
What is Optional?

Optional is a **container object** introduced in **Java 8** that **may or may not contain a non-null value**.
It is designed to **represent optional (i.e., possibly absent) values** instead of using null references.

Purpose

- **Avoid NullPointerException (NPE):** Instead of returning null to indicate "no result," methods can return an Optional object.
- **Make absence of a value explicit:** Clients of your API can see that the value might be missing.
- **Promote functional programming:** Optional supports methods like map, filter, and flatMap to process the contained value in a fluent style.

Key Characteristics

- It is a **final class** in java.util.
- It acts as a **wrapper** for a value that **might be present or absent**.
- It is **not intended as a replacement for every null**, but mainly for **return types** where absence is expected and meaningful.

Important Methods

Some of the commonly used methods in Optional:

- **empty()**
  Creates an empty Optional.
- **of(value)**
  Creates an Optional with a **non-null** value. Throws an exception if the value is null.
- **ofNullable(value)**
  Creates an Optional that can hold a **nullable** value.
- **isPresent()**
  Returns true if a value is present, otherwise false.

- **ifPresent(Consumer)**
  Performs an action if a value is present.
- **get()**
  Returns the value if present, otherwise throws NoSuchElementException.
- **orElse(defaultValue)**
  Returns the value if present, otherwise returns a default value.
- **orElseGet(Supplier)**
  Returns the value if present, otherwise calls a supplier to get a value.
- **orElseThrow()**
  Throws NoSuchElementException if the value is absent.
- **map(Function)**
  Transforms the value if present.
- **flatMap(Function)**
  Similar to map but avoids nested Optional.

When to Use Optional

- As **method return types** when the result may be absent.
- To make the **API more expressive and intention-revealing**.
- To encourage **explicit handling of missing values**.

When NOT to Use Optional

- As **fields** in entities or data structures (because it adds unnecessary overhead).
- For **method parameters** (use method overloading or null checks instead).

Summary

Optional makes your code:

- **Safer** by avoiding null references,
- **Clearer** about possible absence of values,
- **More functional** in style.