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.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads 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.

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.

Methods for Java Daemon thread by Thread class

The java.lang.Thread class provides two methods for java daemon thread.

No.	Method	Description
1)	public void setDaemon(boolean status)	is used to mark the current thread as daemon thread or user thread.
2)	public boolean isDaemon()	is used to check that current is daemon.

Simple example of Daemon thread in java

File: MyThread.java

```
1.  public class TestDaemonThread1 extends Thread{
2.   public void run(){
3.    if(Thread.currentThread().isDaemon()){//checking for daemon thread
4.     System.out.println("daemon thread work");
5.    }
6.    else{
7.    System.out.println("user thread work");
8.   }
9.   }
10.  public static void main(String[] args){
11.   TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
12.   TestDaemonThread1 t2=new TestDaemonThread1();
13.   TestDaemonThread1 t3=new TestDaemonThread1();
```

```
14.
15.   t1.setDaemon(true);//now t1 is daemon thread
16.
17.   t1.start();//starting threads
18.   t2.start();
19.   t3.start();
20.  }
21. }
```

Output:

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

Note: If you want to make a user thread as Daemon, it must not be started otherwise it will throw IllegalThreadStateException.

File: MyThread.java

```
1.   class TestDaemonThread2 extends Thread{
2.    public void run(){
3.     System.out.println("Name: "+Thread.currentThread().getName());
4.     System.out.println("Daemon: "+Thread.currentThread().isDaemon());
5.    }
6.
7.    public static void main(String[] args){
8.     TestDaemonThread2 t1=new TestDaemonThread2();
9.     TestDaemonThread2 t2=new TestDaemonThread2();
10.   t1.start();
11.   t1.setDaemon(true);//will throw exception here
12.   t2.start();
13.  }
14. }
```

Output:

```
exception in thread main: java.lang.IllegalThreadStateException
```

Java Thread Pool

Java Thread pool represents a group of worker threads that are waiting for the job and reused many times.

In the case of a thread pool, a group of fixed-size threads is created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, the thread is contained in the thread pool again.

Thread Pool Methods

newFixedThreadPool(int s): The method creates a thread pool of the fixed size s.

newCachedThreadPool(): The method creates a new thread pool that creates the new threads when needed but will still use the previously created thread whenever they are available to use.

newSingleThreadExecutor(): The method creates a new thread.

Advantage of Java Thread Pool

Better performance It saves time because there is no need to create a new thread.

Real time usage

It is used in Servlet and JSP where the container creates a thread pool to process the request.

Example of Java Thread Pool

Let's see a simple example of the Java thread pool using ExecutorService and Executors.

File: WorkerThread.java

```
1. import java.util.concurrent.ExecutorService;
2. import java.util.concurrent.Executors;
3. class WorkerThread implements Runnable {
4.     private String message;
5.     public WorkerThread(String s){
6.         this.message=s;
7.     }
8.      public void run() {
9.         System.out.println(Thread.currentThread().getName()+" (Start) message = "+message);
10.        processmessage();//call processmessage method that sleeps the thread for 2 seconds
11.        System.out.println(Thread.currentThread().getName()+" (End)");//prints thread name
12.    }
13.    private void processmessage() {
14.        try {  Thread.sleep(2000);  } catch (InterruptedException e) { e.printStackTrace(); }
15.    }
16. }
```

File: TestThreadPool.java

```
1. public class TestThreadPool {
2.     public static void main(String[] args) {
3.         ExecutorService executor = Executors.newFixedThreadPool(5);//creating a pool of 5 threads
4.         for (int i = 0; i < 10; i++) {
5.             Runnable worker = new WorkerThread("" + i);
6.             executor.execute(worker);//calling execute method of ExecutorService
7.           }
8.         executor.shutdown();
9.         while (!executor.isTerminated()) {   }
```

```
10.
11.         System.out.println("Finished all threads");
12.     }
13.  }
```

Output:

```
pool-1-thread-1 (Start) message = 0
pool-1-thread-2 (Start) message = 1
pool-1-thread-3 (Start) message = 2
pool-1-thread-5 (Start) message = 4
pool-1-thread-4 (Start) message = 3
pool-1-thread-2 (End)
pool-1-thread-2 (Start) message = 5
pool-1-thread-1 (End)
pool-1-thread-1 (Start) message = 6
pool-1-thread-3 (End)
pool-1-thread-3 (Start) message = 7
pool-1-thread-4 (End)
pool-1-thread-4 (Start) message = 8
pool-1-thread-5 (End)
pool-1-thread-5 (Start) message = 9
pool-1-thread-2 (End)
pool-1-thread-1 (End)
pool-1-thread-4 (End)
pool-1-thread-3 (End)
pool-1-thread-5 (End)
Finished all threads
```

Thread Pool Example: 2

Let's see another example of the thread pool.

FileName: ThreadPoolExample.java

```
1.  // important import statements
2.  import java.util.Date;
3.  import java.util.concurrent.ExecutorService;
4.  import java.util.concurrent.Executors;
5.  import java.text.SimpleDateFormat;
6.
7.
8.  class Tasks implements Runnable
9.  {
10. private String taskName;
11.
12. // constructor of the class Tasks
13. public Tasks(String str)
14. {
15. // initializing the field taskName
16. taskName = str;
17. }
18.
19. // Printing the task name and then sleeps for 1 sec
```

```
20. // The complete process is getting repeated five times
21. public void run()
22. {
23. try
24. {
25. for (int j = 0; j <= 5; j++)
26. {
27. if (j == 0)
28. {
29. Date dt = new Date();
30. SimpleDateFormat sdf = new SimpleDateFormat("hh : mm : ss");
31.
32. //prints the initialization time for every task
33. System.out.println("Initialization time for the task name: "+ taskName + " = " + sdf.format(dt));
34.
35. }
36. else
37. {
38. Date dt = new Date();
39. SimpleDateFormat sdf = new SimpleDateFormat("hh : mm : ss");
40.
41. // prints the execution time for every task
42. System.out.println("Time of execution for the task name: " + taskName + " = " +sdf.format(dt));
43.
44. }
45.
46. // 1000ms = 1 sec
47. Thread.sleep(1000);
48. }
49.
50. System.out.println(taskName + " is complete.");
51. }
52.
53. catch(InterruptedException ie)
54. {
55. ie.printStackTrace();
56. }
57. }
58. }
59.
60. public class ThreadPoolExample
61. {
62. // Maximum number of threads in the thread pool
63. static final int MAX_TH = 3;
64.
65. // main method
66. public static void main(String argvs[])
67. {
68. // Creating five new tasks
69. Runnable rb1 = new Tasks("task 1");
70. Runnable rb2 = new Tasks("task 2");
```

```
71. Runnable rb3 = new Tasks("task 3");
72. Runnable rb4 = new Tasks("task 4");
73. Runnable rb5 = new Tasks("task 5");
74.
75. // creating a thread pool with MAX_TH number of
76. // threads size the pool size is fixed
77. ExecutorService pl = Executors.newFixedThreadPool(MAX_TH);
78.
79. // passes the Task objects to the pool to execute (Step 3)
80. pl.execute(rb1);
81. pl.execute(rb2);
82. pl.execute(rb3);
83. pl.execute(rb4);
84. pl.execute(rb5);
85.
86. // pool is shutdown
87. pl.shutdown();
88. }
89. }
```

Output:

```
Initialization time for the task name: task 1 = 06 : 13 : 02
Initialization time for the task name: task 2 = 06 : 13 : 02
Initialization time for the task name: task 3 = 06 : 13 : 02
Time of execution for the task name: task 1 = 06 : 13 : 04
Time of execution for the task name: task 2 = 06 : 13 : 04
Time of execution for the task name: task 3 = 06 : 13 : 04
Time of execution for the task name: task 1 = 06 : 13 : 05
Time of execution for the task name: task 2 = 06 : 13 : 05
Time of execution for the task name: task 3 = 06 : 13 : 05
Time of execution for the task name: task 1 = 06 : 13 : 06
Time of execution for the task name: task 2 = 06 : 13 : 06
Time of execution for the task name: task 3 = 06 : 13 : 06
Time of execution for the task name: task 1 = 06 : 13 : 07
Time of execution for the task name: task 2 = 06 : 13 : 07
Time of execution for the task name: task 3 = 06 : 13 : 07
Time of execution for the task name: task 1 = 06 : 13 : 08
Time of execution for the task name: task 2 = 06 : 13 : 08
Time of execution for the task name: task 3 = 06 : 13 : 08
task 2 is complete.
Initialization time for the task name: task 4 = 06 : 13 : 09
task 1 is complete.
Initialization time for the task name: task 5 = 06 : 13 : 09
task 3 is complete.
Time of execution for the task name: task 4 = 06 : 13 : 10
Time of execution for the task name: task 5 = 06 : 13 : 10
Time of execution for the task name: task 4 = 06 : 13 : 11
Time of execution for the task name: task 5 = 06 : 13 : 11
Time of execution for the task name: task 4 = 06 : 13 : 12
Time of execution for the task name: task 5 = 06 : 13 : 12
Time of execution for the task name: task 4 = 06 : 13 : 13
Time of execution for the task name: task 5 = 06 : 13 : 13
Time of execution for the task name: task 4 = 06 : 13 : 14
Time of execution for the task name: task 5 = 06 : 13 : 14
task 4 is complete.
task 5 is complete.
```

Explanation: It is evident by looking at the output of the program that tasks 4 and 5 are executed only when the thread has an idle thread. Until then, the extra tasks are put in the queue.

The takeaway from the above example is when one wants to execute 50 tasks but is not willing to create 50 threads. In such a case, one can create a pool of 10 threads. Thus, 10 out of 50 tasks are assigned, and the rest are put in the queue. Whenever any thread out of 10 threads becomes idle, it picks up the 11^{th} task. The other pending tasks are treated the same way.

Risks involved in Thread Pools

The following are the risk involved in the thread pools.

Deadlock: It is a known fact that deadlock can come in any program that involves multithreading, and a thread pool introduces another scenario of deadlock. Consider a scenario where all the threads that are executing are waiting for the results from the threads that are blocked and waiting in the queue because of the non-availability of threads for the execution.

Thread Leakage: Leakage of threads occurs when a thread is being removed from the pool to execute a task but is not returning to it after the completion of the task. For example, when a thread throws the exception and the pool class is not able to catch this exception, then the thread exits and reduces the thread pool size by 1. If the same thing repeats a number of times, then there are fair chances that the pool will become empty, and hence, there are no threads available in the pool for executing other requests.

Resource Thrashing: A lot of time is wasted in context switching among threads when the size of the thread pool is very large. Whenever there are more threads than the optimal number may cause the starvation problem, and it leads to resource thrashing.

Points to Remember

Do not queue the tasks that are concurrently waiting for the results obtained from the other tasks. It may lead to a deadlock situation, as explained above.

Care must be taken whenever threads are used for the operation that is long-lived. It may result in the waiting of thread forever and will finally lead to the leakage of the resource.

In the end, the thread pool has to be ended explicitly. If it does not happen, then the program continues to execute, and it never ends. Invoke the shutdown() method on the thread pool to terminate the executor. Note that if someone tries to send another task to the executor after shutdown, it will throw a RejectedExecutionException.

One needs to understand the tasks to effectively tune the thread pool. If the given tasks are contrasting, then one should look for pools for executing different varieties of tasks so that one can properly tune them.

To reduce the probability of running JVM out of memory, one can control the maximum threads that can run in JVM. The thread pool cannot create new threads after it has reached the maximum limit.

A thread pool can use the same used thread if the thread has finished its execution. Thus, the time and resources used for the creation of a new thread are saved.

Tuning the Thread Pool

The accurate size of a thread pool is decided by the number of available processors and the type of tasks the threads have to execute. If a system has the P processors that have only got the computation type processes, then the maximum size of the thread pool of P or P + 1 achieves the maximum efficiency. However, the tasks may have to wait for I/O, and in such a scenario, one has to take into consideration the ratio of the waiting time (W) and the service time (S) for the request; resulting in the maximum size of the pool P * (1 + W / S) for the maximum efficiency.

Conclusion

A thread pool is a very handy tool for organizing applications, especially on the server-side. Concept-wise, a thread pool is very easy to comprehend. However, one may have to look at a lot of issues when dealing with a thread pool. It is because the thread pool comes with some risks involved it (risks are discussed above).

ThreadGroup in Java

Java provides a convenient way to group multiple threads in a single object. In such a way, we can suspend, resume or interrupt a group of threads by a single method call.

Note: Now suspend(), resume() and stop() methods are deprecated.

Java thread group is implemented by *java.lang.ThreadGroup* class.

A ThreadGroup represents a set of threads. A thread group can also include the other thread group. The thread group creates a tree in which every thread group except the initial thread group has a parent.

A thread is allowed to access information about its own thread group, but it cannot access the information about its thread group's parent thread group or any other thread groups.

Constructors of ThreadGroup class

There are only two constructors of ThreadGroup class.

No.	Constructor	Description
1)	ThreadGroup(String name)	creates a thread group with given name.
2)	ThreadGroup(ThreadGroup parent, String name)	creates a thread group with a given parent group and name.

Methods of ThreadGroup class

There are many methods in ThreadGroup class. A list of ThreadGroup methods is given below.

S.N.	Modifier and Type	Method	Description

1)	void	checkAccess()	This method determines if the currently running thread has permission to modify the thread group.
2)	int	activeCount()	This method returns an estimate of the number of active threads in the thread group and its subgroups.
3)	int	activeGroupCount()	This method returns an estimate of the number of active groups in the thread group and its subgroups.
4)	void	destroy()	This method destroys the thread group and all of its subgroups.
5)	int	enumerate(Thread[] list)	This method copies into the specified array every active thread in the thread group and its subgroups.
6)	int	getMaxPriority()	This method returns the maximum priority of the thread group.
7)	String	getName()	This method returns the name of the thread group.
8)	ThreadGroup	getParent()	This method returns the parent of the thread group.
9)	void	interrupt()	This method interrupts all threads in the thread group.
10)	boolean	isDaemon()	This method tests if the thread group is a daemon thread group.
11)	void	setDaemon(boolean daemon)	This method changes the daemon status of the thread group.
12)	boolean	isDestroyed()	This method tests if this thread group has been destroyed.
13)	void	list()	This method prints information about the thread group to the standard output.
14)	boolean	parentOf(ThreadGroup g	This method tests if the thread group is either the thread group argument or one of its ancestor thread groups.
15)	void	suspend()	This method is used to suspend all threads in the thread group.
16)	void	resume()	This method is used to resume all threads in the thread group which was suspended using suspend() method.
17)	void	setMaxPriority(int pri)	This method sets the maximum priority of the group.
18)	void	stop()	This method is used to stop all threads in the thread group.
19)	String	toString()	This method returns a string representation of the Thread group.

Let's see a code to group multiple threads.

```
1. ThreadGroup tg1 = new ThreadGroup("Group A");
2. Thread t1 = new Thread(tg1,new MyRunnable(),"one");
3. Thread t2 = new Thread(tg1,new MyRunnable(),"two");
4. Thread t3 = new Thread(tg1,new MyRunnable(),"three");
```

Now all 3 threads belong to one group. Here, tg1 is the thread group name, MyRunnable is the class that implements Runnable interface and "one", "two" and "three" are the thread names.

Now we can interrupt all threads by a single line of code only.

```
1. Thread.currentThread().getThreadGroup().interrupt();
```

ThreadGroup Example

File: ThreadGroupDemo.java

```
1.  public class ThreadGroupDemo implements Runnable{
2.      public void run() {
3.           System.out.println(Thread.currentThread().getName());
4.       }
5.      public static void main(String[] args) {
6.         ThreadGroupDemo runnable = new ThreadGroupDemo();
7.           ThreadGroup tg1 = new ThreadGroup("Parent ThreadGroup");
8.
9.           Thread t1 = new Thread(tg1, runnable,"one");
10.          t1.start();
11.          Thread t2 = new Thread(tg1, runnable,"two");
12.          t2.start();
13.          Thread t3 = new Thread(tg1, runnable,"three");
14.          t3.start();
15.
16.          System.out.println("Thread Group Name: "+tg1.getName());
17.         tg1.list();
18.
19.     }
20.   }
```

Output:

```
one
two
three
Thread Group Name: Parent ThreadGroup
java.lang.ThreadGroup[name=Parent ThreadGroup,maxpri=10]
```

Thread Pool Methods Example: int activeCount()

Let's see how one can use the method activeCount().

FileName: ActiveCountExample.java

```
1.  // code that illustrates the activeCount() method
2.
3.  // import statement
4.  import java.lang.*;
5.
6.
7.  class ThreadNew extends Thread
8.  {
9.  // constructor of the  class
```

```
10. ThreadNew(String tName, ThreadGroup tgrp)
11. {
12. super(tgrp, tName);
13. start();
14. }
15.
16. // overriding the run method
17. public void run()
18. {
19.
20. for (int j = 0; j < 1000; j++)
21. {
22. try
23. {
24. Thread.sleep(5);
25. }
26. catch (InterruptedException e)
27. {
28. System.out.println("The exception has been encountered " + e);
29. }
30. }
31. }
32. }
33.
34. public class ActiveCountExample
35. {
36. // main method
37. public static void main(String argvs[])
38. {
39. // creating the thread group
40. ThreadGroup tg = new ThreadGroup("The parent group of threads");
41.
42. ThreadNew th1 = new ThreadNew("first", tg);
43. System.out.println("Starting the first");
44.
45. ThreadNew th2 = new ThreadNew("second", tg);
46. System.out.println("Starting the second");
47.
48. // checking the number of active thread by invoking the activeCount() method
49. System.out.println("The total number of active threads are: " + tg.activeCount());
50. }
51. }
```

Output:

```
Starting the first
Starting the second
The total number of active threads are: 2
```

Thread Pool Methods Example: int activeGroupCount()

Now, we will learn how one can use the activeGroupCount() method in the code.

FileName: ActiveGroupCountExample.java

```
1.  // Java code illustrating the activeGroupCount() method
2.
3.  // import statement
4.  import java.lang.*;
5.
6.
7.  class ThreadNew extends Thread
8.  {
9.  // constructor of the  class
10. ThreadNew(String tName, ThreadGroup tgrp)
11. {
12. super(tgrp, tName);
13. start();
14. }
15.
16. // overriding the run() method
17. public void run()
18. {
19.
20. for (int j = 0; j < 100; j++)
21. {
22. try
23. {
24. Thread.sleep(5);
25. }
26. catch (InterruptedException e)
27. {
28. System.out.println("The exception has been encountered " + e);
29. }
30.
31. }
32.
33. System.out.println(Thread.currentThread().getName() + " thread has finished executing");
34. }
35. }
36.
37. public class ActiveGroupCountExample
38. {
39. // main method
40. public static void main(String argvs[])
41. {
42. // creating the thread group
43. ThreadGroup tg = new ThreadGroup("The parent group of threads");
44.
45. ThreadGroup tg1 = new ThreadGroup(tg, "the child group");
46.
47. ThreadNew th1 = new ThreadNew("the first", tg);
```

```
48. System.out.println("Starting the first");
49.
50. ThreadNew th2 = new ThreadNew("the second", tg);
51. System.out.println("Starting the second");
52.
53. // checking the number of active thread by invoking the activeGroupCount() method
54. System.out.println("The total number of active thread groups are: " + tg.activeGroupCount());
55. }
56. }
```

Output:

```
Starting the first
Starting the second
The total number of active thread groups are: 1
the second thread has finished executing
the first thread has finished executing
```

Thread Pool Methods Example: void destroy()

Now, we will learn how one can use the destroy() method in the code.

FileName: DestroyExample.java

```
1.  // Code illustrating the destroy() method
2.
3.  // import statement
4.  import java.lang.*;
5.
6.
7.  class ThreadNew extends Thread
8.  {
9.  // constructor of the  class
10. ThreadNew(String tName, ThreadGroup tgrp)
11. {
12. super(tgrp, tName);
13. start();
14. }
15.
16. // overriding the run() method
17. public void run()
18. {
19.
20. for (int j = 0; j < 100; j++)
21. {
22. try
23. {
24. Thread.sleep(5);
25. }
26. catch (InterruptedException e)
27. {
```

```
28. System.out.println("The exception has been encountered " + e);
29. }
30.
31. }
32.
33. System.out.println(Thread.currentThread().getName() + " thread has finished executing");
34. }
35. }
36.
37. public class DestroyExample
38. {
39. // main method
40. public static void main(String argvs[]) throws SecurityException, InterruptedException
41. {
42. // creating the thread group
43. ThreadGroup tg = new ThreadGroup("the parent group");
44.
45. ThreadGroup tg1 = new ThreadGroup(tg, "the child group");
46.
47. ThreadNew th1 = new ThreadNew("the first", tg);
48. System.out.println("Starting the first");
49.
50. ThreadNew th2 = new ThreadNew("the second", tg);
51. System.out.println("Starting the second");
52.
53. // waiting until the other threads has been finished
54. th1.join();
55. th2.join();
56.
57. // destroying the child thread group
58. tg1.destroy();
59. System.out.println(tg1.getName() + " is destroyed.");
60.
61. // destroying the parent thread group
62. tg.destroy();
63. System.out.println(tg.getName() + " is destroyed.");
64. }
65. }
```

Output:

```
Starting the first
Starting the second
the first thread has finished executing
the second thread has finished executing
the child group is destroyed.
the parent group is destroyed.
```

Thread Pool Methods Example: int enumerate()

Now, we will learn how one can use the enumerate() method in the code.

FileName: EnumerateExample.java

```
1.  // Code illustrating the enumerate() method
2.
3.  // import statement
4.  import java.lang.*;
5.
6.
7.  class ThreadNew extends Thread
8.  {
9.  // constructor of the class
10. ThreadNew(String tName, ThreadGroup tgrp)
11. {
12. super(tgrp, tName);
13. start();
14. }
15.
16. // overriding the run() method
17. public void run()
18. {
19.
20. for (int j = 0; j < 100; j++)
21. {
22. try
23. {
24. Thread.sleep(5);
25. }
26. catch (InterruptedException e)
27. {
28. System.out.println("The exception has been encountered " + e);
29. }
30.
31. }
32.
33. System.out.println(Thread.currentThread().getName() + " thread has finished executing");
34. }
35. }
36.
37. public class EnumerateExample
38. {
39. // main method
40. public static void main(String argvs[]) throws SecurityException, InterruptedException
41. {
42. // creating the thread group
43. ThreadGroup tg = new ThreadGroup("the parent group");
44.
45. ThreadGroup tg1 = new ThreadGroup(tg, "the child group");
46.
47. ThreadNew th1 = new ThreadNew("the first", tg);
48. System.out.println("Starting the first");
49.
```

```
50. ThreadNew th2 = new ThreadNew("the second", tg);
51. System.out.println("Starting the second");
52.
53. // returning the number of threads kept in this array
54. Thread[] grp = new Thread[tg.activeCount()];
55. int cnt = tg.enumerate(grp);
56. for (int j = 0; j < cnt; j++)
57. {
58. System.out.println("Thread " + grp[j].getName() + " is found.");
59. }
60. }
61. }
```

Output:

```
Starting the first
Starting the second
Thread the first is found.
Thread the second is found.
the first thread has finished executing
the second thread has finished executing
```

Thread Pool Methods Example: int getMaxPriority()

The following code shows the working of the getMaxPriority() method.

FileName: GetMaxPriorityExample.java

```
1.  // Code illustrating the getMaxPriority() method
2.
3.  // import statement
4.  import java.lang.*;
5.
6.
7.  class ThreadNew extends Thread
8.  {
9.  // constructor of the class
10. ThreadNew(String tName, ThreadGroup tgrp)
11. {
12. super(tgrp, tName);
13. start();
14. }
15.
16. // overriding the run() method
17. public void run()
18. {
19.
20. for (int j = 0; j < 100; j++)
21. {
22. try
23. {
```

```
24. Thread.sleep(5);
25. }
26. catch (InterruptedException e)
27. {
28. System.out.println("The exception has been encountered " + e);
29. }
30.
31. }
32.
33. System.out.println(Thread.currentThread().getName() + " thread has finished executing");
34. }
35. }
36.
37. public class GetMaxPriorityExample
38. {
39. // main method
40. public static void main(String argvs[]) throws SecurityException, InterruptedException
41. {
42. // creating the thread group
43. ThreadGroup tg = new ThreadGroup("the parent group");
44.
45. ThreadGroup tg1 = new ThreadGroup(tg, "the child group");
46.
47. ThreadNew th1 = new ThreadNew("the first", tg);
48. System.out.println("Starting the first");
49.
50. ThreadNew th2 = new ThreadNew("the second", tg);
51. System.out.println("Starting the second");
52.
53. int priority = tg.getMaxPriority();
54.
55. System.out.println("The maximum priority of the parent ThreadGroup: " + priority);
56.
57.
58. }
59. }
```

Output:

```
Starting the first
Starting the second
The maximum priority of the parent ThreadGroup: 10
the first thread has finished executing
the second thread has finished executing
```

Thread Pool Methods Example: ThreadGroup getParent()

Now, we will learn how one can use the getParent() method in the code.

FileName: GetParentExample.java

```
1.  // Code illustrating the getParent() method
2.
3.  // import statement
4.  import java.lang.*;
5.
6.
7.  class ThreadNew extends Thread
8.  {
9.  // constructor of the class
10. ThreadNew(String tName, ThreadGroup tgrp)
11. {
12. super(tgrp, tName);
13. start();
14. }
15.
16. // overriding the run() method
17. public void run()
18. {
19.
20. for (int j = 0; j < 100; j++)
21. {
22. try
23. {
24. Thread.sleep(5);
25. }
26. catch (InterruptedException e)
27. {
28. System.out.println("The exception has been encountered" + e);
29. }
30.
31. }
32.
33. System.out.println(Thread.currentThread().getName() + " thread has finished executing");
34. }
35. }
36.
37. public class GetMaxPriorityExample
38. {
39. // main method
40. public static void main(String argvs[]) throws SecurityException, InterruptedException
41. {
42. // creating the thread group
43. ThreadGroup tg = new ThreadGroup("the parent group");
44.
45. ThreadGroup tg1 = new ThreadGroup(tg, "the child group");
46.
47. ThreadNew th1 = new ThreadNew("the first", tg);
48. System.out.println("Starting the first");
49.
50. ThreadNew th2 = new ThreadNew("the second", tg);
51. System.out.println("Starting the second");
```

```
52.
53. // printing the parent ThreadGroup
54. // of both child and parent threads
55. System.out.println("The ParentThreadGroup for " + tg.getName() + " is " + tg.getParent().getName());
56. System.out.println("The ParentThreadGroup for " + tg1.getName() + " is " + tg1.getParent().getName())
    ;
57.
58.
59. }
60. }
```

Output:

```
Starting the first
Starting the second
The ParentThreadGroup for the parent group is main
The ParentThreadGroup for the child group is the parent group
the first thread has finished executing
the second thread has finished executing
```

Thread Pool Methods Example: void interrupt()

The following program illustrates how one can use the interrupt() method.

FileName: InterruptExample.java

```
1.  // Code illustrating the interrupt() method
2.
3.  // import statement
4.  import java.lang.*;
5.
6.
7.  class ThreadNew extends Thread
8.  {
9.  // constructor of the class
10. ThreadNew(String tName, ThreadGroup tgrp)
11. {
12. super(tgrp, tName);
13. start();
14. }
15.
16. // overriding the run() method
17. public void run()
18. {
19.
20. for (int j = 0; j < 100; j++)
21. {
22. try
23. {
24. Thread.sleep(5);
25. }
```

```
26. catch (InterruptedException e)
27. {
28. System.out.println("The exception has been encountered " + e);
29. }
30.
31. }
32.
33. System.out.println(Thread.currentThread().getName() + " thread has finished executing");
34. }
35. }
36.
37. public class InterruptExample
38. {
39. // main method
40. public static void main(String argvs[]) throws SecurityException, InterruptedException
41. {
42. // creating the thread group
43. ThreadGroup tg = new ThreadGroup("the parent group");
44.
45. ThreadGroup tg1 = new ThreadGroup(tg, "the child group");
46.
47. ThreadNew th1 = new ThreadNew("the first", tg);
48. System.out.println("Starting the first");
49.
50. ThreadNew th2 = new ThreadNew("the second", tg);
51. System.out.println("Starting the second");
52.
53. // invoking the interrupt method
54. tg.interrupt();
55.
56. }
57. }
```

Output:

```
Starting the first
Starting the second
The exception has been encountered java.lang.InterruptedException: sleep interrupted
The exception has been encountered java.lang.InterruptedException: sleep interrupted
the second thread has finished executing
the first thread has finished executing
```

Thread Pool Methods Example: boolean isDaemon()

The following program illustrates how one can use the isDaemon() method.

FileName: IsDaemonExample.java

```
1. // Code illustrating the isDaemon() method
2.
3. // import statement
```

```
4.  import java.lang.*;
5.
6.
7.  class ThreadNew extends Thread
8.  {
9.  // constructor of the class
10. ThreadNew(String tName, ThreadGroup tgrp)
11. {
12. super(tgrp, tName);
13. start();
14. }
15.
16. // overriding the run() method
17. public void run()
18. {
19.
20. for (int j = 0; j < 100; j++)
21. {
22. try
23. {
24. Thread.sleep(5);
25. }
26. catch (InterruptedException e)
27. {
28. System.out.println("The exception has been encountered" + e);
29. }
30.
31. }
32.
33. System.out.println(Thread.currentThread().getName() + " thread has finished executing");
34. }
35. }
36.
37. public class IsDaemonExample
38. {
39. // main method
40. public static void main(String argvs[]) throws SecurityException, InterruptedException
41. {
42. // creating the thread group
43. ThreadGroup tg = new ThreadGroup("the parent group");
44.
45. ThreadGroup tg1 = new ThreadGroup(tg, "the child group");
46.
47. ThreadNew th1 = new ThreadNew("the first", tg);
48. System.out.println("Starting the first");
49.
50. ThreadNew th2 = new ThreadNew("the second", tg);
51. System.out.println("Starting the second");
52.
53. if (tg.isDaemon() == true)
54. {
```

```
55. System.out.println("The group is a daemon group.");
56. }
57. else
58. {
59. System.out.println("The group is not a daemon group.");
60. }
61.
62. }
63. }
```

Output:

```
Starting the first
Starting the second
The group is not a daemon group.
the second thread has finished executing
the first thread has finished executing
```

Thread Pool Methods Example: boolean isDestroyed()

The following program illustrates how one can use the isDestroyed() method.

FileName: IsDestroyedExample.java

```
1.  // Code illustrating the isDestroyed() method
2.
3.  // import statement
4.  import java.lang.*;
5.
6.
7.  class ThreadNew extends Thread
8.  {
9.  // constructor of the class
10. ThreadNew(String tName, ThreadGroup tgrp)
11. {
12. super(tgrp, tName);
13. start();
14. }
15.
16. // overriding the run() method
17. public void run()
18. {
19.
20. for (int j = 0; j < 100; j++)
21. {
22. try
23. {
24. Thread.sleep(5);
25. }
26. catch (InterruptedException e)
27. {
```

```
28. System.out.println("The exception has been encountered" + e);
29. }
30.
31. }
32.
33. System.out.println(Thread.currentThread().getName() + " thread has finished executing");
34. }
35. }
36.
37. public class IsDestroyedExample
38. {
39. // main method
40. public static void main(String argvs[]) throws SecurityException, InterruptedException
41. {
42. // creating the thread group
43. ThreadGroup tg = new ThreadGroup("the parent group");
44.
45. ThreadGroup tg1 = new ThreadGroup(tg, "the child group");
46.
47. ThreadNew th1 = new ThreadNew("the first", tg);
48. System.out.println("Starting the first");
49.
50. ThreadNew th2 = new ThreadNew("the second", tg);
51. System.out.println("Starting the second");
52.
53. if (tg.isDestroyed() == true)
54. {
55. System.out.println("The group has been destroyed.");
56. }
57. else
58. {
59. System.out.println("The group has not been destroyed.");
60. }
61.
62. }
63. }
```

Output:

```
Starting the first
Starting the second
The group has not been destroyed.
the first thread has finished executing
the second thread has finished executing
```

Java Shutdown Hook

A special construct that facilitates the developers to add some code that has to be run when the Java Virtual Machine (JVM) is shutting down is known as the **Java shutdown hook**. The Java shutdown hook comes in very handy in the cases where one needs to perform some special cleanup work when the JVM is shutting down. Note that handling an operation such as invoking a special method before the JVM terminates does not

work using a general construct when the JVM is shutting down due to some external factors. For example, whenever a kill request is generated by the operating system or due to resource is not allocated because of the lack of free memory, then in such a case, it is not possible to invoke the procedure. The shutdown hook solves this problem comfortably by providing an arbitrary block of code.

Taking at a surface level, learning about the shutdown hook is straightforward. All one has to do is to derive a class using the java.lang.Thread class, and then provide the code for the task one wants to do in the run() method when the JVM is going to shut down. For registering the instance of the derived class as the shutdown hook, one has to invoke the method Runtime.getRuntime().addShutdownHook(Thread), whereas for removing the already registered shutdown hook, one has to invoke the removeShutdownHook(Thread) method.

In nutshell, the shutdown hook can be used to perform cleanup resources or save the state when JVM shuts down normally or abruptly. Performing clean resources means closing log files, sending some alerts, or something else. So if you want to execute some code before JVM shuts down, use the shutdown hook.

When does the JVM shut down?

The JVM shuts down when:

- user presses ctrl+c on the command prompt
- System.exit(int) method is invoked
- user logoff
- user shutdown etc.

The addShutdownHook(Thread hook) method

The addShutdownHook() method of the Runtime class is used to register the thread with the Virtual Machine.

Syntax:

```
1. public void addShutdownHook(Thread hook){}
```

The object of the Runtime class can be obtained by calling the static factory method getRuntime(). For example:

```
1. Runtime r = Runtime.getRuntime();
```

The removeShutdownHook(Thread hook) method

The removeShutdownHook() method of the Runtime class is invoked to remove the registration of the already registered shutdown hooks.

Syntax:

```
1. public boolean removeShutdownHook(Thread hook){ }
```

True value is returned by the method, when the method successfully de-register the registered hooks; otherwise returns false.

Factory method

The method that returns the instance of a class is known as factory method.

Simple example of Shutdown Hook

FileName: MyThread.java

```
1. class MyThread extends Thread{
2.     public void run(){
3.         System.out.println("shut down hook task completed..");
4.     }
5. }
6.
7. public class TestShutdown1{
8. public static void main(String[] args)throws Exception {
9.
10. Runtime r=Runtime.getRuntime();
11. r.addShutdownHook(new MyThread());
12.
13. System.out.println("Now main sleeping... press ctrl+c to exit");
14. try{Thread.sleep(3000);}catch (Exception e) {}
15. }
16. }
```

Output:

```
Now main sleeping... press ctrl+c to exit
shut down hook task completed.
```

Same example of Shutdown Hook by anonymous class:

FileName: TestShutdown2.java

```
1. public class TestShutdown2{
2. public static void main(String[] args)throws Exception {
3.
4. Runtime r=Runtime.getRuntime();
5.
6. r.addShutdownHook(new Thread(){
7. public void run(){
8.     System.out.println("shut down hook task completed..");
9.     }
10. }
11. );
12.
13. System.out.println("Now main sleeping... press ctrl+c to exit");
14. try{Thread.sleep(3000);}catch (Exception e) {}
15. }
16. }
```

Output:

```
Now main sleeping... press ctrl+c to exit
shut down hook task completed.
```

Removing the registered shutdown hook example

The following example shows how one can use the removeShutdownHook() method to remove the registered shutdown hook.

FileName: RemoveHookExample.java

```
1. public class RemoveHookExample
2. {
3.
4. // the Msg class is derived from the Thread class
5. static class Msg extends Thread
6. {
7.
8. public void run()
9. {
10. System.out.println("Bye ...");
11. }
12. }
13.
14. // main method
15. public static void main(String[] argvs)
16. {
17. try
18. {
19. // creating an object of the class Msg
20. Msg ms = new Msg();
21.
22. // registering the Msg object as the shutdown hook
23. Runtime.getRuntime().addShutdownHook(ms);
24.
25. // printing the current state of program
26. System.out.println("The program is beginning ...");
27.
28. // causing the thread to sleep for 2 seconds
29. System.out.println("Waiting for 2 seconds ...");
30. Thread.sleep(2000);
31.
32. // removing the hook
33. Runtime.getRuntime().removeShutdownHook(ms);
34.
35. // printing the message program is terminating
36. System.out.println("The program is terminating ...");
37. }
38. catch (Exception ex)
39. {
40. ex.printStackTrace();
41. }
```

42. }
43. }

Output:

```
The program is beginning ...
Waiting for 2 seconds ...
The program is terminating ...
```

Points to Remember

There are some important points to keep in mind while working with the shutdown hook.

No guarantee for the execution of the shutdown hooks: The first and the most important thing to keep in mind is that there is no certainty about the execution of the shutdown hook. In some scenarios, the shutdown hooks will not execute at all. For example, if the JVM gets crashed due to some internal error, then there is no scope for the shutdown hooks. When the operating system gives the SYSKILL signal, then also it is not possible for the shutdown hooks to come into picture.

Note that when the application is terminated normally the shutdown hooks are called (all threads of the application is finished or terminated). Also, when the operating system is shut down or the user presses the ctrl + c the shutdown hooks are invoked.

Before completion, the shutdown hooks can be stopped forcefully: It is a special case of the above discussed point. Whenever a shutdown hooks start to execute, one can forcefully terminate it by shutting down the system. In this case, the operating system for a specific amount of time. If the job is not done in that frame of time, then the system has no other choice than to forcefully terminate the running hooks.

There can be more than one shutdown hooks, but there execution order is not guaranteed: The JVM can execute the shutdown hooks in any arbitrary order. Even concurrent execution of the shutdown hooks are also possible.

Within shutdown hooks, it is not allowed to unregister or register the shutdown hooks: When the JVM initiates the shutdown sequence, one can not remove or add more any existing shutdown hooks. If one tries to do so, the IllegalStateException is thrown by the JVM.

The Runtime.halt() can stop the shutdown sequence that has been started: Only the Runtime.halt(), which terminates the JVM forcefully, can stop the started shutdown sequence, which also means that invoking the System.exit() method will not work within a shutdown hook.

Security permissions are required when using shutdown hooks: If one is using the Java Security Managers, then the Java code that is responsible for removing or adding the shutdown hooks need to get the shutdown hooks permission at the runtime. If one invokes the method without getting the permission in the secure environment, then it will raise the SecurityException.

How to perform single task by multiple threads in Java?

If you have to perform a single task by many threads, have only one run() method. For example:

Program of performing single task by multiple threads

FileName: TestMultitasking1.java

```
1. class TestMultitasking1 extends Thread{
2.  public void run(){
3.    System.out.println("task one");
4.  }
5.  public static void main(String args[]){
6.   TestMultitasking1 t1=new TestMultitasking1();
7.   TestMultitasking1 t2=new TestMultitasking1();
8.   TestMultitasking1 t3=new TestMultitasking1();
9.
10.  t1.start();
11.  t2.start();
12.  t3.start();
13. }
14. }
```

Output:

```
task one
task one
task one
```

Program of performing single task by multiple threads

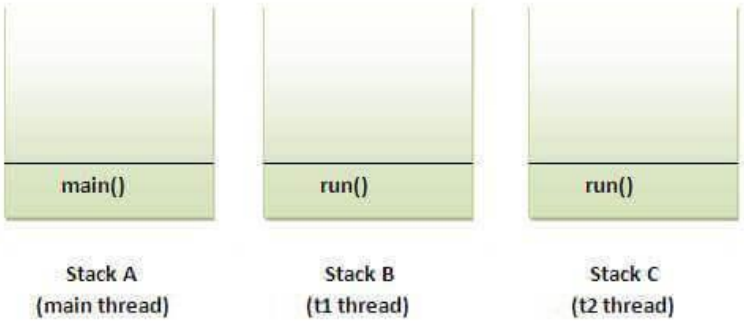
FileName: TestMultitasking2.java

```
1. class TestMultitasking2 implements Runnable{
2. public void run(){
3. System.out.println("task one");
4. }
5.
6. public static void main(String args[]){
7. Thread t1 =new Thread(new TestMultitasking2());//passing annonymous object of TestMultitasking2 cla
   ss
8. Thread t2 =new Thread(new TestMultitasking2());
9.
10. t1.start();
11. t2.start();
12.
13.  }
14. }
```

Output:

```
task one
task one
```

Note: Each thread run in a separate callstack.



How to perform multiple tasks by multiple threads (multitasking in multithreading)?

If you have to perform multiple tasks by multiple threads,have multiple run() methods.For example:

Program of performing two tasks by two threads

FileName: TestMultitasking3.java

```
1.  class Simple1 extends Thread{
2.   public void run(){
3.     System.out.println("task one");
4.   }
5.  }
6.
7.  class Simple2 extends Thread{
8.   public void run(){
9.     System.out.println("task two");
10.  }
11. }
12.
13.  class TestMultitasking3{
14.  public static void main(String args[]){
15.   Simple1 t1=new Simple1();
16.   Simple2 t2=new Simple2();
17.
18.   t1.start();
19.   t2.start();
20.  }
21. }
```

Output:

```
task one
```

```
task two
```

Same example as above by anonymous class that extends Thread class:

Program of performing two tasks by two threads

FileName: TestMultitasking4.java

```
1.  class TestMultitasking4{
2.   public static void main(String args[]){
3.    Thread t1=new Thread(){
4.     public void run(){
5.      System.out.println("task one");
6.     }
7.    };
8.    Thread t2=new Thread(){
9.     public void run(){
10.     System.out.println("task two");
11.    }
12.   };
13.
14.
15.   t1.start();
16.   t2.start();
17.  }
18. }
```

Output:

```
task one
task two
```

Same example as above by anonymous class that implements Runnable interface:

Program of performing two tasks by two threads

FileName: TestMultitasking5.java

```
1.  class TestMultitasking5{
2.   public static void main(String args[]){
3.    Runnable r1=new Runnable(){
4.     public void run(){
5.      System.out.println("task one");
6.     }
7.    };
8.
9.    Runnable r2=new Runnable(){
10.    public void run(){
11.     System.out.println("task two");
12.    }
13.   };
```

```
14.
15.   Thread t1=new Thread(r1);
16.   Thread t2=new Thread(r2);
17.
18.   t1.start();
19.   t2.start();
20.  }
21. }
```

Output:

```
task one
task two
```

Printing even and odd numbers using two threads

To print the even and odd numbers using the two threads, we will use the synchronized block and the notify() method. Observe the following program.

FileName: OddEvenExample.java

```
1.  // Java program that prints the odd and even numbers using two threads.
2.  // the time complexity of the program is O(N), where N is the number up to which we
3.  // are displaying the numbers
4.  public class OddEvenExample
5.  {
6.  // Starting the counter
7.  int contr = 1;
8.  static int NUM;
9.  // Method for printing the odd numbers
10. public void displayOddNumber()
11. {
12. // note that synchronized blocks are necessary for the code for getting the desired
13. // output. If we remove the synchronized blocks, we will get an exception.
14. synchronized (this)
15. {
16. // Printing the numbers till NUM
17. while (contr < NUM)
18. {
19. // If the contr is even then display
20. while (contr % 2 == 0)
21. {
22. // handling the exception handle
23. try
24. {
25. wait();
26. }
27. catch (InterruptedException ex)
28. {
29. ex.printStackTrace();
30. }
```

```
31. }
32. // Printing the number
33. System.out.print(contr + " ");
34. // Incrementing the contr
35. contr = contr + 1;
36. // notifying the thread which is waiting for this lock
37. notify();
38. }
39. }
40. }
41. // Method for printing the even numbers
42. public void displayEvenNumber()
43. {
44. synchronized (this)
45. {
46. // Printing the number till NUM
47. while (contr < NUM)
48. {
49. // If the count is odd then display
50. while (contr % 2 == 1)
51. {
52. // handling the exception
53. try
54. {
55. wait();
56. }
57. catch (InterruptedException ex)
58. {
59. ex.printStackTrace();
60. }
61. }
62. // Printing the number
63. System.out.print(contr + " ");
64. // Incrementing the contr
65. contr = contr +1;
66. // Notifying to the 2nd thread
67. notify();
68. }
69. }
70. }
71. // main method
72. public static void main(String[] argvs)
73. {
74. // The NUM is given
75. NUM = 20;
76. // creating an object of the class OddEvenExample
77. OddEvenExample oe = new OddEvenExample();
78. // creating a thread th1
79. Thread th1 = new Thread(new Runnable()
80. {
81. public void run()
```

```
82. {
83. // invoking the method displayEvenNumber() using the thread th1
84. oe.displayEvenNumber();
85. }
86. });
87. // creating a thread th2
88. Thread th2 = new Thread(new Runnable()
89. {
90. public void run()
91. {
92. // invoking the method displayOddNumber() using the thread th2
93. oe.displayOddNumber();
94. }
95. });
96. // starting both of the threads
97. th1.start();
98. th2.start();
99. }
100.          }
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

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.

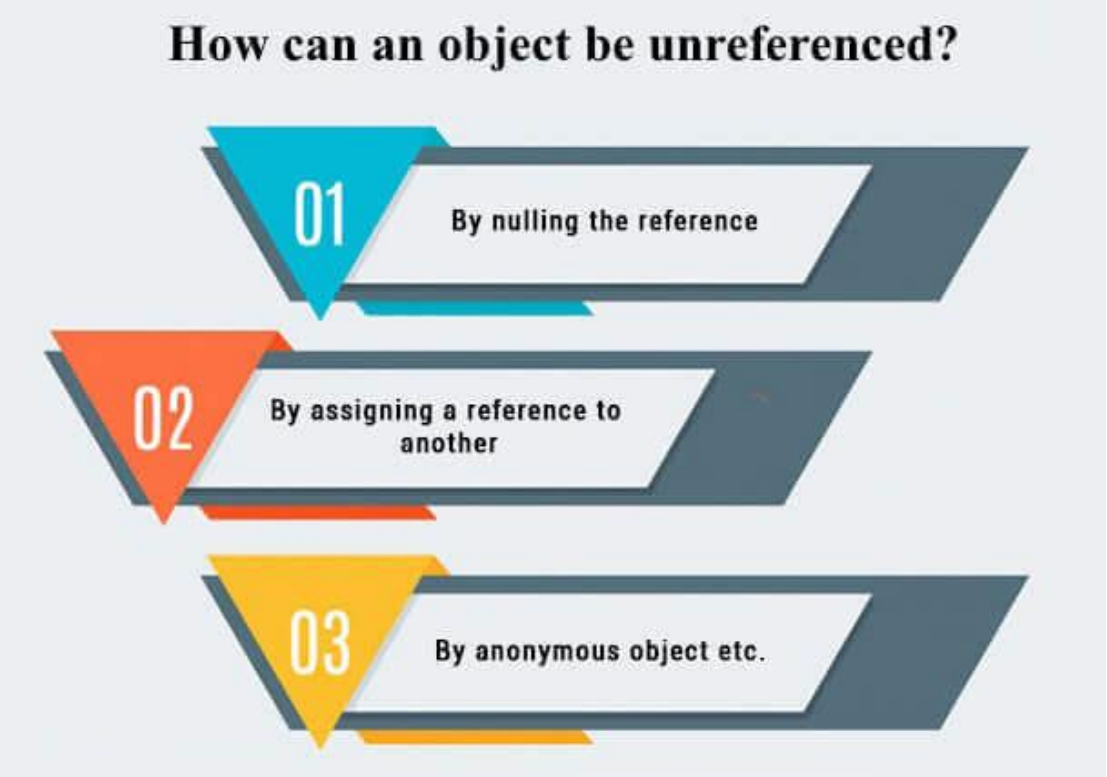
Advantage of Garbage Collection

- 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.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.



1) By nulling a reference:

2) By assigning a reference to another:

```
1. Employee e1=new Employee();
2. Employee e2=new Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By anonymous object:

```
1. new Employee();
```

finalize() 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:

```
1. protected void finalize(){}
```

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).

gc() method

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

1. public static void gc(){}

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

Simple Example of garbage collection in java

```
1. public class TestGarbage1{
2.  public void finalize(){System.out.println("object is garbage collected");}
3.  public static void main(String args[]){
4.   TestGarbage1 s1=new TestGarbage1();
5.   TestGarbage1 s2=new TestGarbage1();
6.   s1=null;
7.   s2=null;
8.   System.gc();
9.  }
10. }
```

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

Note: Neither finalization nor garbage collection is guaranteed.

Java Runtime class

Java Runtime class is used *to interact with java runtime environment*. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory etc. There is only one instance of java.lang.Runtime class is available for one java application.

The **Runtime.getRuntime()** method returns the singleton instance of Runtime class.

Important methods of Java Runtime class

No.	Method	Description
1)	public static Runtime getRuntime()	returns the instance of Runtime class.
2)	public void exit(int status)	terminates the current virtual machine.
3)	public void addShutdownHook(Thread hook)	registers new hook thread.
4)	public Process exec(String command)throws IOException	executes given command in a separate process.
5)	public int availableProcessors()	returns no. of available processors.
6)	public long freeMemory()	returns amount of free memory in JVM.
7)	public long totalMemory()	returns amount of total memory in JVM.

Java Runtime exec() method

```
1. public class Runtime1{
2.  public static void main(String args[])throws Exception{
3.   Runtime.getRuntime().exec("notepad");//will open a new notepad
4.  }
5. }
```

How to shutdown system in Java

You can use *shutdown -s* command to shutdown system. For windows OS, you need to provide full path of shutdown command e.g. c:\\Windows\\System32\\shutdown.

Here you can use -s switch to shutdown system, -r switch to restart system and -t switch to specify time delay.

```
1. public class Runtime2{
2.  public static void main(String args[])throws Exception{
3.   Runtime.getRuntime().exec("shutdown -s -t 0");
4.  }
5. }
```

How to shutdown windows system in Java

```
1. public class Runtime2{
2.  public static void main(String args[])throws Exception{
3.   Runtime.getRuntime().exec("c:\\Windows\\System32\\shutdown -s -t 0");
4.  }
5. }
```

How to restart system in Java

```
1. public class Runtime3{
2.  public static void main(String args[])throws Exception{
3.   Runtime.getRuntime().exec("shutdown -r -t 0");
4.  }
5. }
```

Java Runtime availableProcessors()

```
1. public class Runtime4{
2.  public static void main(String args[])throws Exception{
3.   System.out.println(Runtime.getRuntime().availableProcessors());
4.  }
5. }
```

Java Runtime freeMemory() and totalMemory() method

In the given program, after creating 10000 instance, free memory will be less than the previous free memory. But after gc() call, you will get more free memory.

```
1.  public class MemoryTest{
2.   public static void main(String args[])throws Exception{
3.    Runtime r=Runtime.getRuntime();
4.    System.out.println("Total Memory: "+r.totalMemory());
5.    System.out.println("Free Memory: "+r.freeMemory());
6.
7.    for(int i=0;i<10000;i++){
8.     new MemoryTest();
9.    }
10.   System.out.println("After creating 10000 instance, Free Memory: "+r.freeMemory());
11.   System.gc();
12.   System.out.println("After gc(), Free Memory: "+r.freeMemory());
13.  }
14. }
```

```
Total Memory: 100139008
Free Memory: 99474824
After creating 10000 instance, Free Memory: 99310552
After gc(), Free Memory: 100182832
```