In **Java 8**, the Optional<T> class was introduced in java.util to **avoid NullPointerException** and **clearly express optional values** (values that may or may not be present).

**🔍 Why Optional<T>?**

Traditionally, Java APIs return null to indicate absence of a value, which leads to:

if (user != null && user.getAddress() != null) {

System.out.println(user.getAddress().getCity());

}

This is **error-prone and messy**. Java 8’s Optional handles this more elegantly.

**✅ Definition**

public final class Optional<T>

* It’s a **container object** that may or may not contain a non-null value.
* Helps to write **null-safe, expressive, and readable code**.

**🛠️ Creating Optional**

| **Method** | **Description** |
| --- | --- |
| Optional.of(value) | Wraps a **non-null** value |
| Optional.ofNullable(value) | Wraps a value that may be null |
| Optional.empty() | Returns an empty Optional |

**Example:**

Optional<String> name = Optional.of("Avinash");

Optional<String> maybeNull = Optional.ofNullable(null);

Optional<String> empty = Optional.empty();

**🔧 Common Methods**

| **Method** | **Purpose** |
| --- | --- |
| isPresent() | Returns true if value is present |
| get() | Returns value (⚠️ throws if empty) |
| ifPresent(Consumer) | Runs block if value exists |
| orElse(T other) | Returns value or fallback if empty |
| orElseGet(Supplier) | Lazy version of orElse |
| orElseThrow() | Throws NoSuchElementException if empty |
| map(Function) | Transforms value |
| flatMap(Function) | Transforms nested Optionals |
| filter(Predicate) | Keeps value if condition matches |

**💡 Usage Examples**

**1. Basic Usage**

Optional<String> name = Optional.of("John");

System.out.println(name.get()); // John

**2. Avoid null checks**

Optional<String> name = Optional.ofNullable(null);

System.out.println(name.orElse("Default")); // Default

**3. ifPresent()**

name.ifPresent(n -> System.out.println("Hello " + n));

**4. map() and flatMap()**

Optional<String> name = Optional.of("Avinash");

int length = name.map(String::length).orElse(0);

System.out.println(length); // 7

**🔄 Chaining Optionals (Safe navigation)**

public class User {

private Optional<Address> address;

public Optional<Address> getAddress() { return address; }

}

public class Address {

private Optional<String> city;

public Optional<String> getCity() { return city; }

}

Optional<User> user = Optional.ofNullable(getUser());

String city = user

.flatMap(User::getAddress)

.flatMap(Address::getCity)

.orElse("Unknown");

**🔥 orElse() vs orElseGet()**

String value = "Avinash";

System.out.println(

Optional.ofNullable(value).orElse(getDefault()) // getDefault() is always called

);

System.out.println(

Optional.ofNullable(value).orElseGet(() -> getDefault()) // only called if value is null

);

**🧪 Using Optional in Repositories (Spring Style)**

Optional<User> user = userRepository.findById(10);

user.ifPresent(u -> System.out.println(u.getName()));

**❌ When *not* to use Optional**

* As class fields (use sparingly)
* In method parameters (prefer null or overloading)
* In high-performance scenarios (adds slight overhead)

**✅ Best Practices**

| **Do** | **Avoid** |
| --- | --- |
| Use Optional as return type | Using Optional in fields/args |
| Use orElseGet lazily | Calling get() without isPresent() |
| Use chaining (map, filter) | Overusing deep if nesting |

**📌 Summary**

* Optional<T> is a container to avoid null.
* Helps write **cleaner, safer, and more readable** code.
* Use map(), flatMap(), orElse(), ifPresent() effectively.
* It’s **not a replacement for null everywhere** — use wisely.