What is Program?

A computer program is nothing but a set of instructions (smallest unit of execution) that are used to execute particular tasks to get particular results.

Why does the JVM terminate daemon threads when no user threads are running?

Daemon threads exist only to support user threads in the background (e.g., garbage collection).

If no user threads are running, there is no need for daemon threads.

Keeping them running would waste resources, so JVM automatically stops them.

Example:

A thread auto-saves your work (daemon thread).

If you close the application (no user threads), there’s no work to save, so the auto-save stops.

Java Thread Pool (Efficient Thread Management)

A Thread Pool is a group of pre-created threads that can be reused instead of creating new ones each time.

🔹 How it works?

Instead of creating a new thread for every task, Java reuses existing threads.

This saves time and system resources.

When a task is completed, the thread returns to the pool for future use.

🔹 Real-life use:

Used in Servlets and JSP (Java web applications) to handle multiple user requests efficiently.

Thread Pool Methods

Java provides different ways to create a thread pool:

newFixedThreadPool(int s): Creates a thread pool with a fixed number (s) of threads.

newCachedThreadPool(): Creates threads as needed but reuses existing ones when available.

newSingleThreadExecutor(): Creates a single thread that executes tasks one by one.

Advantages of Java Thread Pool

Better Performance:

Saves time by reusing existing threads instead of creating new ones.

Reduces the overhead of thread creation and destruction.

Real-time Usage:

Used in Servlets and JSP where a thread pool handles multiple user requests efficiently.

Improves system scalability and responsiveness in web applications.

Constructors of ThreadGroup Class

Constructor Description

ThreadGroup(String name) Creates a new thread group with the given name.

ThreadGroup(ThreadGroup parent, String name) Creates a new thread group inside another group, linking it to a parent.

Multitasking:

Its performing multiple task at a sing time.

Use: It increase the performance of cpu.

Types:

(i). Process based Multitasking(Mp)

(ii).Thread based Multasking(Mt)

I

Multiprocessing:

When one system is connected to multiple processor(CPU) in order to complete the task.

It is best suitable at system level or Os level.



Multithreading:

Executing multiple threads (sub-process,smalltask)at a single time.

Multithreading is best suitable at programming level.

Java provides predefined API for Multithreading.

Ex.Thread,Runnable,ThreadGroup,Cocurrency,Thread Pool.

Uses: Softwares,Games,Animations etc.



Difference between Process and Threads.

|  |  |
| --- | --- |
| Process(Software) | Threads |
| **Process means any program is in execution.** | **It is subpart of process(Smalltask).** |
| It is heavy height. | It is lightweight. |
| Process take more time for context switching | Threads take less lime at context switching. |
| Take more time for inter process communication | Less time |
| Each process has different address space. | Thread share same address space. |
| Process are not dependent on each other. | Threads are dependent on each other. |
| Process does not required synchronization. | Thread required synchronization. |
| Process require more resourses. | Less resources. |
| Process require more time for creation. | Less time |
| Process require more time for termination. | Less time. |

Two ways to create Threads:

Better ways to create thread is runnable interface.

1).Thread class

2.)Runnable interface

1.Thread class

package.java.lang;

public class Thread

{

//constructor

//Methods

1.run()

2.start()

3.sleep()

4.join()

5.getName & setName()

6.interepereted,Priority

7.demon

}

//1.Extends the thread class

Class Test extends Thread

{

//2.Overide the run method

Public void run()

{

//Thread task

}

Public static void main(String[] args)

{

//3.Create an object of the class

Test t = new Test();

//4.Start the thread

t.start();

}

Thread Lifecycle of five stages:



Thread class inherit the properties of Runnable interface.

We can create thread into (Thread) and (Runnable) interface.

Example:

public class Test extends Thread {  
 public void run(){  
 System.*out*.println("Kaushik");  
 }  
  
 public static void main(String[] args) {  
 Test t = new Test();  
 t.start();  
 }  
}

Output:Kaushik

2.Runnable interface.

Runnable interface only one method run().

Implements the runnable intrerface.

Package java.lang;

//1.Implement the runnable interface

Public class Test implements Runnable

{

//2.Overide the run method

Public void run()

{

Sout(“ );

}

Psvm()

{

//3.create an object of test class

Test t = new Test();

//4.Create an object of thread class & pass the parameter in constructor

Thread th = new Thread(t);

//5.Start the thread

Th.start();

}

}

Example:

public class Test implements Runnable {  
 public void run(){  
 System.*out*.println("Kaushik");  
 }  
  
 public static void main(String[] args) {  
 Test t = new Test();  
 Thread th = new Thread(t);  
 th.start();  
 }  
}

Output:kaushik

1.Performing Single task from single thread.

Class Test extends Thread

{

Public void run()

{

System.out.println(“kaushik”);

}

Public static void main(String[] args)

{

Test t = new Test();

t.start();

}

Example:

Simple explanation:

public class Tests extends Thread {  
 public void run()  
 {  
 System.*out*.println("My name is kaushik");  
 }  
 public static void main(String[] args) {  
 Tests t = new Tests();  
 t.start();  
 }  
}

Output:

My name is kaushik

Project implementation:

class Mythread1 extends Thread {  
 public void run() {  
 System.*out*.println("My name is kaushik");  
 }  
}  
 public class Tests{  
 public static void main(String[] args) {  
 Mythread1 t = new Mythread1();  
 t.start();  
 }  
}

Output:

My name is kaushik

2.Performing single task from multiple thread.

class Task implements Runnable {

public void run() {

System.out.println("Hello from thread: " + Thread.currentThread().getName());

}

public static void main(String[] args) {

Task task = new Task(); // Single task

Thread t1 = new Thread(task);

Thread t2 = new Thread(task);

Thread t3 = new Thread(task);

t1.start();

t2.start();

t3.start();

}

}This method create 3 threads.

class Mythread1 extends Thread {  
 public void run() {  
 System.*out*.println("My name is kaushik");  
 }  
}  
 public class Tests{  
 public static void main(String[] args) {  
 Mythread1 thread1 = new Mythread1();  
 thread1.start();  
 Mythread1 thread2 = new Mythread1();  
 thread2.start();  
 }  
}

Output:

My name is kaushik

My name is kaushik

3.Performing multiple task from single thread.

It is not possible.

4. Performing multiple task from multiple thread.

class Task1 extends Thread {

public void run() {

System.out.println("Task 1 is running...");

}

}

class Task2 extends Thread {

public void run() {

System.out.println("Task 2 is running...");

}

}

public class MultiTaskingExample {

public static void main(String[] args) {

Task1 t1 = new Task1();

Task2 t2 = new Task2();

t1.start(); // Start thread t1 (Task 1)

t2.start(); // Start thread t2 (Task 2)

}

}

2. Using Runnable interface:

class Task1 implements Runnable {

public void run() {

System.out.println("Task 1 is running...");

}

}

class Task2 implements Runnable {

public void run() {

System.out.println("Task 2 is running...");

}

}

public class MultiTaskingExample {

public static void main(String[] args) {

Thread t1 = new Thread(new Task1()); // Task 1

Thread t2 = new Thread(new Task2()); // Task 2

t1.start();

t2.start();

}

}

Every thread excute together.

Thread execution depend on JVM.

Example:

class Mythread1 extends Thread {  
 public void run() {  
 int a =10;  
 int b =20;  
 int c =a+b;  
 System.*out*.println("Add:"+c);  
 }  
}  
class Mythread2 extends Thread{  
 public void run()  
 {  
 int a =2;  
 for(int i=1;i<=10;i++)  
 {  
 System.*out*.println(a\*i);  
 }  
 }  
}  
class Mythread3 extends Thread  
{  
 public void run()  
 {  
 int a=54;  
 if(a%2==0)  
 {  
  
 System.*out*.println("Even");  
 }else {  
 System.*out*.println("Odd");  
 }  
 }  
}  
 public class Tests{  
 public static void main(String[] args) {  
 Mythread1 thread1 = new Mythread1();  
 thread1.start();  
 Mythread2 thread2 = new Mythread2();  
 thread2.start();  
 Mythread3 thread3 = new Mythread3();  
 thread3.start();  
 }  
}

Output:

2

4

6

8

10

12

14

16

18

20

Even

Add:30

import java.util.Scanner;  
  
class Mythread extends Thread  
{  
 public int a;  
 Mythread(Scanner sc) {  
 System.*out*.print("Enter a number: ");  
 this.a = sc.nextInt();  
 }  
 public void run()  
 {  
  
 if(a%2==0)  
 {  
 System.*out*.println("Even");  
 }else{  
 System.*out*.println("Odd");  
 }  
 }  
}  
class Mythread2 extends Thread{  
 public int a;  
 Mythread2(Scanner sc) {  
 System.*out*.print("Enter a number: ");  
 this.a = sc.nextInt();  
 }  
 public void run()  
 {  
 for(int i=1;i<=10;i++)  
 {  
 System.*out*.println(a\*i);  
 }  
 }  
}  
public class Newthred  
{  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 Mythread m = new Mythread(sc);  
 m.start();  
  
 Mythread2 m2 = new Mythread2(sc);  
 m2.start();  
  
 sc.close();  
 }  
}

Output:

Enter a number: 97

Odd

Enter a number: 8

8

16

24

32

40

48

56

64

72

80

Another example:

class Mythread1 extends Thread {  
 public void run() {  
  
 System.*out*.println("Play Video");  
 }  
}  
class Mythread2 extends Thread{  
 public void run() {  
 System.*out*.println("Play Music");  
 }  
}  
class Mythread3 extends Thread  
{  
 public void run()  
 {  
 System.*out*.println("Process bar is executing:");  
 }  
}  
class Mythread4 extends Thread  
{  
 public void run()  
 {  
 System.*out*.println("Timer is executing:");  
 }  
}  
  
 public class Tests{  
 public static void main(String[] args) {  
 Mythread1 thread1 = new Mythread1();  
 thread1.start();  
  
 Mythread2 thread2 = new Mythread2();  
 thread2.start();  
  
 Mythread3 thread3 = new Mythread3();  
 thread3.start();  
  
 Mythread4 thread4 = new Mythread4();  
 thread4.start();  
 }  
}

Output:

Play Video

Timer is executing:

Process bar is executing:

Play Music

“Thread” class Constructors:

Public class Thread implement Runnable

{

//Simple constructors:

Public Thread(){ }

Public Thread(Runnable target){ }

Public Thread(String name){ }

Public Thread(Runnable target,String name){ }

//Group constructor:

Public Thread(ThreadGroup group,Runnable target){ }

Public Thread(ThreadGroup group,String name){ }

Public Thread(ThreadGroup group,Runnable target,String name){ }

Public Thread(ThreadGroup group,Runnable target,String name,longstackSize){ }

}

“Thread” class methods.

//Basic Methods:

Public void run(){ }

Public synchronized void run(){ }

It is not in java:

Public static native Thread currentThread(); //static method

Public final native boolean isAlive();

//Naming Methods:

Public final String getName(){ }

Public final synchronized void setName(String name){ }

//Demon Thread methods:

public final boolean isDaemon(){ }

public final void setDaemon(Boolean on){ }

//Priority Based Methods:

public final int getPriority(){ }

public final void setPriority(int newPriority){ }

//Depricated methods:

Public final void suspend(){ }

Public final void resume(){ }

Public final void stop(){ }

Public void destroy(){ }

Important Methods:

//Inter-Thread Communication Methods:

Public class Object{

Public final void wait() throws InterruptedException{ }

Public final native void notify();

Public final native void notify();

}

Example of Basic and Naming Methods:

public class Threadss {  
 public static void main(String[] args) {  
 System.*out*.println("hello");  
 System.*out*.println(Thread.*currentThread*().getName());  
 Thread.*currentThread*().setName("kaushik");  
 System.*out*.println(Thread.*currentThread*().getName());  
 }  
}

Output:

hello

main

kaushik

Example:

class Test extends Thread

{

Public void main()

{

Sop(Thread.currentThread.getname()); // Execute this process Thread 0

Sop(“Kaushik”);

}

Psvm

{

Test t = new Test(); // Execute this process Main Thread

t.start();

Sop(Thread.currentThread.getname());

}

}

public class Threadss extends Thread {  
  
 public void run()  
 {  
 System.*out*.println("hello:"+Thread.*currentThread*().getName());  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println(Thread.*currentThread*().getName());  
 Threadss t1 = new Threadss();  
 t1.setName("Kaushik");  
 t1.start();  
 Threadss t2 = new Threadss();  
 t2.setName("Akash");  
 t2.start();  
 Threadss t3 = new Threadss();  
 t3.start();  
 Threadss t4 = new Threadss();  
 t4.start();  
 System.*out*.println(Thread.*currentThread*().isAlive());  
  
 }  
}

Output:

Main

true

hello:Thread-3

hello:Thread-2

hello:Akash

hello:Kaushik

Demon Thread methods:

A daemon thread is a special type of thread in Java that runs in the background to perform tasks such as garbage collection, housekeeping, or other maintenance tasks. These threads are typically low-priority threads that help keep the program running smoothly without blocking the program from terminating.

Demon thread run in the background of another thread.

Use:It provide service to the threads.

Its life depends on another thread.

It inherit the properties from its parent thread.

Jvm not depend on daemon thread.

Most of the times,Daemon threads haws low priority,but we can change its priority sccording to our needs.

Ex:Garbage Collector,Spelling checker in ms word.

Methods:

public final boolean isDaemon(){ }

public final void setDaemon(Boolean on){ }

Program:

Class Test extends Thread

{

Public void run()

{

System.out.println(“child thread”);

}

Public static void main(String[] args)

{

Sop(“Main thread”)

Test t = new Test();

t.setDaemon(True);

t.start();

}

Case 1: we have to create daemon thread before starting the thread,if we create daemon thread after starting it, it will throw run-time exception i.e IllegegalThreadException.

Ex. class demois extends Thread{  
 public void run(){  
 if(Thread.*currentThread*().isDaemon())  
 {  
 System.*out*.println(getName()+"My name is kaushik");  
 }  
 else{  
 System.*out*.println(getName()+"kaushik");  
 }  
 }  
}  
public class Setsdemon {  
 public static void main(String[] args) {  
 Thread.*currentThread*().setDaemon(true);  
 System.*out*.println("Main");  
 demois d = new demois();  
 d.setDaemon(true);  
 d.start();  
  
 demois d2= new demois();  
  
 d2.start();  
 d2.setDaemon(false);  
 }  
}

Output:

Exception in thread "main" java.lang.IllegalThreadStateException

at java.base/java.lang.Thread.setDaemon(Thread.java:1414)

at Setsdemon.main(Setsdemon.java:14)

Case2: we cannot create main thread of daemon thread.

Case2 example:

class demois extends Thread{  
 public void run(){  
 if(Thread.*currentThread*().isDaemon())  
 {  
 System.*out*.println(getName()+"My name is kaushik");  
 }  
 else{  
 System.*out*.println(getName()+"kaushik");  
 }  
 }  
}  
public class Setsdemon {  
 public static void main(String[] args) {  
 Thread.*currentThread*().setDaemon(true);  
 System.*out*.println("Main");  
 demois d = new demois();  
 d.setDaemon(true);  
 d.start();  
  
 demois d2= new demois();  
 d2.setDaemon(false);  
 d2.start();  
 }  
}

Output:

Exception in thread "main" java.lang.IllegalThreadStateException

at java.base/java.lang.Thread.setDaemon(Thread.java:1414)

at Setsdemon.main(Setsdemon.java:14)

Example:

public class Demon extends Thread {  
  
 public void run()  
 {  
 //System.out.println(Thread.currentThread().isDaemon());  
 System.*out*.println("Child thread:");  
 }  
 public static void main(String[] args) {  
 System.*out*.println("Parent Thread:");  
 Demon d = new Demon();  
 d.setDaemon(true);  
 d.start();  
 }  
}

Output:

Parent Thread:

Child thread:

class demois extends Thread{  
 public void run(){  
 if(Thread.*currentThread*().isDaemon())  
 {  
 System.*out*.println(getName()+"My name is kaushik");  
 }  
 else{  
 System.*out*.println(getName()+"kaushik");  
 }  
 }  
}  
public class Setsdemon {  
 public static void main(String[] args) {  
 System.*out*.println("Main");  
 demois d = new demois();  
 d.setDaemon(true);  
 d.start();  
  
 demois d2= new demois();  
 d2.setDaemon(false);  
 d2.start();  
 }  
}

Output:

Main

Thread-0My name is kaushik

Thread-1kaushik

Priority based Methods:

Jvm provides the priority to each thread & according to these priorities jvm allocate the processor.

Priority are represented in the form of integer value which range from 1-10.

Priority are inherited from default parent thread.

By default Priority for jvm creates 5.

If priority value is not between 1 -10,then it will throw runtime Exception i.e IllegalargumentException.

Multiple thread same priority that execution depends upon jvm.

Priority depends on the platform(Windows does not support priorities).

1 - MIN\_PRIORITY

5 - NORM\_ PRIORITY

10 - MAX\_ PRIORITY

Below are not Priorities:

0 , <1 , >10

MINIMUM\_PRORITY

LOW\_PRIORITY

Methods:

Public final void setPriority(int value)

Public final int getPriority()

Program:

Class Test extends Thread

{

Public void run()

{

System.out.println(“child thread”);

}

Public static void main(String[] args)

{

Sop(“Main thread”)

Test t = new Test();

t.setPriority(10);

t.start();

}

Example:

public class Priorities extends Thread {  
 public void run(){  
 System.*out*.println("child thread");  
 System.*out*.println(Thread.*currentThread*().getPriority());  
 }  
 public static void main(String[] args) {  
  
 System.*out*.println("Main thread:"+Thread.*currentThread*().getPriority());  
 //Thread.currentThread().setPriority(10);  
 Thread.*currentThread*().setPriority(*MIN\_PRIORITY*);  
 System.*out*.println("Main thread new priority:"+Thread.*currentThread*().getPriority());  
 Priorities p = new Priorities();  
 p.setPriority(6);  
 p.start();  
 }  
}

Output:

Main thread:5

Main thread new priority:1

child thread:6

Prevent Thread execution Methods:

1.sleep() Methods:

Class Thread

{

1.public static native void sleep(long milisecond ) throws InteruptedException

2.public static void sleep(long milisec , int nanomilisec) throws InteruptedException

}

Example: Main thread in sleep mode:

public class Preventthread {  
 public static void main(String[] args) {  
 try {  
 for (int i = 0; i <= 5; i++) {  
 Thread.*sleep*(1000);  
 System.*out*.println(i);  
 }  
 }  
 catch (Exception e) {  
 System.*out*.println(e);  
 }  
 }  
}

Output:0,1,2,3,4,5

class myth extends Thread {  
 public void run() {  
 for (int i = 0; i <= 5; i++) {  
 try {  
  
 Thread.*sleep*(1000);  
 //Thread.sleep(1000,-1);  
 //java.lang.IllegalArgumentException: nanosecond timeout value out of range  
  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
 System.*out*.println(i);  
 }  
 }  
}  
public class Preventthread {  
 public static void main(String[] args) {  
 myth m = new myth();  
 m.start();  
 }  
}

Output:0,1,2,3,4,5

Sleep () method important points:

1.If the value of milliseconds is negative then thow”IllegalArgumemntException” .

2. If the value of nanoseconds is not in range 0-999999 then thow”IllegalArgumemntException” .

3.Whenever we use sleep method we also need to handle the “interruptedException”.If we not handle ,Jvm will show compilation error.

4.when any thread is sleeping and if any other thread interrupts it,then it throws”InteruptedException”.

5.The sleep() method always pauses the current thread execution.When the jvm find the sleep method in code,It checks which thread is running and pause the execution of thread.

6.When we use sleep() method to pause the execution of thread.If we send any thread in sleep mode,then if execute another thread then cpu will assign another thread,if cpu is busy so,there in no gurantee that thread wakes up in time specified in sleep() method.it totaly depend on thread scheduler.

7.while the thread is sleeping, it doesn’t lose any locks or monitors that it had acquired before sleeping.

Example:

public class Sleepdemo extends Thread {  
 public void run(){  
 try{  
 for(int i=1;i<=5;i++)  
 {  
 System.*out*.println(i+" :"+Thread.*currentThread*().getName());  
 Thread.*sleep*(1000);  
 }  
 }  
 catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
 public static void main(String[] args) {  
 Sleepdemo s = new Sleepdemo();  
 s.start();  
 Sleepdemo s2 = new Sleepdemo();  
 s2.start();  
 }  
}

Output:

1 :Thread-0

1 :Thread-1

2 :Thread-1

2 :Thread-0

3 :Thread-1

3 :Thread-0

4 :Thread-0

4 :Thread-1

5 :Thread-0

5 :Thread-1

We use run method for calling:

public class Sleepdemo extends Thread {  
 public void run(){  
 try{  
 for(int i=1;i<=5;i++)  
 {  
 System.*out*.println(i+" :"+Thread.*currentThread*().getName());  
 Thread.*sleep*(1000);  
 }  
 }  
 catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
 public static void main(String[] args) {  
 Sleepdemo s = new Sleepdemo();  
 s.run();  
 Sleepdemo s2 = new Sleepdemo();  
 s2.run();  
 }  
}

Output:

1 :main

2 :main

3 :main

4 :main

5 :main

1 :main

2 :main

3 :main

4 :main

5 :main

public class Sleepdemo extends Thread {  
 public void run(){  
 try{  
 System.*out*.println("play video");  
 Thread.*sleep*(1000);  
 System.*out*.println("Forwad 10sec ");  
 Thread.*sleep*(1000);  
 System.*out*.println("backward 10sec");  
 Thread.*sleep*(1000);  
 System.*out*.println("stop");  
 Thread.*sleep*(1000);  
 }  
 catch (Exception e)  
 {  
 System.*out*.println(e);  
 }  
 }  
 public static void main(String[] args) {  
 Sleepdemo s = new Sleepdemo();  
 s.start();  
  
 }  
}

Output:

play video //before 1sec

Forwad 10sec //before 1sec

backward 10sec //before 1sec

stop

2.Yield method():

it stop the current executing thread & give a chance to another thread for execution.

Working:

java 5:Internally it used sleep()

java 6: Thread provides the hint to the thread–sheduler ,then it depends on thread scheduler to accept or ignore the hint.

Output may be (vary) different to machine to machine.

If priority are high and same of another thread then execute another thread priority when low of another thread then execute first thread first.

Method();

Public static native void yield();

public class Yields extends Thread{  
 public void run()  
 {  
 Thread.*yield*();  
 for(int i=0;i<=5;i++)  
 {  
 System.*out*.println(Thread.*currentThread*().getName()+i+" ");  
 }  
 }  
 public static void main(String[] args) {  
 Yields y= new Yields();  
 y.start();  
 //Thread.yield(); if you want main method to stop and provide chance to other threads for execution.  
 for(int i=0;i<=4;i++) {  
 System.*out*.println(i+" "+Thread.*currentThread*().getName());  
  
 }  
 }  
}

Output:

Thread-00

Thread-01

Thread-02

Thread-03

Thread-04

Thread-05

0 main

1 main

2 main

3 main

4 main