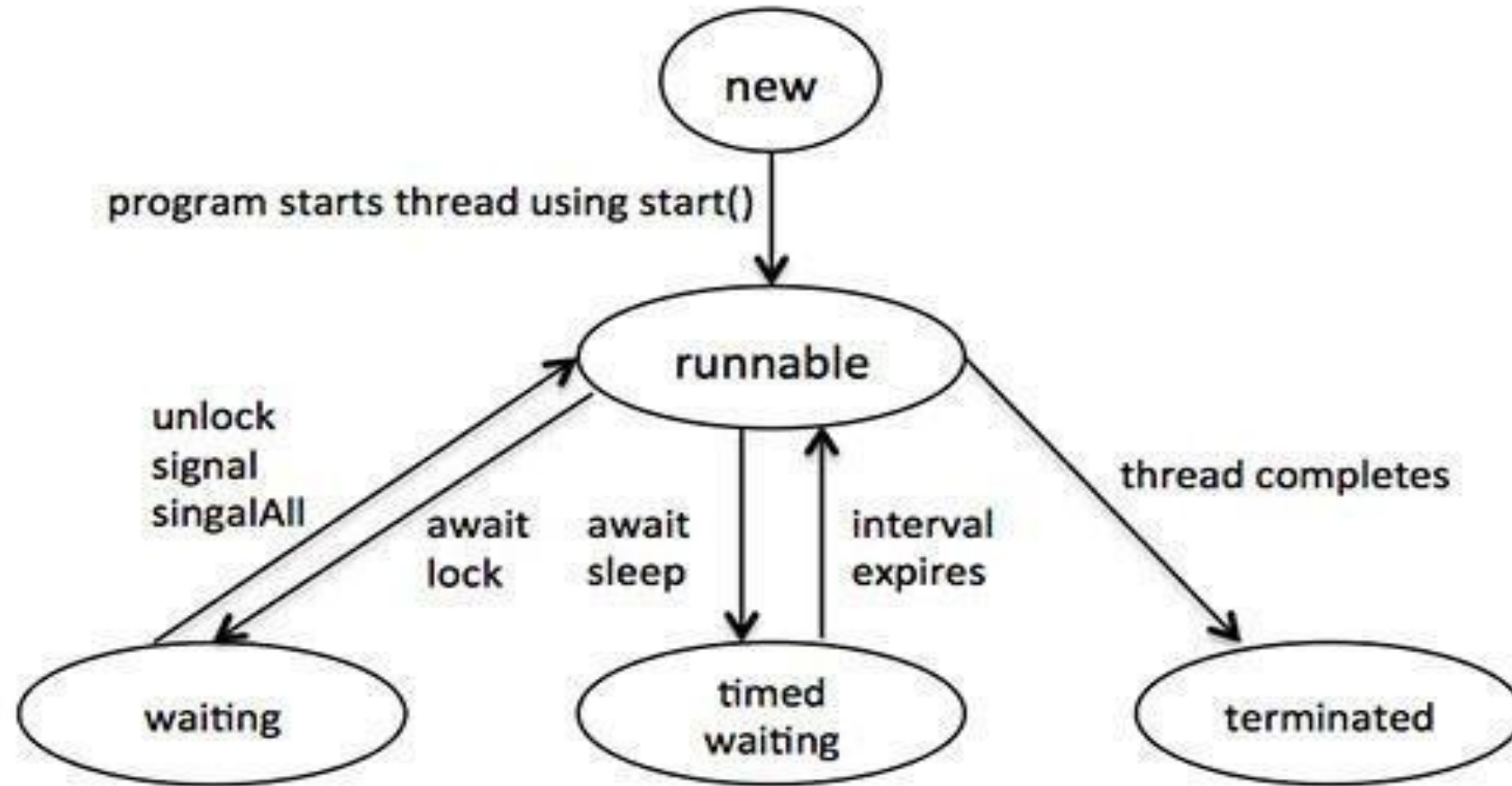
# Advantages of Multithreading

- ▶ Increased performance
- ▶ Better utilization of resources (Ex. multiple core)
- ▶ Handle multiple instance of data simultaneously

# Disadvantages of Multithreading

- ▶ Added complexity to maintain multiple threads
- ▶ Maintaining the shared resources (figure out a way to make sharing between threads as efficient and as safe as possible)

# Thread Life-cycle



# Stages Explained

- ▶ **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread
- ▶ **Runnable:** After a newly born thread is started, the thread becomes runnable.
- ▶ **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time.
- ▶ **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates

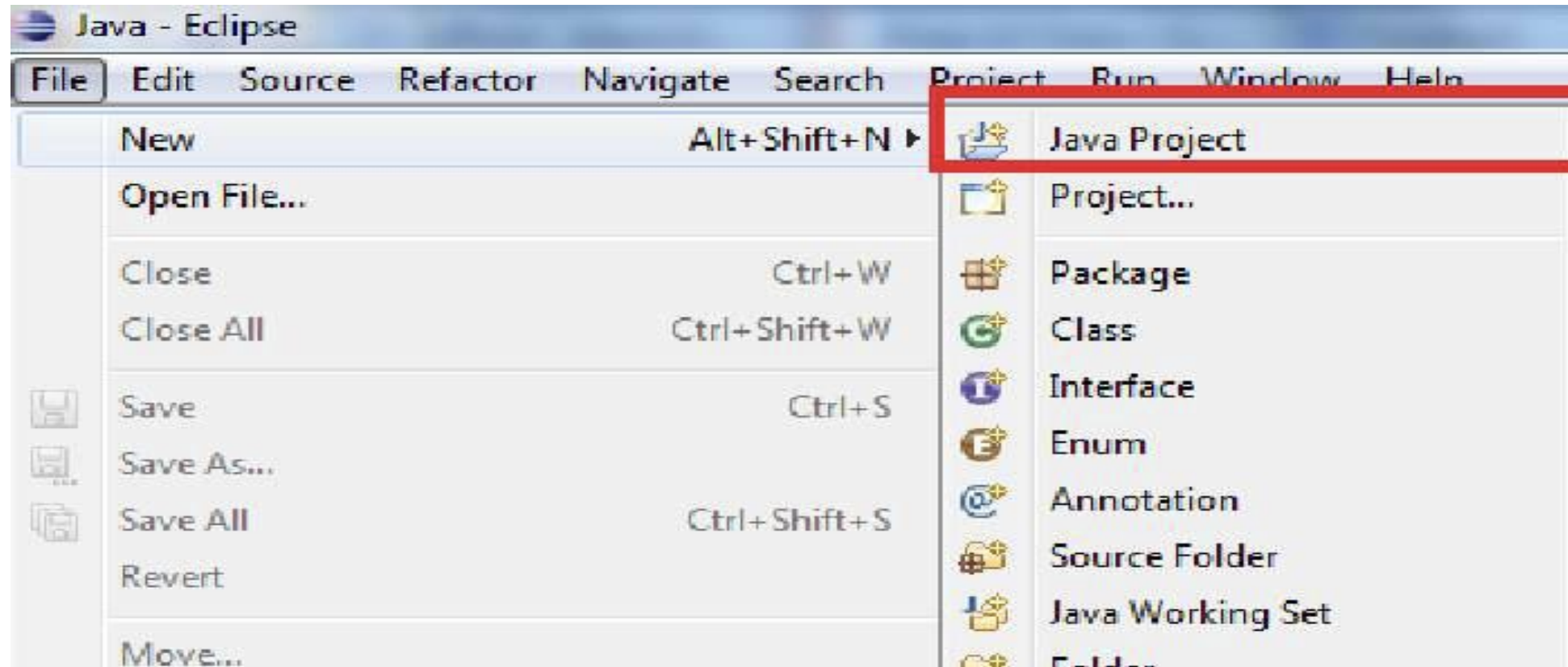


# Multithreading in Java

- ▶ 3 ways to multithread in Java
  - ▶ Extending the Thread Class
  - ▶ Use the Runnable Interface
  - ▶ Using anonymous Thread Class

# 1) Extending Thread Class

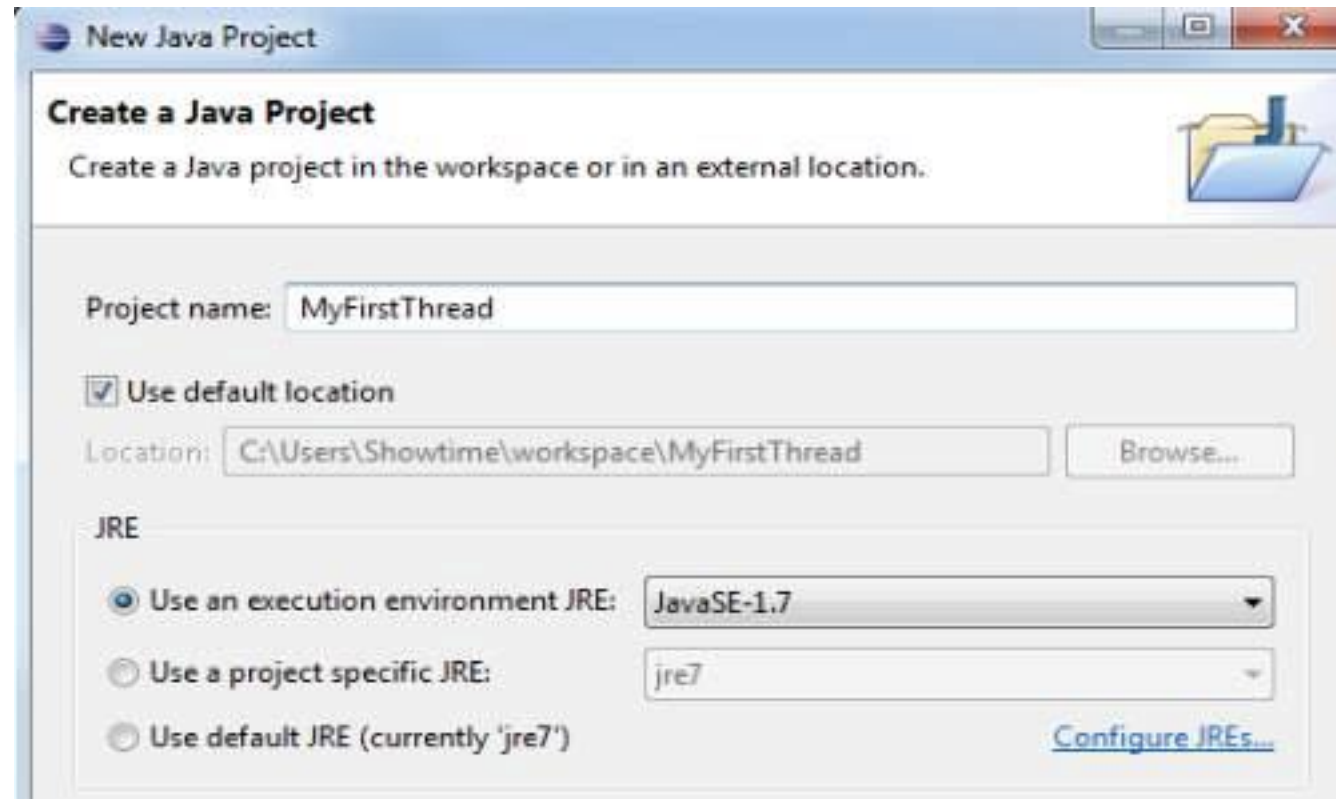
## 1. Create a project



# Extending Thread Class (cont.)

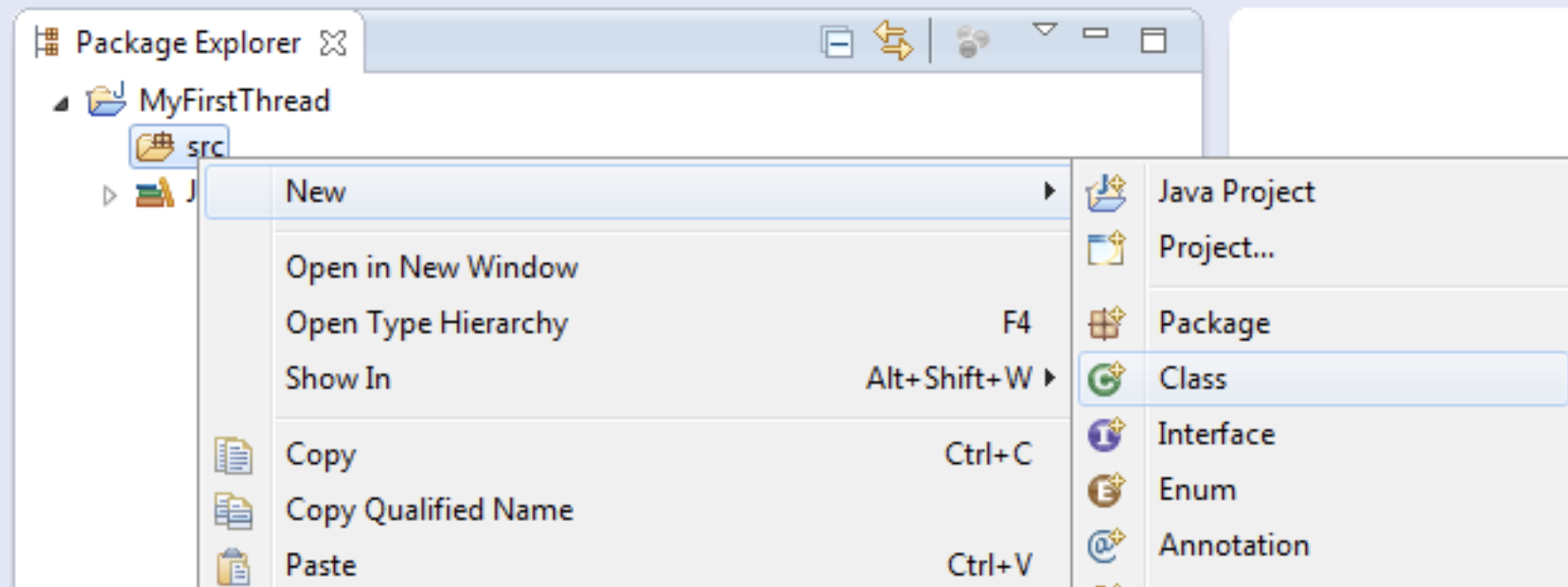
2. Project name :- MyFirstThread

3. Click on finish



# Extending Thread Class (cont.)

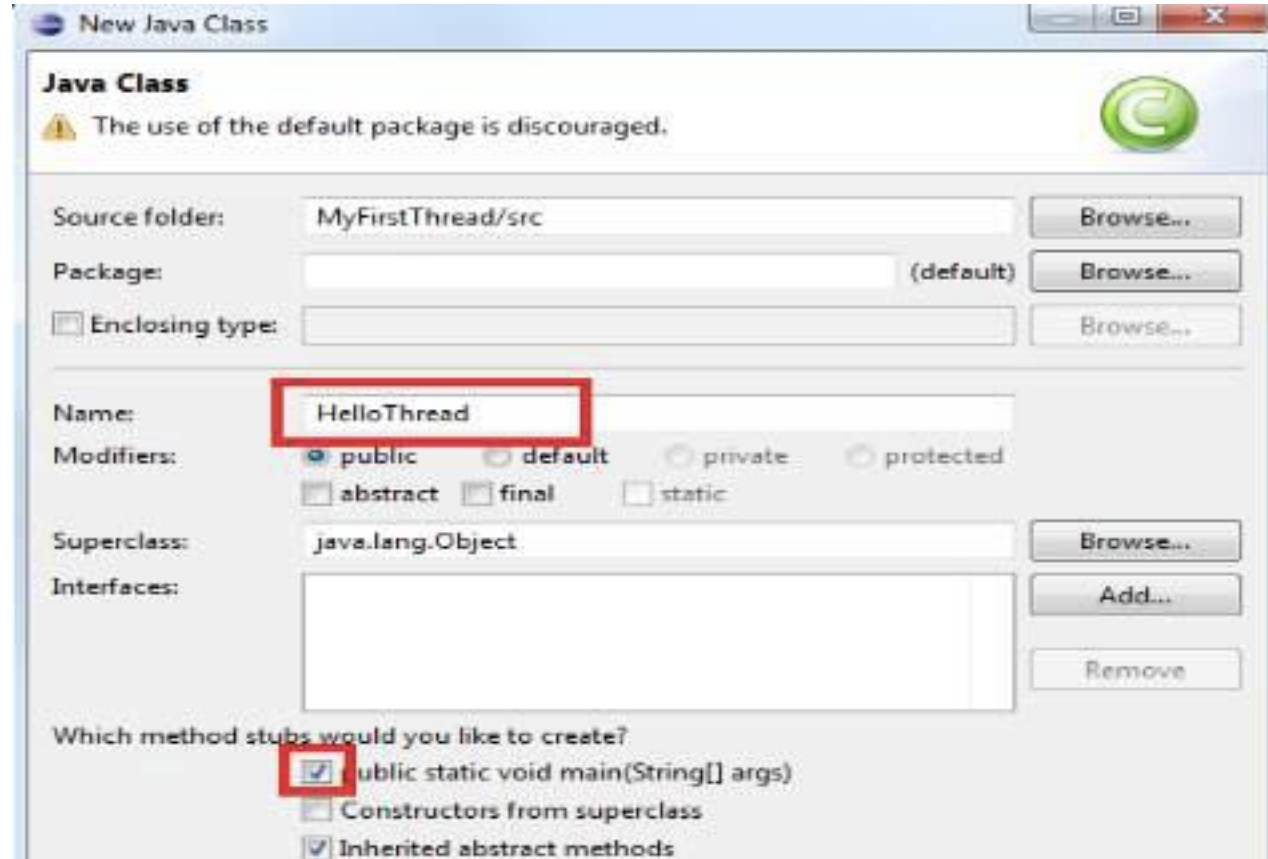
4. Expand the project folder
5. Right click on src
6. Create a new class name "HelloThread"



# Extending Thread Class (cont.)

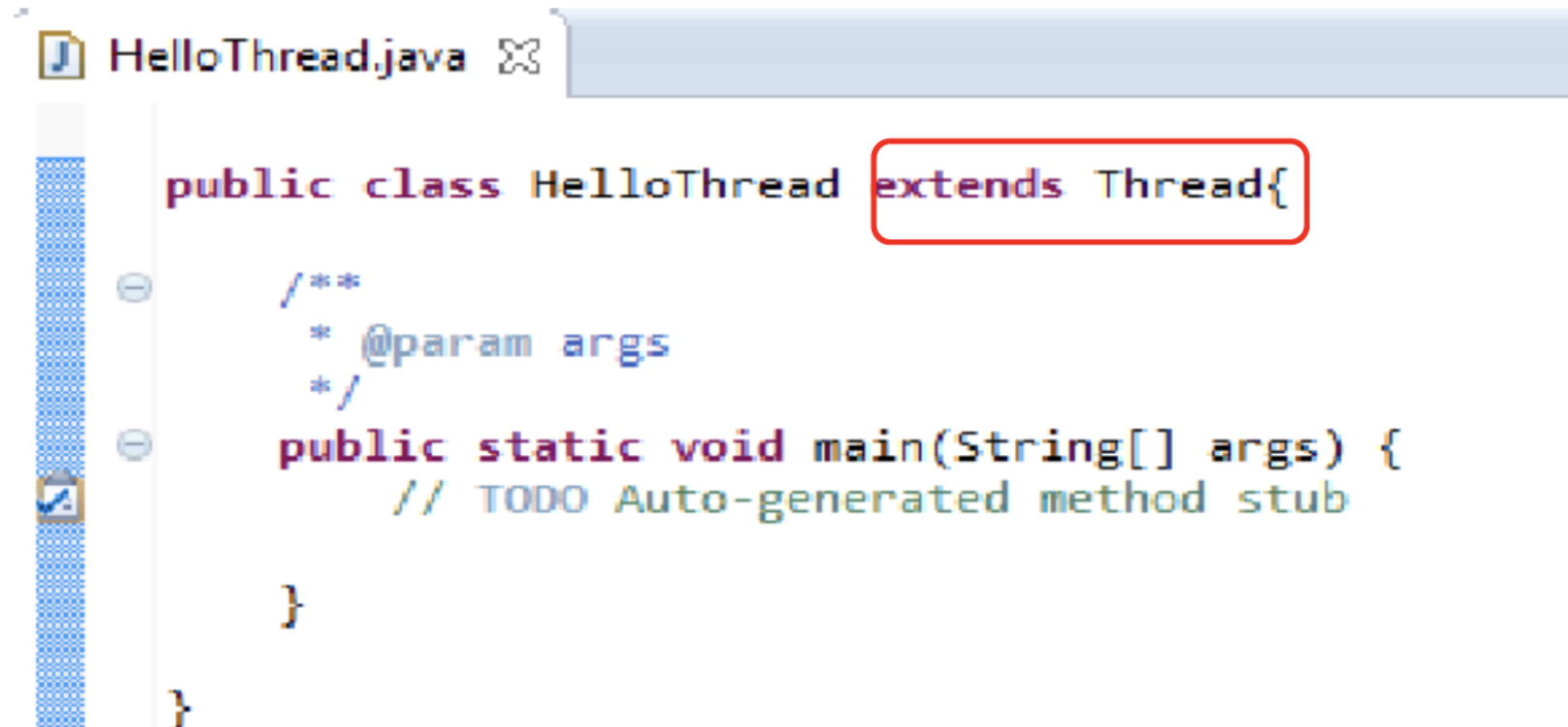
7. Check box “public static void main(String[] args)”

8. Click on finish



# Extending Thread Class (cont.)

## 9. Extend thread



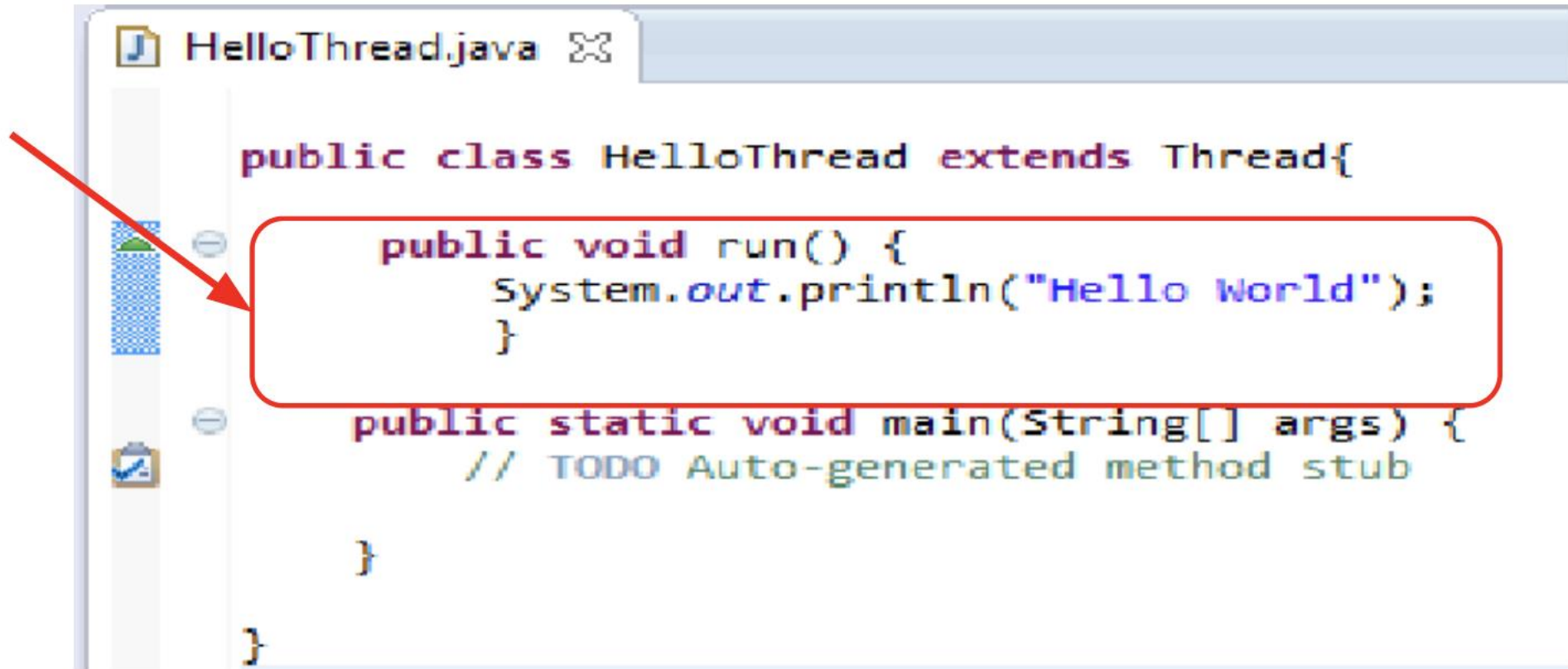
```
HelloThread.java

public class HelloThread extends Thread{

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}
```

# Extending Thread Class (cont.)


## 10. Override method run()



# Extending Thread Class (cont.)

## 11. Create and start thread

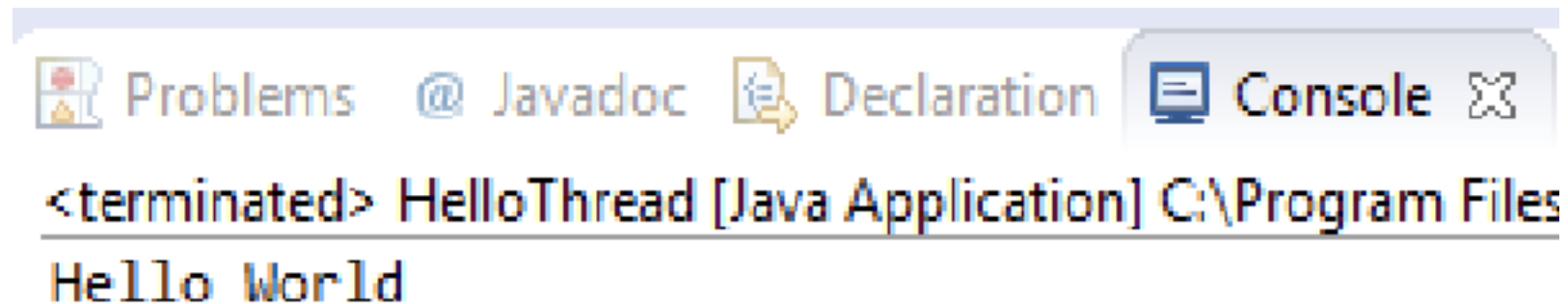
```
public class HelloThread extends Thread{  
    public void run() {  
        System.out.println("Hello World");  
    }  
  
    public static void main(String[] args) {  
        (new HelloThread()).start();  
    }  
}
```





# Extending Thread Class (cont.)

12. Run the file and output should look like this



The screenshot shows a console window with a tab bar at the top containing 'Problems', '@ Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active. The output text in the console is:   
<terminated> HelloThread [Java Application] C:\Program Files  
Hello World

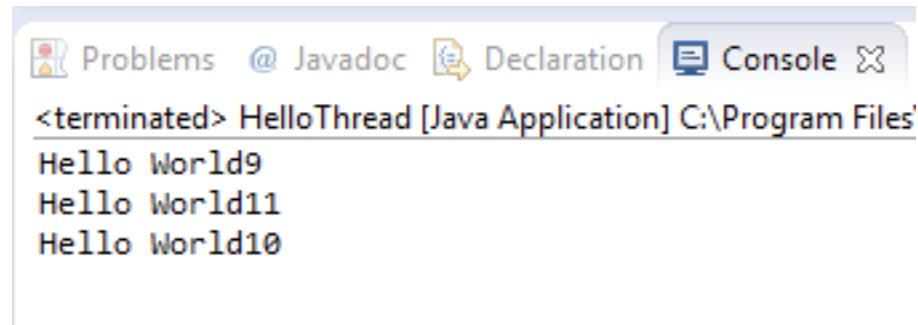
# Extending Thread Class (cont.)

- Examine a multithreaded run getId() returns thread unique identifier

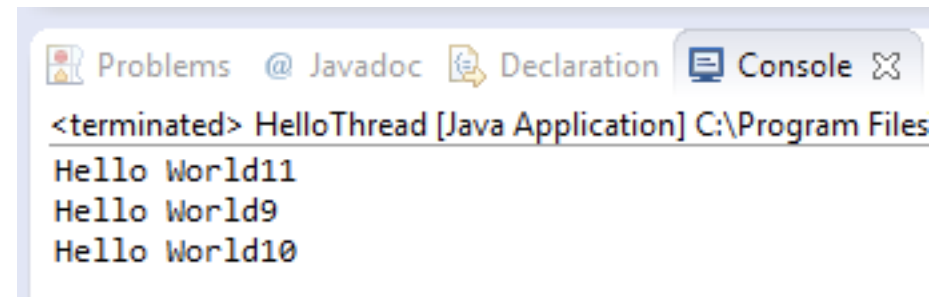
```
public class HelloThread extends Thread{  
  
    public void run() {  
        System.out.println("Hello World" + this.getId());  
    }  
  
    public static void main(String[] args) {  
        (new HelloThread()).start();  
        (new HelloThread()).start();  
        (new HelloThread()).start();  
    }  
}
```

# Extending Thread Class (cont.)

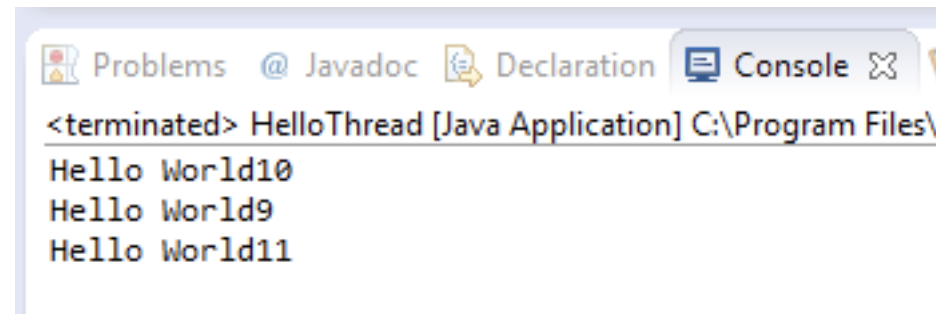
- WHY??



```
<terminated> HelloThread [Java Application] C:\Program Files\  
Hello World9  
Hello World11  
Hello World10
```



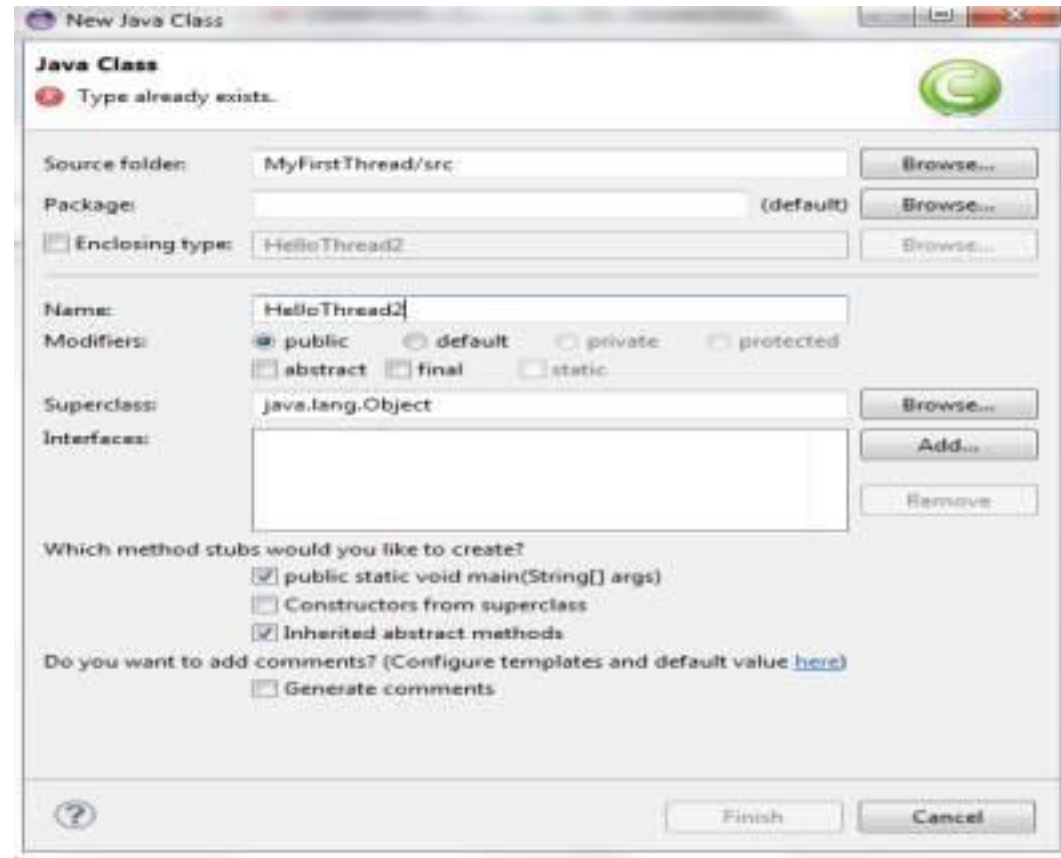
```
<terminated> HelloThread [Java Application] C:\Program Files\  
Hello World11  
Hello World9  
Hello World10
```



```
<terminated> HelloThread [Java Application] C:\Program Files\  
Hello World10  
Hello World9  
Hello World11
```

## 2) Using the Runnable Interface

- In the project, create a class file “HelloThread2”



# Code:

```
HelloThread2.java 88

public class HelloThread2 implements Runnable
{
    @Override
    public void run()
    {
        System.out.println("Thread id" + Thread.currentThread().getId());
    }

    public static void main(String args[])
    {
        HelloThread2 ht2 = new HelloThread2();

        //Need to pass the instance of the class into
        //into the constructor of the thread
        Thread t = new Thread(ht2);
        t.start();

        //or
        //new Thread(new HelloThread2()).start();
    }
}
```

# Difference between extends Thread and Runnable Interface

- ▶ When you extend Thread class, you can't extend any other class which you require. (As you know, Java does not allow inheriting more than one class). When you implement Runnable, you can save a space for your class to extend any other class in future or now.
- ▶ When you extends Thread class, each of your thread creates unique object and associate with it. When you implements Runnable, it shares the same object to multiple threads.

# 3) Thread creation using Anonymous Creation

Directly creating a thread when we need it using new Thread()

```
public class AnonymousThreadCreation
{
    public static void main(String args[])
    {
        new Thread( new Runnable() {
            @Override
            public void run()
            {
                for(int i=10; i>1; i--)
                {
                    System.out.println(i + " looping in Thread ...");
                    // Sleep for a while
                    try
                    {
                        Thread.sleep(200);
                    }
                    catch (InterruptedException e)
                    {
                        // Interrupted exception will occur if
                        // the Worker object's interrupt() method
                        // is called. interrupt() is inherited
                        // from the Thread class.
                        System.out.println("Thread interrupted.");
                        break;
                    }
                }
            }
        }).start();

        for (int i=10;i>1;i--)
        {
            System.out.println(i+ "looping ... in the main");
        }
    }
}
```

# Synchronization

- ▶ Necessary to conserve integrity of critical portions of the code.

- ▶ Example:

We can have multiple people reading an article at the same time.

However if they want to edit the article, should they be able to do it all at the same time? What if someone saves changes while another was editing the article without knowledge of the changes?



# Synchronization

- In the Same package , create a new class name it SynchronizedCounter

**New Java Class**

Java Class

The use of the default package is discouraged.

Source folder: Soen\_423\_threads/src Browse...

Package: (default) Browse...

☐ Enclosing type: Browse...

Name: SynchronizedCounter

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...  
Remove

Which method stubs would you like to create?

☒ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

? Finish Cancel

# Synchronization

## ► Add Few Methods to the class :

```
public class SynchronizedCounter
{
    private int c=0;

    public void increament()
    {
        c++;
    }

    public void decreament()
    {
        c--;
    }

    public int retunValue()
    {
        return c;
    }
}
```

# Problem in above code:

- ▶ As one thread is incrementing the value and other thread is decrementing the value hence the resultant value of c should be 0.
- ▶ But in this case the value of the c is -1 .
- ▶ Hence to overcome this problem we can use the Synchronized functions or the Synchronized blocks .

# Synchronization

- ▶ Now imagine the following scenario:
- ▶ Suppose you create 2 Threads (A and B)
- ▶ A increments the counter
- ▶ B decrements the counter
- ▶ They execute:
  - 1.) Thread A: Retrieve c.
  - 2.) Thread B: Retrieve c.
  - 3.) Thread A: Increment retrieved value; result is 1.
  - 4.) Thread B: Decrement retrieved value; result is -1.
  - 5.) Thread A: Store result in c; c is now 1.
  - 6.) Thread B: Store result in c; c is now -1.

# Synchronized

```
public class SynchronizedCounter
{
    private int c=0;

    public synchronized void increment()
    {
        c++;
    }

    public synchronized void decreament()
    {
        c--;
    }

    public synchronized int retunValue()
    {
        return c;
    }
}
```

Method level synchronization

# Synchronized

- ▶ With the use of "Synchronized" variable the JVM allows only one Thread to enter the Synchronized regions so the problem that we experienced before can be emitted using the Synchronized block, methods (whichever you seem fit you can use).

# Synchronized(this)

```
public class SynchronizedCounter
{
    private int c=0;

    public void increament()
    {
        synchronized(this)
        {
            c++;
        }
    }
}
```

Block level synchronization

# Difference between both techniques:

<http://stackoverflow.com/questions/4394976/what-is-the-differencebetween synchronizedthis-and-synchronizedmethod>

<http://www.geeksforgeeks.org/method-block-synchronization-java/>

Use block-level synchronization which has much better performance vs method-level synchronization.



# Locks

```
Lock l=lock;  
l.lock();  
try  
{  
    // access or modify the resources protected by the lock  
}  
finally  
{  
    l.unlock();  
}
```

# References

1. <http://tutorials.jenkov.com/java-concurrency/locks.html>
2. [http://www.javamex.com/tutorials/synchronization\\_concurrency\\_1.shtml](http://www.javamex.com/tutorials/synchronization_concurrency_1.shtml)
3. <http://docs.oracle.com/javase/tutorial/essential/concurrency/interfere.html>
4. [https://en.wikipedia.org/wiki/Thread\\_%28computing%29](https://en.wikipedia.org/wiki/Thread_%28computing%29)
5. <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

Further Reading:

<http://docs.oracle.com/javase/tutorial/essential/concurrency/>