HowToDoInJava

Java Wrapper Classes, Autoboxing and Unboxing

Last Updated: January 25, 2022 By: Shailender Java Basics Data Types, Wrapper Classes

Learn about **Java wrapper classes**, their usage, conversion between primitives and objects; and autoboxing and unboxing with examples.

1. Java Wrapper Classes

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

Wrapper Class	Boolean

2. 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.

3. Conversions

3.1. 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 **constrcutors**
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 perfomance and memory unilization.

3.2. Converting Wrapper Class to Primitive Type

Converting from wrapper class to primitive type is simple. We can use the corresponding instance methods to get the primitive type. e.g. **intValue()**, **doubleValue()**, **shortValue()** etc.

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

int val = object.intValue();    //wrapper to primitive
```

4. Autoboxing and Unboxing

Beginning with JDK 5, Java added two important features:

- Autoboxing
- Auto-Unboxing

4.1. Autoboxing

Autoboxing is the **automatic conversion of the primitive types into their corresponding wrapper class**.

For example, converting an `int` to an `Integer`, a `char` to a `Character`, and so on.

We can simply pass or assign a primitive type to an argument or reference accepting wrapper class type.

Java Autoboxing Example

```
List<Integer> integerList = new ArrayList<>();

for (int i = 1; i < 10; i ++)
{
    integerList.add(i);     //int to Integer
}
```

In given example, `integerList` is a `List` of `Integer`s. It is not a list of primitive type int values.

Here compiler automatically creates an `Integer` object from `int` and adds the object to `integerList`. Thus, the previous code turns into the following at runtime:

```
List<Integer> integerList = new ArrayList<>();

for (int i = 1; i < 10; i ++)
{
      integerList.add(Integer.valueOf(i));      //autoboxing
}
```

4.2. Unboxing

Unboxing happens **when the conversion happens from wrapper class to its corresponding primitive type**. It means we can pass or assign a wrapper object to an argument or reference accepting primitive type.

Java Unboxing Example

```
public static int sumOfEven(List<Integer> integerList)
{
    int sum = 0;
    for (Integer i: integerList) {
        if (i % 2 == 0)
            sum += i;           //Integer to int
    }
    return sum;
}
```

In the above example, the remainder `(%) and unary plus (+=) operators do` not apply on Integer objects. The compiler automatically converts an Integer to an int at runtime by invoking the `intValue()` method.

Autoboxing and unboxing lets developers write **cleaner code**, make it easier to read.

Happy Learning !!

Recommended Reading:

1. Java – Internal Caching in Wrapper Classes
2. Java Classes and Objects
3. Generate XSD from JAXB Java Classes using Eclipse
4. Java abstract keyword – abstract classes and methods
5. Sealed Classes and Interfaces
6. Java Primitive Data Types
7. Java Data Types
8. Difference between 32-bit Java vs. 64-bit Java
9. Java main() Method
0. What is Block Statement in Java

Leave a Comment

Name *

Email *

Website

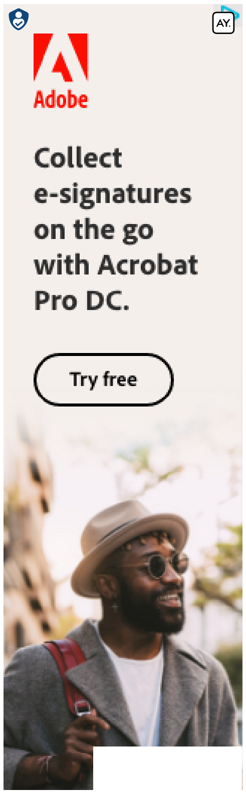
☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

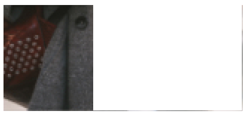
Post Comment

Search ...

Promoted by amazon.jobs
Sponsored

A message from our sponsor

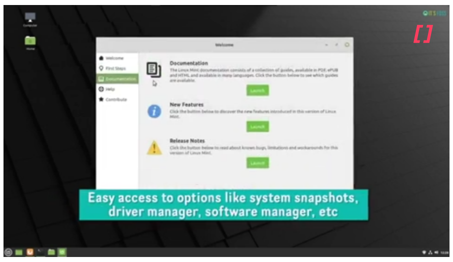




Promoted by Adobe
Sponsored

Use Lightroom to sharpen

Read more


Easy access to options like system snapshots, driver manager, software manager, etc

Promoted by amazon.jobs
Sponsored

A message from our sponsor