**Multithreading**

**🔷 What is Multithreading?**

Multithreading is a programming concept where multiple threads run **concurrently** within a single process.

| **Term** | **Description** |
| --- | --- |
| **Thread** | A lightweight process. It's the smallest unit of execution. |
| **Process** | An independent program in execution. A process can have multiple threads. |
| **Multithreading** | Ability of a CPU (or a program) to execute multiple threads simultaneously. |

**🔹 Basic Terminology**

**🔸 Thread Lifecycle (States)**

A thread can be in one of the following states:

New → Runnable → Running → (Waiting/Blocked/Sleeping) → Terminated

**Description:**

* **New**: Thread is created but not started.
* **Runnable**: Thread is ready to run but waiting for CPU.
* **Running**: Thread is executing.
* **Blocked/Waiting**: Thread is waiting for some resource.
* **Terminated**: Thread has finished execution or has been stopped.

**🔸 Ways to Create Threads in Java**

**✅ 1. By Extending the Thread class**

class MyThread extends Thread {

public void run() {

System.out.println("Thread is running...");

}

}

public class Main {

public static void main(String[] args) {

MyThread t = new MyThread();

t.start(); // starts a new thread and calls run()

}

}

**✅ 2. By Implementing the Runnable Interface**

class MyRunnable implements Runnable {

public void run() {

System.out.println("Runnable thread is running...");

}

}

public class Main {

public static void main(String[] args) {

MyRunnable r =new MyRunnable();

Thread t = new Thread(r);

t.start();

}

}

**Note**: Prefer Runnable if you only need to run code in a thread and don't want to inherit from Thread.

**🔐 What is Thread Synchronization?**

**Thread synchronization** is the process of controlling the access of multiple threads to **shared resources** to prevent **data inconsistency** (also known as race conditions).

When multiple threads try to modify the same data simultaneously, the result can be unpredictable. Synchronization ensures that only **one thread** can access the critical section (shared code/resource) at a time.

**🔸 The Problem Without Synchronization**

Let’s say two threads are adding money to a bank account at the same time:

class BankAccount {

int balance = 1000;

void deposit(int amount) {

balance += amount; // Critical section

}

}

Without synchronization, if two threads modify balance at the same time, the final value may be incorrect due to **race conditions**.

**✅ Solution: synchronized Keyword**

You can **synchronize**:

1. A method → synchronized method
2. A block of code → synchronized block

**🔹 1. Synchronized Method**

class BankAccount {

int balance = 1000;

public synchronized void deposit(int amount) {

balance += amount; // Only one thread can execute this at a time

}

}

* **Lock** is associated with the object.
* Only one thread can access this method for an object at a time.

**🔹 2. Synchronized Block**

Use this when you want to synchronize **only part** of the method:

class BankAccount {

int balance = 1000;

public void deposit(int amount) {

synchronized (this) {

balance += amount;

}

}

}

This is more **efficient** than synchronizing the whole method.

**🔸 Static Synchronization**

If you make a static method synchronized, the **class-level lock** is used (not object-level).

class Logger {

public static synchronized void log(String message) {

System.out.println(message);

}

}

**Java Multithreading Interview Questions with Answers**

**✅ 1. What is a thread in Java?**

A thread is a lightweight sub-process, the smallest unit of execution. It runs independently within a program and shares the process memory**.**

**✅ 2. Difference between process and thread?**

| **Process** | **Thread** |
| --- | --- |
| Independent | Part of a process |
| Has its own memory | Shares memory |
| Heavyweight | Lightweight |
| Slower communication | Faster communication |

**✅ 3. Difference between start() and run()?**

* start() → starts a new thread and internally calls run().
* run() → just a method call; does not start a new thread.

**✅ 4. What is Thread.sleep()?**

Puts the thread to sleep (pause) for the given time (in milliseconds), allowing other threads to execute.

**✅ 5. What is the purpose of Thread.join()?**

It pauses the current thread until the thread on which join() was called finishes its execution.

**✅ 6. What is synchronization?**

Synchronization ensures that only one thread can access a block of code or method at a time, preventing race conditions.

**✅ 7. Difference between synchronized method and block?**

* Synchronized method locks the entire method.
* Synchronized block locks only a specific section of code, offering better performance.

**✅ 8. Can a constructor be synchronized?**

No, because constructors are not inherited and locks are associated with object instances.

**✅ 9. What is a daemon thread?**

A background thread that runs in the background and dies when all non-daemon threads finish.

**✅ 10. Difference between wait() and sleep()?**

| **wait()** | **sleep()** |
| --- | --- |
| Releases the lock | Doesn’t release lock |
| Defined in Object class | Defined in Thread class |
| Used for inter-thread communication | Used to pause execution |

**✅ 11. What is a deadlock?**

A situation where two or more threads are blocked forever, waiting for each other’s lock.