Java Wrapper Class

In Java, we have 8 primitive data types. Java provides **type wrappers**, which are classes that encapsulate a primitive type within an Object.

- A wrapper class wraps (encloses) around a primitive datatype and gives it an object appearance. Wherever the primitive datatype is required as an object type, this type wrapper can be used.
- Wrapper classes include methods to unwrap the object and give back the data type.
- The *type wrappers* classes are part of *java.lang* package.
- Each primitive type has a corresponding wrapper class.

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

When to use Wrapper Classes?

Java wrapper classes are used in scenarios –

- When two methods wants to refer to the same instance of an primitive type, then pass wrapper class as **method arguments**.
- Java **Generics works only with object types** and does not support primitive types.
- Java **Collections deal only with objects**; to store a primitive type in one of these classes, you need to wrap the primitive type in a class.
- When you want to refer null from data type, the you need object. **Primitives cannot have null** as value.

Converting Primitive Types to Wrapper Classes

There are two ways for converting a primitive type into an instance of the corresponding wrapper class –

1. Using **constructors (Deprecated since Java 9)**

2. Using **static factory methods**

```
// 1. using constructor
Integer object = new Integer(10);


// 2. using static factory method
Integer anotherObject = Integer.valueOf(10);
```

In the above example, the valueOf() method is a static factory method that returns an instance of Integer class representing the specified int value.

Similarly, we can convert the other primitive types like boolean to Boolean, char to Character, short to Short, etc.

Java wrapper classes use internal caching which returns internally cached values upto a limit. This internal caching of instances makes the wrapper classes more efficient in performance and memory utilization.

Converting Wrapper Class to Primitive Type

Converting from wrapper class to primitive type is simple. Each wrapper class (Integer, Double, etc.) provides a set of xxxValue() methods to obtain the corresponding primitive value. e.g. intValue(), doubleValue(), shortValue() etc.

```
Integer integerObject = new Integer(10);
int intValue = integerObject.intValue();
```

Generally, using `intValue()` is the preferred method for converting a Wrapper type (like `Integer`) to its corresponding primitive type (like `int`).

Here's why:

- **Clarity:** It explicitly expresses the intent to convert the wrapper to a primitive.
- **Consistency:** It aligns with other wrapper classes' methods for getting primitive values (e.g., `doubleValue`, `longValue`).
- **Avoids potential pitfalls:** Unboxing (implicit conversion) can sometimes lead to unexpected behavior, especially when dealing with null values. Using `intValue()` makes the code more explicit and less prone to errors.

Unboxing

Java automatically converts wrapper objects to primitive types in certain contexts, a process known as unboxing. This happens implicitly in arithmetic operations, method parameters, and conditional statements.

```
Integer integerObject = new Integer(10);

int intValue = integerObject + 5; // Unboxing occurs here
```

Typecasting (Not Preferred)

While possible, explicit type casting is generally not recommended as it might lead to unexpected behavior if the wrapper object is null.

```
Integer integerObject = new Integer(10);

int intValue = (int) integerObject; // Type casting
```

Note:

- Using xxxValue() methods is the preferred way to convert wrapper objects to primitive types.
- Unboxing is often the most convenient approach, but be aware of potential NullPointerException if the wrapper object is null.
- Explicit type casting should be used with caution.

Additional Considerations:

- For boolean values, use booleanValue() method.
- For character values, use charValue() method.
- Be mindful of potential data loss when converting between numeric types (e.g., from double to int).

By understanding these methods, you can effectively convert wrapper objects to primitive types in your Java code.