|  |
| --- |
|  |
|  |

|  |
| --- |
| **Optional** |
| Optional<T> is a container introduced in Java 8 to handle nullable values safely and avoid NullPointerException (NPE).   * Prevents NullPointerException by making nullability explicit. * Reduces excessive null checks. * Encourages functional programming (map, flatMap, etc.).  ****Creating Optional Instances**** Optional<String> opt1 = Optional.of("Hello"); // Throws NPE if null  Optional<String> opt2 = Optional.ofNullable(null); // Can hold null safely  Optional<String> opt3 = Optional.empty(); // Creates an empty Optional ****Retrieving Values**** String value1 = opt1.get(); // Use cautiously, throws if empty  String value2 = opt2.orElse("Default"); // Returns "Default" if empty  String value3 = opt2.orElseGet(() -> "Generated Default");  String value4 = opt2.orElseThrow(() -> new RuntimeException("Value not present")); ****Checking Presence of Value**** if(opt1.isPresent()) {  System.out.println(opt1.get());  }  opt1.ifPresent(System.out::println); // Prints "Hello" if present ****Transforming Values**** Optional<Integer> length = opt1.map(String::length); // Transforming value  Optional<String> result = Optional.of(Optional.of("Nested")).flatMap(opt -> opt); // Avoids nested Optional ****Best Practices**** ✅ Use Optional as a return type for methods that may return null. ❌ Avoid using Optional for **method parameters** and **class fields**. ****Real-World Example: Avoiding NullPointerException******Without Optional** public String getEmployeeName(Employee emp) {  if (emp != null) {  Address addr = emp.getAddress();  if (addr != null) {  return addr.getCity();  }  }  return "Unknown";  } **With Optional** public String getEmployeeName(Employee emp) {  return Optional.ofNullable(emp)  .map(Employee::getAddress)  .map(Address::getCity)  .orElse("Unknown");  } ****9. Output Scenarios****  | **Employee Object State** | **Output** | | --- | --- | | emp == null | "Unknown" | | emp.getAddress() == null | "Unknown" | | emp.getAddress().getCity() == null | "Unknown" | | emp.getAddress().getCity() == "New York" | "New York" |  ****10. Key Takeaways****  * Optional prevents NullPointerException and improves code readability. * Use orElse(), orElseGet(), and orElseThrow() for handling absent values. * Use map() and flatMap() for transformations. * Avoid using Optional in class fields and method parameters. |

|  |
| --- |
| **Function.identity()** |
| Function.identity() is a static method in the Function interface in Java. It returns a function that always returns its input argument. This can be useful when you need to pass a function as an argument but want the function to simply return the value it receives.  **Syntax:**  Function<T, T> identity()  **Usage:**   * It is commonly used in streams, map, groupingBy, or when dealing with transformations that don’t require any changes to the input. * It returns a function that takes an input and returns the same input without modification.   **Example:**  import java.util.function.Function;  public class IdentityFunctionExample {  public static void main(String[] args) {  Function<String, String> identityFunction = Function.identity();  String result = identityFunction.apply("Hello, World!");  System.out.println(result); // Output: Hello, World!  }  }  In this example, Function.identity() is used to create a function that simply returns the input string unchanged. |

|  |
| --- |
| **Static Imports** |
| A static import in Java allows you to import a class's static members (fields and methods) directly, so you can use them without prefixing them with the class name. Introduced in Java 5  **Syntax:**  import static package\_name.ClassName.staticMember;  **Example:**  **Without Static import** public class MathExample {  public static void main(String[] args) {  double result = Math.sqrt(25); // Using Math class explicitly  System.out.println("Square Root: " + result);  }  } **Without Static Import**import static java.lang.Math.sqrt; // Static importpublic class MathExample {public static void main(String[] args) {double result = sqrt(25); // No need for Math.sqrt()System.out.println("Square Root: " + result);}} **Example:Junit**  // Static import for assertions  import static org.junit.jupiter.api.Assertions.\*; import org.junit.jupiter.api.Test;class TestExample {@Testvoid testAddition() {int sum = 5 + 3;assertEquals(8, sum); // No need for Assertions.assertEquals()}} |

|  |
| --- |
|  |
|  |