

**Threads in Java**

1. **What are Threads?**

Threads are a fundamental concept in **computer science** and are widely used in various applications to achieve concurrency, improve responsiveness, and utilize resources efficiently.

1. **Applications of Threads:**

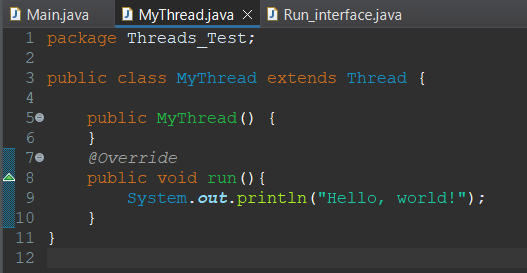
* **Multithreaded Servers:** where threads handle multiple client requests simultaneously, allowing the server to serve multiple clients concurrently.
* **Parallel Processing:** to execute tasks concurrently, enhancing performance in tasks such as scientific simulations and data processing.
* **Background Tasks:** Threads handle background tasks like periodic maintenance, logging, and monitoring, ensuring they run independently of the main program flow.
* **Thread Pools:** Thread pools are employed to manage reusable threads efficiently, enhancing performance by minimizing thread creation and destruction overhead.

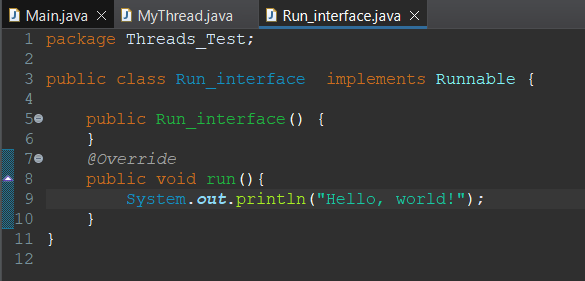
1. **Types of Threads:**

**User-level Threads** (ULTs): Managed entirely by the application and not visible to the operating system kernel. They are lightweight but can't benefit from multiple processor cores simultaneously. **Kernel-level Threads** (KLTs): Managed by the operating system kernel. They are more heavyweight but can take advantage of multiple processor cores.

1. **Threads in Java:**

* How to **create** a Thread in Java?

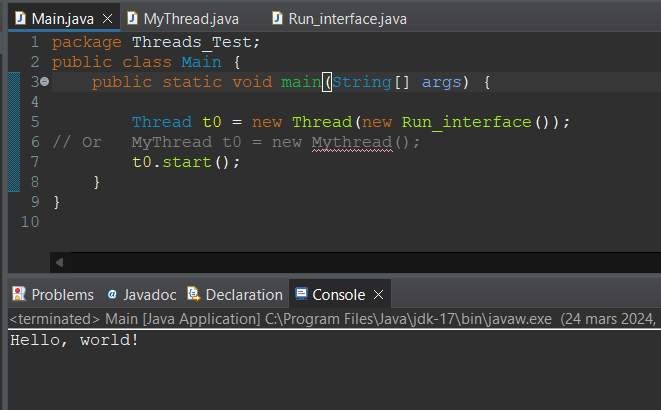
In Java, you can create threads using either of two methods: **Subclassing** the **Thread** class and overriding its run() method:

or **Implementing** the **Runnable** interface and implements its run() method:

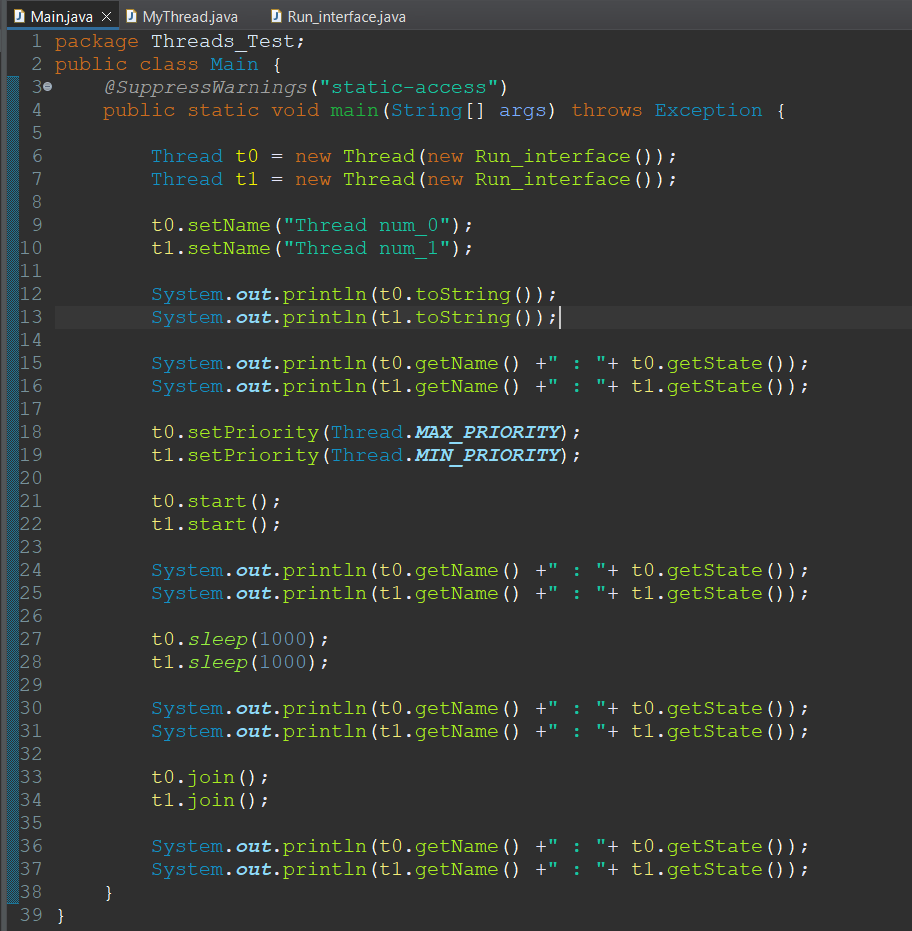
When extending the Thread class, your class cannot extend any other class due to **Java's single inheritance limitation**. Conversely, implementing the Runnable interface allows your class to extend other classes while still enabling multithreading.

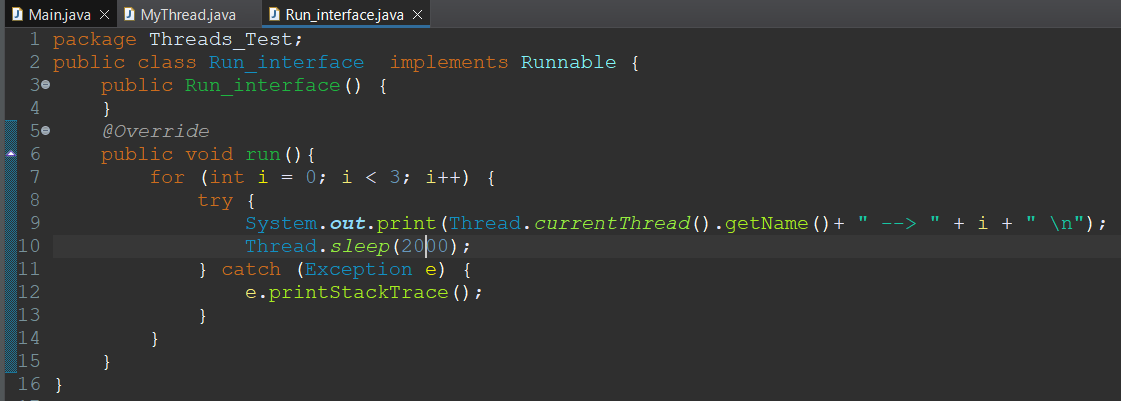
* **Running** Threads in Java:

You can run a thread either by creating an instance of **Run\_interface** class and pass it to Thread constructor, or by creating an instance of **MyThread** class then start the thread by using the method **start()**.



* Java Thread Common Methods:
* **sleep(long** millis**)**: Pauses the execution of the current thread for the specified amount of time.
* **currentThread()**: Returns a reference to the currently executing thread object.
* **join()**: Waits for the thread on which it's called to die.
* **getPriority()**: Returns the priority of the thread.
* **setPriority(int** priority**)**: Changes the priority of the thread.
* **getName()**: Returns the name of the thread.
* **setName(String** name**)**: Changes the name of the thread.
* **getId()**: Returns the id of the thread.
* **toString()**: Returns a string representation of this thread, including the thread's name, priority, and thread group.
* **isAlive()**: Tests if the thread is alive.
* **getState()**: Returns the state of the thread.
* **suspend()**: Suspends the thread.
* **interrupt()**: Interrupts the thread.
* **isInterrupted()**: Tests whether the thread has been interrupted.
* **resume()**: Resumes the suspended thread.
* **stop()**: Stops the thread (deprecated and not recommended for general use).

**Example:**



**Output:**

