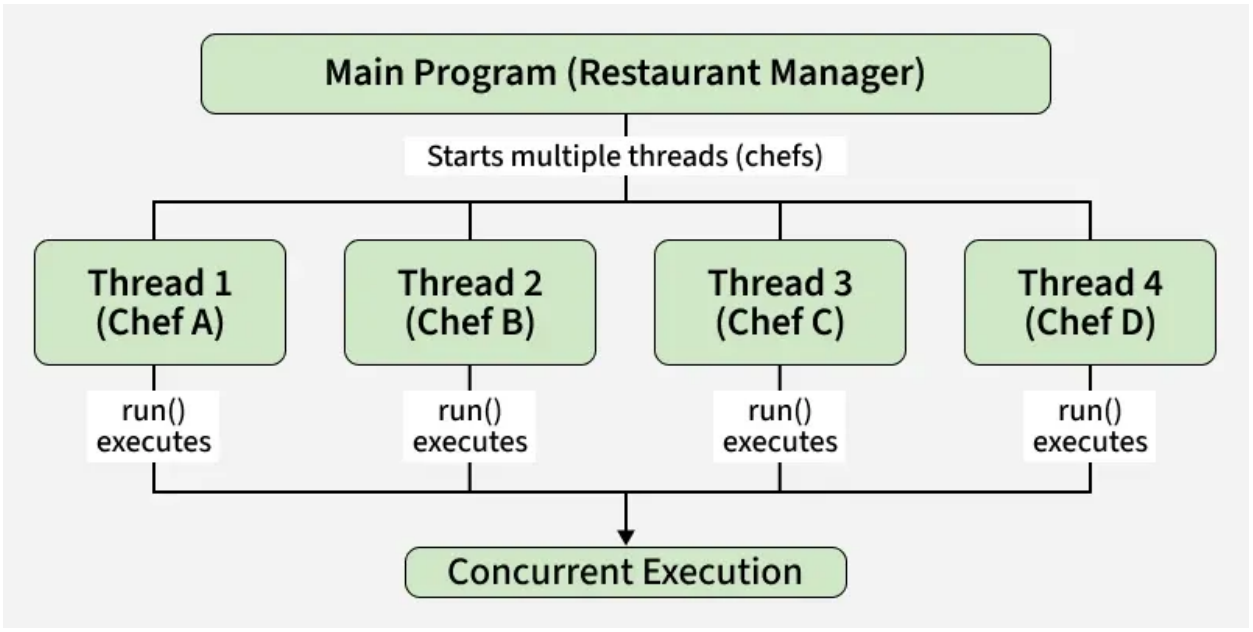
Multithreading in Java

Last Updated : 08 Sep, 2025

Multithreading in Java is a feature that enables a program to run multiple threads simultaneously, allowing tasks to execute in parallel and utilize the CPU more efficiently. A thread is a lightweight, independent unit of execution inside a program (process).

- A process can have multiple threads.
- Each thread runs independently but shares the same memory.

Example: Imagine a restaurant kitchen. Multiple chefs (threads) are preparing different dishes at the same time. This speeds up service and utilizes all available resources (CPU).



Multithreading

Different Ways to Create Threads

Threads can be created by using two mechanisms:

1. Extending the Thread class

We create a class that extends Thread and override its run() method to define the task. Then, we make an object of this class and call start(), which automatically calls run() and begins the thread’s execution.

Example: Restaurant Kitchen (Extending Thread)

```
class CookingTask extends Thread {
    private String task;

    CookingTask(String task) {
        this.task = task;
    }

    public void run() {
        System.out.println(task + " is being prepared
by " +
            Thread.currentThread().getName());
    }
}

public class Restaurant {
    public static void main(String[] args) {
        Thread t1 = new CookingTask("Pasta");
        Thread t2 = new CookingTask("Salad");
        Thread t3 = new CookingTask("Dessert");
        Thread t4 = new CookingTask("Rice");

        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

Output

```
Salad is being prepared by Thread-1
```

```
Rice is being prepared by Thread-3
Dessert is being prepared by Thread-2
Pasta is being prepared by Thread-0
```

Explanation:

- We created multiple threads (t1–t4) using the CookingTask class.
- Each thread represents a dish being prepared.
- Calling start() runs them concurrently.

2. Implementing the Runnable Interface

We create a new class which implements java.lang.Runnable interface and define the run() method there. Then we instantiate a Thread object and call start() method on this object.

Example: Restaurant Kitchen (Runnable Interface)

```
class CookingJob implements Runnable {
    private String task;

    CookingJob(String task) {
        this.task = task;
    }

    public void run() {
        System.out.println(task + " is being prepared
by " +
            Thread.currentThread().getName());
    }
}

public class RestaurantRunnable {
    public static void main(String[] args) {
        Thread t1 = new Thread(new CookingJob("Soup"));
        Thread t2 = new Thread(new
CookingJob("Pizza"));
        Thread t3 = new Thread(new
CookingJob("Burger"));

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

Output

```
Burger is being prepared by Thread-2
```

```
Pizza is being prepared by Thread-1
Soup is being prepared by Thread-0
```

Explanation:

- CookingJob implements Runnable and overrides run().
- We pass a Runnable object to the Thread constructor.
- Calling start() runs them in parallel.

When to Use Which?

- **Use extends Thread:** if your class does not extend any other class.
- **Use implements Runnable:** if your class already extends another class (preferred because Java doesn't support multiple inheritance).

Advantages of Multithreading in Java

1. **Improved Performance:** Multiple tasks can run simultaneously, reducing execution time.
2. **Efficient CPU Utilization:** Threads keep the CPU busy by running tasks in parallel.
3. **Responsiveness:** Applications (like GUIs) remain responsive while performing background tasks.
4. **Resource Sharing:** Threads within the same process share memory and resources, avoiding duplication.
5. **Better User Experience:** Smooth execution of tasks like file downloads, animations, and real-time updates.

K **kartik** 754

Article Tags : Java Java-Multithreading