Multithreading

Agenda:

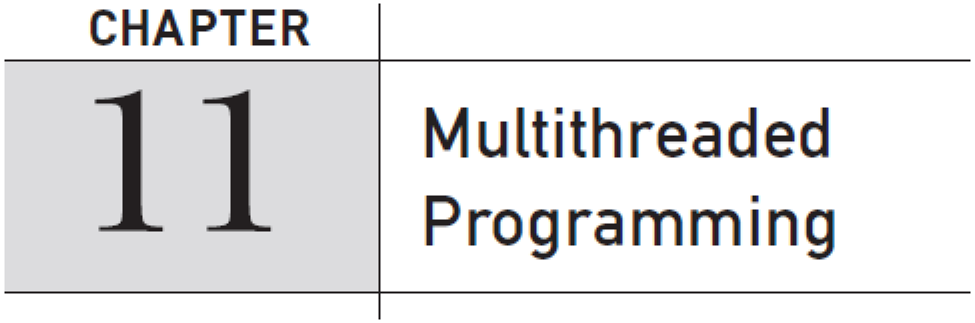
Multithreaded Programming

Md. Mamun Hossain
B.Sc. (Engg.) & M.Sc. (Thesis) in CSE , SUST
Assistant Professor, Dept. of CSE, BAUST

A thread is a lightweight subprocess, the smallest unit of processing.

Reference Text : Chap. - 11

Multithreaded Programming


CHAPTER

11

Multithreaded Programming

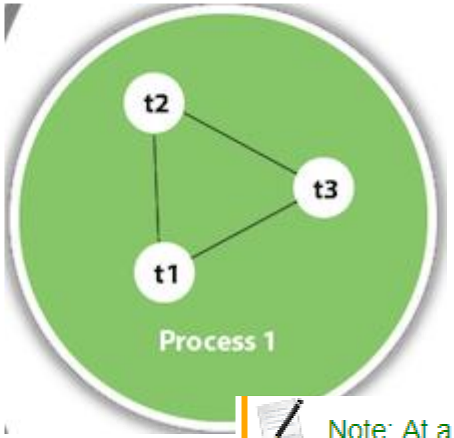
Keywords:

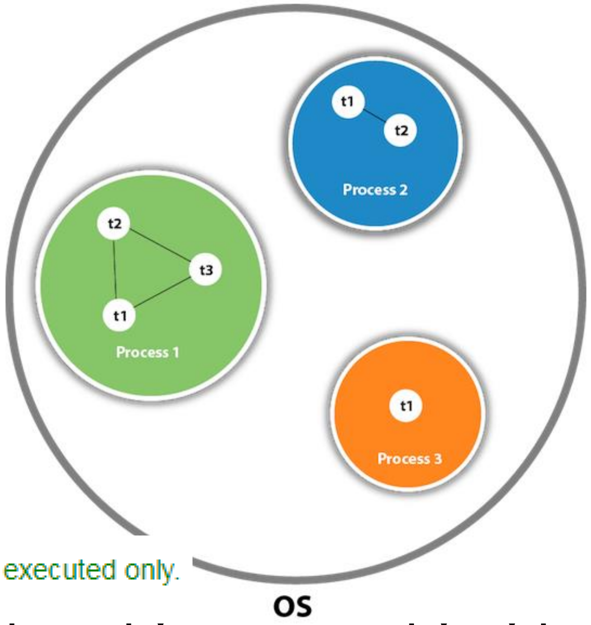
- **Thread & Threading**
- **Multi Threading**
- **Advantages of Thread**
- **Multitasking & Thread**
- **Thread Life Cycle**
- **Creating Thread**
- **Thread Schedular**
- **Daemon Thread**
- **Thread Priority**
- **Concurrency**

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

What is Thread in java?

- A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.
- Threads are independent. If there occurs exception in one thread, it doesn't affect other threads.
- It uses a shared memory area.





Note: At a time one thread is executed only.

- As shown in the above figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

What is Multi-threading?

- Multithreading in java is a process of executing multiple threads simultaneously.

- Multiprocessing and multithreading, both are used to achieve multitasking.

- However, we use multithreading than multiprocessing because *threads use a shared memory area. They don't allocate separate memory area so saves memory*, *and context-switching between the threads takes less time than process*.

- Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It doesn't block the user because threads are independent and you can perform multiple operations at the same time.

2) You can perform many operations together, so it saves time.

3) Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

Multitasking & Thread

Multitasking is a process of executing multiple tasks simultaneously.

We use multitasking to utilize the CPU.

Multitasking can be achieved in two ways:

- ✓ Process-based Multitasking (Multiprocessing)
- ✓ Thread-based Multitasking (Multithreading)

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Multitasking & Thread

1) **Process-based Multitasking (Multiprocessing)**

Each process has an address in memory. In other words, each process allocates a separate memory area.

- ✓ A process is heavyweight.
- ✓ Cost of communication between the process is high.
- ✓ Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

- ✓ Threads share the same address space.
- ✓ A thread is lightweight.
- ✓ Cost of communication between the thread is low.

Note: At least one process is required for each thread.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

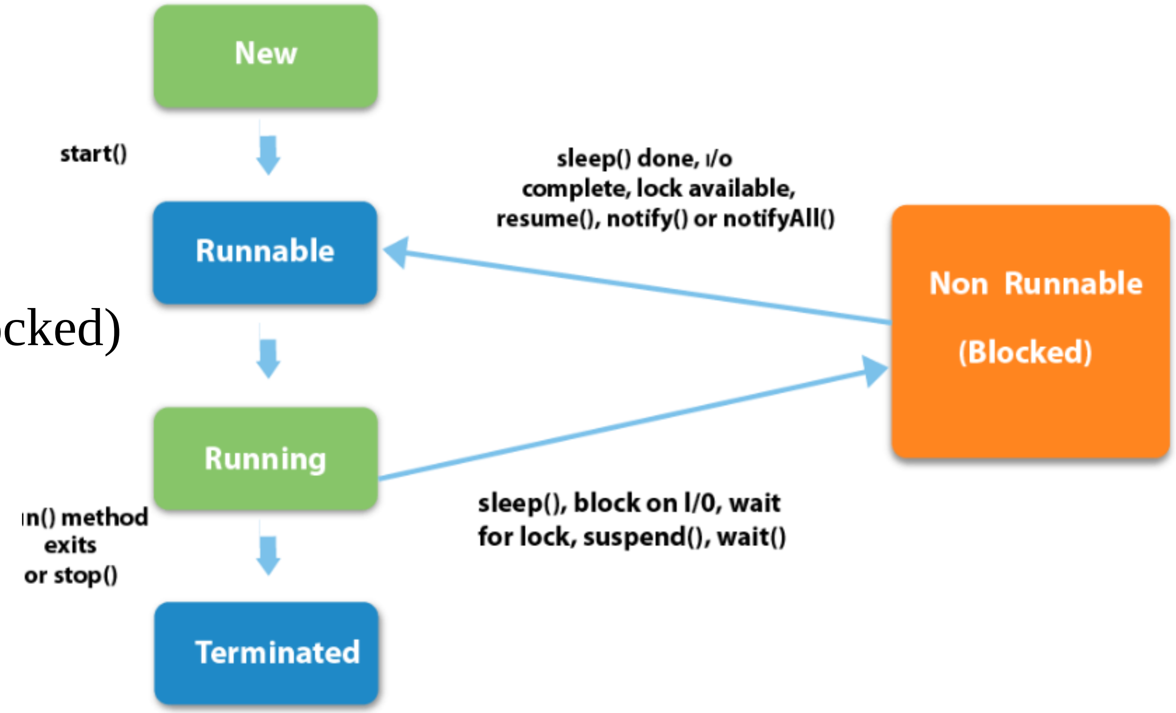
Life cycle of a Thread (Thread States)

A thread can be in one of the five states. According to sun, there is only 4 states in thread life cycle in java 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 in java is controlled by JVM. The java thread states are as follows:

i. New
ii. Runnable
iii. Running
iv. Non-Runnable (Blocked)
v. Terminated


New
start()
Runnable
sleep() done, I/o complete, lock available, resume(), notify() or notifyAll()
Non Runnable (Blocked)
Running
sleep(), block on I/0, wait for lock, suspend(), wait()
n() method exits or stop()
Terminated

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Life cycle of a Thread (Thread States)

1) New

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

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

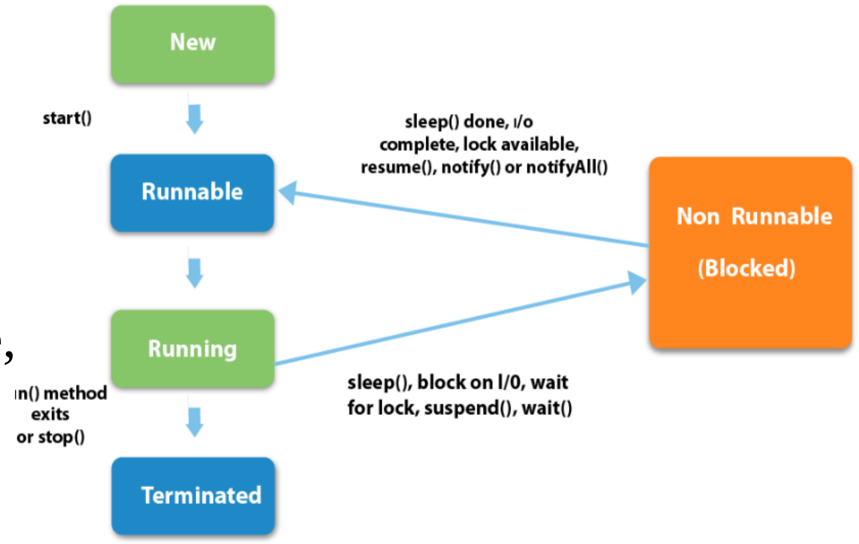
The thread is in running state

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.


New
start()
Runnable
sleep() done, i/o complete, lock available, resume(), notify() or notifyAll()
Non Runnable (Blocked)
Running
sleep(), block on I/0, wait for lock, suspend(), wait()
n() method exits or stop()
Terminated

5) Terminated

A thread is in terminated or dead state when its run() method exits.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Java Thread class

The Thread Class

- Java provides Thread class to achieve thread programming.
- Thread class provides constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- ✓ Thread()
- ✓ Thread(String name)
- ✓ Thread(Runnable r)
- ✓ Thread(Runnable r,String name)

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Runnable interface

Runnable interface:

- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.
- ***Runnable interface have only one method named run().***

public void run(): is used to perform action for a thread.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Commonly used methods of Thread class

public void **run**(): is used to perform action for a thread.

public void **start**(): starts the execution of the thread.JVM calls the run() method on the thread.

public void **sleep(long** miliseconds): Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

public int **getPriority**(): returns the priority of the thread.

public int **setPriority(int** priority): changes the priority of the thread.

public String **getName**(): returns the name of the thread.

public void **setName(String** name): changes the name of the thread.

public **currentThread**(): returns the reference of currently executing thread.

public int **getId**(): returns the id of the thread.

public boolean **isAlive**(): tests if the thread is alive.

public boolean **isDaemon**(): tests if the thread is a daemon thread.

public void **setDaemon(boolean** b): marks the thread as daemon or user thread.

Commonly used methods of Thread class

public void join(): waits for a thread to die.

public void join(long miliseconds): waits for a thread to die for the specified miliseconds.

public Thread.State getState(): returns the state of the thread.

public void yield(): causes the currently executing thread object to temporarily pause and allow other threads to execute.

public void suspend(): is used to suspend the thread(depricated).

public void resume(): is used to resume the suspended thread(depricated).

public void stop(): is used to stop the thread(depricated).

public void interrupt(): interrupts the thread.

p blic boolean isInterrupted(): tests if the thread has been interrupted.

public static boolean interrupted(): tests if the current thread has been interrupted.

Starting a thread

start() method of Thread class is used to start a newly created thread.

It performs following tasks:

- ✓ A new thread starts(with new callstack).
- ✓ The thread moves from New state to the Runnable state.
- ✓ When the thread gets a chance to execute, its target run() method will run.

How to create thread?

There are two ways to create a thread:

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

extending Thread class

```
class A extends Thread{

Public void run()
{
  System.out.println("Thread is running");
}

Public static Void main(String args[]){
        A t1=new A();
        t1.run();

}
```

Output : Thread is running.

If you are not extending the Thread class, your class object would not be treated as a thread object.

implementing Runnable interface

```
class B implements Runnable{

Public void run()
{
  System.out.println("Thread is running");
}

Public static Void main(String args[]){
        B ob=new B();
        Thread t1=new Thread(ob);
        t1.run();
}
```

So we need to explicitly create Thread class object. We are passing the object of the class that implements Runnable so that the class run() method may execute.

Thread Scheduler in Java

- Thread scheduler in java is the part of the JVM that decides which thread should run.
- There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
- *Only one thread at a time can run in a single process.*
- The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

Preemptive Vs. Time slicing

Difference between preemptive scheduling and time slicing

- ✓ Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence.
- ✓ Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

Sleep method: Syntax of sleep() method

The Thread class provides two methods for sleeping a thread:

- public static void sleep(long miliseconds)throws InterruptedException
- public static void sleep(long miliseconds, int nanos)throws InterruptedException

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

```
class TestSleepMethod1 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();

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

Output:

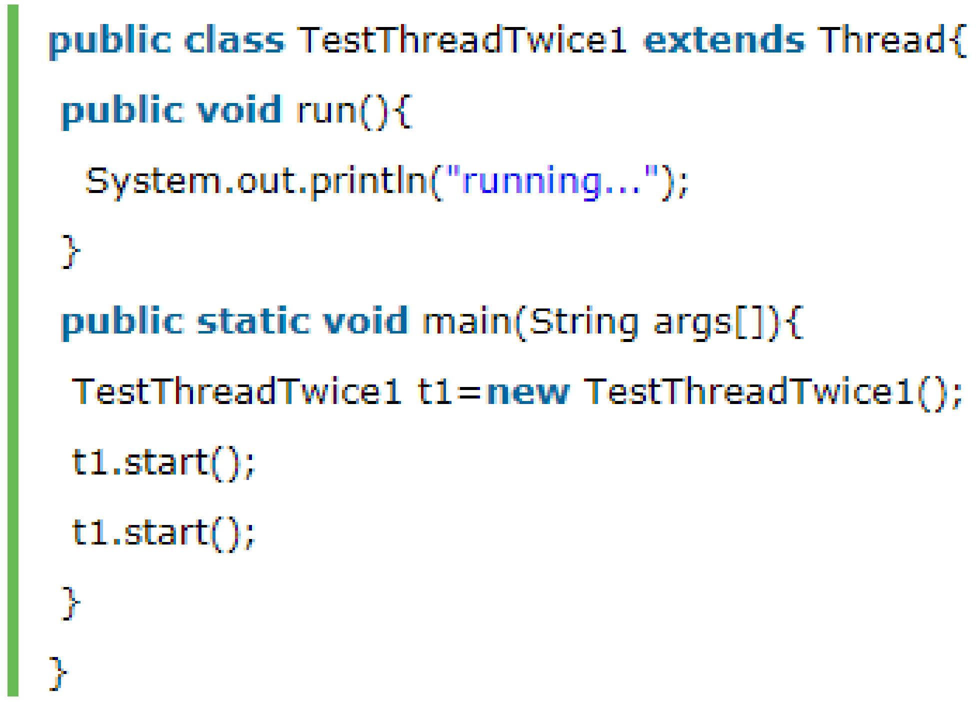
```
1
1
2
2
3
3
4
4
```

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time, the thread shedular picks up another thread and so on.

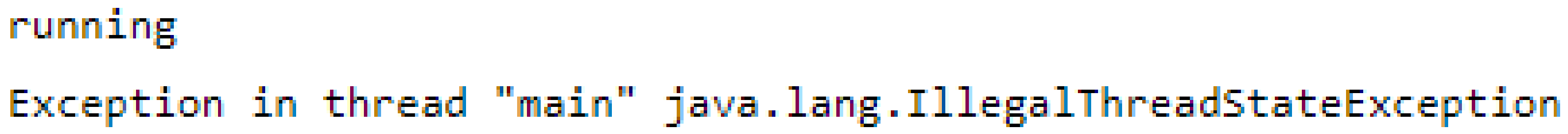
Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Can we start a thread twice

No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.



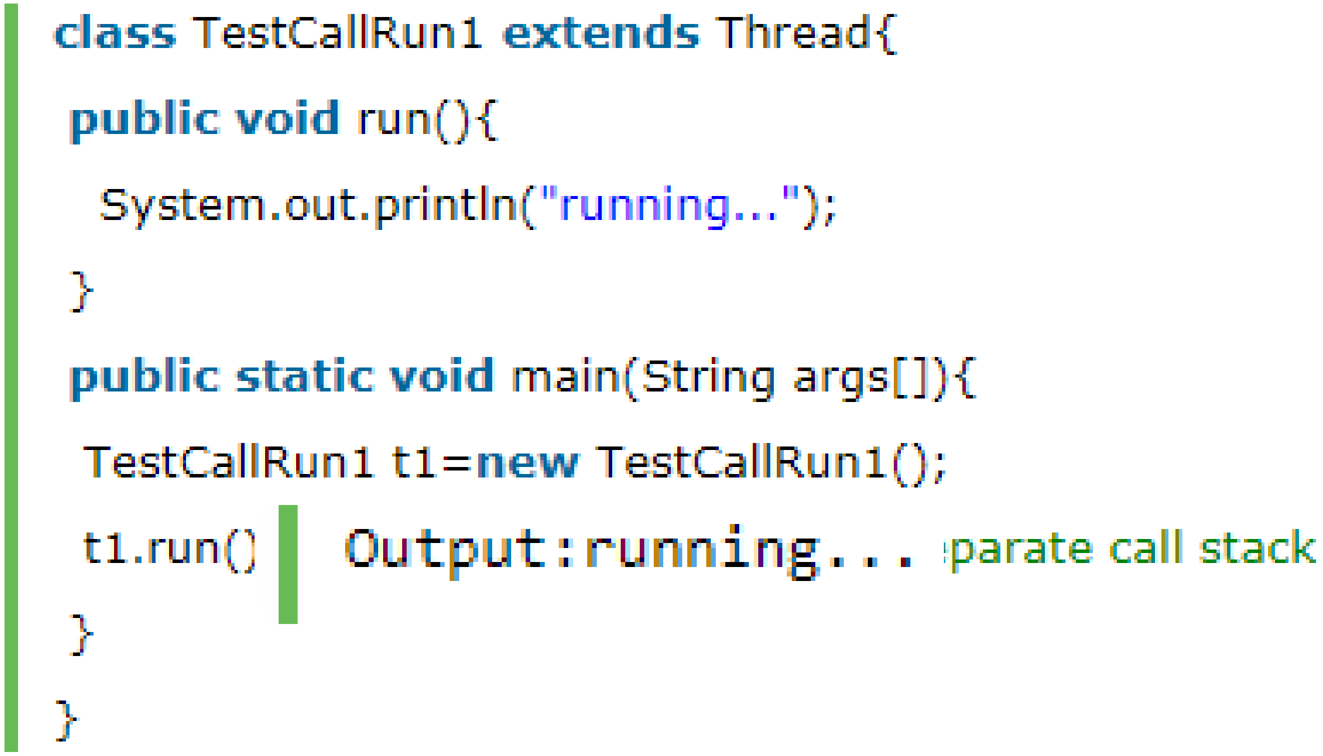
```
public class TestThreadTwice1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestThreadTwice1 t1=new TestThreadTwice1();
  t1.start();
  t1.start();
 }
}
```



```
running
Exception in thread "main" java.lang.IllegalThreadStateException
```

What if we call run() method directly instead start() method?

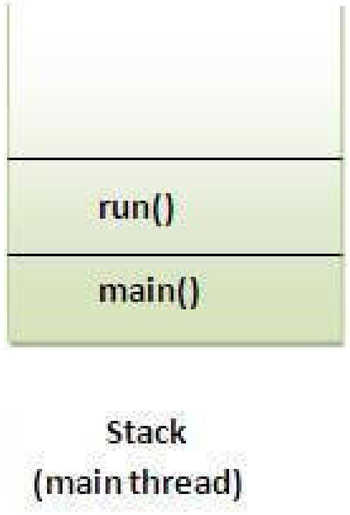
Each thread starts in a separate call stack.
Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.



```
class TestCallRun1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestCallRun1 t1=new TestCallRun1();
  t1.run()
 }
}
```

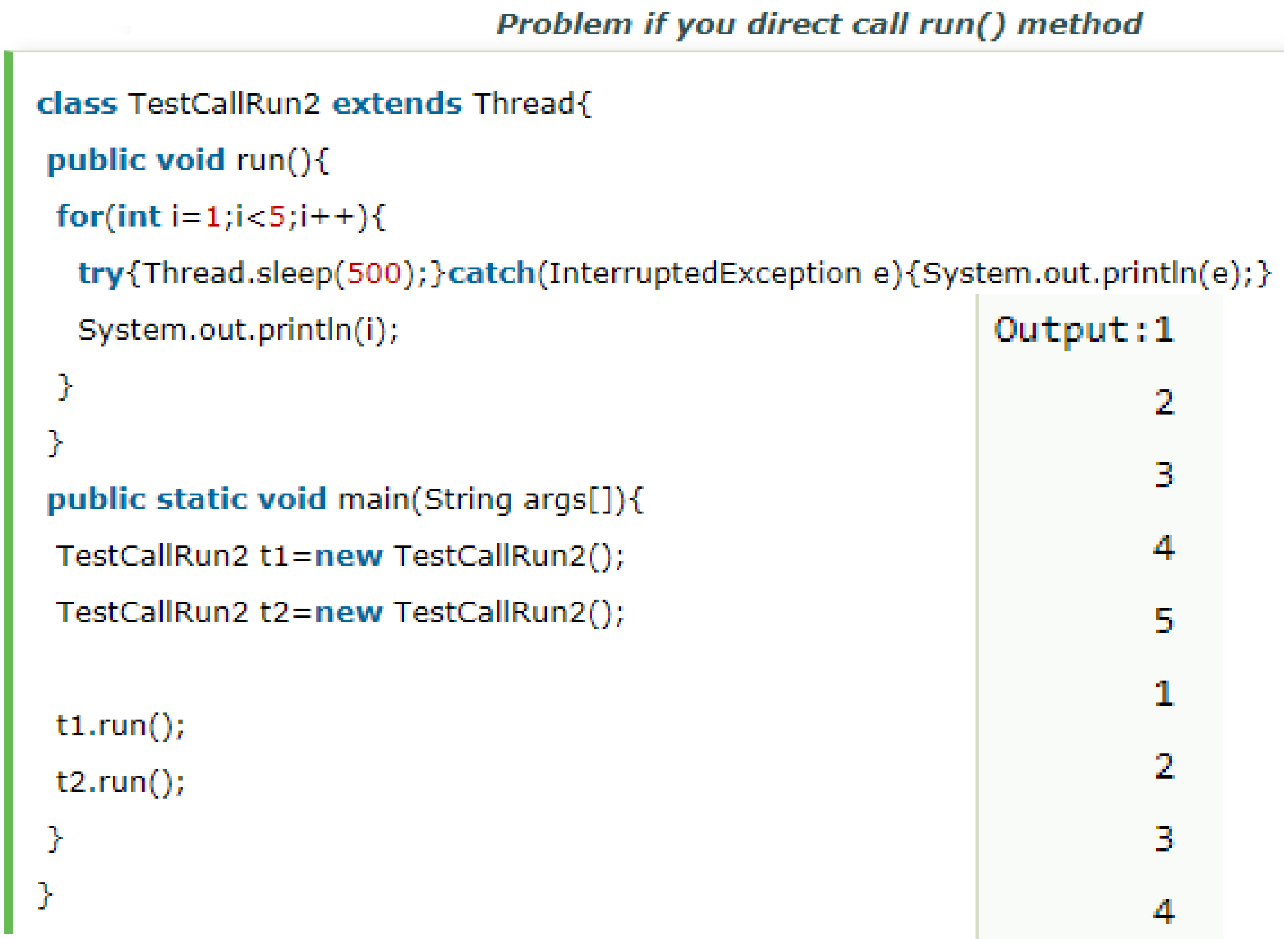
```
Output:running...
```

parate call stack



Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

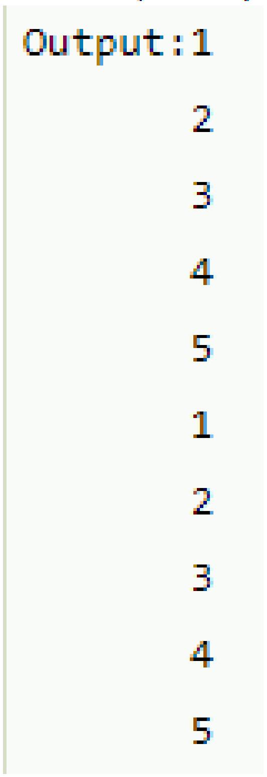
What if we call run() method directly instead start() method?



Problem if you direct call run() method

```
class TestCallRun2 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestCallRun2 t1=new TestCallRun2();
  TestCallRun2 t2=new TestCallRun2();

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

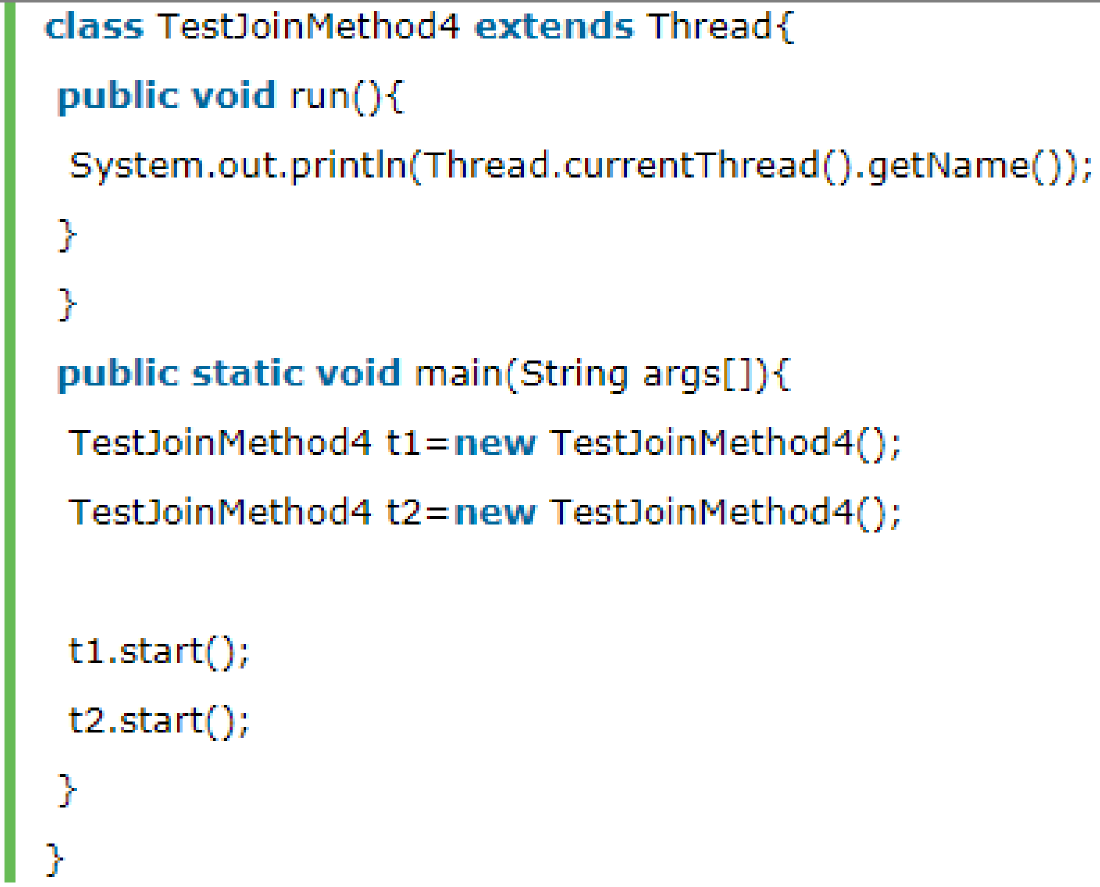


```
Output:1
       2
       3
       4
       5
       1
       2
       3
       4
       5
```

As you can see in the above program that there is no context-switching because here t1 and t2 will be treated as normal object not thread object.

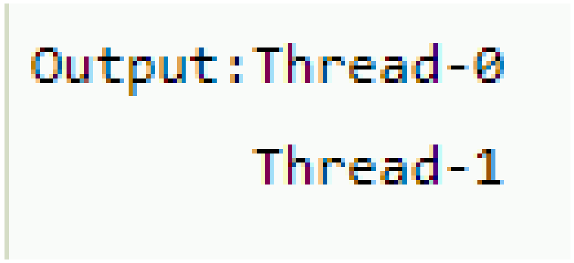
Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

The current Thread() method:



```
class TestJoinMethod4 extends Thread{
 public void run(){
  System.out.println(Thread.currentThread().getName());
 }
 }
 public static void main(String args[]){
  TestJoinMethod4 t1=new TestJoinMethod4();
  TestJoinMethod4 t2=new TestJoinMethod4();

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



```
Output:Thread-0
       Thread-1
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Thread Naming

```
class TestMultiNaming1 extends Thread{
  public void run(){
   System.out.println("running...");
  }
 public static void main(String args[]){
  TestMultiNaming1 t1=new TestMultiNaming1();
  TestMultiNaming1 t2=new TestMultiNaming1();
  System.out.println("Name of t1:"+t1.getName());
  System.out.println("Name of t2:"+t2.getName());

  t1.start();
  t2.start();

  t1.setName("Sonoo Jaiswal");
  System.out.println("After changing name of t1:"+t1.getName());
 }
}
```

```
Output:Name of t1:Thread-0
       Name of t2:Thread-1
       id of t1:8
       running...
       After changeling name of t1:Sonoo Jaiswal
       running...
```

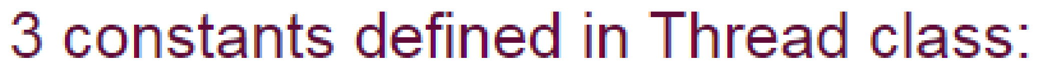
Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Thread Priority

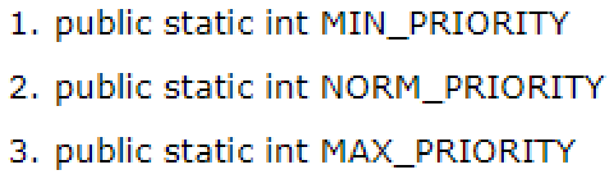
Each thread have a priority. Priorities are represented by a number between 1 and 10.

In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling).

But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.



3 constants defined in Thread class:



1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Example of priority of a Thread

```
class TestMultiPriority1 extends Thread{
 public void run(){
   System.out.println("running thread name is:"+Thread.currentThread().getName());
   System.out.println("running thread priority is:"+Thread.currentThread().getPriority());


  }
 public static void main(String args[]){
  TestMultiPriority1 m1=new TestMultiPriority1();
  TestMultiPriority1 m2=new TestMultiPriority1();
  m1.setPriority(Thread.MIN_PRIORITY);
  m2.setPriority(Thread.MAX_PRIORITY);
  m1.start();
  m2.start();


 }
}
```

```
Output:running thread name is:Thread-0
       running thread priority is:10
       running thread name is:Thread-1
       running thread priority is:1
```

Daemon Thread in Java

Daemon thread in java is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically
e.g. ***gc, finalizer*** etc.

Points to remember for Daemon Thread in Java

- ✓ It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
- ✓ Its life depends on user threads.
- ✓ It is a low priority thread.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Daemon Thread: Example

```
public class TestDaemonThread1 extends Thread{
 public void run(){
  if(Thread.currentThread().isDaemon()){//checking for daemon thread
   System.out.println("daemon thread work");
  }
  else{
  System.out.println("user thread work");
 }
 }
 public static void main(String[] args){
  TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
  TestDaemonThread1 t2=new TestDaemonThread1();
  TestDaemonThread1 t3=new TestDaemonThread1();

  t1.setDaemon(true);//now t1 is daemon thread

  t1.start();//starting threads
  t2.start();
  t3.start();
 }
}
```

Output

```
daemon thread work
user thread work
user thread work
```

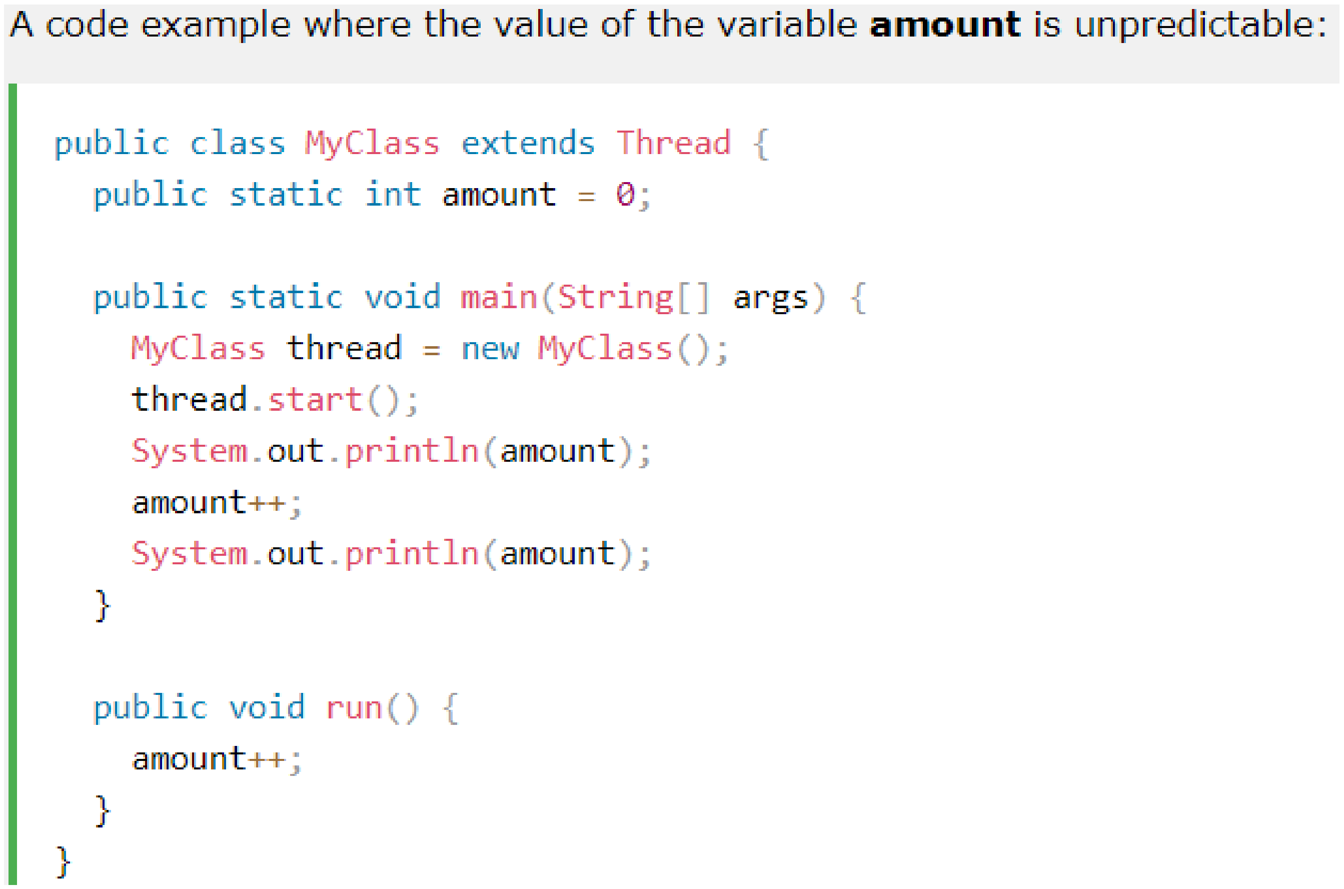
Why JVM terminates the daemon thread if there is no user thread?

- The sole purpose of the daemon thread is that it provides services to user thread for background supporting task.
- If there is no user thread, why should JVM keep running this thread.
- That is why JVM terminates the daemon thread if there is no user thread.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Concurrency Problems

- Because threads run at the same time as other parts of the program, there is no way to know in which order the code will run. When the threads and main program are reading and writing the same variables, the values are unpredictable. The problems that result from this are called concurrency problems.



A code example where the value of the variable **amount** is unpredictable:

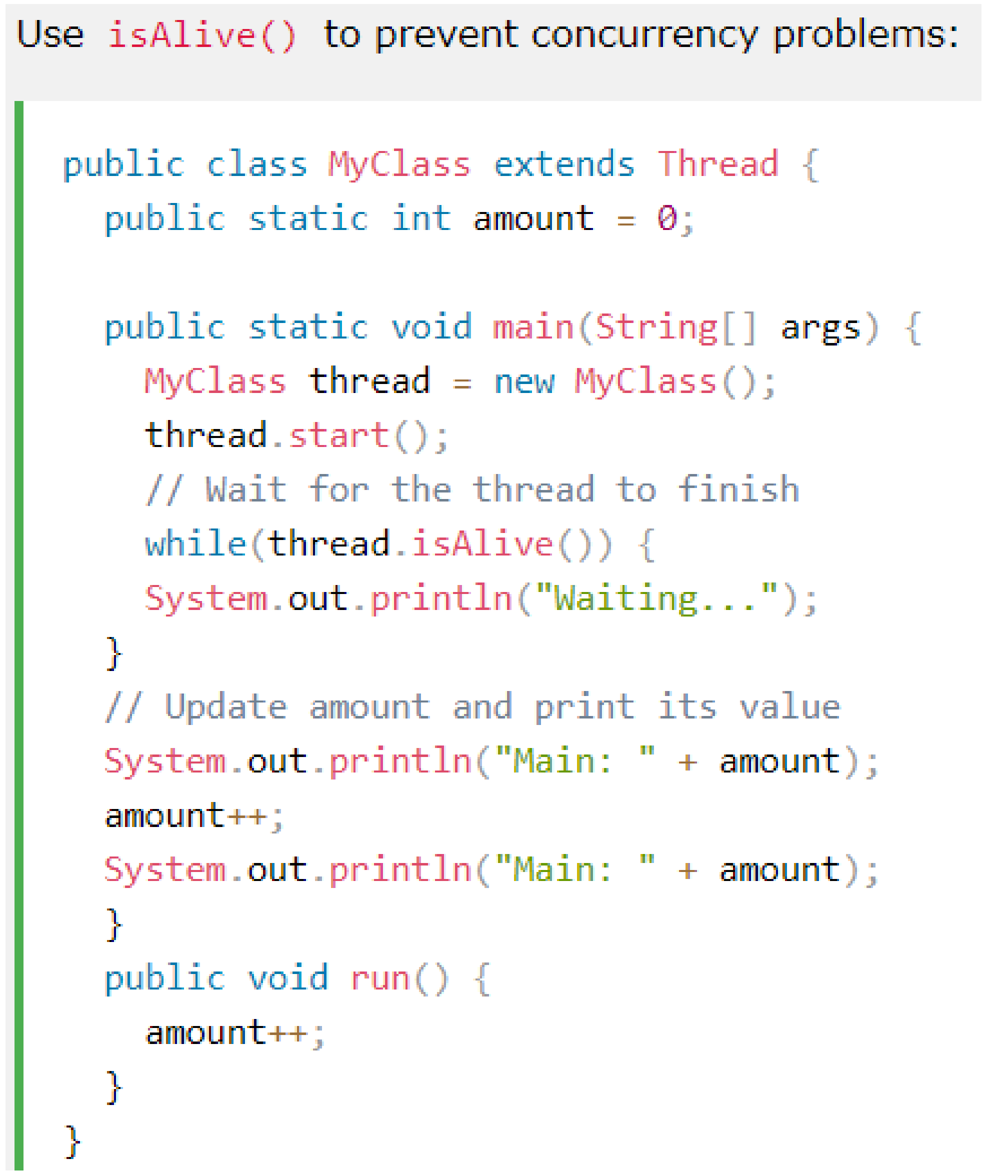
```
public class MyClass extends Thread {
  public static int amount = 0;

  public static void main(String[] args) {
    MyClass thread = new MyClass();
    thread.start();
    System.out.println(amount);
    amount++;
    System.out.println(amount);
  }

  public void run() {
    amount++;
  }
}
```

Concurrency Solution

To avoid concurrency problems, it is best to share as few attributes between threads as possible. If attributes need to be shared, one possible solution is to use the isAlive() method of the thread to check whether the thread has finished running before using any attributes that the thread can change.



Use `isAlive()` to prevent concurrency problems:

```
public class MyClass extends Thread {
  public static int amount = 0;

  public static void main(String[] args) {
    MyClass thread = new MyClass();
    thread.start();
    // Wait for the thread to finish
    while(thread.isAlive()) {
    System.out.println("Waiting...");
  }
  // Update amount and print its value
  System.out.println("Main: " + amount);
  amount++;
  System.out.println("Main: " + amount);
  }
  public void run() {
    amount++;
  }
}
```

Java Garbage Collection

In java, ***garbage means unreferenced objects***.

- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

✓ It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

✓ It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

How can an object be unreferenced?

There are many ways:

1) **By nulling a reference:**

Employee e=new Employee(); // Here Employee() is a Class

e=null;

2) **By assigning a reference to another:**

Employee e1=new Employee();

Employee e2=new Employee();

e1=e2;// now the first object referred by e1 is available for garbage collection

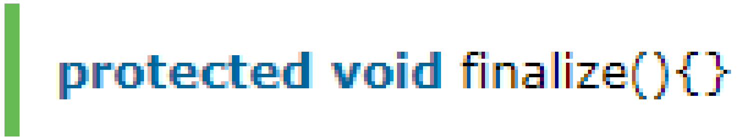
3) **By anonymous object:**

new Employee();

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

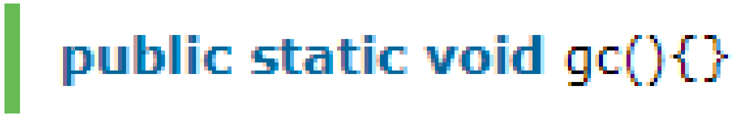
finalize() and *gc()* method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

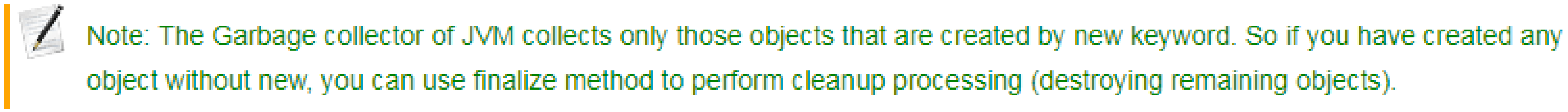


```java
protected void finalize(){}
```

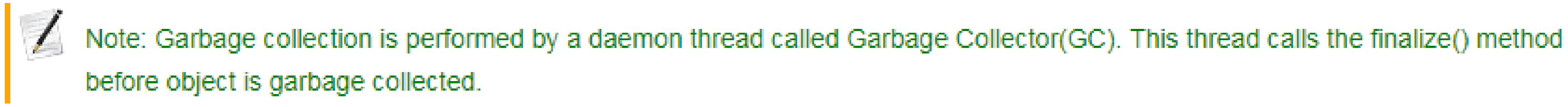
The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.



```java
public static void gc(){}
```



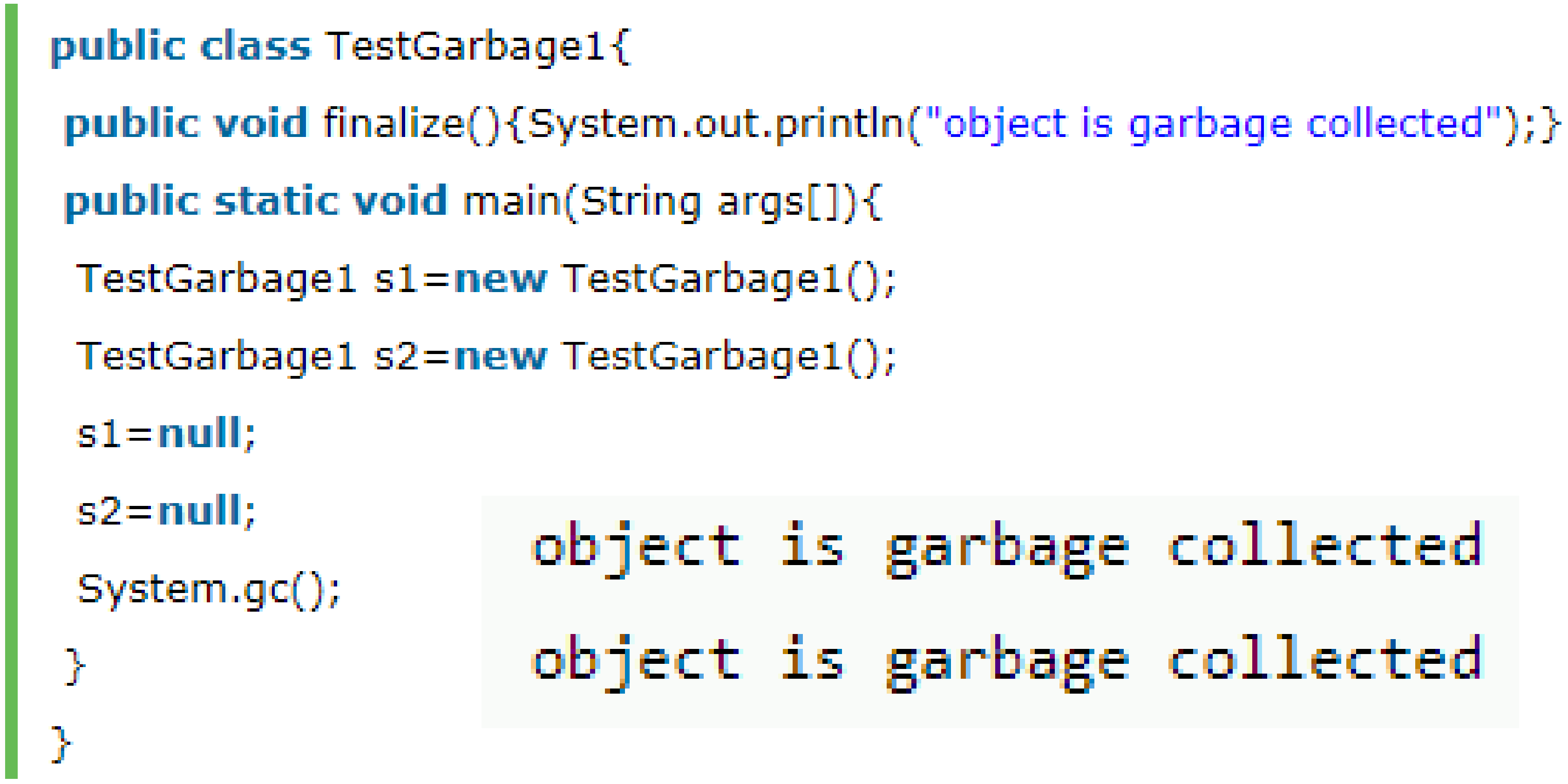
Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).



Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Garbage Collection Example



```java
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 }
}
```

```
object is garbage collected
object is garbage collected
```

Note: Neither finalization nor garbage collection is guaranteed.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST