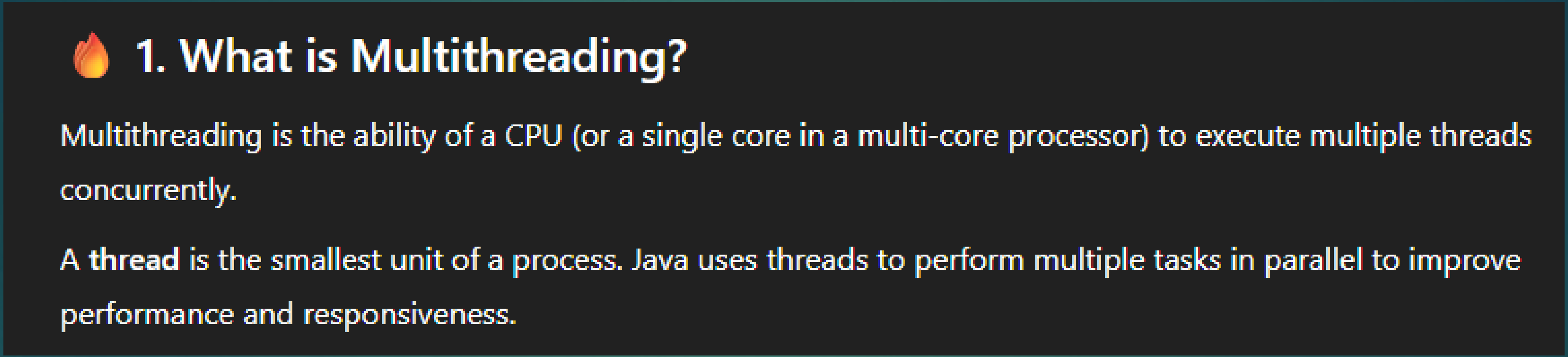
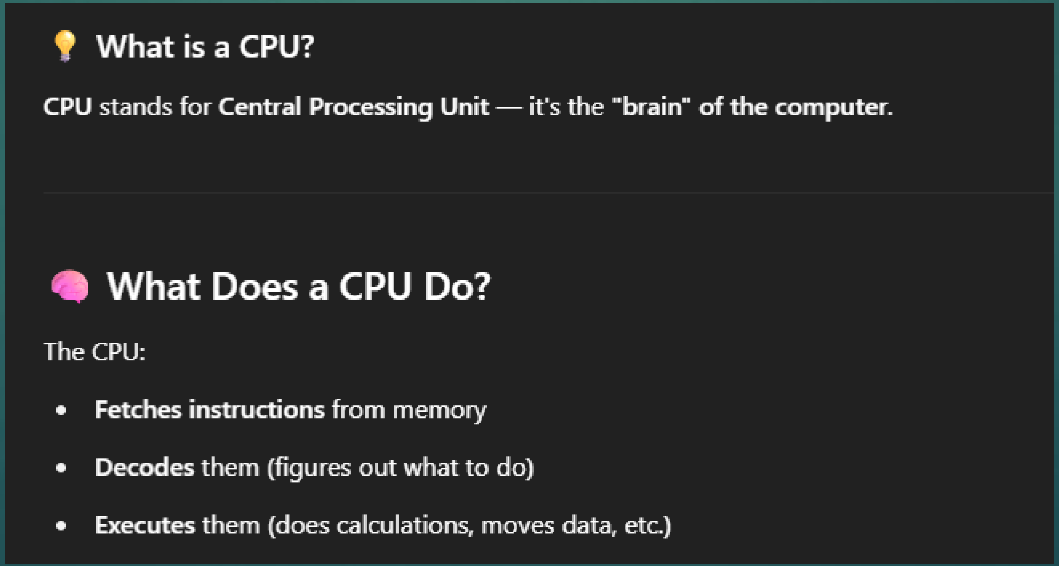


🔥 1. What is Multithreading?

Multithreading is the ability of a CPU (or a single core in a multi-core processor) to execute multiple threads concurrently.

A **thread** is the smallest unit of a process. Java uses threads to perform multiple tasks in parallel to improve performance and responsiveness.



💡 What is a CPU?

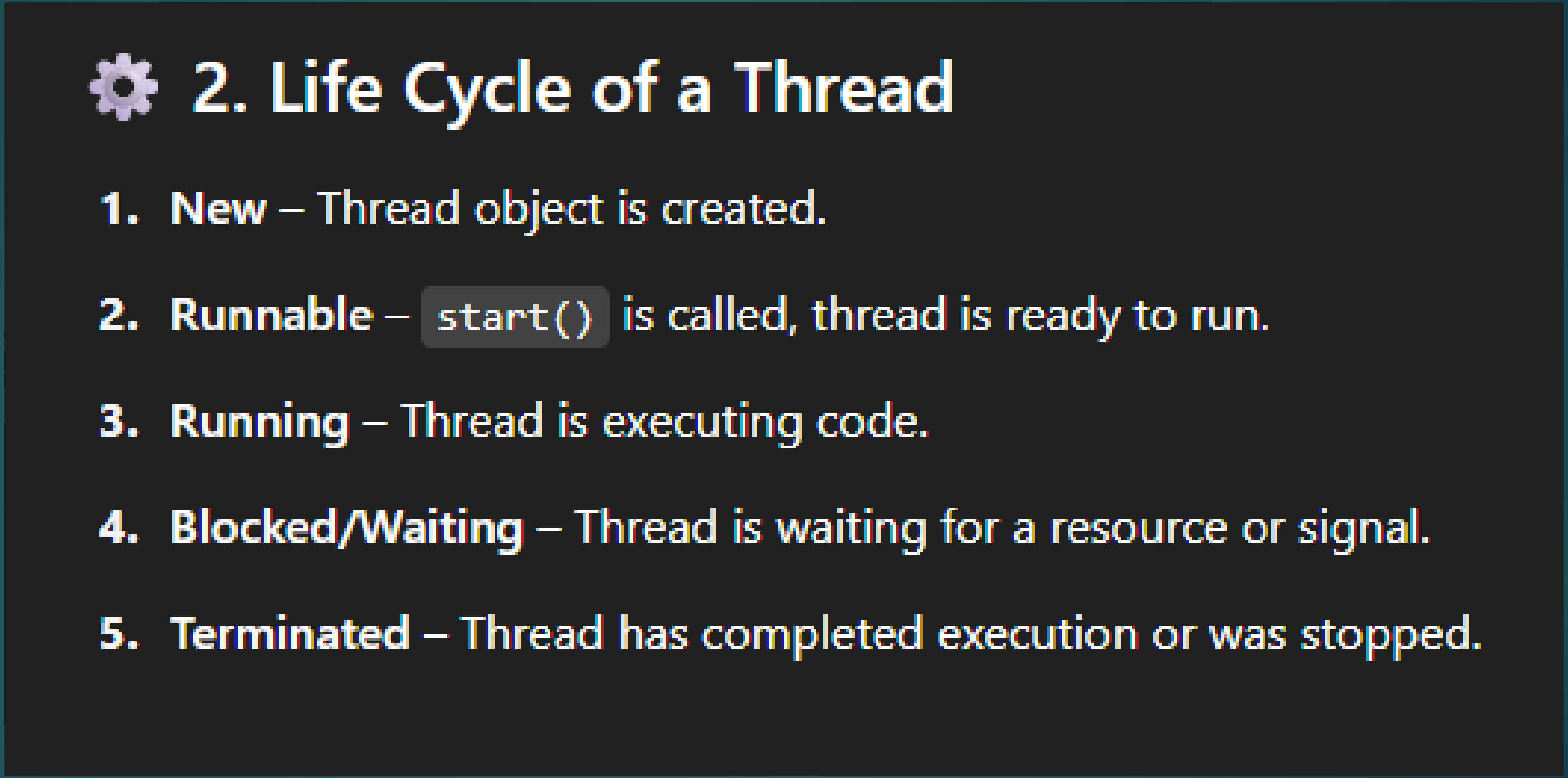
CPU stands for **Central Processing Unit** — it's the **"brain" of the computer.**

🧠 What Does a CPU Do?

The CPU:

- **Fetches instructions** from memory
- **Decodes** them (figures out what to do)
- **Executes** them (does calculations, moves data, etc.)

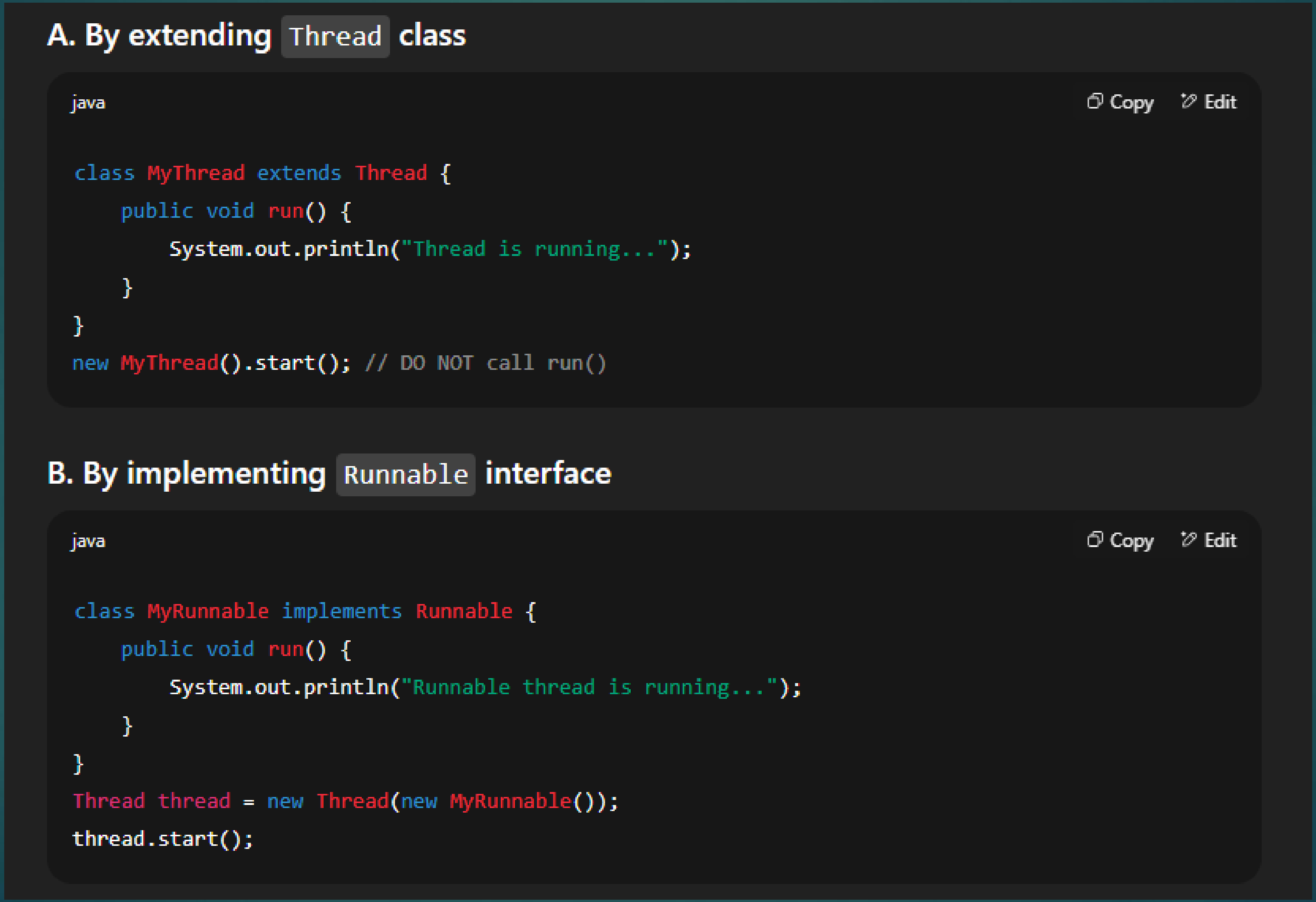
By Eslam Khder



⚙️ 2. Life Cycle of a Thread

1. **New** – Thread object is created.
2. **Runnable** – `start()` is called, thread is ready to run.
3. **Running** – Thread is executing code.
4. **Blocked/Waiting** – Thread is waiting for a resource or signal.
5. **Terminated** – Thread has completed execution or was stopped.

By Eslam Khder



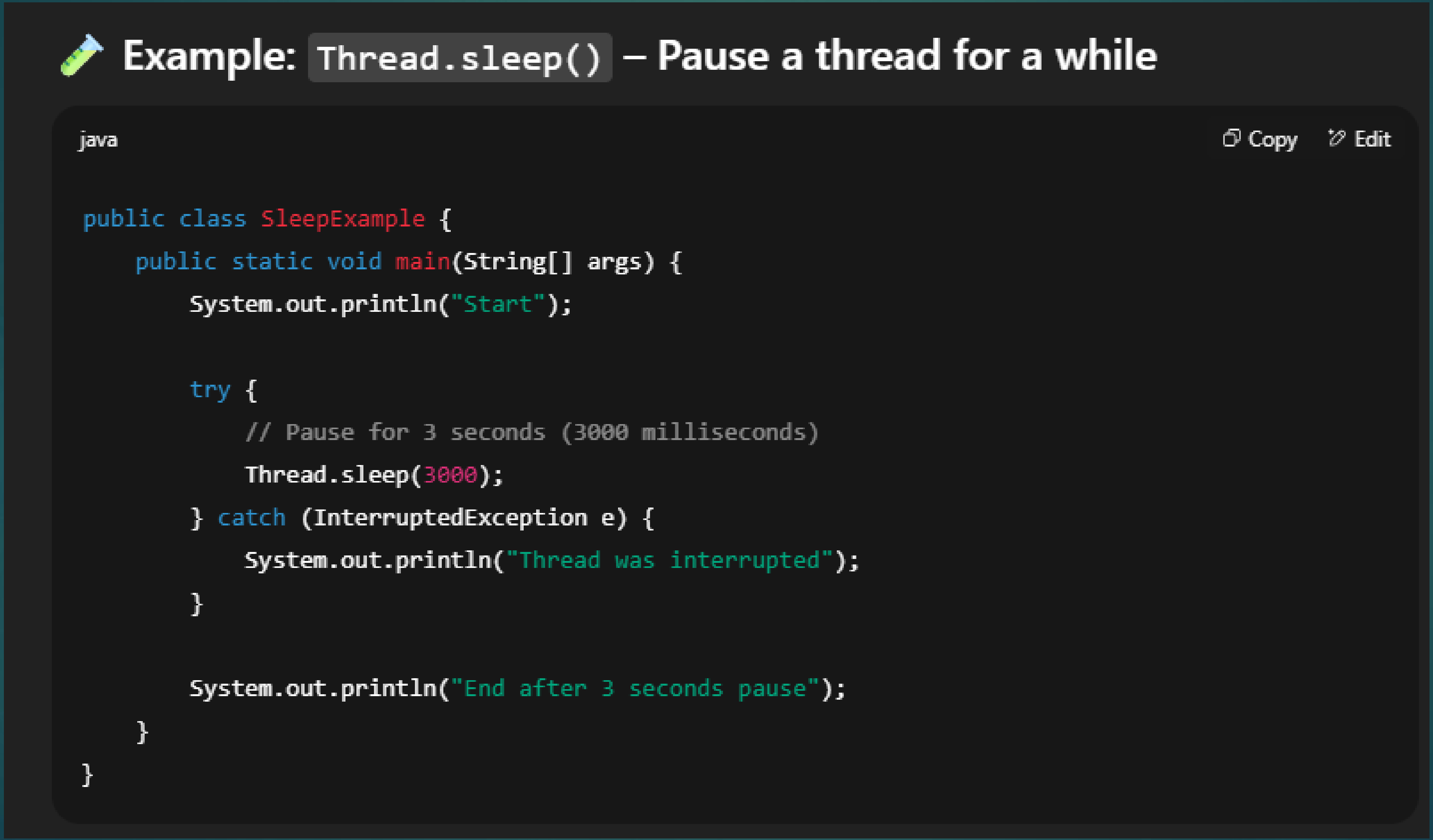
A. By extending `Thread` class

```java
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running...");
    }
}
new MyThread().start(); // DO NOT call run()
```

B. By implementing `Runnable` interface

```java
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Runnable thread is running...");
    }
}
Thread thread = new Thread(new MyRunnable());
thread.start();
```

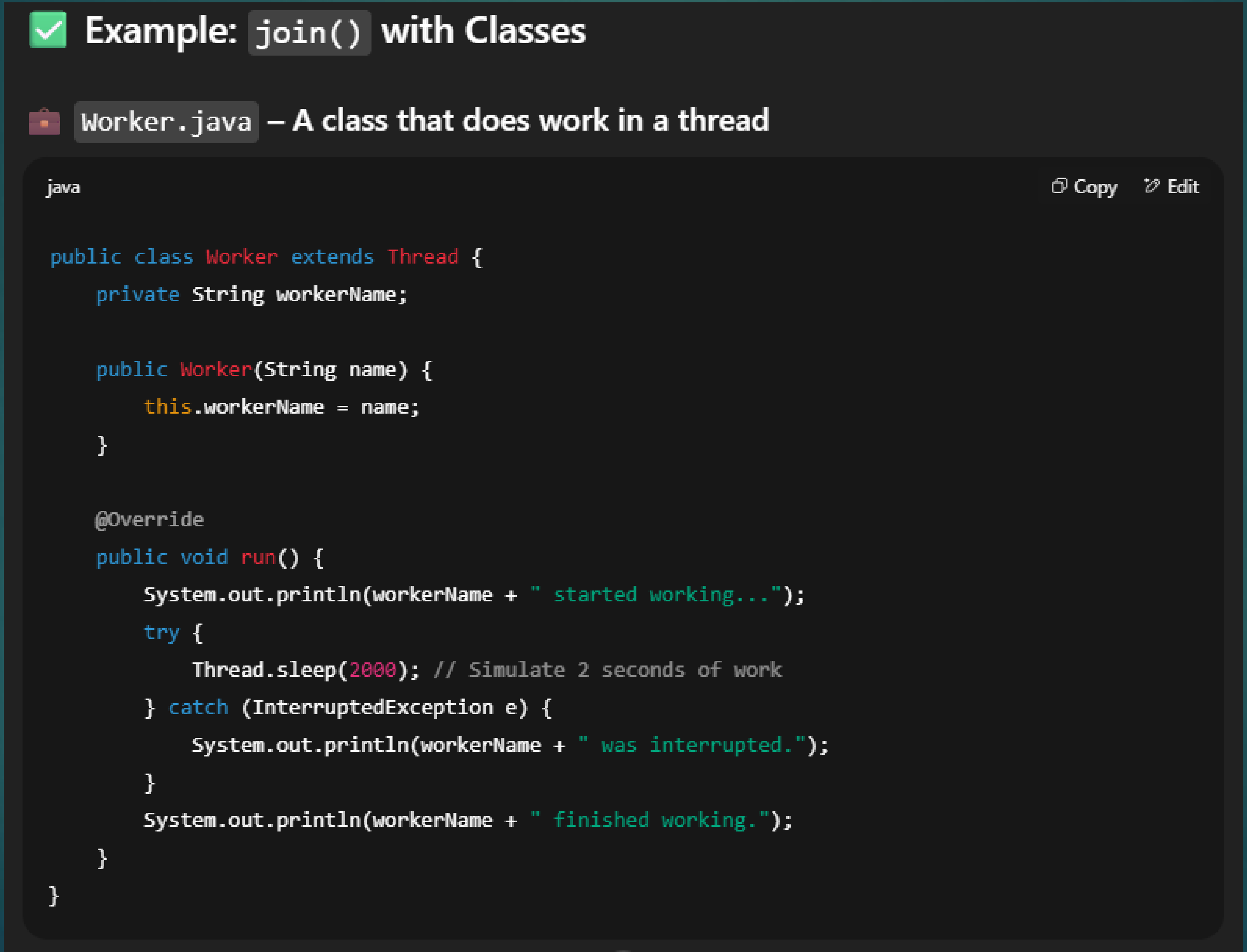
By Eslam Khder



🧪 Example: `Thread.sleep()` – Pause a thread for a while

```java
public class SleepExample {
    public static void main(String[] args) {
        System.out.println("Start");

        try {
            // Pause for 3 seconds (3000 milliseconds)
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            System.out.println("Thread was interrupted");
        }

        System.out.println("End after 3 seconds pause");
    }
}
```

By Eslam Khder

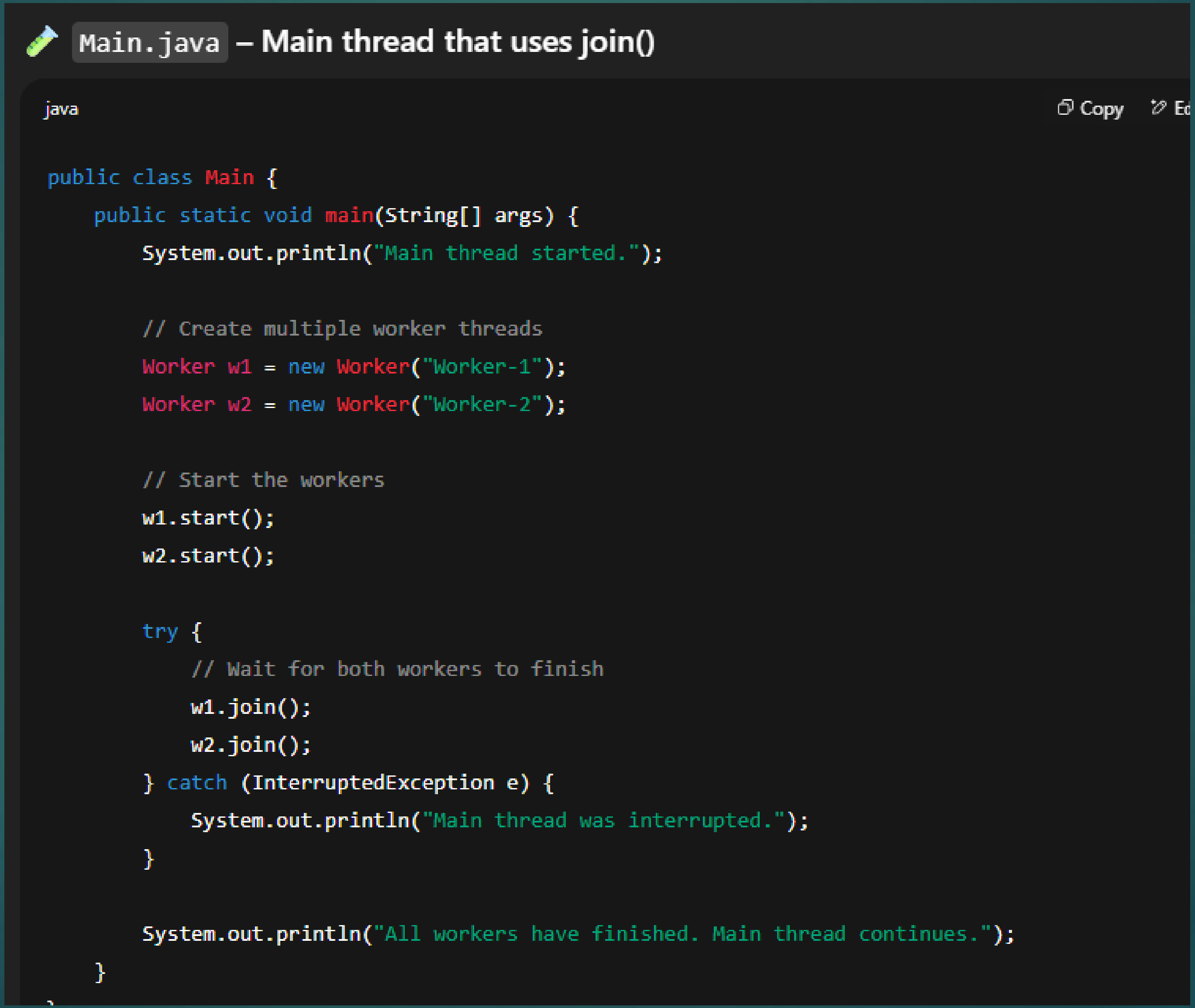


✅ Example: `join()` with Classes

💼 `Worker.java` – A class that does work in a thread

java Copy Edit

```java
public class Worker extends Thread {
    private String workerName;

    public Worker(String name) {
        this.workerName = name;
    }

    @Override
    public void run() {
        System.out.println(workerName + " started working...");
        try {
            Thread.sleep(2000); // Simulate 2 seconds of work
        } catch (InterruptedException e) {
            System.out.println(workerName + " was interrupted.");
        }
        System.out.println(workerName + " finished working.");
    }
}
```

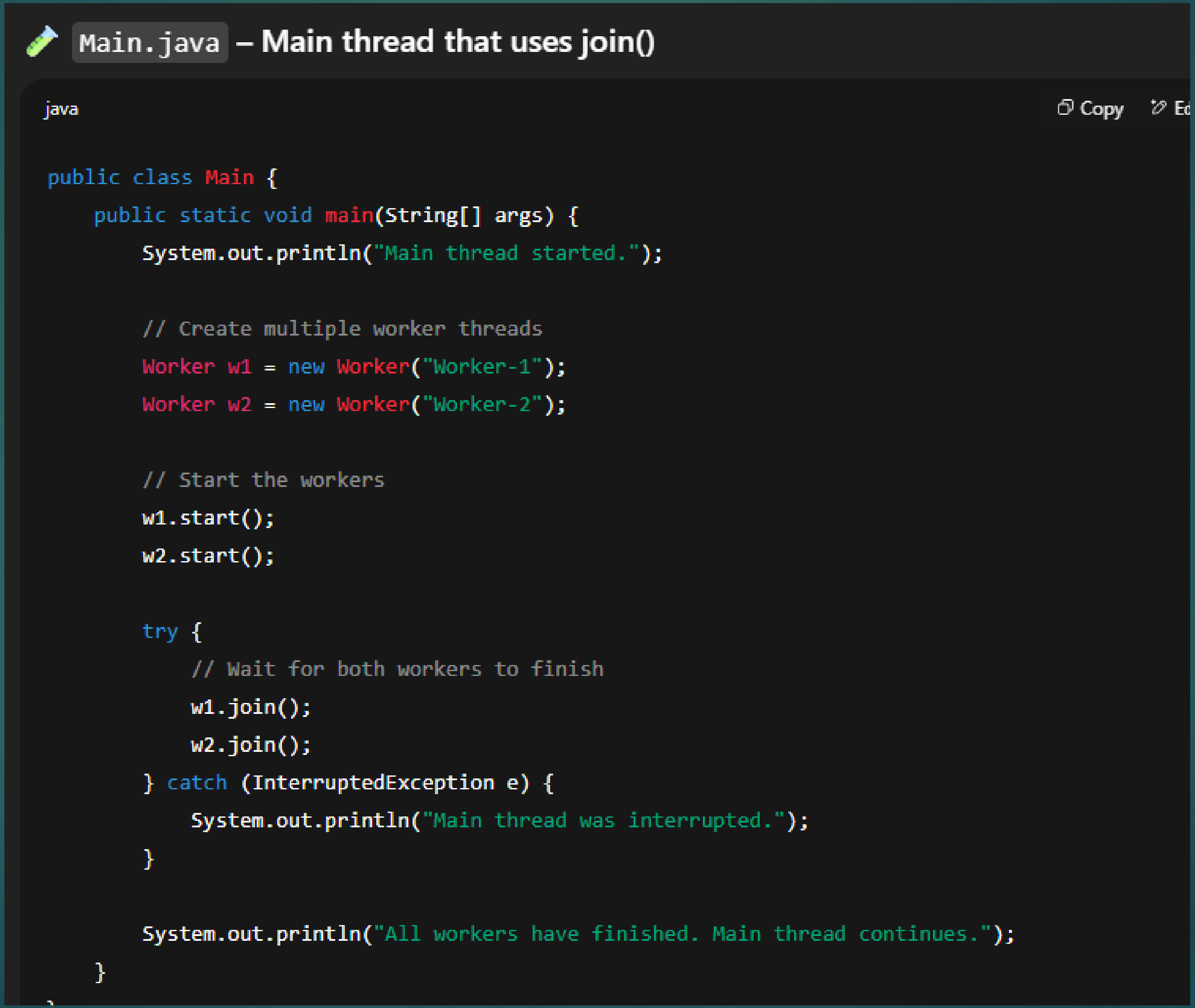


Main.java – Main thread that uses join()

java

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Main thread started.");

        // Create multiple worker threads
        Worker w1 = new Worker("Worker-1");
        Worker w2 = new Worker("Worker-2");

        // Start the workers
        w1.start();
        w2.start();

        try {
            // Wait for both workers to finish
            w1.join();
            w2.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread was interrupted.");
        }

        System.out.println("All workers have finished. Main thread continues.");
    }
```

By Eslam Khder



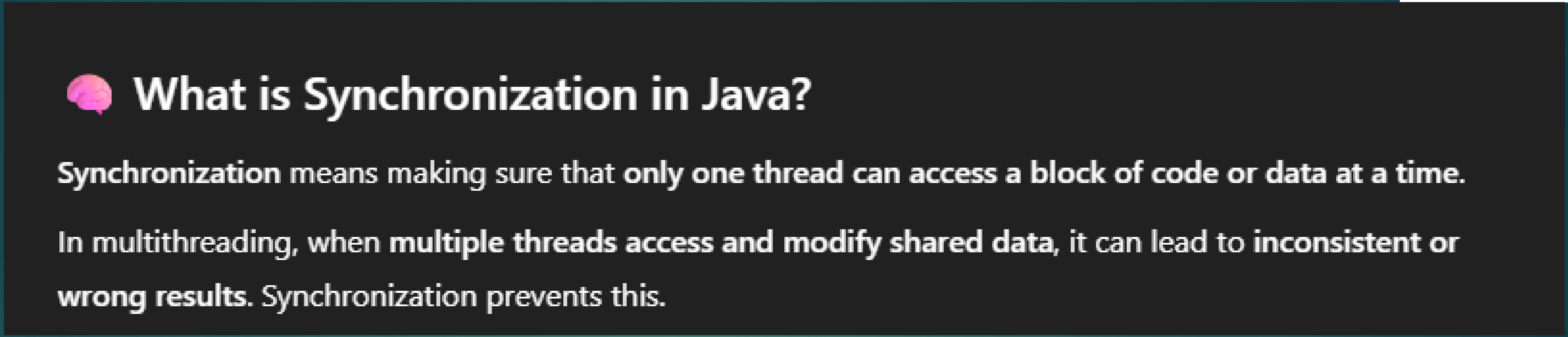
Main.java – Main thread that uses join()

java

Copy Ed

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Main thread started.");

        // Create multiple worker threads
        Worker w1 = new Worker("Worker-1");
        Worker w2 = new Worker("Worker-2");

        // Start the workers
        w1.start();
        w2.start();

        try {
            // Wait for both workers to finish
            w1.join();
            w2.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread was interrupted.");
        }

        System.out.println("All workers have finished. Main thread continues.");
    }
```

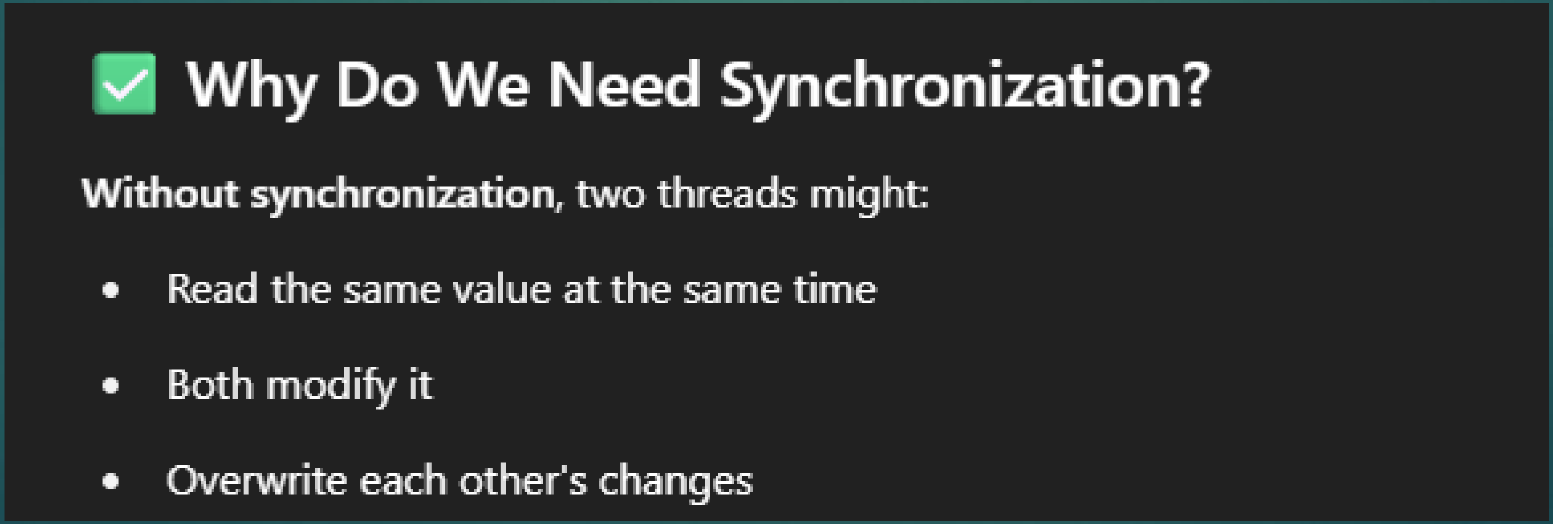
By Eslam Khder



🧠 What is Synchronization in Java?

Synchronization means making sure that **only one thread can access a block of code or data at a time.**

In multithreading, when **multiple threads access and modify shared data,** it can lead to **inconsistent or wrong results.** Synchronization prevents this.

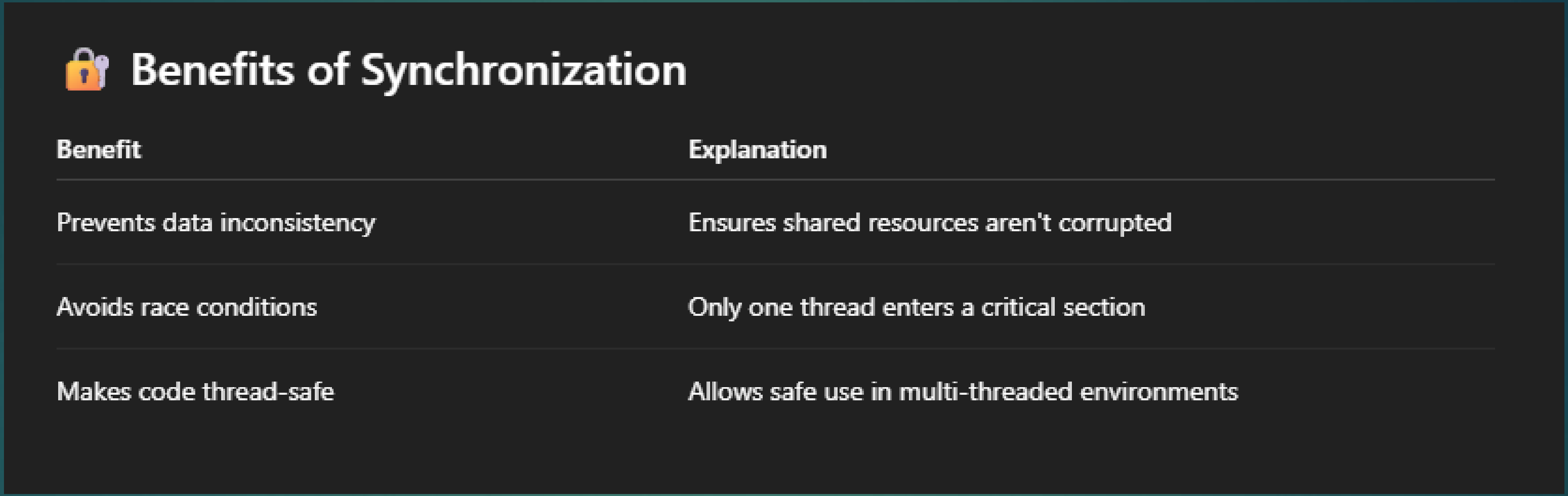


✅ Why Do We Need Synchronization?

Without synchronization, two threads might:

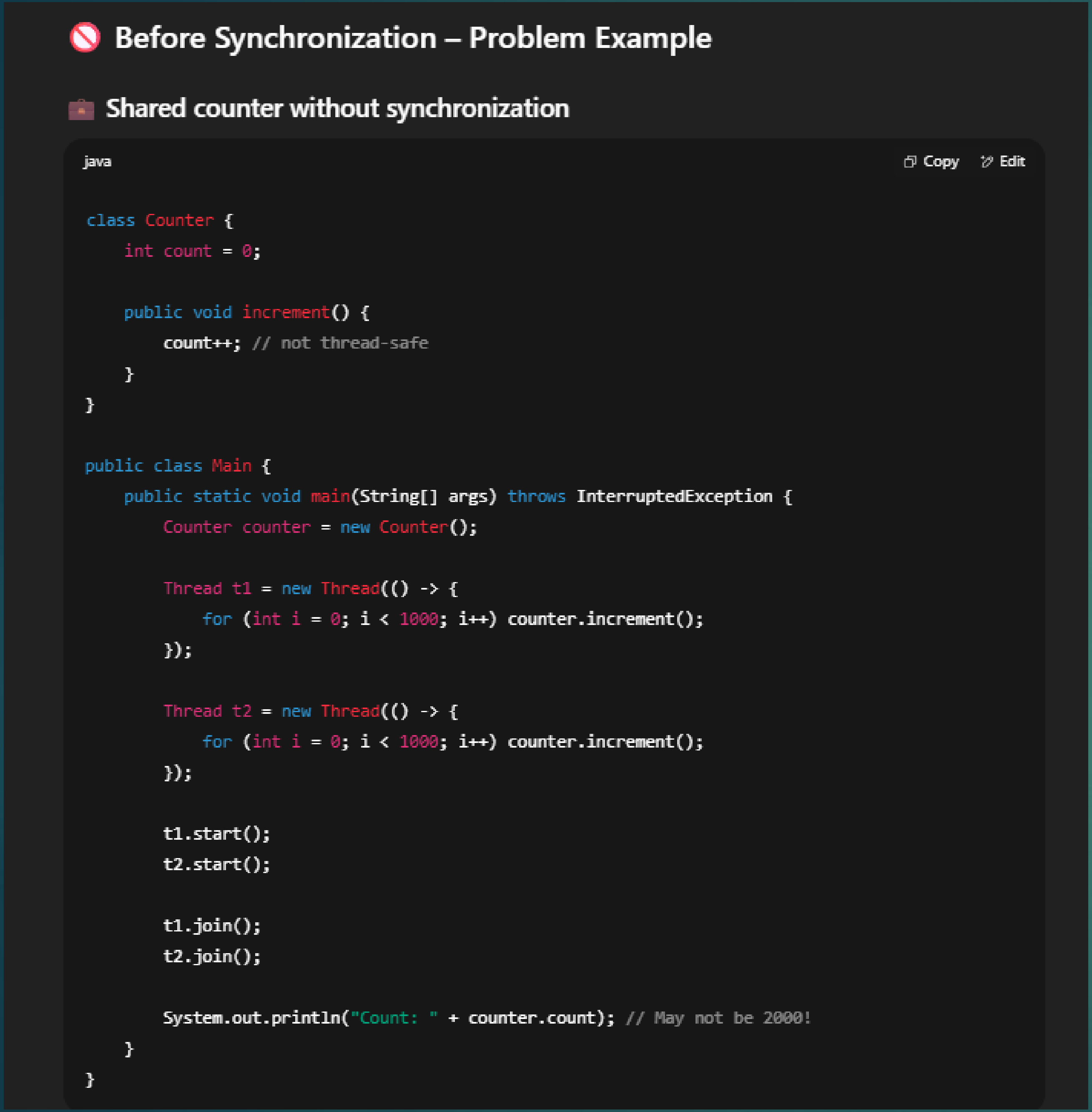
- Read the same value at the same time
- Both modify it
- Overwrite each other's changes

By Eslam Khder



🔐 Benefits of Synchronization

Benefit	Explanation
Prevents data inconsistency	Ensures shared resources aren't corrupted
Avoids race conditions	Only one thread enters a critical section
Makes code thread-safe	Allows safe use in multi-threaded environments



🚫 Before Synchronization – Problem Example

💼 Shared counter without synchronization

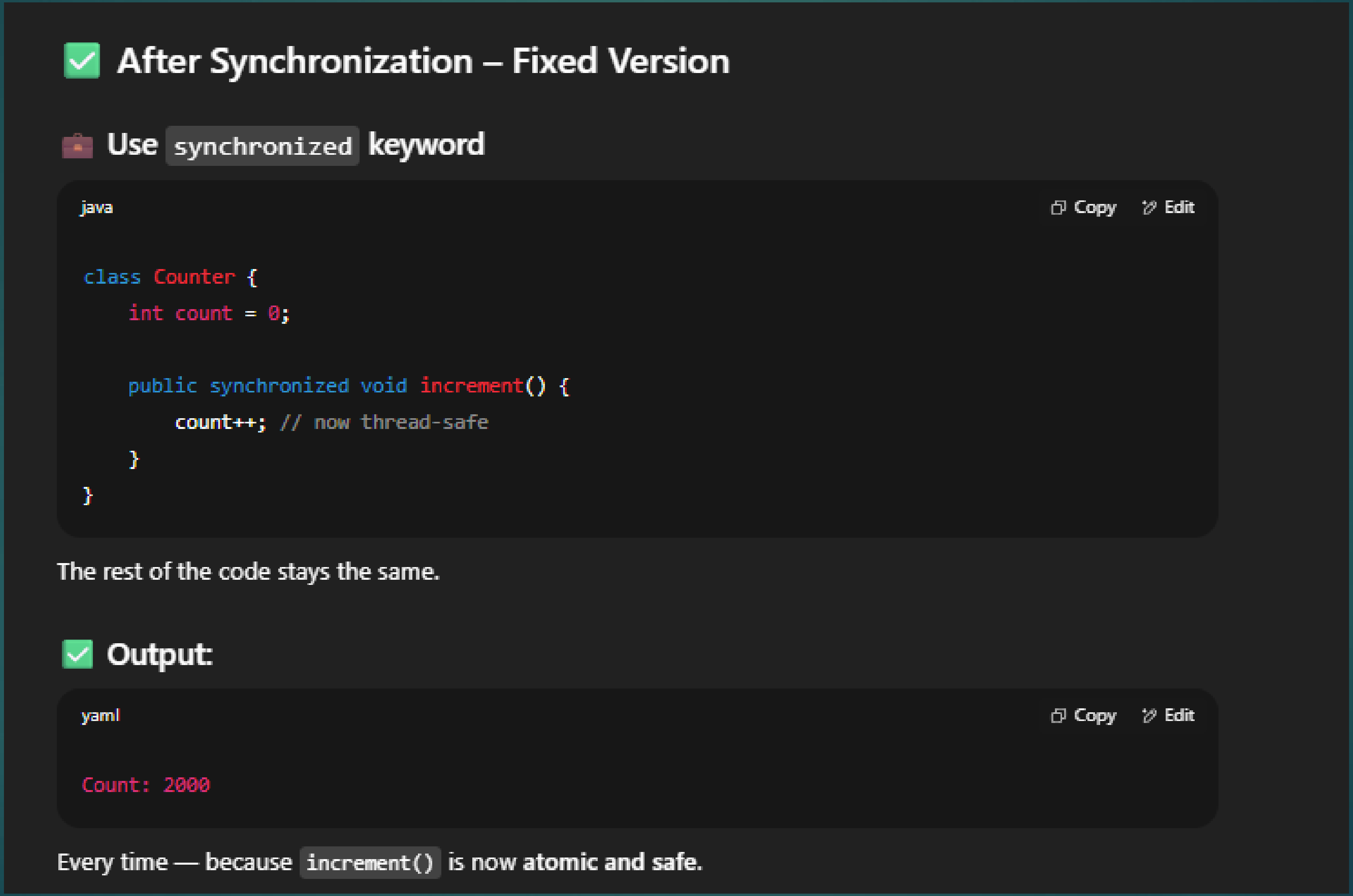
```java
class Counter {
    int count = 0;

    public void increment() {
        count++; // not thread-safe
    }
}

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Counter counter = new Counter();

        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) counter.increment();
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) counter.increment();
        });

        t1.start();
        t2.start();

        t1.join();
        t2.join();

        System.out.println("Count: " + counter.count); // May not be 2000!
    }
}
```

❗ Problem:

- Expected output: `Count: 2000`
- Actual output: Sometimes less (e.g., 1687, 1999...) due to **race condition**

EraaSoft

By Eslam Khder



✅ After Synchronization – Fixed Version

💼 Use `synchronized` keyword

java Copy Edit

```java
class Counter {
    int count = 0;

    public synchronized void increment() {
        count++; // now thread-safe
    }
}
```

The rest of the code stays the same.

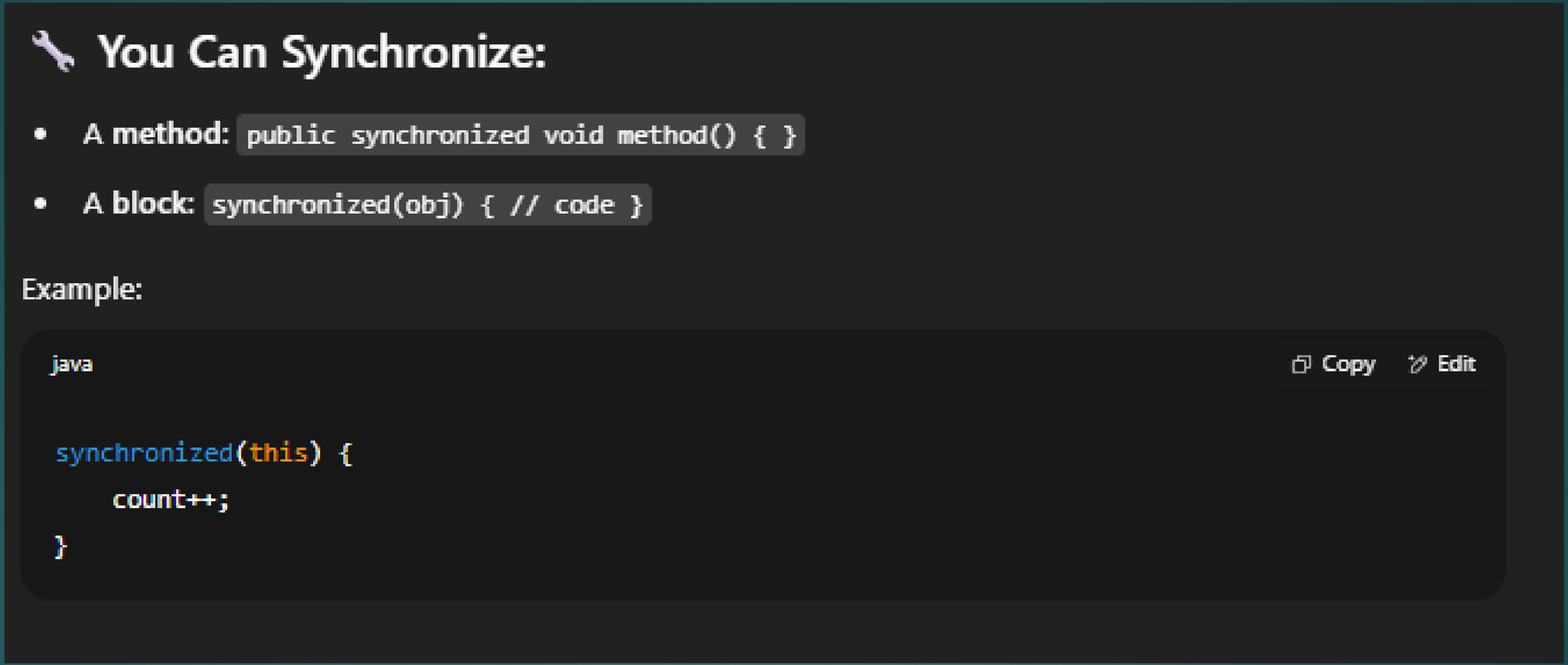
✅ Output:

yaml Copy Edit

```yaml
Count: 2000
```

Every time — because `increment()` is now **atomic and safe**.

By Eslam Khder



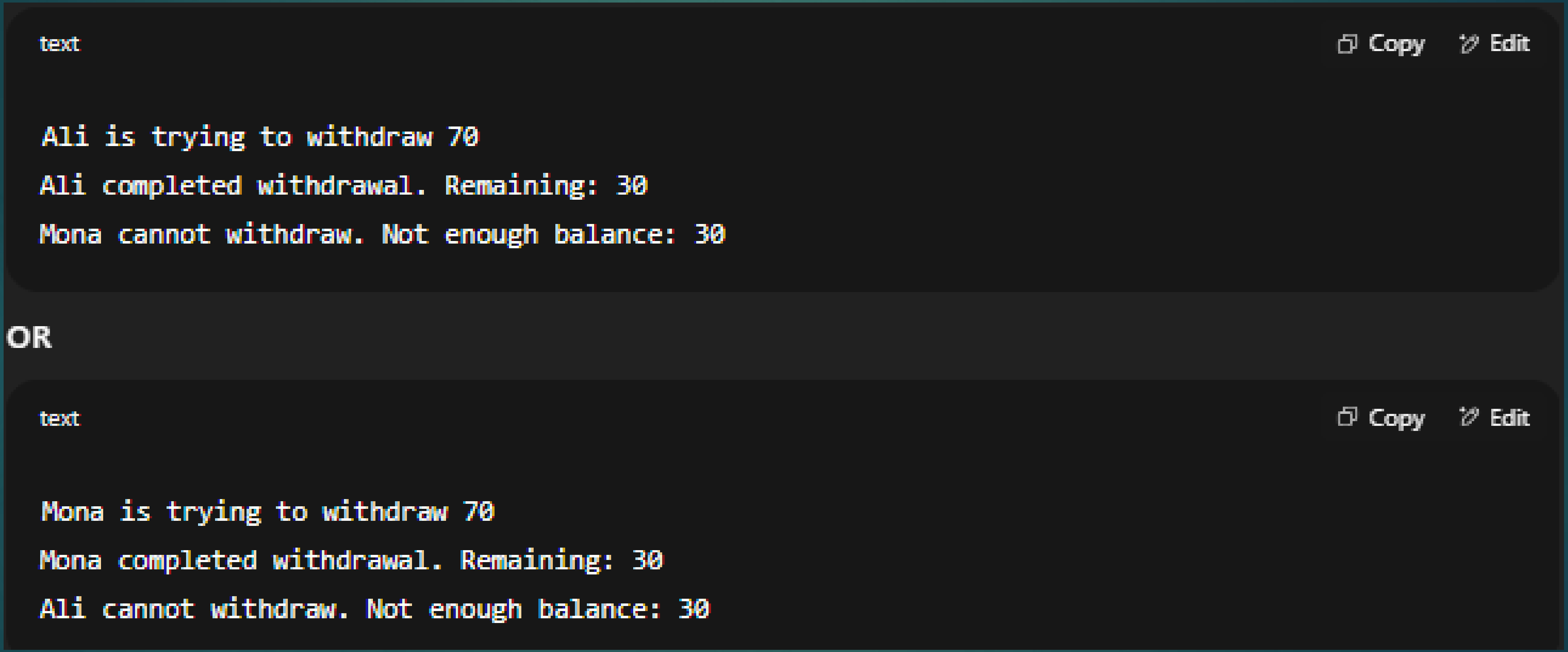
🔧 You Can Synchronize:

- A **method:** `public synchronized void method() { }`
- A **block:** `synchronized(obj) { // code }`

Example:

```java
synchronized(this) {
    count++;
}
```

By Eslam Khder



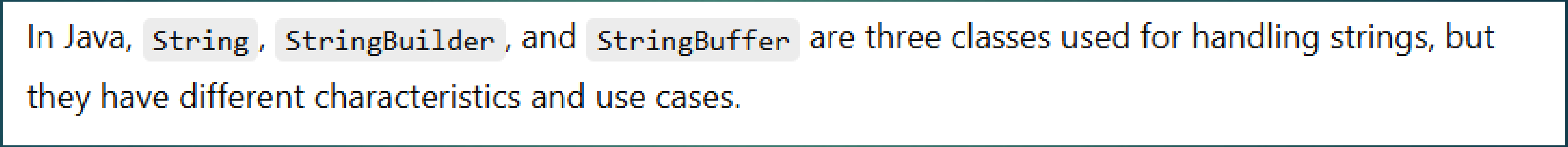
```text
Ali is trying to withdraw 70
Ali completed withdrawal. Remaining: 30
Mona cannot withdraw. Not enough balance: 30
```

OR

```text
Mona is trying to withdraw 70
Mona completed withdrawal. Remaining: 30
Ali cannot withdraw. Not enough balance: 30
```

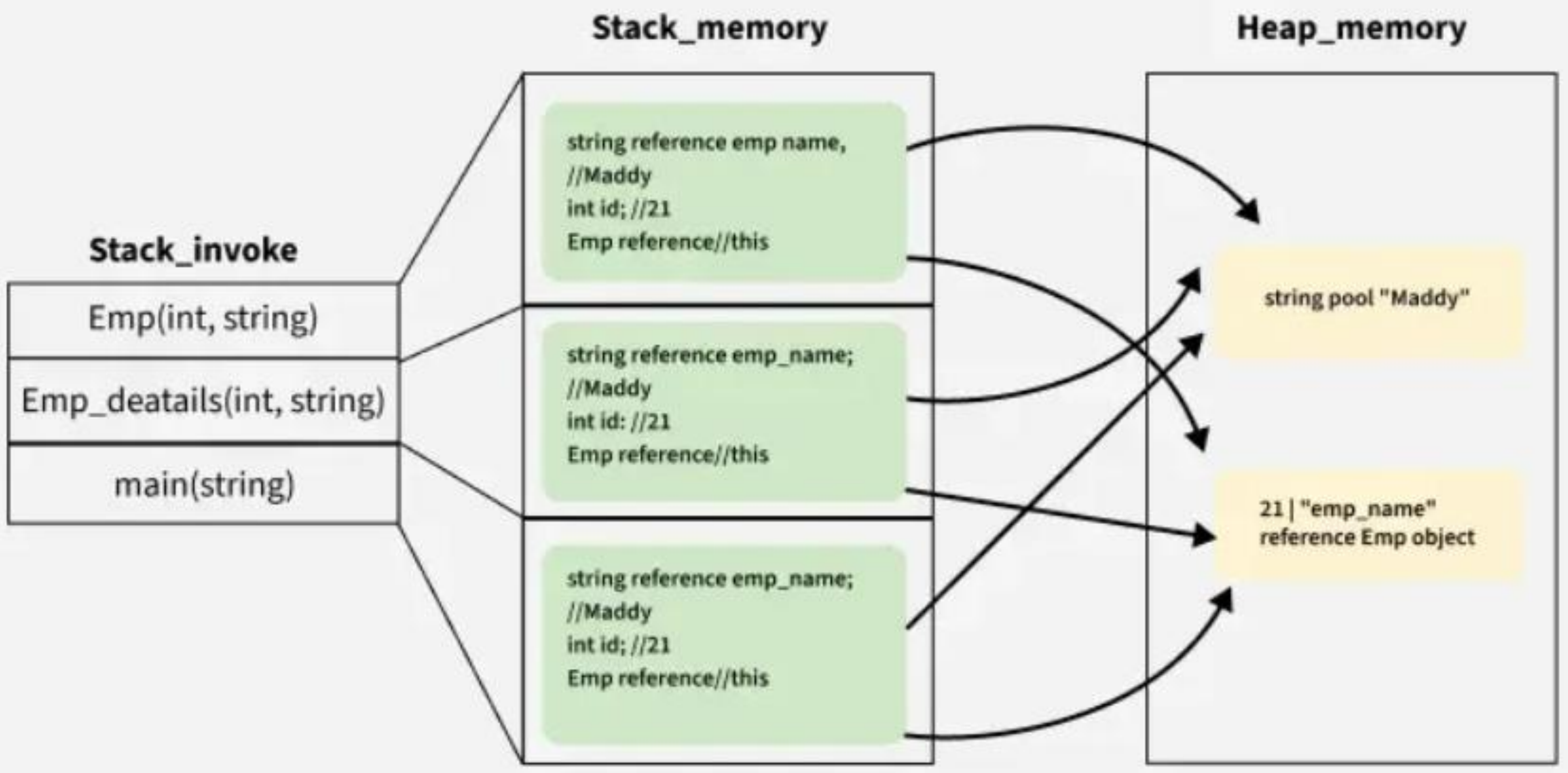
By Eslam Khder

String, StringBuilder, and StringBuffer



In Java, `String`, `StringBuilder`, and `StringBuffer` are three classes used for handling strings, but they have different characteristics and use cases.

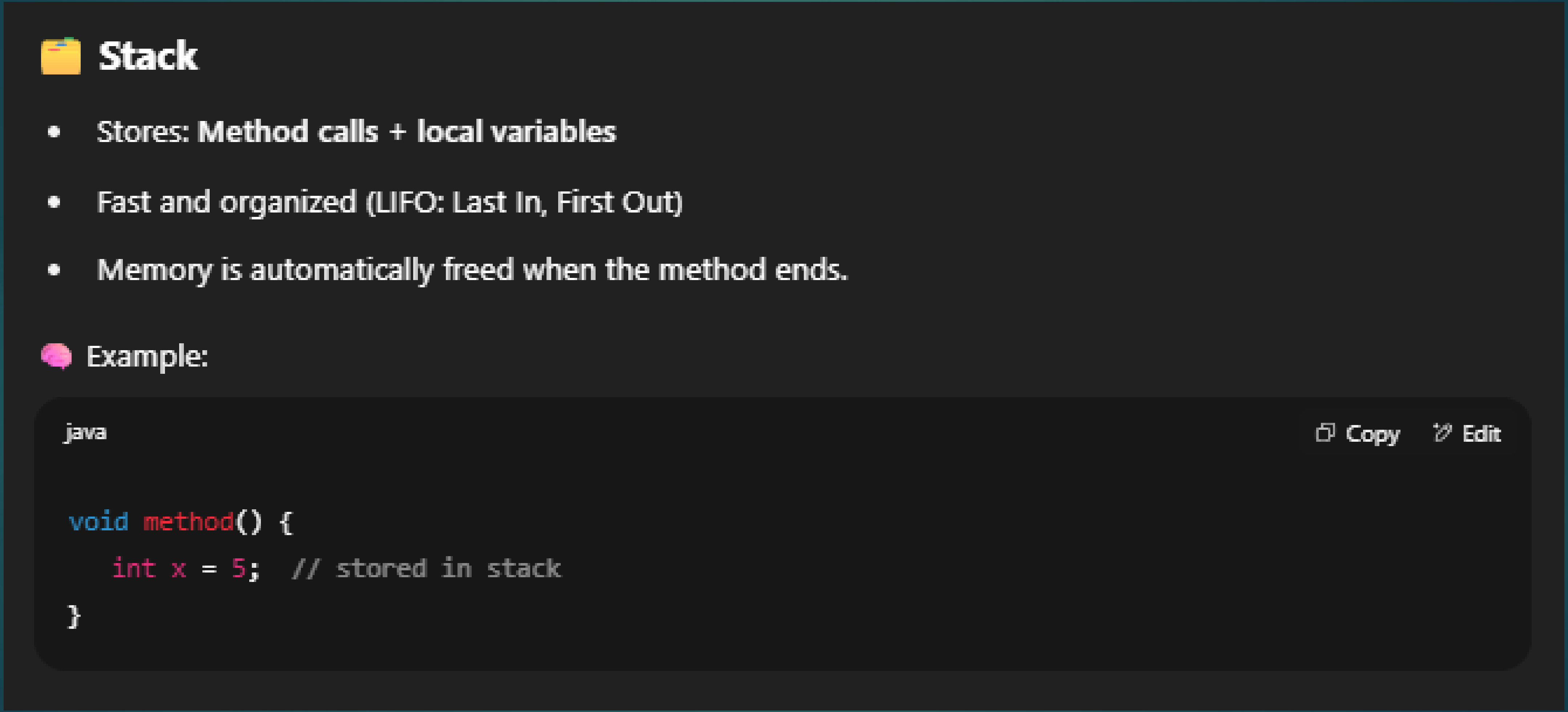
Stack Vs Heap Memory Allocation



By Eslam Khder

spring

EraaSoft



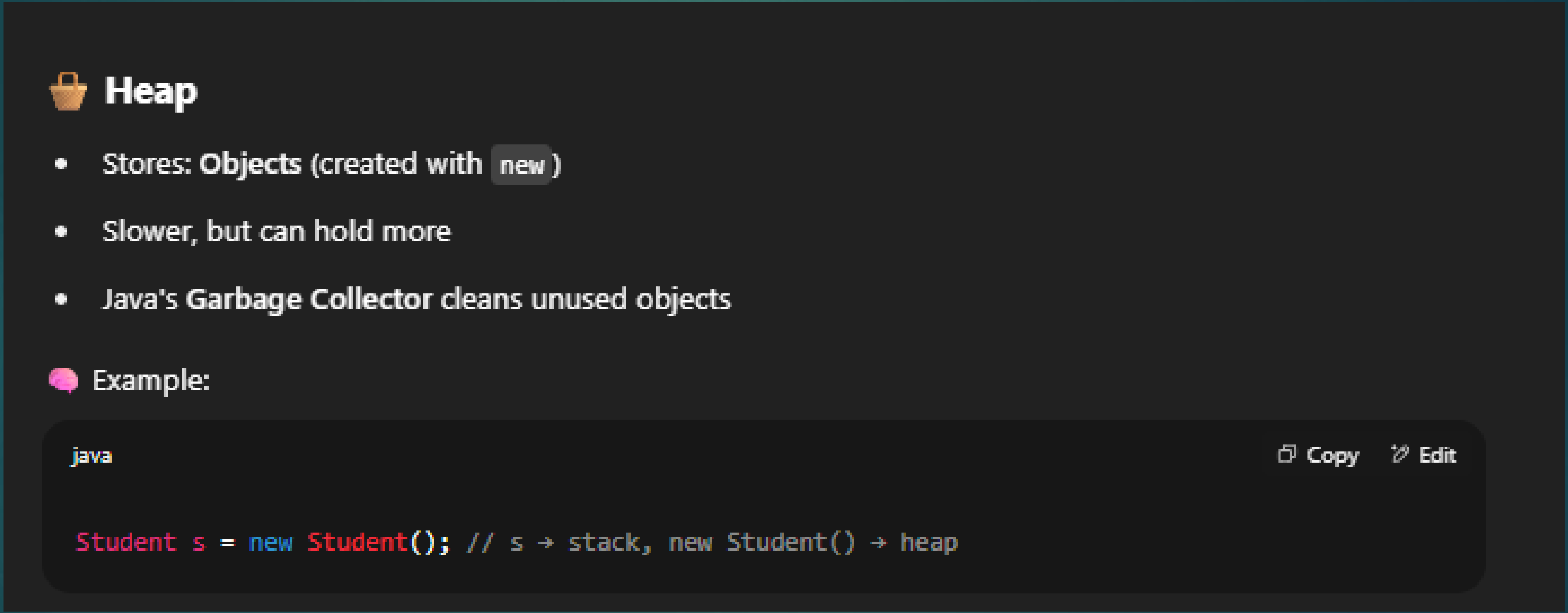
🗂️ Stack

- Stores: **Method calls + local variables**
- Fast and organized (LIFO: Last In, First Out)
- Memory is automatically freed when the method ends.

🧠 Example:

```java
void method() {
    int x = 5;  // stored in stack
}
```

By Eslam Khder

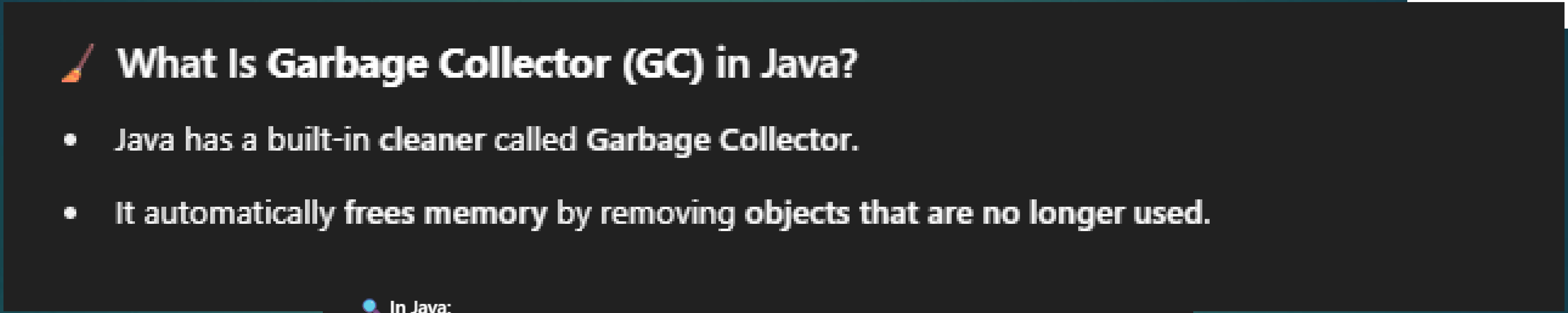


🧺 Heap

- Stores: **Objects** (created with `new`)
- Slower, but can hold more
- Java's **Garbage Collector** cleans unused objects

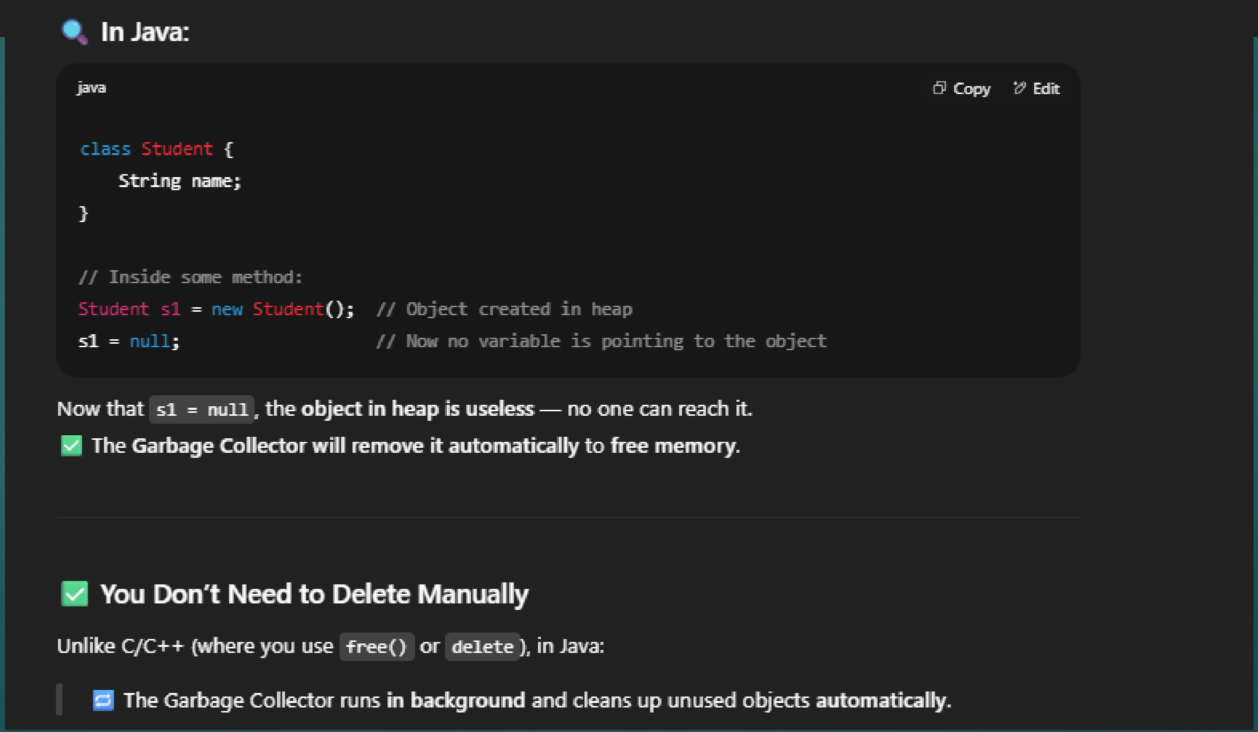
🧠 Example:

```java
Student s = new Student(); // s → stack, new Student() → heap
```



🧹 What Is Garbage Collector (GC) in Java?

- Java has a built-in cleaner called Garbage Collector.
- It automatically frees memory by removing objects that are no longer used.



🔍 In Java:

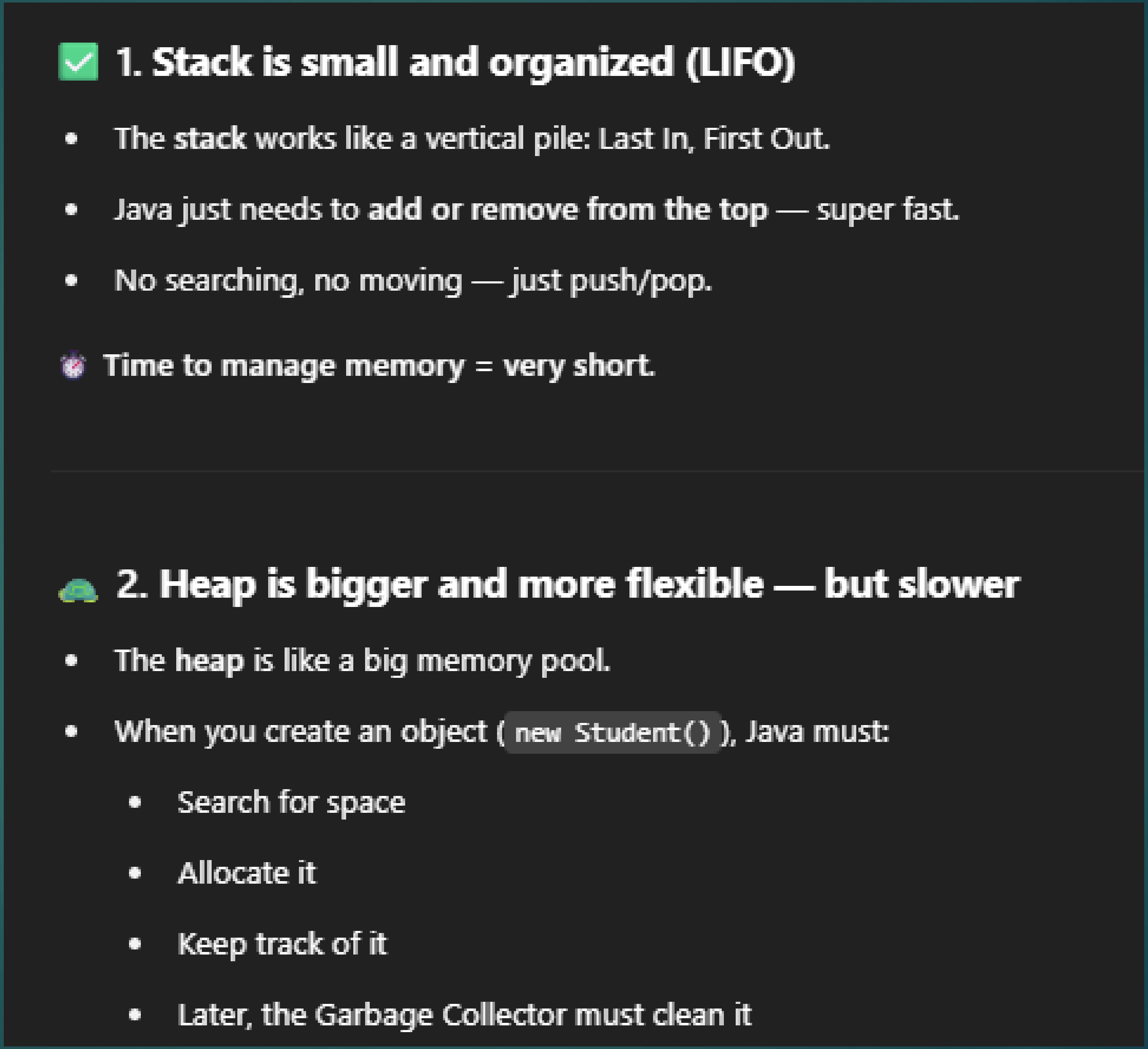
```java
class Student {
    String name;
}

// Inside some method:
Student s1 = new Student();  // Object created in heap
s1 = null;                   // Now no variable is pointing to the object
```

Now that `s1 = null`, the **object in heap is useless** — no one can reach it.
✅ The **Garbage Collector will remove it automatically** to **free memory.**

✅ You Don't Need to Delete Manually

Unlike C/C++ (where you use `free()` or `delete`), in Java:

> 🔄 The Garbage Collector runs **in background** and cleans up unused objects **automatically.**



✅ 1. Stack is small and organized (LIFO)

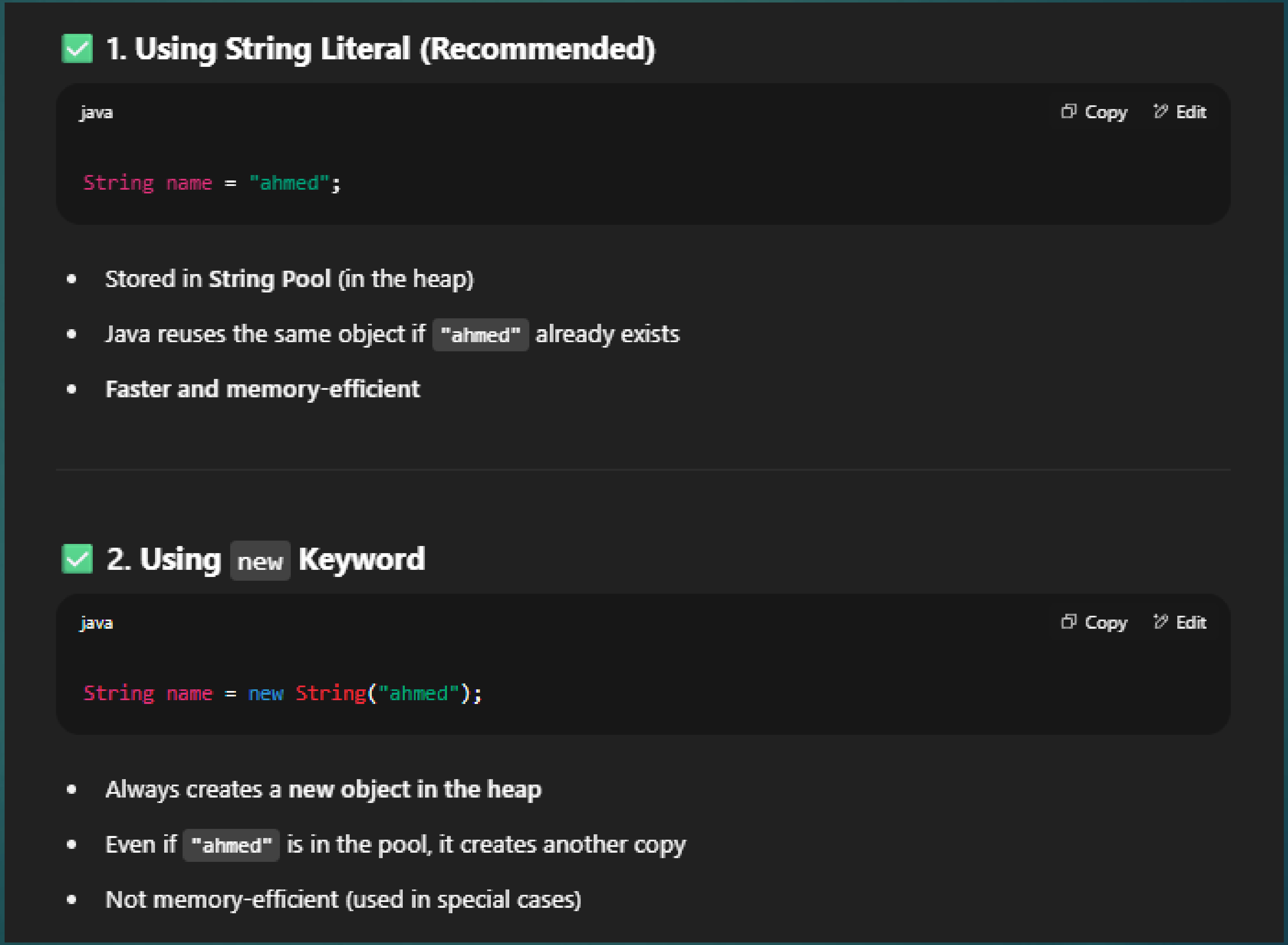
- The stack works like a vertical pile: Last In, First Out.
- Java just needs to add or remove from the top — super fast.
- No searching, no moving — just push/pop.

⏱ Time to manage memory = very short.

🐢 2. Heap is bigger and more flexible — but slower

- The heap is like a big memory pool.
- When you create an object (`new Student()`), Java must:
  - Search for space
  - Allocate it
  - Keep track of it
  - Later, the Garbage Collector must clean it

By Eslam Khder



✅ 1. Using String Literal (Recommended)

java

Copy Edit

```java
String name = "ahmed";
```

- Stored in **String Pool** (in the heap)
- Java reuses the same object if `"ahmed"` already exists
- **Faster and memory-efficient**

✅ 2. Using `new` Keyword

java

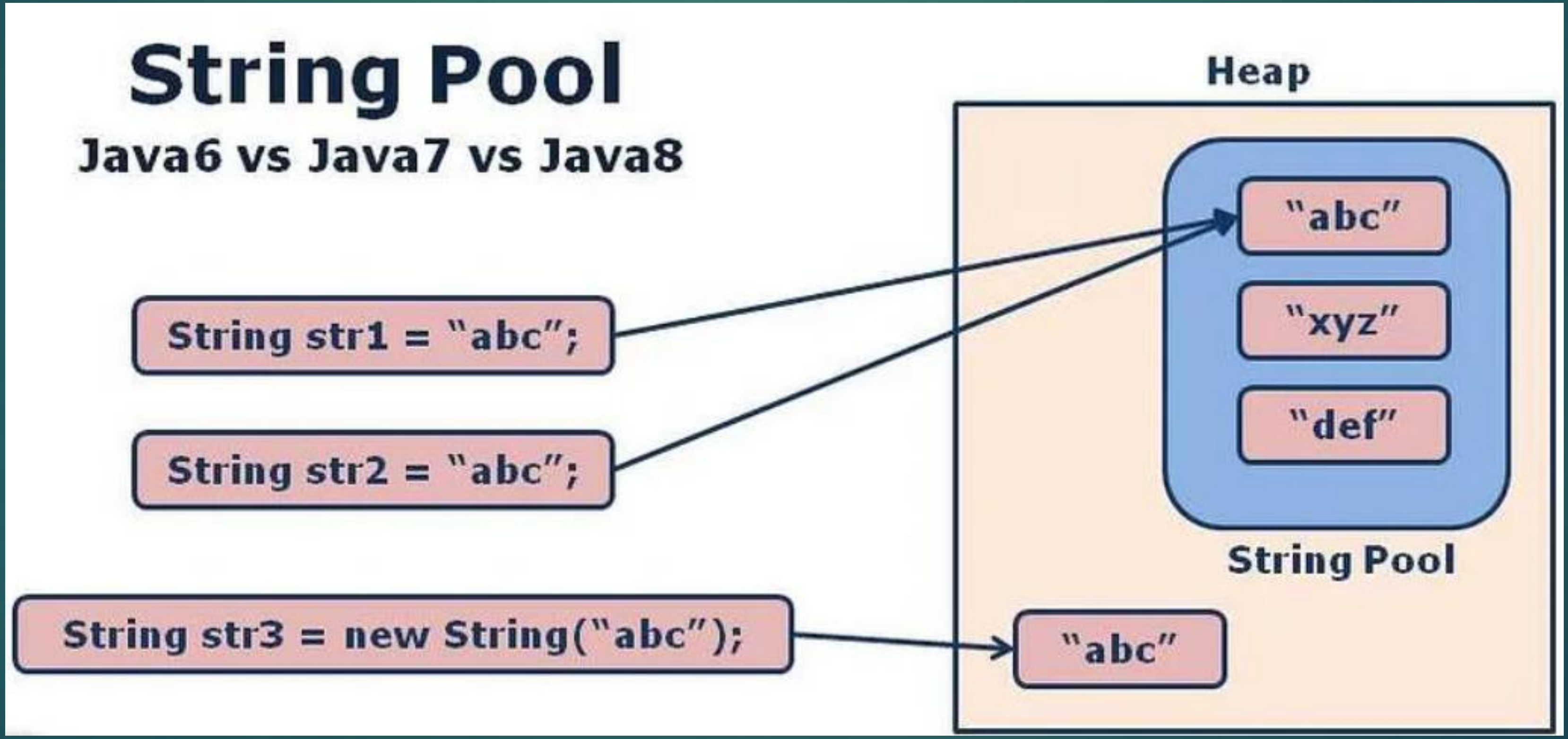
Copy Edit

```java
String name = new String("ahmed");
```

- Always creates a **new object in the heap**
- Even if `"ahmed"` is in the pool, it creates another copy
- Not memory-efficient (used in special cases)

By Eslam Khder

In Java, the term "pool" often refers to a specific area of memory where certain types of objects are stored to optimize resource management and improve performance. Here are some common types



By Eslam Khder

1. String

- **Immutability:** `String` objects are immutable. Once a `String` is created, it cannot be changed. Any modification (like concatenation) creates a new `String` object.
- **Usage:** Ideal for situations where the string content does not change, such as constants or fixed values.
- **Performance:** Since every modification creates a new object, frequent modifications can lead to performance overhead.

java

```java
String str1 = "Hello";
str1 += " World"; // Creates a new String object
```

By Eslam Khder

2. StringBuilder

- **Mutability:** `StringBuilder` is mutable. It allows modifications without creating new objects.
- **Thread Safety:** Not synchronized, which makes it faster than `StringBuffer` but not thread-safe. Use it in single-threaded scenarios.
- **Usage:** Best for situations where you need to modify the string frequently, such as in loops or when constructing large strings.

java

```java
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World"); // Modifies the same object
```

By Eslam Khder

3. StringBuffer

- **Mutability:** Like `StringBuilder`, `StringBuffer` is also mutable and allows modifications.
- **Thread Safety:** It is synchronized, making it thread-safe but slower due to the overhead of synchronization. Use it in multi-threaded scenarios.
- **Usage:** Best when working with strings in a multi-threaded environment where you need to ensure that multiple threads do not modify the same string concurrently.

java

```java
StringBuffer sbf = new StringBuffer("Hello");
sbf.append(" World"); // Modifies the same object in a thread-safe manner
```

By Eslam Khder