📘 Executor Framework in Java & Spring Boot

---

1. Executor Framework in Java

🕰️What is it?

The **Executor Framework** is a set of classes and interfaces in Java that simplify the execution of tasks in a multithreaded environment. Instead of manually creating and managing threads, you submit tasks to an executor, and it handles thread management for you.

🕰️Why use it?

- Avoids manual thread creation.
- Improves performance by reusing threads.
- Provides better control over concurrency.
- Built-in support for scheduling and retrieving results.

🕰️Core Interfaces

1. **Executor** – basic interface with `execute(Runnable)`.
2. **ExecutorService** – adds advanced features like `submit()`, `invokeAll()`, `shutdown()`.
3. **ScheduledExecutorService** – supports delayed and periodic execution.

🕰️Common Implementations

- `Executors.newFixedThreadPool(n)` → Pool of fixed-size threads.
- `Executors.newCachedThreadPool()` → Creates new threads as needed, reuses idle ones.
- `Executors.newSingleThreadExecutor()` → Executes tasks sequentially with one thread.
- `Executors.newScheduledThreadPool(n)` → Executes tasks periodically or with delay.

🕰️Example: Fixed Thread Pool

```
import java.util.concurrent.*;

public class ExecutorExample {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(3);
```

```
        for (int i = 1; i <= 5; i++) {
            final int taskId = i;
            executor.submit(() -> {
                System.out.println("Task " + taskId + " is running in " +
Thread.currentThread().getName());
            });
        }

        executor.shutdown();
    }
}
```

Output:

```
Task 1 is running in pool-1-thread-1
Task 2 is running in pool-1-thread-2
Task 3 is running in pool-1-thread-3
Task 4 is running in pool-1-thread-1
Task 5 is running in pool-1-thread-2
```

👉Only 3 threads are reused for 5 tasks.

🕰️Advantages

👈Performance boost via thread reuse.
👈Easy concurrency control.
👈Supports async execution & scheduling.
👈Simplifies thread lifecycle management.

---

2. Executor Framework in Spring Boot

Spring Boot integrates the Executor Framework using **TaskExecutor** and `@Async` support.

🕰️Step 1: Enable Async

```
@SpringBootApplication
@EnableAsync
public class ExecutorSpringApp {
    public static void main(String[] args) {
        SpringApplication.run(ExecutorSpringApp.class, args);
```

```
    }
}
```

🕰️Step 2: Configure Executor

```
@Configuration
public class AsyncConfig {

    @Bean(name = "taskExecutor")
    public Executor taskExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(5);
        executor.setMaxPoolSize(10);
        executor.setQueueCapacity(20);
        executor.setThreadNamePrefix("MyExecutor-");
        executor.initialize();
        return executor;
    }
}
```

🕰️Step 3: Use `@Async` in Service

```
@Service
public class MyService {

    @Async("taskExecutor")
    public void processTask(int taskId) {
        System.out.println("Task " + taskId + " started by " +
Thread.currentThread().getName());
        try { Thread.sleep(2000); } catch (InterruptedException e) {
e.printStackTrace(); }
        System.out.println("Task " + taskId + " finished by " +
Thread.currentThread().getName());
    }
}
```

🕰️Step 4: Trigger from Controller

```
@RestController
public class MyController {

    private final MyService myService;

    public MyController(MyService myService) {
        this.myService = myService;
    }

    @GetMapping("/start-tasks")
    public String startTasks() {
        for (int i = 1; i <= 10; i++) {
            myService.processTask(i);
        }
        return "Tasks submitted!";
    }
}
```

🕰️Example Output

```
Task 1 started by MyExecutor-1
Task 2 started by MyExecutor-2
Task 3 started by MyExecutor-3
Task 4 started by MyExecutor-4
Task 5 started by MyExecutor-5
Task 6 queued...
```

🔑Key Takeaways

- **Java Executor Framework** → Provides thread pool & async task execution.
- **Spring Boot Executor Integration** → Uses `@Async` + `TaskExecutor` for async processing.
- Helps APIs handle **huge incoming requests** without blocking.

👉Use **Java Executor** for general multithreading.
👉Use **Spring Boot Executor** when building APIs/services to simplify async execution.