**Java Wrapper Class**

The wrapper classes in Java are used to convert primitive types (int, char, float, etc) into corresponding objects.

Each of the 8 primitive types has corresponding wrapper classes.

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

**Convert Primitive Type to Wrapper Objects**

We can also use the valueOf() method to convert primitive types into corresponding objects.

**Example 1: Primitive Types to Wrapper Objects**

class Main {

public static void main(String[] args) {

// create primitive types

int a = 5;

double b = 5.65;

//converts into wrapper objects

Integer aObj = Integer.valueOf(a);

Double bObj = Double.valueOf(b);

if(aObj instanceof Integer) {

System.out.println("An object of Integer is created.");

}

if(bObj instanceof Double) {

System.out.println("An object of Double is created.");

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

An object of Integer is created.

An object of Double is created.

In the above example, we have used the valueOf() method to convert the primitive types into objects.

Here, we have used the instanceof operator to check whether the generated objects are of Integer or Double type or not.

However, the Java compiler can directly convert the primitive types into corresponding objects. For example,

int a = 5;

// converts into object

Integer aObj = a;

double b = 5.6;

// converts into object

Double bObj = b;

This process is known as **auto-boxing**. To learn more, visit [Java autoboxing and unboxing](https://www.programiz.com/java-programming/autoboxing-unboxing).

**Note**: We can also convert primitive types into wrapper objects using Wrapper class constructors. But the use of constructors is discarded after Java 9.

**Wrapper Objects into Primitive Types**

To convert objects into the primitive types, we can use the corresponding value methods (intValue(), doubleValue(), etc) present in each wrapper class.

**Example 2: Wrapper Objects into Primitive Types**

class Main {

public static void main(String[] args) {

// creates objects of wrapper class

Integer aObj = Integer.valueOf(23);

Double bObj = Double.valueOf(5.55);

// converts into primitive types

int a = aObj.intValue();

double b = bObj.doubleValue();

System.out.println("The value of a: " + a);

System.out.println("The value of b: " + b);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

The value of a: 23

The value of b: 5.55

In the above example, we have used the intValue() and doubleValue() method to convert the Integer and Double objects into corresponding primitive types.

However, the Java compiler can automatically convert objects into corresponding primitive types. For example,

Integer aObj = Integer.valueOf(2);

// converts into int type

int a = aObj;

Double bObj = Double.valueOf(5.55);

// converts into double type

double b = bObj;

This process is known as **unboxing**. To learn more, visit [Java autoboxing and unboxing](https://www.programiz.com/java-programming/autoboxing-unboxing).

**Advantages of Wrapper Classes**

* In Java, sometimes we might need to use objects instead of primitive data types. For example, while working with collections.
* // error
* ArrayList<int> list = new ArrayList<>();
* // runs perfectly

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

In such cases, wrapper classes help us to use primitive data types as objects.

* We can store the null value in wrapper objects. For example,
* // generates an error
* int a = null;
* // runs perfectly
* Integer a = null;

**Note**: Primitive types are more efficient than corresponding objects. Hence, when efficiency is the requirement, it is always recommended primitive types.