➢ 31-Wrapper Classes

➢ Introduction

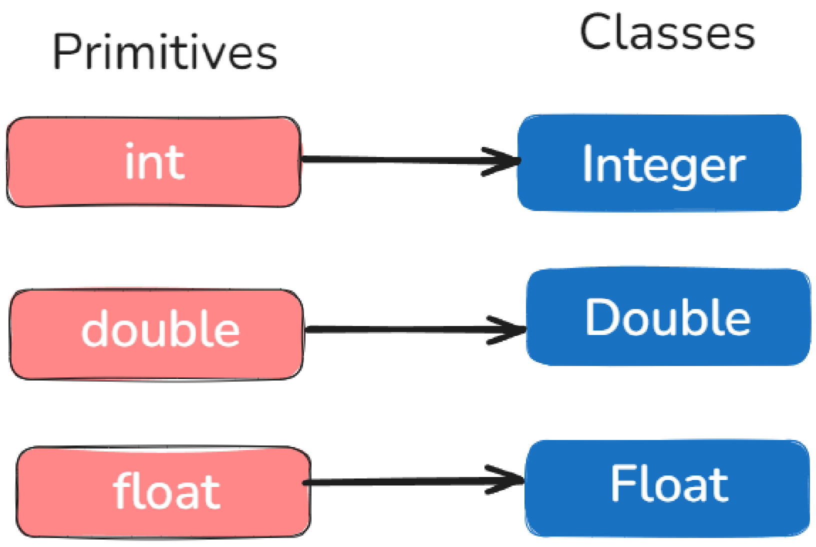
When discussing Java and Object-Oriented Programming (OOP), it's important to note that Java is not a purely object-oriented language. This is because it uses primitive data types such as int for integers, double or float for decimal values, and char for characters. These primitive data types are not objects, which contrasts with the OOP principle that everything should be an object.

➢ Why Java is Not Purely Object-Oriented

In a purely object-oriented language, all entities, including data types, must be represented as objects. However, in Java, primitive data types do not follow this principle, which is why Java is considered a **partially object-oriented language**.

➢ Wrapper Classes in Java

To bridge the gap between primitive types and the object-oriented nature of Java, the language provides **Wrapper Classes**. A Wrapper class in Java is a class whose object wraps or contains a primitive data type. When we create an object of a wrapper class, it encapsulates the primitive data type inside it. This allows the primitive data to be used in contexts that require objects.



➢ Primitive Data Types and Their Corresponding Wrapper Classes

For every primitive data type in Java, there is a corresponding wrapper class:

Primitive Data Type	Wrapper Class
char	Character
byte	Byte

Primitive Data Type	Wrapper Class
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

- **Examples and Scenarios**

Scenario 1: Primitive to Wrapper Type

In this scenario, we convert a primitive type to its corresponding wrapper type.

```java
public class Demo {
    public static void main(String[] args) {
        int num = 7;
        Integer num1 = new Integer(8);  // Deprecated
        System.out.println(num1);  // Output: 8
    }
}
```

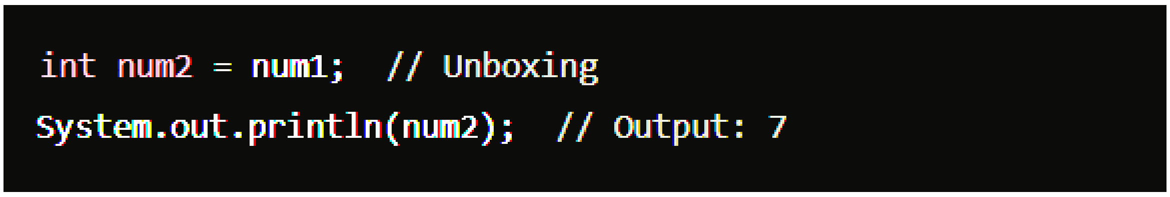
Note: *The use of new Integer(8) is deprecated in recent versions of Java. Instead, you should rely on autoboxing, where the conversion from a primitive type to a wrapper class is done automatically.*

- **Scenario 2: Autoboxing and Unboxing**

Autoboxing: The automatic conversion of a primitive type to the object of its corresponding wrapper class.

```java
int num = 7;
Integer num1 = num;  // Autoboxing
```

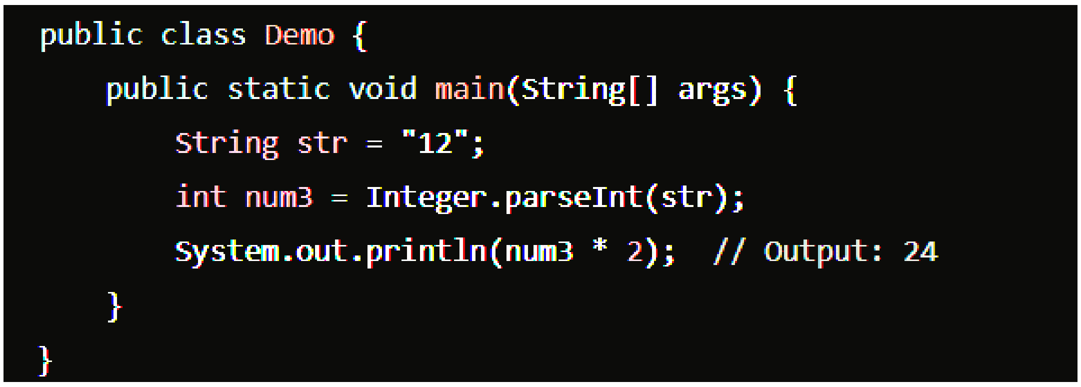
Unboxing: The reverse process, where the object of a wrapper class is automatically converted back to its corresponding primitive type.



```
int num2 = num1;  // Unboxing
System.out.println(num2);  // Output: 7
```

➢ **Scenario 3: String to int Conversion**
This scenario demonstrates how to convert a String to an int using a wrapper class method.

➢ **Advantages of Wrapper Classes**



```
public class Demo {
    public static void main(String[] args) {
        String str = "12";
        int num3 = Integer.parseInt(str);
        System.out.println(num3 * 2);  // Output: 24
    }
}
```

Collections Framework: Collections in Java can only hold objects, not primitive types. Wrapper classes enable the use of primitive data in collections like ArrayList, HashSet, etc.

Method Operations: Wrapper classes provide useful methods like compareTo(), equals(), and toString().

Cloning: Only objects can be cloned in Java, not primitives.

Null Values: Wrapper objects can be null, while primitives cannot.

Serialization: Serialization, the process of converting an object into a byte stream, only works with objects.

➢ **FAQs on Wrapper Classes**

Which are the wrapper classes in Java?

Wrapper classes in Java are classes whose objects encapsulate primitive data types.

Why use the wrapper class in Java?

Wrapper classes are used to convert primitive data types into objects, which is necessary when you want to modify arguments passed into a method or store primitives in collections.

What are the 8 wrapper classes in Java?

- The 8 wrapper classes in Java are Boolean, Byte, Short, Character, Integer, Long, Float, and Double.