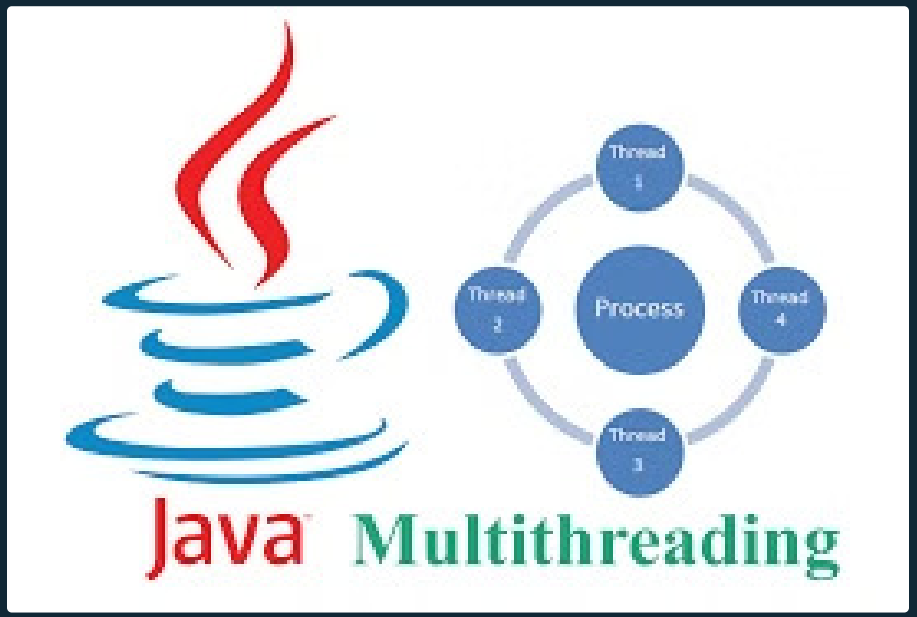
Multithreading in Java



1. CPU, Core, Program, and Process
2. Threads, Multitasking, and Multithreading
3. Context Switching
4. Thread Lifecycle & States
5. Creating Threads in Java
6. Priority & Synchronization
7. Reentrant Lock

CPU, Core, Program, and Process

CPU

The CPU executes instructions from programs.

Core

A core is a processing unit within a CPU.

Program

A program is a set of instructions for a task.

Process

A process is an instance of a running program.

Threads, Multitasking, and Multithreading

Thread

Smallest unit of execution within a process.

Multitasking

OS runs multiple processes simultaneously.

Multiprocessing

Multiple processors executing tasks.

Multithreading

Multiple threads execute within a process.

Context Switching

- Context switching is the process of saving the state of a currently running process or thread and restoring the state of another process or thread so that execution can continue.

Process Context Switch

- Slower, more overhead.

Thread Context Switch

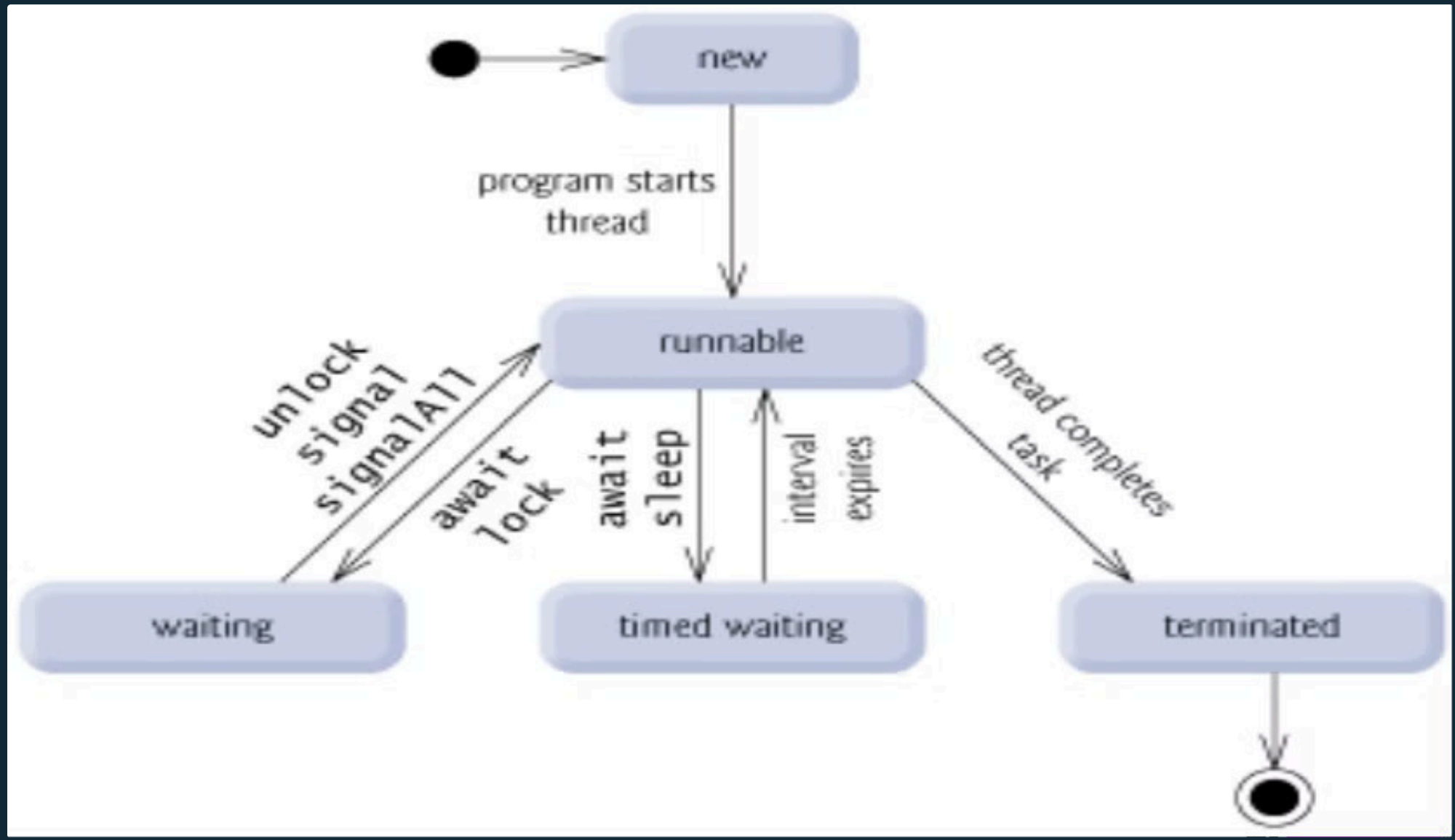
- Faster, less overhead.

Context Switching

- Switching between processes is more expensive than switching between threads.

Thread Lifecycle & States

Thread States



Creating Threads in Java

Extending **Thread**

Create a class that extends the **Thread** class. Override the **run()** method with your task's code.

- Simple to implement.
- Inherits thread functionality.
- Limited, can't extend other classes.

Implementing **Runnable**

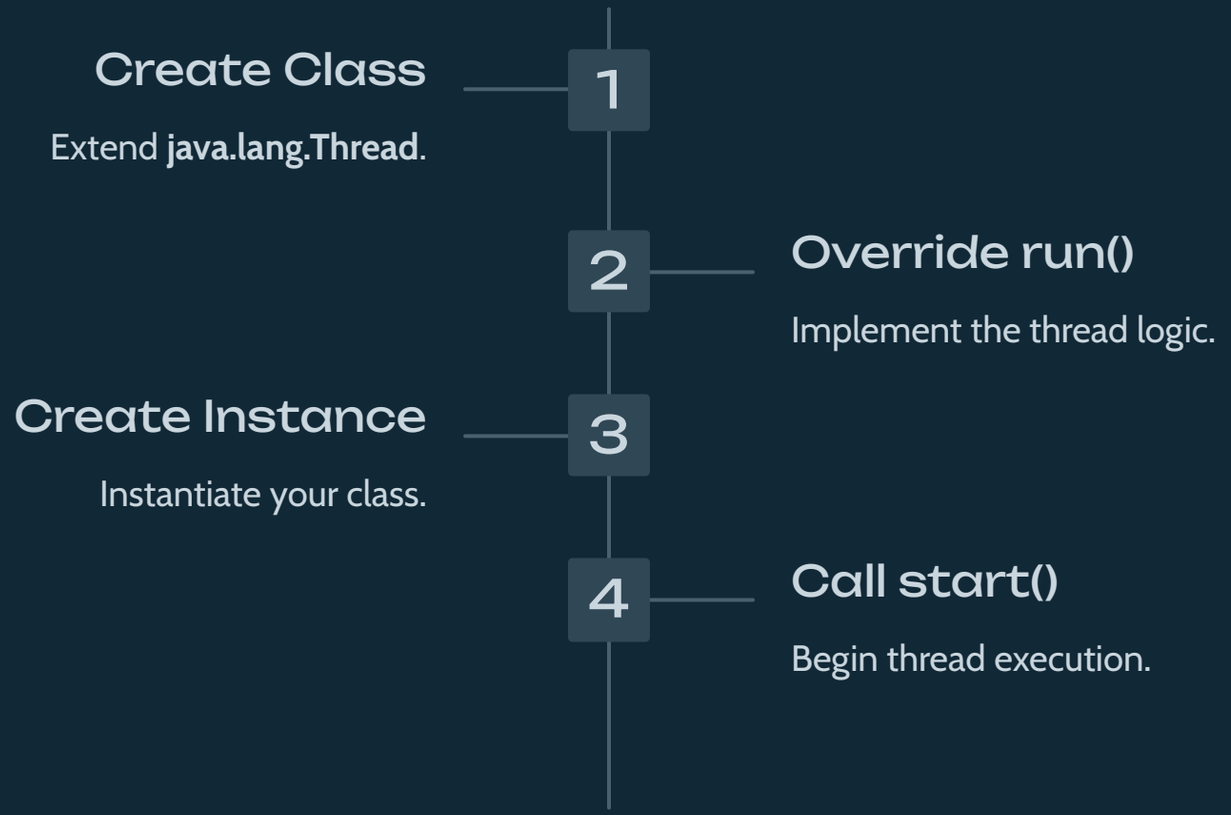
Create a class that implements the **Runnable** interface. Define the task within the **run()** method.

- More flexible approach.
- Allows extending other classes.
- Recommended for most cases.

Extending the Thread Class

To create a thread by extending the **Thread** class, you need to:

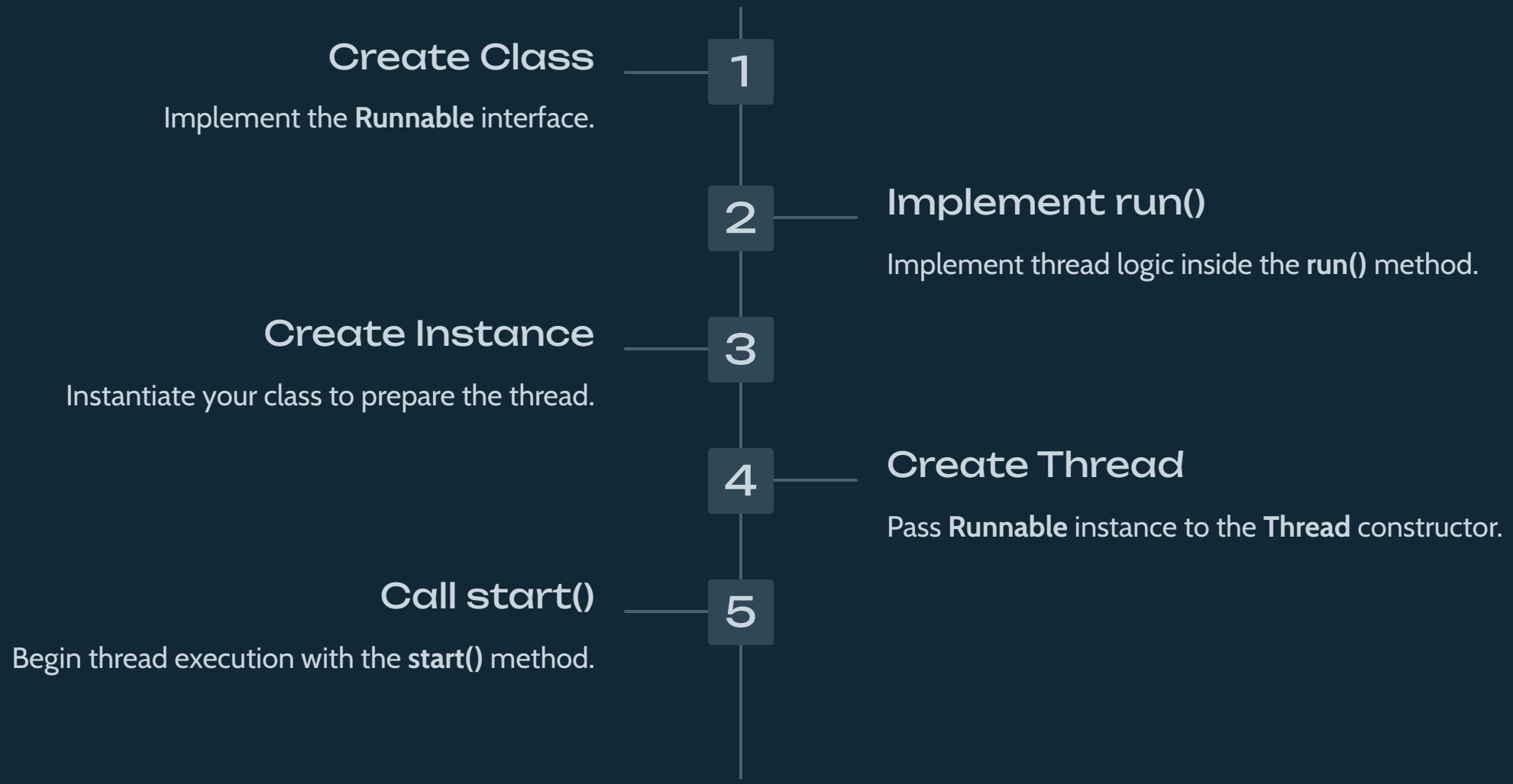
1. Create a class that inherits from **Thread**.
2. Override the **run()** method, which contains the code that will be executed by the thread.
3. Create an instance of your class and call the **start()** method to begin execution.



Implementing the Runnable Interface

To create a thread by implementing the **Runnable** interface, you need to:

1. Create a class that implements **java.lang.Runnable**.
2. Implement the **run()** method, which contains the code to be executed by the thread.
3. Create an instance of your class.
4. Create a **Thread** object, passing your **Runnable** instance to the **Thread** constructor.
5. Call the **start()** method of the **Thread** object.



Priority Thread

Each thread has a priority. Priorities determine the order of execution.

```
l.setPriority(Thread.MIN_PRIORITY);// 1
m.setPriority(Thread.NORM_PRIORITY);// 5
n.setPriority(Thread.MAX_PRIORITY);// 10
```

Set Priority

Use **setPriority()** method.

Get Priority

Use **getPriority()** method.

Range

From 1 to 10, 10 is highest.

Thread Synchronization

It ensures data integrity by managing access to shared resources.

Mutual Exclusion

Only one thread can access the resource at a time. Prevents multiple threads from simultaneously reading or writing.

Monitors

Controls access to objects with synchronized methods. Ensures proper locking and unlocking of resources.

Reentrant Lock

lock()

Acquires the lock, blocking until available.

unlock()

Releases the lock held by the current thread.

A Reentrant Lock in Java allows a thread to acquire the same lock multiple times. It prevents deadlock, offering flexibility over synchronized. Use ReentrantLock from java.util.concurrent.locks for advanced thread coordination.

Thanks