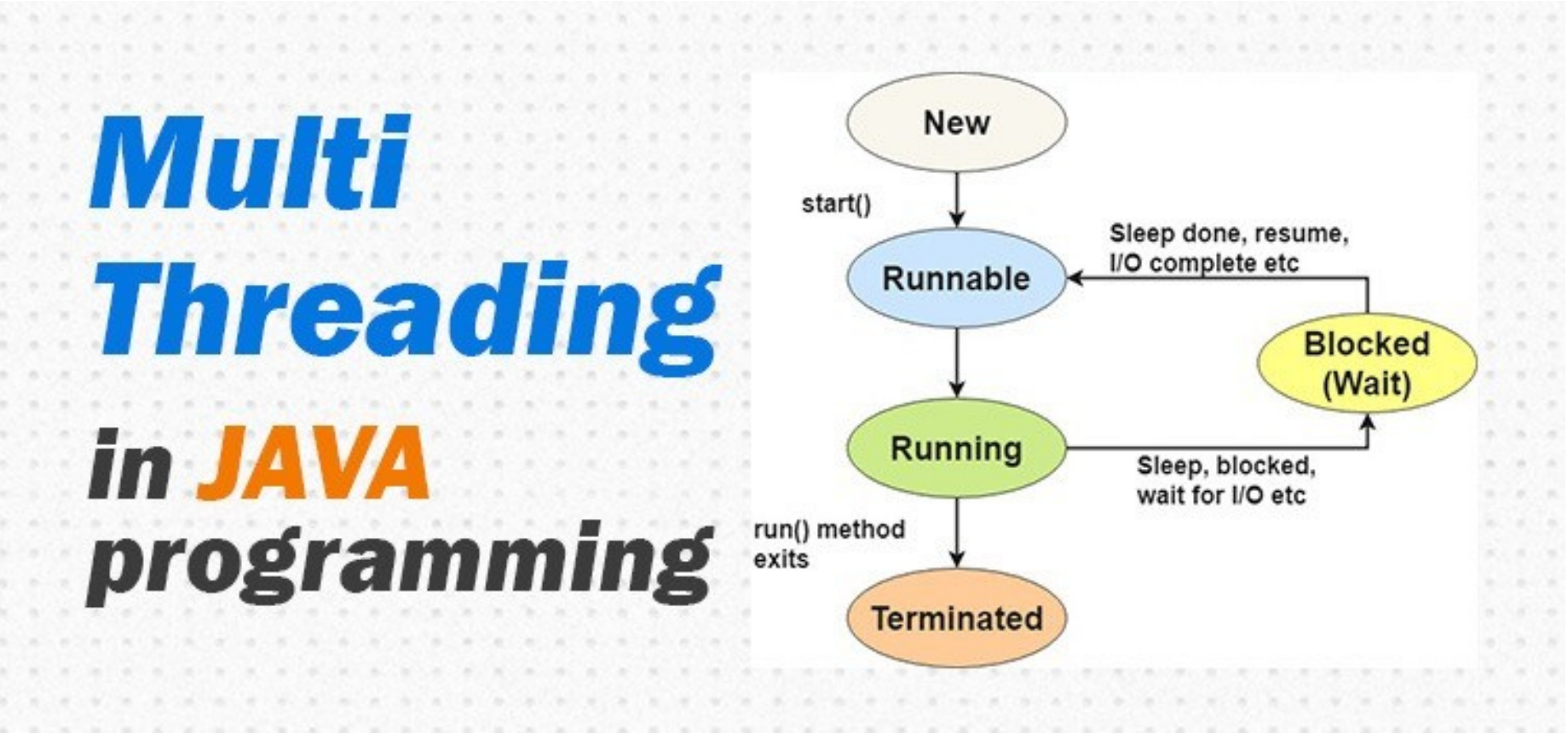
Multithreading

Multithreading



Multithreading

- Threads are the smallest unit of execution within a process in a multitasking operating system.
- They provide a way to execute multiple tasks concurrently, improving the overall performance and responsiveness of a program or system.
- In Java, threads are implemented through the Thread class or the Runnable interface.
- Threads allows a program to operate more efficiently by doing multiple things at the same time.
- Threads can be used to perform complicated tasks in the background without interrupting the main program.

Multithreading

- **Thread**:
- A thread in Java is the direction or path that is taken while a program is being executed.
- Generally, all the programs have at least one thread, known as the main thread, that is provided by the JVM or Java Virtual Machine at the starting of the program's execution.
- At this point, when the main thread is provided, the main() method is invoked by the main thread.
- A thread is an execution thread in a program.
- Multiple threads of execution can be run concurrently by an application running on the Java Virtual Machine.

Multithreading

- **Thread:**
- The priority of each thread varies.
- Higher priority threads are executed before lower priority threads.
- Thread is critical in the program because it enables multiple operations to take place within a single method.
- Each thread in the program often has its own program counter, stack, and local variable.

Multithreading

- **Creating a Thread**
- There are two ways to create a thread.
  - It can be created by extending the Thread class and overriding its run() method:
  - Another way to create a thread is to implement the Runnable interface:

Multithreading

```
1 public class Main extends Thread {
2   public void run() {
3 System.out.println("This code is running in a thread");
4   }
5 }
```

```
1 public class Main implements Runnable {
2   public void run() {
3     System.out.println("This code is running in a thread");
4   }
5 }
```

Multithreading

- ***Running Threads:***
- *If the class extends the Thread class, the thread can be run by creating an instance of the class and call its **start()** method:*

```
1  public class Main extends Thread {
2    public static void main(String[] args) {
3      Main thread = new Main();
4      thread.start();
5      System.out.println("This code is outside of the thread");
6    }
7    public void run() {
8      System.out.println("This code is running in a thread");
9    }
10 }
```

Multithreading

- ***Running Threads:***
- *If the class implements the Runnable interface, the thread can be run by passing an instance of the class to a Thread object's constructor and then calling the thread's start() method:*

```
1  public class Main implements Runnable {
2    public static void main(String[] args) {
3      Main obj = new Main();
4      Thread thread = new Thread(obj);
5      thread.start();
6      System.out.println("This code is outside of the thread");
7    }
8    public void run() {
9      System.out.println("This code is running in a thread");
10   }
11 }
```

Multithreading

- *Differences between "extending" and "implementing" Threads*
  - *The major difference is that when a class extends the Thread class, you cannot extend any other class, but by implementing the Runnable interface, it is possible to extend from another class as well, like: class MyClass extends OtherClass implements Runnable.*

Multithreading

- Thread Priorities
  - In Java, thread priorities are used to influence the order in which threads are scheduled to run by the thread scheduler.
  - Each thread is assigned a priority that determines its relative importance or urgency compared to other threads.
  - Thread priorities are represented as integers and range from Thread.
    - MIN_PRIORITY (1) to Thread.MAX_PRIORITY (10), with Thread.NORM_PRIORITY being the default priority

Multithreading

- **Thread Priorities**:
- *In this example, Thread-1 has the minimum priority, and Thread-2 has the maximum priority.*
- *The output might reflect the scheduling decisions made by the thread scheduler based on these priorities.*
- *Keep in mind that the actual behavior may vary depending on the operating system and other factors.*

```
1  class MyRunnable implements Runnable {
2      public void run() {
3          for (int i = 0; i < 5; i++) {
4              System.out.println(Thread.currentThread().getName() + " - Count: "
   + i);
5          }
6      }
7  }
8  public class PriorityExample {
9      public static void main(String[] args) {
10         Thread thread1 = new Thread(new MyRunnable(), "Thread-1");
11         Thread thread2 = new Thread(new MyRunnable(), "Thread-2");
12         // Setting thread priorities
13         thread1.setPriority(Thread.MIN_PRIORITY);
14         thread2.setPriority(Thread.MAX_PRIORITY);
15         // Starting threads
16         thread1.start();
17         thread2.start();
18     }
19 }
20
```

Multithreading

- *Life cycle of a Thread (Thread states)*
  - *The life cycle of a thread in Java represents the different states that a thread can go through from its creation to its termination.*
  - *The Java Thread class provides methods to query and influence the state of a thread.*
  - *The thread states are as follows:*
    - *New*
    - *Runnable*
    - *Blocked*
    - *Waiting*
    - *Timed Waiting*
    - *Terminated*

Multithreading

- *New:*
  - *The thread is in this state when it is first created using the new Thread() constructor.*
  - *The thread is not yet started with the start() method.*
- Runnable:
  - After calling the start() method, the thread becomes runnable.
  - The thread scheduler selects it to run, but it may not start immediately.
  - It's waiting for CPU time.
  - Threads in this state are eligible to run, but the actual execution is controlled by the operating system.

Multithreading

- *Blocked:*
  - *A thread transitions to the blocked state when it needs to wait for a monitor lock to enter a synchronized block/method or when waiting for I/O operations to complete.*
  - *When the condition for which the thread was waiting is satisfied, it returns to the runnable state*

- Waiting:
  - A thread enters the waiting state when it calls the wait() method.
  - It remains in this state until another thread calls notify() or notifyAll() on the same object.

Multithreading

- *Timed Waiting:*
  - *Similar to the waiting state but with a specified time duration.*
  - *Threads in this state will automatically transition back to the runnable state when the specified time period elapses.*

- Terminated:
  - A thread enters the terminated state when it completes its execution or when an uncaught exception occurs.
  - Once a thread is terminated, it cannot be restarted.

Multithreading

- *In this example, the main thread creates a new thread (myThread), starts it, and then simulates different states such as sleep, waiting, etc.*
- *Keep in mind that the actual transitions between states may vary depending on the thread scheduler and other factors.*

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Thread is in the Runnable state");
        try {
            Thread.sleep(2000); // Simulate some work
        } catch (InterruptedException e) {
            e.printStackTrace();}
        System.out.println("Thread is now in the Terminated state");}}
public class ThreadLifeCycleExample {
    public static void main(String[] args) {
        Thread myThread = new Thread(new MyRunnable());
        // New state
        System.out.println("Thread is in the New state");
        // Runnable state
        myThread.start();
        // Main thread may continue its execution
        // ...
        // Thread may be in Blocked or Timed Waiting state
        // ...
        // Thread may enter Waiting state
        synchronized (myThread) {
            try {
                myThread.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();}}
        // Terminated state
        System.out.println("Thread is in the Terminated state");}}
```

Assignment

- What is a thread in the context of computer programming?
- Explain the difference between a process and a thread.
- How does multithreading contribute to the performance improvement of a program?
- Describe two ways to create a thread in Java.
- When is it preferable to implement the Runnable interface instead of extending the Thread class for creating a thread?
- Explain the purpose of the start() method in the Thread class.
- What is the purpose of thread priorities in Java?
- Enumerate the different states a thread can be in during its life cycle.