Threads in Java - Complete Notes

What is a Thread?

A Thread is a lightweight subprocess, the smallest unit of execution. Java supports multithreading, allowing multiple threads to run concurrently. Threads share the same memory space, which helps in better performance.

Advantages of Multithreading

- Improved Performance
- Better Resource Utilization
- Asynchronous Behavior
- Responsiveness
- Less Memory Overhead

Ways to Create a Thread

1. By Extending the Thread Class

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running: " + Thread.currentThread().getName());
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start();
    }
}
```

2. By Implementing the Runnable Interface

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Runnable Thread: " + Thread.currentThread().getName());
    }
}

public class Main {
    public static void main(String[] args) {
        Thread t1 = new Thread(new MyRunnable());
        t1.start();
```

```
    }
}
```

Thread Lifecycle

States: New -> Runnable -> Running -> Blocked/Waiting -> Terminated

```
class DemoThread extends Thread {
    public void run() {
        System.out.println("Thread Running...");
    }
}

public class Test {
    public static void main(String[] args) {
        DemoThread t = new DemoThread();
        t.start();
    }
}
```

Thread Methods Explained

1. start()

Starts a new thread. Internally calls the run() method in a separate call stack.

MyThread t = new MyThread();
t.start();

2. run()

Contains the code that the thread will execute. If you call run() directly, it won't start a new thread.

```
public void run() {
    System.out.println("Thread code here");
}
```

3. sleep(milliseconds)

Temporarily pauses the thread for specified milliseconds.

```
try {
    Thread.sleep(1000); // pauses for 1 second
```

```
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

4. join()

Waits for the thread to die. Useful when one thread must finish before others continue.

```
Thread t1 = new Thread(() -> {
    System.out.println("Thread 1");
});
t1.start();
t1.join(); // main thread waits for t1 to finish
```

5. yield()

Hints to the thread scheduler that the current thread is willing to yield CPU to other threads.

```
Thread.yield();
```

6. isAlive()

Returns true if the thread is still running.

```
Thread t = new Thread(() -> {});
t.start();
System.out.println(t.isAlive()); // true if thread is active
```

7. setName() and getName()

Used to set or get the name of a thread.

t.setName("WorkerThread");
System.out.println(t.getName());

8. setPriority() and getPriority()

Sets or gets the priority of a thread. Ranges from 1 (MIN) to 10 (MAX). Default is 5.

t.setPriority(Thread.MAX_PRIORITY);
System.out.println(t.getPriority());

Threads in Java - Complete Notes

Synchronization in Threads

Used to prevent race conditions in multithreaded programs. Ensures only one thread can access a block of code at a time.

```
class Counter {
    int count = 0;
    public synchronized void increment() {
        count++;
    }
}
```

```
public void increment() {
    synchronized(this) {
        count++;
    }
}
```

```
Thread: Lightweight subprocess
Creation: Thread class or Runnable
Life Cycle: New -> Runnable -> Running -> Blocked -> Terminated
Important Methods: start(), sleep(), join(), yield(), isAlive(), setName(), getName(), setPriority(), getPriority()
Synchronization: To avoid inconsistency
```