Multithreading in Java means running **multiple threads (small units of a program) concurrently** within the same Java application.

Simple example

Imagine a music app doing several things:

- **Thread 1:** Plays music
- **Thread 2:** Downloads a song
- **Thread 3:** Updates the user interface

Instead of waiting for one task to finish before starting another, multithreading allows these tasks to make progress at the same time.

What is a Thread?

A **thread** is a lightweight path of execution inside a process.

For example:

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running");
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start();

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

Here there are two threads:

1. The **main thread**
2. The **t1 thread**

Calling `t1.start()` creates/starts a separate thread, which executes the `run()` method.

Why use multithreading?

Multithreading is useful for improving **responsiveness and throughput**, especially when tasks can overlap—for example, handling multiple client requests on a server, downloading files while doing other work, or performing background operations.
One important point: multithreading can introduce problems such as **race conditions** when multiple threads access shared data. Java provides tools such as `synchronized`, locks, concurrent collections, and executors to manage this safely.
In short:
Multithreading in Java is the execution of multiple threads concurrently within a single program.

Here's a simple example where **3 different tasks run using 3 different threads**.

```
class Task1 extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Task 1: Printing " + i);
        }
    }
}

class Task2 extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Task 2: Downloading file " + i);
        }
    }
}

class Task3 extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Task 3: Sending email " + i);
        }
    }
}

public class Mythread  {
    public static void main(String[] args) {

        Task1 t1 = new Task1();
        Task2 t2 = new Task2();
        Task3 t3 = new Task3();

        // Start all 3 threads
        t1.start();
        t2.start();
        t3.start();
    }
}
```

What are the 3 tasks?

- **Thread 1 (**t1**)** → Printing numbers
- **Thread 2 (**t2**)** → Downloading files
- **Thread 3 (**t3**)** → Sending emails

When you call:

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

all three threads can execute **concurrently**.

```
D:\java-code>javac Mythread.java

D:\java-code>java Mythread
Task 1: Printing 1
Task 1: Printing 2
Task 1: Printing 3
Task 1: Printing 4
Task 1: Printing 5
Task 2: Downloading file 1
Task 2: Downloading file 2
Task 2: Downloading file 3
Task 2: Downloading file 4
Task 2: Downloading file 5
Task 3: Sending email 1
Task 3: Sending email 2
Task 3: Sending email 3
Task 3: Sending email 4
Task 3: Sending email 5
```

The same **3-task example** can be written using Runnable. This is a very common way to create threads in Java.

```
class Task1 implements Runnable {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Task 1: Printing " + i);
        }
    }
}

class Task2 implements Runnable {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Task 2: Downloading file " + i);
        }
    }
}

class Task3 implements Runnable {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Task 3: Sending email " + i);
        }
    }
}

public class Main {
    public static void main(String[] args) {

        // Create Runnable objects
        Task1 task1 = new Task1();
        Task2 task2 = new Task2();
        Task3 task3 = new Task3();

        // Create Thread objects
        Thread t1 = new Thread(task1);
        Thread t2 = new Thread(task2);
        Thread t3 = new Thread(task3);

        // Start the threads
        t1.start();
        t2.start();
        t3.start();
    }
}
```

How it works

The important difference is:

```
class Task1 implements Runnable
```

Task1 is now a **task**, not a thread itself.

Then we create a thread and give it the task:

```
Thread t1 = new Thread(task1);
```

Finally:

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

starts the thread, which automatically calls:

```
task1.run();
```

So the flow is:

```
Task1 implements Runnable
        ↓
   task1 object
        ↓
new Thread(task1)
        ↓
     t1.start()
        ↓
    task1.run()
```

And similarly for Task2 and Task3.