How to Assign Different Runs to Different Threads in Java

Approaches Overview

Different threads can be assigned different tasks using:

1. **Separate Classes (using `Runnable`)**
2. **Anonymous Inner Classes**
3. **Lambda Expressions (Java 8+)**

Each approach is useful depending on the **size**, **complexity**, and **reusability** of the tasks.

Approach 1: Separate Classes (Using Runnable)

Concept

- Define **individual classes** that implement the `Runnable` interface.
- Each class contains its own **run()** method (its unique task).
- Create `Thread` objects and pass the Runnable instances to them.

Example

```
class TaskA implements Runnable {
    public void run() {
        System.out.println("Running Task A");
    }
}

class TaskB implements Runnable {
    public void run() {
```

```
        System.out.println("Running Task B");
    }
}

Thread t1 = new Thread(new TaskA());
Thread t2 = new Thread(new TaskB());

t1.start();
t2.start();
```

Approach 2: Anonymous Inner Classes

Concept

- No separate class files are required.
- The `run()` method is written **inside the Thread constructor**.
- Ideal for short, one-time tasks.

Example

```
Thread t1 = new Thread(new Runnable() {
    public void run() {
        System.out.println("Task 1 Running");
    }
});

Thread t2 = new Thread(new Runnable() {
    public void run() {
        System.out.println("Task 2 Running");
    }
});

t1.start();
t2.start();
```

Approach 3: Lambda Expressions (Java 8+)

Concept

- A concise and modern way to implement `Runnable` .
- Reduces boilerplate code.
- Best for simple tasks.

Example

```
Thread t1 = new Thread(() -> System.out.println("Lambda Task 1"));
Thread t2 = new Thread(() -> System.out.println("Lambda Task 2"));

t1.start();
t2.start();
```
