In Java, when creating a new thread, you have two primary options:

1. **Extending the Thread class**
2. **Implementing the Runnable interface**

**Why implements Runnable is generally better than extends Thread:**

**1. Java supports single inheritance**

* Java does not support multiple inheritance, meaning a class can only extend **one** other class.
* If you extend Thread, your class **cannot** extend any other class, which limits its flexibility.
* However, by implementing Runnable, your class can still extend another class and gain additional functionalities.

**Example** (Problem with extends Thread):

class MyThread extends Thread {

// Custom thread implementation

}

class MyOtherClass extends MyThread { // This is fine

}

Now, if MyThread was already extending another class:

class SomeBaseClass {

// Some functionality

}

class MyThread extends SomeBaseClass { // Now it can't extend Thread

}

class MyOtherClass extends MyThread { // Cannot extend Thread anymore

}

**Solution: Use Runnable**

class SomeBaseClass {

// Some functionality

}

class MyRunnable implements Runnable {

public void run() {

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

}

}

class MyOtherClass extends SomeBaseClass { // Can still extend another class

}

**2. Better separation of concerns (Encapsulation & Code Reusability)**

* Using Runnable keeps thread behavior separate from the actual Thread class, leading to better code organization.
* It allows **different threads to execute the same Runnable instance**, which is not possible with Thread.

**Example (Using extends Thread)**:

class MyThread extends Thread {

public void run() {

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

}

}

public class Main {

public static void main(String[] args) {

MyThread t1 = new MyThread();

MyThread t2 = new MyThread();

t1.start();

t2.start();

}

}

Here, t1 and t2 are **two separate objects**, meaning they do not share any state.

**Example (Using implements Runnable)**:

class MyRunnable implements Runnable {

public void run() {

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

}

}

public class Main {

public static void main(String[] args) {

MyRunnable r = new MyRunnable();

Thread t1 = new Thread(r);

Thread t2 = new Thread(r);

t1.start();

t2.start();

}

}

Here, both t1 and t2 **share the same instance of MyRunnable**, meaning they can access shared resources efficiently.

**3. Thread Pooling (Recommended for large-scale applications)**

* Using Runnable, you can submit tasks to a thread pool (ExecutorService), which is **not possible with Thread**.
* Thread pools **reuse** threads, making them more efficient than creating new Thread objects each time.

**Example (Using ExecutorService)**:

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class MyRunnable implements Runnable {

public void run() {

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

}

}

public class Main {

public static void main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(5); // Thread pool of 5 threads

for (int i = 0; i < 10; i++) {

executor.execute(new MyRunnable());

}

executor.shutdown();

}

}

* This **improves performance** because it **reuses** threads instead of creating and destroying them repeatedly.

**4. More Flexible Object-Oriented Design**

* When extending Thread, each instance is a **separate thread**. This is **not reusable**.
* With Runnable, you can pass the same Runnable instance to **multiple Thread objects**, making it more reusable and manageable.

**5. Avoids Unnecessary Overhead**

* When you extend Thread, your class **inherits all properties and methods** from Thread, which may not always be needed.
* This increases memory overhead since it loads unnecessary members.
* With Runnable, your class only implements run(), keeping it **lightweight**.

**When to use extends Thread?**

Though implements Runnable is generally preferred, extends Thread can be useful when:

1. You **need to override** other Thread class methods such as start(), interrupt(), etc.
2. You are working on **very small, simple applications** where thread reusability and flexibility are not a concern.