

# MultiThreading

## Multitasking

Multitasking allows several activities to occur concurrently on the computer.

- \* process-based multitasking
- \* Thread-based multitasking

process based multitasking: Allows processes (i.e. programs) to run concurrently on the computer.

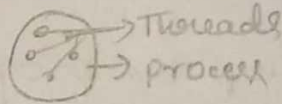
eg: Running the MS paint while also working with the word processor.

Thread based Multitasking: Allows part of the same program to run concurrently on the computer.

eg: MS word is printing & formatting text at the same time.

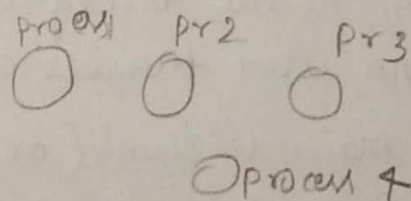
## Thrd v/s process

Thread → A thread is the smallest unit of execution within a process. Multiple threads share the same memory and run in parallel.



process → A process is an independent program, executes with its own memory space.

Multiple processes run separately and don't share memory.



## Why Multithreading?

- \* In a single-threaded environment, only one task at a time can be performed.
- \* CPU cycles are wasted, for example, when waiting for user input.
- \* Multitasking allows idle CPU time to be put to good use.

### ⇒ The Main Thread

- \* When you run a Java program, a special thread (called the main thread) is automatically started to run the `main()` method.
- \* If your program does not create any extra threads, it will stop running as soon as the `main()` method finishes.
- \* If your program creates additional threads, they are called child threads bcoz they are created by the main thread (parent thread).
- \* User threads : Keep the program running until they finish.
- \* Daemon Threads : These are background threads that do tasks like garbage collection.

Run in the background but automatically stop when no user threads are left.

- \* If the main thread (or any user thread) finishes, daemon threads will also stop immediately, even if they are in the middle of a task.



\* setDaemon(boolean)  $\Rightarrow$  used to create the Daemon thread

## Thread Creation

2 ways

- Implementing the `java.lang.Runnable` Interface
- Extending the `java.lang.Thread` class.

Bg:- class Thread1 extends Thread {

@Override

public void run() {

for (int i = 0; i < 5; i++) {

    s.o.pln ("Inside thread1" + i);  
}

public class Day33 {

public static void main(String[] args) {

    s.o.pln ("Main thread running");

    Thread1 t1 = new Thread1();

    t1.start();

    s.o.pln ("Main thread exiting");  
}

The main thread does not wait for t1 to finish.

O/p:- Main thread running  
Main thread exiting

Inside Thread1 0

Inside Thread1 1

Inside Thread1 2

Inside Thread1 3

Inside Thread1 4

\* The t1 thread starts running, but the main thread continues its own execution and exits before Thread1 finishes

\* Threads are executed based on the JVM Schedule

Ex:-

```
class Thread1 extends Thread {  
    public Thread1(String name) { } → constructor  
        super(name);  
}
```

@Override

```
public void run() {
```

```
    for (int i=0; i<5; i++) {
```

Represent the current thread name

```
        System.out.println("Inside " + Thread.currentThread().
```

```
            getName() + " " + i);
```

If u mention Thread1 it represents

thread only but if u mention

Thread(main) ⇒ represents currently running thread.

```
    }  
}
```

```
public class Day33 {
```

```
    public static void main(String[] args) {
```

```
        System.out.println("Main thread running");
```

```
        Thread t1 = new Thread1("JavaThread");
```

```
        t1.start();
```

```
        System.out.println("Main thread exiting");
```

```
    }  
}
```

O/p:-

- main thread running
- main thread exiting
- Inside JavaThread 0
- Inside JavaThread 1
- Inside JavaThread 2
- Inside JavaThread 3
- Inside JavaThread 4



120 Days ChallengegVersion control

Day20.javaDay21.javaDay22.javaDay31.javaDay32.javaDay33.java x

1class Thread1 extends Thread{ 2 usages

2@Override

3public void run(){

4for(int i=0;i<5;i++){

5System.out.println("Inside Thread1 "+i);

6}

7}

8}

9public class Day33 {

10public static void main(String[] args){

11System.out.println("Main thread running");

12Thread1 t1=new Thread1();

13t1.start();

14System.out.println("Main thread exiting");

15}

16}

17}

18}

RunDay33 x

C:\Program Files\Java\jdk-19\bin\java.exe "-java

Main thread running

Main thread exiting

Inside Thread1 0

Inside Thread1 1

Inside Thread1 2

Inside Thread1 3

Inside Thread1 4

Process finished with exit code 0

The image shows an IDE window with a file explorer on the left, a code editor in the center, and a Run console on the right. The code editor displays two Java classes: `Thread1` and `Day33`. `Thread1` is a subclass of `Thread` with a `run()` method that prints "Inside" followed by the thread name and an incrementing counter from 0 to 4. `Day33` contains a `main` method that creates a `Thread1` object named `t1` with the name "JavaThread", starts it, and prints "Main thread running" and "Main thread exiting". The Run console on the right shows the execution output, including the command `"C:\Program Files\Java\jdk-19\bin\java.exe" "-java` and the sequence of "Inside" messages for each thread, followed by "Main thread exiting" and "Main thread running". The process finished with exit code 0.

```
1 class Thread1 extends Thread{ 2 usages
2     public Thread1(String name){ 1 usage
3         super(name);
4     }
5     @Override
6     public void run(){
7         for(int i=0;i<5;i++){
8             System.out.println("Inside "+Thread.currentThread().getName()+" "+i);
9         }
10    }
11 }
12 public class Day33 {
13     public static void main(String[] args){
14         System.out.println("Main thread running");
15         Thread1 t1=new Thread1(name: "JavaThread");
16         t1.start();
17         System.out.println("Main thread exiting");
18     }
19 }
20 }
21
```

Run Day33 x

"C:\Program Files\Java\jdk-19\bin\java.exe" "-java  
Main thread running  
Main thread exiting  
Inside JavaThread 0  
Inside JavaThread 1  
Inside JavaThread 2  
Inside JavaThread 3  
Inside JavaThread 4  
Process finished with exit code 0