**Why Multithreading**

A thread is a light-weight smallest part of a process that can run concurrently with the other parts(other threads) of the same process. Threads are independent because they all have separate path of execution that’s the reason if an exception occurs in one thread, it doesn’t affect the execution of other threads. All threads of a process share the common memory. The process of executing multiple threads simultaneously is known as multithreading.

Let’s summarize the discussion in points:  
1. The main purpose of multithreading is to provide simultaneous execution of two or more parts of a program to maximum utilize the CPU time. A multithreaded program contains two or more parts that can run concurrently. Each such part of a program called thread.

2. Threads are lightweight sub-processes, they share the common memory space. In Multithreaded environment, programs that are benefited from multithreading, utilize the maximum CPU time so that the idle time can be kept to minimum.

3. A thread can be in one of the following states:  
NEW – A thread that has not yet started is in this state.  
RUNNABLE – A thread executing in the Java virtual machine is in this state.  
BLOCKED – A thread that is blocked waiting for a monitor lock is in this state.  
WAITING – A thread that is waiting indefinitely for another thread to perform a particular action is in this state.  
TIMED\_WAITING – A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.  
TERMINATED – A thread that has exited is in this state.  
A thread can be in only one state at a given point in time.

**Multiprocessing**

Multiprocessing refers to using multiple CPUs/processors in a single system. Multiple CPUs can act in a parallel fashion and executes multiple processes together. They increase computing power to a great extent. Symmetric multiprocessing and asymmetric multiprocessing are two types of multiprocessing.

**Difference between Multiprogramming, multitasking, multithreading and multiprocessing**

Multiprogramming – A computer running more than one program at a time (like running Excel and Firefox simultaneously).

Multiprocessing – A computer using more than one CPU at a time.

Multitasking – Tasks sharing a common resource (like 1 CPU).

Multithreading is an extension of multitasking.

**1. Multi programming –**

In a modern computing system, there are usually several concurrent application processes which want to execute. Now it is the responsibility of the Operating System to manage all the processes effectively and efficiently.  
One of the most important aspects of an Operating System is to multi program.  
In a computer system, there are multiple processes waiting to be executed, i.e. they are waiting when the CPU will be allocated to them and they begin their execution. These processes are also known as jobs. Now the main memory is too small to accommodate all of these processes or jobs into it. Thus, these processes are initially kept in an area called job pool. This job pool consists of all those processes awaiting allocation of main memory and CPU.  
CPU selects one job out of all these waiting jobs, brings it from the job pool to main memory and starts executing it. The processor executes one job until it is interrupted by some external factor or it goes for an I/O task.

Non-multi programmed system’s working –

In a non multi programmed system, As soon as one job leaves the CPU and goes for some other task (say I/O ), the CPU becomes idle. The CPU keeps waiting and waiting until this job (which was executing earlier) comes back and resumes its execution with the CPU. So CPU remains free for all this while.

Now it has a drawback that the CPU remains idle for a very long period of time. Also, other jobs which are waiting to be executed might not get a chance to execute because the CPU is still allocated to the earlier job.  
This poses a very serious problem that even though other jobs are ready to execute, CPU is not allocated to them as the CPU is allocated to a job which is not even utilizing it (as it is busy in I/O tasks).

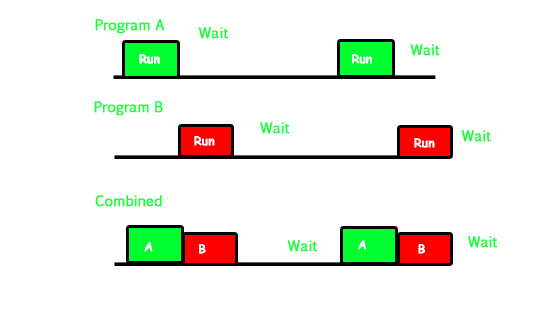
It cannot happen that one job is using the CPU for say 1 hour while the others have been waiting in the queue for 5 hours. To avoid situations like this and come up with efficient utilization of CPU, the concept of multi programming came up.

The main idea of multi programming is to maximize the CPU time.  
Multi programmed system’s working –

In a multi-programmed system, as soon as one job goes for an I/O task, the Operating System interrupts that job, chooses another job from the job pool (waiting queue), gives CPU to this new job and starts its execution. The previous job keeps doing its I/O operation while this new job does CPU bound tasks. Now say the second job also goes for an I/O task, the CPU chooses a third job and starts executing it. As soon as a job completes its I/O operation and comes back for CPU tasks, the CPU is allocated to it.

In this way, no CPU time is wasted by the system waiting for the I/O task to be completed.  
Therefore, the ultimate goal of multi programming is to keep the CPU busy as long as there are processes ready to execute. This way, multiple programs can be executed on a single processor by executing a part of a program at one time, a part of another program after this, then a part of another program and so on, hence executing multiple programs. Hence, the CPU never remains idle.

In the image below, program A runs for some time and then goes to waiting state. In the mean time program B begins its execution. So the CPU does not waste its resources and gives program B an opportunity to run.



**2. Multiprocessing –**

In a uni-processor system, only one process executes at a time.  
Multiprocessing is the use of two or more CPUs (processors) within a single Computer system. The term also refers to the ability of a system to support more than one processor within a single computer system. Now since there are multiple processors available, multiple processes can be executed at a time. These multi processors share the computer bus, sometimes the clock, memory and peripheral devices also.

Multi processing system’s working –

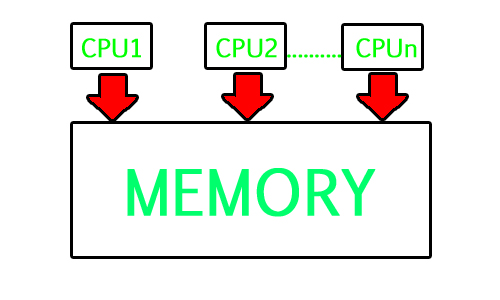
With the help of multiprocessing, many processes can be executed simultaneously. Say processes P1, P2, P3 and P4 are waiting for execution. Now in a single processor system, firstly one process will execute, then the other, then the other and so on.

But with multiprocessing, each process can be assigned to a different processor for its execution. If its a dual-core processor (2 processors), two processes can be executed simultaneously and thus will be two times faster, similarly a quad core processor will be four times as fast as a single processor.

**Why use multi processing –**

The main advantage of multiprocessor system is to get more work done in a shorter period of time. These types of systems are used when very high speed is required to process a large volume of data. Multi processing systems can save money in comparison to single processor systems because the processors can share peripherals and power supplies.

It also provides increased reliability in the sense that if one processor fails, the work does not halt, it only slows down. e.g. if we have 10 processors and 1 fails, then the work does not halt, rather the remaining 9 processors can share the work of the 10th processor. Thus the whole system runs only 10 percent slower, rather than failing altogether.



Multiprocessing refers to the hardware (i.e., the CPU units) rather than the software (i.e., running processes). If the underlying hardware provides more than one processor then that is multiprocessing. It is the ability of the system to leverage multiple processors’ computing power.

**Difference between Multi programming and Multi processing –**

A System can be both multi programmed by having multiple programs running at the same time and multiprocessing by having more than one physical processor. The difference between multiprocessing and multi programming is that Multiprocessing is basically executing multiple processes at the same time on multiple processors, whereas multi programming is keeping several programs in main memory and executing them concurrently using a single CPU only.

Multiprocessing occurs by means of parallel processing whereas Multi programming occurs by switching from one process to other (phenomenon called as context switching).

**3. Multitasking –**

As the name itself suggests, multi tasking refers to execution of multiple tasks (say processes, programs, threads etc.) at a time. In the modern operating systems, we are able to play MP3 music, edit documents in Microsoft Word, surf the Google Chrome all simultaneously, this is accomplished by means of multi tasking.

Multitasking is a logical extension of multi programming. The major way in which multitasking differs from multi programming is that multi programming works solely on the concept of context switching whereas multitasking is based on time sharing alongside the concept of context switching.

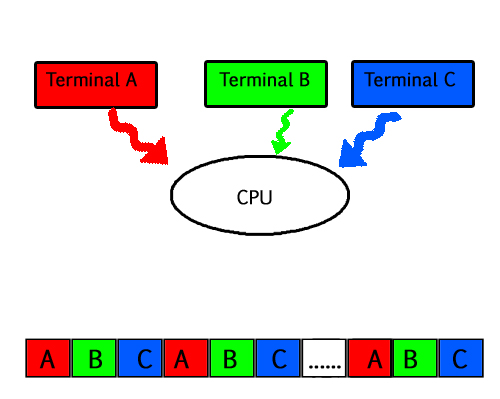
Multi tasking system’s working –

In a time sharing system, each process is assigned some specific quantum of time for which a process is meant to execute. Say there are 4 processes P1, P2, P3, P4 ready to execute. So each of them are assigned some time quantum for which they will execute e.g time quantum of 5 nanoseconds (5 ns). As one process begins execution (say P2), it executes for that quantum of time (5 ns). After 5 ns the CPU starts the execution of the other process (say P3) for the specified quantum of time.

Thus the CPU makes the processes to share time slices between them and execute accordingly. As soon as time quantum of one process expires, another process begins its execution.

Here also basically a context switch is occurring but it is occurring so fast that the user is able to interact with each program separately while it is running. This way, the user is given the illusion that multiple processes/ tasks are executing simultaneously. But actually only one process/ task is executing at a particular instant of time. In multitasking, time sharing is best manifested because each running process takes only a fair quantum of the CPU time.

In a more general sense, multitasking refers to having multiple programs, processes, tasks, threads running at the same time. This term is used in modern operating systems when multiple tasks share a common processing resource (e.g., CPU and Memory).



As depicted in the above image, At any time the CPU is executing only one task while other tasks are waiting for their turn. The illusion of parallelism is achieved when the CPU is reassigned to another task. i.e all the three tasks A, B and C are appearing to occur simultaneously because of time sharing.

So for multitasking to take place, firstly there should be multiprogramming i.e. presence of multiple programs ready for execution. And secondly the concept of time sharing.

**4. Multi threading –**

A thread is a basic unit of CPU utilization. Multi threading is an execution model that allows a single process to have multiple code segments (i.e., threads) running concurrently within the “context” of that process.  
e.g. VLC media player, where one thread is used for opening the VLC media player, one thread for playing a particular song and another thread for adding new songs to the playlist.

Multi threading is the ability of a process to manage its use by more than one user at a time and to manage multiple requests by the same user without having to have multiple copies of the program.

Multi threading system’s working –

Example 1 –

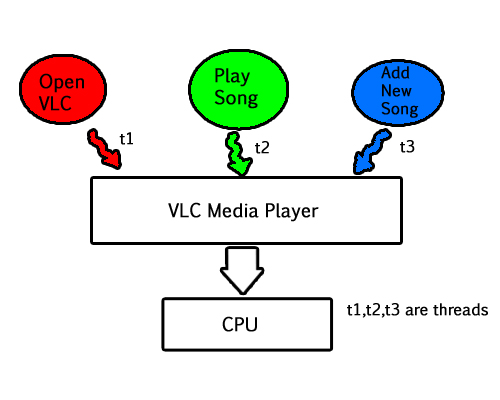
Say there is a web server which processes client requests. Now if it executes as a single threaded process, then it will not be able to process multiple requests at a time. Firstly one client will make its request and finish its execution and only then, the server will be able to process another client request. This is really costly, time consuming and tiring task. To avoid this, multi threading can be made use of.

Now, whenever a new client request comes in, the web server simply creates a new thread for processing this request and resumes its execution to hear more client requests. So the web server has the task of listening to new client requests and creating threads for each individual request. Each newly created thread processes one client request, thus reducing the burden on web server.

Example 2 –

We can think of threads as child processes that share the parent process resources but execute independently. Now take the case of a GUI. Say we are performing a calculation on the GUI (which is taking very long time to finish). Now we can not interact with the rest of the GUI until this command finishes its execution. To be able to interact with the rest of the GUI, this command of calculation should be assigned to a separate thread. So at this point of time, 2 threads will be executing i.e. one for calculation, and one for the rest of the GUI. Hence here in a single process, we used multiple threads for multiple functionality.

The image below completely describes the VLC player example:



**Advantages of Multi threading –**

Benefits of Multi threading include increased responsiveness. Since there are multiple threads in a program, so if one thread is taking too long to execute or if it gets blocked, the rest of the threads keep executing without any problem. Thus the whole program remains responsive to the user by means of remaining threads.

Another advantage of multi threading is that it is less costly. Creating brand new processes and allocating resources is a time consuming task, but since threads share resources of the parent process, creating threads and switching between them is comparatively easy. Hence multi threading is the need of modern Operating Systems.

**Garbage Collection in Java**

**Introduction**

In C/C++, programmer is responsible for both creation and destruction of objects. Usually programmer neglects destruction of useless objects. Due to this negligence, at certain point, for creation of new objects, sufficient memory may not be available and entire program will terminate abnormally causing OutOfMemoryErrors.

But in Java, the programmer need not to care for all those objects which are no longer in use. Garbage collector destroys these objects.

Garbage collector is best example of [Daemon thread](https://www.geeksforgeeks.org/daemon-thread-java/) as it is always running in background.

Main objective of Garbage Collector is to free heap memory by destroying unreachable objects.

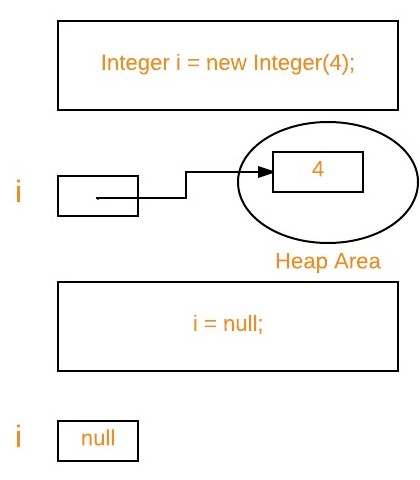
Unreachable objects : An object is said to be unreachable if it doesn’t contain any reference to it. Also note that objects which are part of [island of isolation](https://www.geeksforgeeks.org/island-of-isolation-in-java/) are also unreachable.

Integer i = new Integer(4);

// the new Integer object is reachable via the reference in 'i'

i = null;

// the Integer object is no longer reachable.



Eligibility for garbage collection : An object is said to be eligible for GC(garbage collection) iff it is unreachable. In above image, after i = null; integer object 4 in heap area is eligible for garbage collection.

Ways to make an object eligible for GC

Even though the programmer is not responsible to destroy useless objects but it is highly recommended to make an object unreachable(thus eligible for GC) if it is no longer required.

There are generally four different ways to make an object eligible for garbage collection.

Nullifying the reference variable

Re-assigning the reference variable

Object created inside method

[Island of Isolation](https://www.geeksforgeeks.org/island-of-isolation-in-java/)

All above ways with examples are discussed in separate article : [How to make object eligible for garbage collection](https://www.geeksforgeeks.org/how-to-make-object-eligible-for-garbage-collection/)

Ways for requesting [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/) to run Garbage Collector

Once we made object eligible for garbage collection, it may not destroy immediately by the garbage collector. Whenever JVM runs the Garbage Collector program, then only the object will be destroyed. But when JVM runs Garbage Collector, we can not expect.

We can also request JVM to run Garbage Collector. There are two ways to do it :

Using System.gc() method : System class contain static method gc() for requesting JVM to run Garbage Collector.

Using Runtime.getRuntime().gc() method : [Runtime class](https://www.geeksforgeeks.org/java-lang-runtime-class-in-java/) allows the application to interface with the JVM in which the application is running. Hence by using its gc() method, we can request JVM to run Garbage Collector.

|  |
| --- |
| // Java program to demonstrate requesting  // JVM to run Garbage Collector  public class Test  {      public static void main(String[] args) throws InterruptedException      {          Test t1 = new Test();          Test t2 = new Test();            // Nullifying the reference variable          t1 = null;            // requesting JVM for running Garbage Collector          System.gc();            // Nullifying the reference variable          t2 = null;            // requesting JVM for running Garbage Collector          Runtime.getRuntime().gc();        }        @Override      // finalize method is called on object once      // before garbage collecting it      protected void finalize() throws Throwable      {          System.out.println("Garbage collector called");          System.out.println("Object garbage collected : " + this);      }  } |

Output:

Garbage collector called

Object garbage collected : Test@46d08f12

Garbage collector called

Object garbage collected : Test@481779b8

Note :

There is no guarantee that any one of above two methods will definitely run Garbage Collector.

The call System.gc() is effectively equivalent to the call : Runtime.getRuntime().gc()

Finalization

Just before destroying an object, Garbage Collector calls finalize() method on the object to perform cleanup activities. Once finalize() method completes, Garbage Collector destroys that object.

finalize() method is present in [Object class](https://www.geeksforgeeks.org/object-class-in-java/) with following prototype.

protected void finalize() throws Throwable

Based on our requirement, we can override finalize() method for perform our cleanup activities like closing connection from database.

Note :

The finalize() method called by Garbage Collector not [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/). Although Garbage Collector is one of the module of JVM.

[Object class](https://www.geeksforgeeks.org/object-class-in-java/) finalize() method has empty implementation, thus it is recommended to override finalize() method to dispose of system resources or to perform other cleanup.

The finalize() method is never invoked more than once for any given object.

If an uncaught exception is thrown by the finalize() method, the exception is ignored and finalization of that object terminates.

For examples on finalize() method, please see [Output of Java programs | Set 10 (Garbage Collection)](https://www.geeksforgeeks.org/output-of-java-programs-set-10-garbage-collection/)

Let’s take a real-life example, where we use the concept of garbage collector.

Suppose you go for the internship at GeeksForGeeks and their you were told to write a program, to count the number of Employees working in the company(excluding interns).To make this program, you have to use the concept of a garbage collector.  
This is the actual task you were given at the company:-

Q.Write a program to create a class called Employee having the following data members.  
1.An ID for storing unique id allocated to every employee.  
2.Name of employee.  
3.age of an employee.

Also, provide the following methods-

A parameterized constructor to initialize name and age. The ID should be initialized in this constructor.

A method show() to display ID, name, and age.

A method showNextId() to display the ID of the next employee.

Now any beginner, who doesn’t have knowledge on garbage collector will code like this:

|  |
| --- |
| //Program to count number  //of employees working  //in a company    class Employee  {      private int ID;      private String name;      private int age;      private static int nextId=1;      //it is made static because it      // is keep common among all and      // shared by all objects      public Employee(String name,int age)      {          this.name = name;          this.age = age;          this.ID = nextId++;      }      public void show()      {          System.out.println          ("Id="+ID+"\nName="+name+"\nAge="+age);      }      public void showNextId()      {          System.out.println          ("Next employee id will be="+nextId);      }  }  class UseEmployee  {      public static void main(String []args)      {          Employee E=new Employee("GFG1",56);          Employee F=new Employee("GFG2",45);          Employee G=new Employee("GFG3",25);          E.show();          F.show();          G.show();          E.showNextId();          F.showNextId();          G.showNextId();                { //It is sub block to keep              // all those interns.              Employee X=new Employee("GFG4",23);              Employee Y=new Employee("GFG5",21);              X.show();              Y.show();              X.showNextId();              Y.showNextId();          }          //After countering this brace, X and Y          //will be removed.Therefore,          //now it should show nextId as 4.          E.showNextId();//Output of this line          //should be 4 but it will give 6 as output.      }  } |

Output:

Id=1

Name=GFG1

Age=56

Id=2

Name=GFG2

Age=45

Id=3

Name=GFG3

Age=25

Next employee id will be=4

Next employee id will be=4

Next employee id will be=4

Id=4

Name=GFG4

Age=23

Id=5

Name=GFG5

Age=21

Next employee id will be=6

Next employee id will be=6

Next employee id will be=6

Now to get the correct output:  
Now garbage collector(gc) will see 2 objects free. Now to decrement nextId,gc(garbage collector) will call method finalize() only when we programmers have override it in our class. And as mentioned previously, we have to request gc(garbage collector) and for this, we have to write the following 3 steps before closing brace of sub-block.

Set references to null(i.e X = Y = null;)

Call, System.gc();

Call, System.runFinalization();

Now the correct code for counting the number of employees(excluding interns)

|  |
| --- |
| // Correct code to count number  // of employees excluding interns.  class Employee  {      private int ID;      private String name;      private int age;      private static int nextId=1;      //it is made static because it      // is keep common among all and      // shared by all objects      public Employee(String name,int age)      {          this.name = name;          this.age = age;          this.ID = nextId++;      }      public void show()      {          System.out.println          ("Id="+ID+"\nName="+name+"\nAge="+age);      }      public void showNextId()      {          System.out.println          ("Next employee id will be="+nextId);      }      protected void finalize()      {          --nextId;          //In this case,          //gc will call finalize()          //for 2 times for 2 objects.      }  }    // it is closing brace of Employee class  class UseEmployee  {      public static void main(String []args)      {          Employee E=new Employee("GFG1",56);          Employee F=new Employee("GFG2",45);          Employee G=new Employee("GFG3",25);          E.show();          F.show();          G.show();          E.showNextId();          F.showNextId();          G.showNextId();            {              //It is sub block to keep              // all those interns.              Employee X=new Employee("GFG4",23);              Employee Y=new Employee("GFG5",21);              X.show();              Y.show();              X.showNextId();              Y.showNextId();              X = Y = null;              System.gc();              System.runFinalization();          }      E.showNextId();      }  } |

Output:

Id=1

Name=GFG1

Age=56

Id=2

Name=GFG2

Age=45

Id=3

Name=GFG3

Age=25

Next employee id will be=4

Next employee id will be=4

Next employee id will be=4

Id=4

Name=GFG4

Age=23

Id=5

Name=GFG5

Age=21

Next employee id will be=6

Next employee id will be=6

Next employee id will be=4

**Java Thread start() method**

The start() method of thread class is used to begin the execution of thread. The result of this method is two threads that are running concurrently: the current thread (which returns from the call to the start method) and the other thread (which executes its run method).

The start() method internally calls the run() method of Runnable interface to execute the code specified in the run() method in a separate thread.

The start thread performs the following tasks:

It stats a new thread

The thread moves from New State to Runnable state.

When the thread gets a chance to execute, its target run() method will run.

Syntax

public void start()

Return value

It does not return any value.

Exception

IllegalThreadStateException - This exception throws if the start() method is called more than one times.

Example 1: By Extending thread class

public class StartExp1 extends Thread

{

    public void run()

    {

        System.out.println("Thread is running...");

    }

    public static void main(String args[])

    {

        StartExp1 t1=new StartExp1();

        // this will call run() method

        t1.start();

    }

}

Output:

Thread is running...

Example 2: By Implementing Runnable Interface

public class StartExp2  implements Runnable

{

    public void run()

    {

        System.out.println("Thread is running...");

    }

    public static void main(String args[])

    {

        StartExp2  m1=new StartExp2 ();

        Thread t1 =new Thread(m1);

        // this will call run() method

        t1.start();

    }

}

[Test it Now](https://compiler.javatpoint.com/opr/test.jsp?filename=StartExp2)

Output:

Thread is running...

Example 3: When you call the start() method more than one time

public class StartExp3 extends Thread

{

    public void run()

    {

    System.out.println("First thread running...");

    }

    public static void main(String args[])

    {

        StartExp3 t1=new StartExp3();

        t1.start();

        // It will through an exception because you are calling start() method more than one time

        t1.start();

    }

}

[Test it Now](https://compiler.javatpoint.com/opr/test.jsp?filename=StartExp3)

Output:

First thread running...

Exception in thread "main" java.lang.IllegalThreadStateException

at java.lang.Thread.start(Thread.java:708)

at StartExp3.main(StartExp3.java:12)

**Priority of a Thread (Thread Priority)**

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler 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. Note that not only JVM a Java programmer can also assign the priorities of a thread explicitly in a Java program.

Setter & Getter Method of Thread Priority

Let's discuss the setter and getter method of the thread priority.

public final int getPriority(): The java.lang.Thread.getPriority() method returns the priority of the given thread.

public final void setPriority(int newPriority): The java.lang.Thread.setPriority() method updates or assign the priority of the thread to newPriority. The method throws IllegalArgumentException if the value newPriority goes out of the range, which is 1 (minimum) to 10 (maximum).

3 constants defined in Thread class:

public static int MIN\_PRIORITY

public static int NORM\_PRIORITY

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.