Java Multithreading

Explained Like You're in a Startup Standup Meeting

Manav Juneja

Pariभाषा

From Zero to Hero, One Thread at a Time
Ever wished Java could do multiple things at once—like compile code and cry with you during prod bugs?

Welcome to Multithreading, where Java multitasks like a stressed team lead on Monday. 🚀

Manav Juneja

Thread Class

👶 Start Simple: The Thread Class

```java
class MyThread extends Thread {
    public void run() {
        System.out.println("Hello from " + Thread.currentThread().getName());
    }
}

MyThread t1 = new MyThread();
t1.start();
```

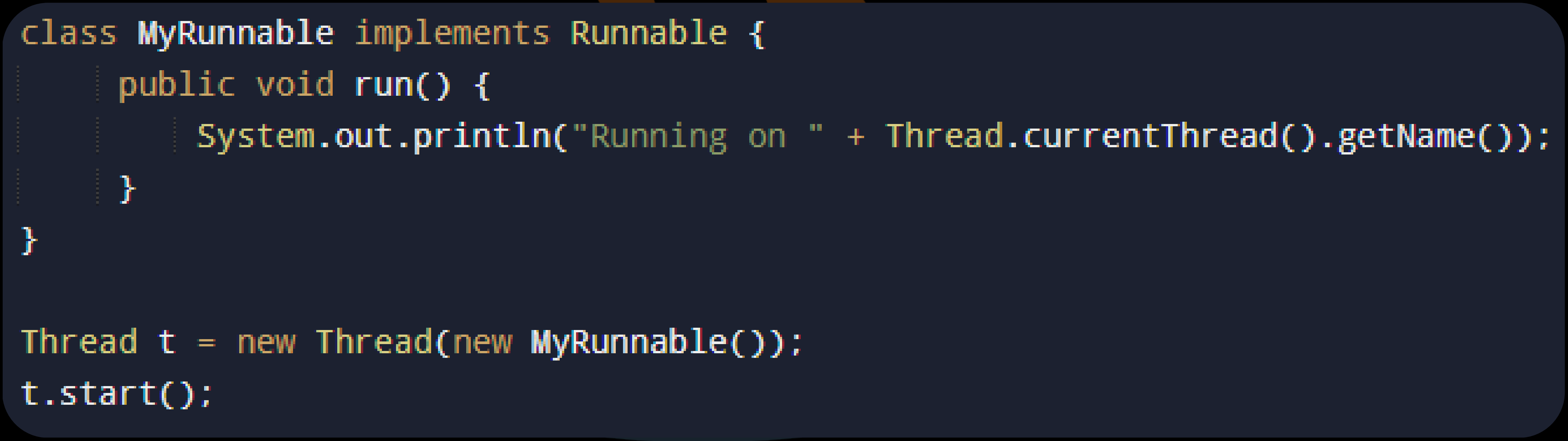
Output

```
Main thread is not lazy either 😎
Hello from Thread-0
```

Manav Juneja

Runnable Interface

🧩 Runnable Interface - Java’s Favorite Minimalist



```java
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Running on " + Thread.currentThread().getName());
    }
}

Thread t = new Thread(new MyRunnable());
t.start();
```

✅ **Cleaner, used more in real-world code.**

Manav Juneja

Industry Use Case

🛠️ But Wait... In Industry, We Use Executors

```java
ExecutorService executor = Executors.newFixedThreadPool(3);
executor.submit(() -> System.out.println("Task by " +
Thread.currentThread().getName()));
executor.shutdown();
```

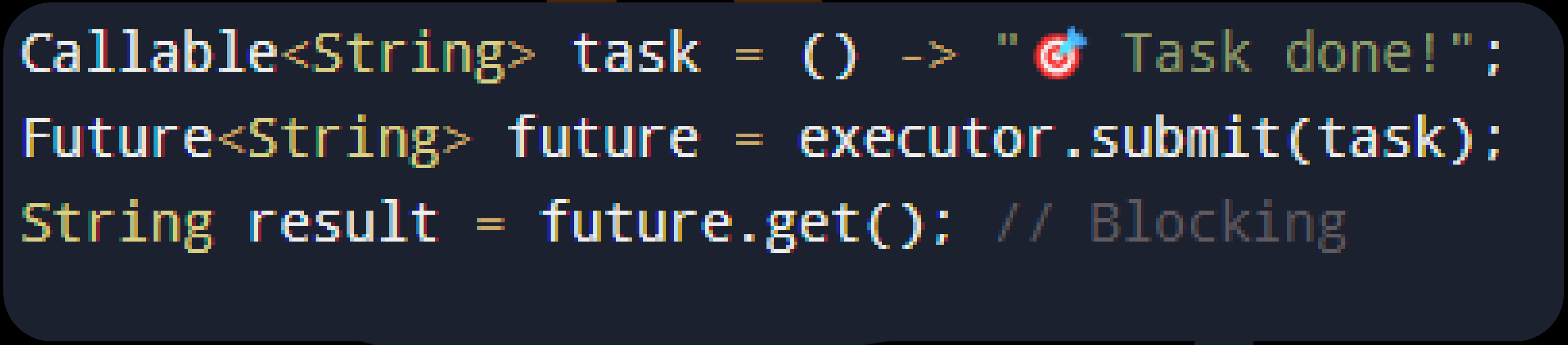
💼 **Used for: Background jobs, handling API calls, batch tasks.**

🧠 **Why? Because creating threads manually is like writing SQL queries in Notepad.**

Manav Juneja

To Return Values

🧠 **Need Return Values? Use Callable + Future**



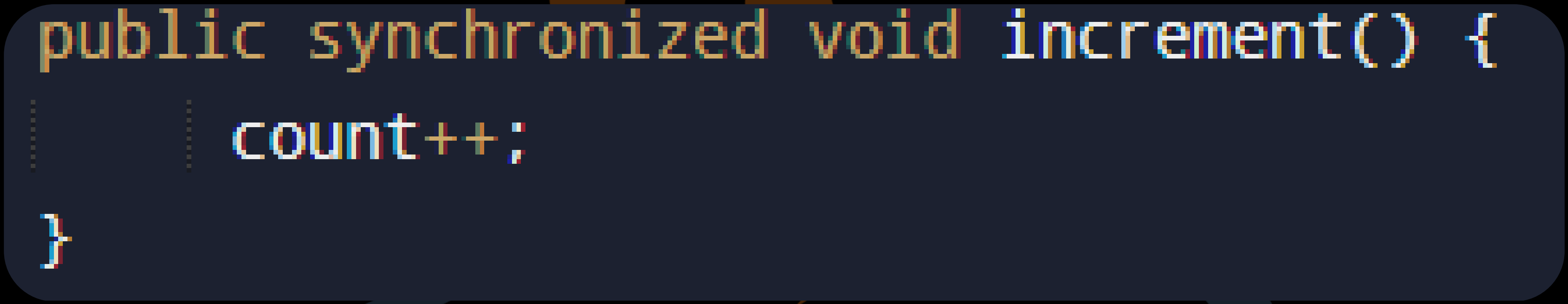
```java
Callable<String> task = () -> "🎯 Task done!";
Future<String> future = executor.submit(task);
String result = future.get(); // Blocking
```

💬 **"Runnable gives you nothing but vibes. Callable brings results." - Every dev ever.**

in Manav Juneja

Synchronization

🔒 Synchronize or Suffer



```
public synchronized void increment() {
        count++;
}
```

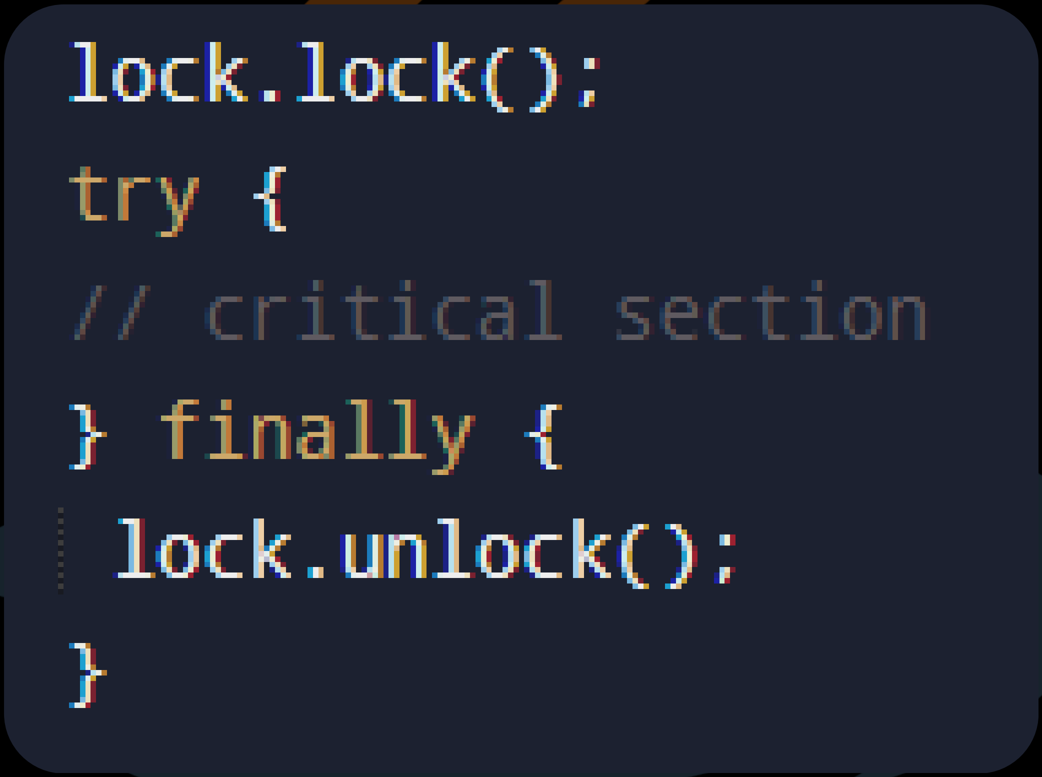
Prevents race conditions where threads fight over variables like kids over toys.

🛡️ Rule #1: Lock it before you rock it.

Manav Juneja

Reentrant Lock

🔄 ReentrantLock - When You Want Full Control



```java
lock.lock();
try {
// critical section
} finally {
 lock.unlock();
}
```

👨‍✈️ Use cases: Try-locks, fairness policies, fine-grained locking.

Manav Juneja

CountDown Latch

⏳ CountDownLatch - Coordinating Threads Like a Boss

```
CountDownLatch latch = new CountDownLatch(3);
// Threads call latch.countDown()
latch.await(); // Waits until all are done
```

Use when you need to wait for a team of threads to finish.
Think: "All hands on deck before launch." 🚀

Manav Juneja

Real-World Tips

✅ Use thread pools (ExecutorService)

✅ Prefer Callable for results

✅ Lock shared data (but don't overlock)

✅ Don't forget executor.shutdown()

✅ volatile helps with visibility, not locking

✅ Use ConcurrentHashMap, not HashMap in multi-threaded hell

Manav Juneja

TL:DR

🧵 Threads = Multiple flows in one app

🚀 Executors = Smart thread management

🎯 Callable = Async tasks with results

🔐 Sync/Locks = Safety from chaos

⛓️ Latches = Group coordination

Manav Juneja

Follow
For
More

Manav Juneja