Mono and Flux Tutorial

What is Project Reactor?

Project Reactor is a fully non-blocking reactive programming foundation for the JVM. It provides two main types:

- Mono<T>: Emits 0 or 1 item.
- Flux<T>: Emits 0 to N items (a stream of data).

1. What is Mono?

Mono is a container for a single value (or empty). It represents asynchronous computation that returns one result or completes empty.

Creating a Mono:

```
Mono<String> mono = Mono.just("Hello, Mono!");
```

Subscribe to Mono:

```
mono.subscribe(System.out::println); // Output: Hello, Mono!
```

Mono.empty() and Mono.error():

```
Mono<String> emptyMono = Mono.empty();
Mono<String> errorMono = Mono.error(new RuntimeException("Something went wrong"));
```

Mono Chaining (map, flatMap):

```
Mono<String> result = Mono.just("hello")
    .map(str -> str.toUpperCase()); // Output: HELLO
```

2. What is Flux?

Flux represents a stream that can emit multiple values over time.

Creating a Flux:

```
Flux<String> flux = Flux.just("One", "Two", "Three");
```

Mono and Flux Tutorial

Subscribe to Flux:

```
flux.subscribe(System.out::println);
// Output:
// One
// Two
// Three
```

Flux Operators (map, filter, delay):

```
Flux<Integer> numbers = Flux.range(1, 5)
    .filter(n -> n % 2 == 1)
    .map(n -> n * 10);
// Output: 10, 30, 50
```

Mono vs Flux:

Feature	Mono	Flux
Items emitted	0 or 1	0 to N
Use case	Single item	Multiple items
Examples	Auth, ID	Streams, Feeds

Combine Mono & Flux:

```
Flux<String> names = Flux.just("A", "B", "C");
Mono<List<String>> namesList = names.collectList();
```

Real-World Example (Spring WebFlux):

```
@GetMapping("/user/{id}")
public Mono<User> getUser(@PathVariable String id) {
    return userService.findById(id);
}
```

Mono and Flux Tutorial

```
@GetMapping("/users")
public Flux<User> getAllUsers() {
    return userService.findAll();
}
```