دانشکده مهندسی و علوم کامپیوتر

برنامه نویسی پیشرفته

وحیدی اصل

مالتی تریدینگ

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

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

```
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
```

Simple example of Daemon thread in java

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

```
class TestDaemonThread2 extends Thread{
 public void run(){
  System.out.println("Name: "+Thread.currentThread().getName());
  System.out.println("Daemon: "+Thread.currentThread().isDaemon());
 }

 public static void main(String[] args){
  TestDaemonThread2 t1=new TestDaemonThread2();
  TestDaemonThread2 t2=new TestDaemonThread2();
  t1.start();
  t1.setDaemon(true);//will throw exception here
  t2.start();
 }
}
```

Test it Now

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

ThreadGroup in Java

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

ThreadGroup Example

- Let's see a code to group multiple threads.

```
ThreadGroup tg1 = new ThreadGroup("Group A");
Thread t1 = new Thread(tg1,new MyRunnable(),"one");
Thread t2 = new Thread(tg1,new MyRunnable(),"two");
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.

ThreadGroup Example

```
public class ThreadGroupDemo implements Runnable{
    public void run() {
        System.out.println(Thread.currentThread().getName());
    }
   public static void main(String[] args) {
     ThreadGroupDemo runnable = new ThreadGroupDemo();
        ThreadGroup tg1 = new ThreadGroup("Parent ThreadGroup");

        Thread t1 = new Thread(tg1, runnable,"one");
        t1.start();
        Thread t2 = new Thread(tg1, runnable,"two");
        t2.start();
        Thread t3 = new Thread(tg1, runnable,"three");
        t3.start();

        System.out.println("Thread Group Name: "+tg1.getName());
       tg1.list();

    }
}
```

```
one
two
three
Thread Group Name: Parent ThreadGroup
java.lang.ThreadGroup[name=Parent ThreadGroup,maxpri=10]
    Thread[one,5,Parent ThreadGroup]
    Thread[two,5,Parent ThreadGroup]
    Thread[three,5,Parent ThreadGroup]
```

How to perform single task by multiple threads?

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

```
class TestMultitasking1 extends Thread{
 public void run(){
   System.out.println("task one");
 }
 public static void main(String args[]){
  TestMultitasking1 t1=new TestMultitasking1();
  TestMultitasking1 t2=new TestMultitasking1();
  TestMultitasking1 t3=new TestMultitasking1();

  t1.start();
  t2.start();
  t3.start();
 }
}
```

Test it Now

```
Output:task one
       task one
       task one
```

How to perform single task by multiple threads?

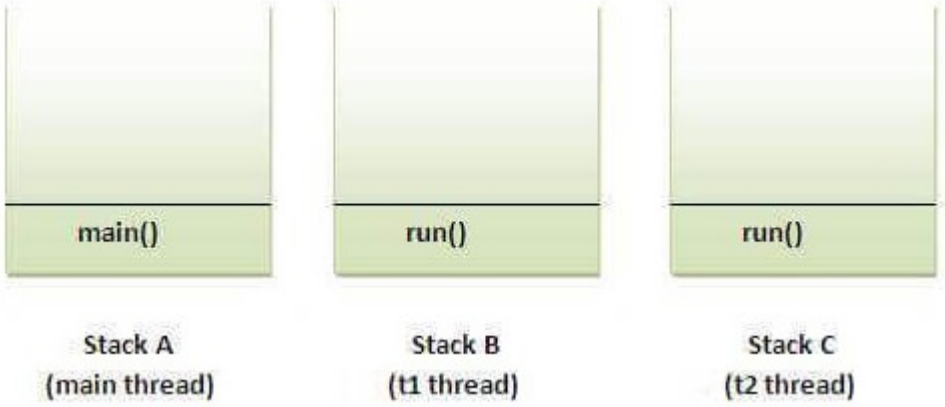
- If you have to perform single task by many threads, have only one run() method.For

```
class TestMultitasking2 implements Runnable{
public void run(){
System.out.println("task one");
}

public static void main(String args[]){
Thread t1 =new Thread(new TestMultitasking2());//passing annonymous object of TestMultitasking2 class
Thread t2 =new Thread(new TestMultitasking2());

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

 }
}
```

Test it Now

```
Output:task one
       task one
```

Note: Each thread run in a separate callstack.



perform multiple tasks by multiple threads (multitasking in multithreading)

```
class Simple1 extends Thread{
 public void run(){
   System.out.println("task one");
 }
}

class Simple2 extends Thread{
 public void run(){
   System.out.println("task two");
 }
}

 class TestMultitasking3{
 public static void main(String args[]){
  Simple1 t1=new Simple1();
  Simple2 t2=new Simple2();

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

```
Output:task one
       task two
```

Synchronization in Java

- Synchronization in java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- **Why use Synchronization**
- The synchronization is mainly used to
  - To prevent thread interference.
  - To prevent consistency problem.
- There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   1. Synchronized method.
   2. Synchronized block.
   3. static synchronization.
2. Cooperation (Inter-thread communication in java)

Concept of Lock in Java

- Synchronization is built around an internal entity known as the lock or monitor.
- Every object has an lock associated with it.
- By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.
- In this example, there is no synchronization, so output is inconsistent. Let's see the example:

```
class Table{
void printTable(int n){//method not synchronized
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
     Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}
  }

 }
}
```

```
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
```

Understanding the problem without Synchronization

```
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

```
Output: 5
        100
        10
        200
        15
        300
        20
        400
        25
        500
```

Java synchronized method

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
//example of java synchronized method
class Table{
 synchronized void printTable(int n){//synchronized method
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
     Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}
  }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
```

```
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

```
Output: 5
        10
        15
        20
        25
        100
        200
        300
        400
        500
```

Synchronized Block in Java

- Synchronized block can be used to perform synchronization on any specific resource of the method.
- Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
- Points to remember for Synchronized block
  - Synchronized block is used to lock an object for any shared resource.
  - Scope of synchronized block is smaller than the method.

Syntax to use synchronized block

```
synchronized (object reference expression) {
  //code block
}
```

Example of synchronized block

- Let's see the simple example of synchronized block.

```
class Table{

 void printTable(int n){
   synchronized(this){//synchronized block
     for(int i=1;i<=5;i++){
      System.out.println(n*i);
      try{
       Thread.sleep(400);
      }catch(Exception e){System.out.println(e);}
     }
   }
 }//end of the method
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
```

```
public class TestSynchronizedBlock1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Test it Now

```
Output:5
       10
       15
       20
       25
       100
       200
       300
       400
       500
```

Java - Thread Control

- Core Java provides complete control over multithreaded program. You can develop a multithreaded program which can be suspended, resumed, or stopped completely based on your requirements.
- There are various static methods which you can use on thread objects to control their behavior. Following table lists down those methods
- Be aware that the latest versions of Java has deprecated the usage of suspend( ), resume( ), and stop( ) methods due to their possible damages to data and so you need to use available alternatives.

Sr.No.	Method & Description
1	**public void suspend()** This method puts a thread in the suspended state and can be resumed using resume() method.
2	**public void stop()** This method stops a thread completely.
3	**public void resume()** This method resumes a thread, which was suspended using suspend() method.
4	**public void wait()** Causes the current thread to wait until another thread invokes the notify().
5	**public void notify()** Wakes up a single thread that is waiting on this object's monitor.

Killing threads in Java

- A thread is automatically destroyed when the run() method has completed.
- But it might be required to kill/stop a thread before it has completed its life cycle.
- Previously, methods suspend(), resume() and stop() were used to manage the execution of threads.
- But these methods were deprecated by Java 2 because they could result in system failures.
- Modern ways to suspend/stop a thread are by using a boolean flag and Thread.interrupt() method
- Using a boolean flag: We can define a boolean variable which is used for stopping/killing threads say ‘exit’. Whenever we want to stop a thread, the ‘exit’ variable will be set to true.

Example of killing threads in Java

```
class MyThread implements Runnable {

    // to stop the thread
    private boolean exit;

    private String name;
    Thread t;

    MyThread(String threadname)
    {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
        exit = false;
        t.start(); // Starting the thread
    }

    // execution of thread starts from run() method
    public void run()
    {
        int i = 0;
        while (!exit) {
            System.out.println(name + ": " + i);
            i++;
            try {
                Thread.sleep(100);
            }
            catch (InterruptedException e) {
                System.out.println("Caught:" + e);
            }
        }
        System.out.println(name + " Stopped.");
    }

    // for stopping the thread
    public void stop()
    {
        exit = true;
    }
}

// Main class
public class Main {
    public static void main(String args[])
    {
        // creating two objects t1 & t2 of MyThread
        MyThread t1 = new MyThread("First  thread");
        MyThread t2 = new MyThread("Second thread");
        try {
            Thread.sleep(500);
            t1.stop(); // stopping thread t1
            t2.stop(); // stopping thread t2
            Thread.sleep(500);
        }
        catch (InterruptedException e) {
            System.out.println("Caught:" + e);
        }
        System.out.println("Exiting the main Thread");
    }
}
```

Example of killing threads in Java

Output:

```
New thread: Thread[First  thread, 5, main]
New thread: Thread[Second thread, 5, main]
First  thread: 0
Second thread: 0
First  thread: 1
Second thread: 1
First  thread: 2
Second thread: 2
First  thread: 3
Second thread: 3
First  thread: 4
Second thread: 4
First  thread: 5
Second thread Stopped.
First  thread Stopped.
Exiting the main Thread
```

Note: The output may vary every time.