Short Note on Optional Methods in Java

In Java, the Optional class is a container object used to contain non-null objects. Optional helps in preventing NullPointerExceptions by offering methods like orElse(), orElseGet(), and map() to handle absent values. Below are some of the commonly used methods of the Optional class, explained with examples.

1. orElse(T other)

Purpose: Returns the value inside the Optional if present, otherwise returns the provided fallback value.

Key Point: The fallback value is always evaluated, even if not used.

Use Case: When the fallback value is simple and readily available.

Example:

```
Optional<String> optional = Optional.empty();
String result = optional.orElse("Default Value"); // Always evaluates "Default Value"
System.out.println(result); // Output: Default Value
```

2. orElseGet(Supplier<? extends T> supplier)

Purpose: Returns the value inside the Optional if present, otherwise calls the supplier to compute a fallback value.

Key Point: The supplier is lazily evaluated (executed only when needed).

Use Case: When the fallback value is expensive to compute or dynamic.

Example:

```
Optional<String> optional = Optional.empty();
String result = optional.orElseGet(() -> "Lazy Default Value");
System.out.println(result); // Output: Lazy Default Value
```

3. orElseThrow()

Purpose: Returns the value inside the Optional if present, otherwise throws an exception.

Key Point: Used for stricter null handling.

Variants:

- Default Exception: Throws NoSuchElementException by default.

- Custom Exception: Accepts a Supplier to throw a custom exception.

Example:

```
Optional<String> optional = Optional.empty();
String result = optional.orElseThrow(() -> new RuntimeException("Value not found")); //
Throws RuntimeException
```

4. map(Function<? super T, ? extends U> mapper)

Purpose: Transforms the value inside the Optional using a mapping function, if present.

Key Point: Returns a new Optional containing the transformed value.

Example:

```
Optional<String> optional = Optional.of("Hello");
Optional<Integer> length = optional.map(String::length);
System.out.println(length.get()); // Output: 5
```

5. filter(Predicate<? super T> predicate)

Purpose: Filters the value inside the Optional based on a condition.

Key Point: If the condition is not met, an empty Optional is returned.

Example:

```
Optional<String> optional = Optional.of("Hello");
Optional<String> result = optional.filter(s -> s.length() > 3);
System.out.println(result.isPresent()); // Output: true
```

Comparison Summary

Method	When to Use	Key Feature
orElse()	When fallback is simple and always available	Always evaluates fallback value
orElseGet()	When fallback is expensive or dynamic	Lazily evaluates fallback
orElseThrow()	When an exception should be thrown if empty	Throws exception if no value
map()	To transform the value inside Optional	Returns transformed Optional
filter()	To conditionally keep the value	Returns Optional based on condition