Wrapper Classes in Java (In-Depth)

1. Introduction

Java provides two types of data handling mechanisms:

- **Primitive data types** (int, double, char, etc.)
- **Objects / Classes**

Java is a *purely object-oriented language at usage level*, but primitives exist for **performance reasons**. However, many Java features work **only with objects**. To bridge this gap, Java provides **Wrapper Classes**.

---

2. Definition of Wrapper Classes

Wrapper classes are predefined classes in the `java.lang` package that wrap primitive data types into objects.

They convert:

```
primitive  →  object
```

This allows primitives to be treated like objects whenever required.

---

3. Primitive Types and Their Wrapper Classes

Primitive Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Important Notes:

- Wrapper classes start with **capital letters**
- All wrapper classes are **final** and **immutable**
- They belong to the `java.lang` package (no import required)

---

4. Why Wrapper Classes Are Needed

4.1 Limitations of Primitive Data Types

Primitives:

- Are **not objects**
- Cannot be stored in **Collections**
- Cannot be used with **Generics**
- Do not support **methods**

Example (Invalid):

```
ArrayList<int> list;  // Compilation Error
```

4.2 Advantages of Wrapper Classes

Wrapper classes:

- Are objects
- Can be stored in Collections
- Work with Generics
- Provide utility methods
- Support null values

Example (Valid):

```
ArrayList<Integer> list = new ArrayList<>();
```

---

5. Creating Wrapper Class Objects

5.1 Using Constructors (Deprecated)

```
Integer a = new Integer(10);
Integer b = new Integer("100");
```

 Deprecated from Java 9 due to inefficiency

5.2 Using `valueOf()` Method (Recommended)

```
Integer a = Integer.valueOf(10);
Integer b = Integer.valueOf("100");
```

✔ Efficient
✔ Uses internal caching

5.3 Autoboxing (Most Common)

```
Integer a = 10;   // primitive int → Integer object
```

The compiler automatically converts the primitive to its wrapper object.

---

6. Autoboxing and Unboxing

6.1 Autoboxing

Converting primitive → wrapper

```
int x = 5;
Integer y = x;   // Autoboxing
```

6.2 Unboxing

Converting wrapper → primitive

```
Integer a = 20;
int b = a;       // Unboxing
```

Internally, the compiler calls:

```
int b = a.intValue();
```

---

7. Wrapper Class Methods

7.1 Parsing Methods (String → Primitive)

```
int i = Integer.parseInt("123");
double d = Double.parseDouble("12.5");
boolean b = Boolean.parseBoolean("true");
```

 If the string is invalid → `NumberFormatException`

7.2 `valueOf()` VS `parseXxx()`

Method	Returns
parseInt()	primitive type
valueOf()	wrapper object

Example:

```
int x = Integer.parseInt("10");
Integer y = Integer.valueOf("10");
```

7.3 `xxxValue()` Methods (Wrapper → Primitive)

```
Integer i = 100;
int a = i.intValue();
double d = i.doubleValue();
```

---

8. Immutability of Wrapper Classes

Wrapper classes are **immutable** — once created, their value cannot be changed.

```
Integer a = 10;
a = a + 5;
```

Explanation:

- A **new Integer object** is created
- Old object becomes unused

Similar to `String` immutability.

---

9. Wrapper Classes and Collections

Collections store **only objects**, not primitives.

```
ArrayList<Integer> list = new ArrayList<>();

list.add(10);   // Autoboxing
list.add(20);

int x = list.get(0); // Unboxing
```

10. Wrapper Classes and Generics

Generics do not support primitives.

```
class Test<T> { }

Test<int> t1;        // Invalid
Test<Integer> t2;   // Valid
```

---

11. Integer Caching (Important Interview Topic)

Java caches wrapper objects for performance.

Integer Cache Range:

```
-128 to 127
```

Example:

```
Integer a = 100;
Integer b = 100;
System.out.println(a == b);  // true
```

Outside cache range:

```
Integer x = 200;
Integer y = 200;
System.out.println(x == y);  // false
```

✔ Always use `.equals()` for value comparison

---

12. NullPointerException Risk

```
Integer a = null;
int b = a;  // NullPointerException
```

Reason:

- Unboxing tries to extract value from `null`

13. Wrapper Classes vs Primitive Types

Feature	Primitive	Wrapper
Object	□	□
Memory	Low	Higher
Methods	□	□
Collections	□	□
Generics	□	□
Null Allowed	□	□

---

16. Exam & Interview Key Points

- Wrapper classes convert primitives into objects
- Immutable and final
- Used in Collections and Generics
- Autoboxing & Unboxing handled by compiler
- Integer caching improves performance
- Prefer `.equals()` over `==`