Autoboxing and Unboxing

Autoboxing and Unboxing

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an `int` to an `Integer`, a `double` to a `Double`, and so on. If the conversion goes the other way, this is called unboxing.

Here is the simplest example of autoboxing:

```
Character ch = 'a';
```

The rest of the examples in this section use generics. If you are not yet familiar with the syntax of generics, see the Generics section.

Consider the following code:

```
List<Integer> ints = new ArrayList<>();
for (int i = 1; i < 50; i += 2)
    ints.add(i);
```

Although you add the `int` values as primitive types, rather than `Integer` objects, to `ints`, the code compiles. Because `ints` is a list of `Integer` objects, not a list of `int` values, you may wonder why the Java compiler does not issue a compile-time error. The compiler does not generate an error because it creates an `Integer` object from `i` and adds the object to `ints`. Thus, the compiler converts the previous code to the following at runtime:

```
List<Integer> ints = new ArrayList<>();
for (int i = 1; i < 50; i += 2)
```

```
        ints.add(Integer.valueOf(i));
```

Converting a primitive value (an `int`, for example) into an object of the corresponding wrapper class `Integer` is called autoboxing. The Java compiler applies autoboxing when a primitive value is:

- Passed as a parameter to a method that expects an object of the corresponding wrapper class.
- Assigned to a variable of the corresponding wrapper class.

Consider the following method:

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

Because the remainder (`%`) and unary plus (`+=`) operators do not apply to `Integer` objects, you may wonder why the Java compiler compiles the method without issuing any errors. The compiler does not generate an error because it invokes the `intValue()` method to convert an `Integer` to an `int` at runtime:

```
public static int sumEven(List<Integer> ints){
    int sum=0;
    for(Integer i:ints) {
        if(i.intValue()%2==0) {
            sum+=i.intValue();
        }
    }
    return sum;
}
```

Converting an object of a wrapper type `Integer` to its corresponding primitive (`int`) value is called unboxing. The Java compiler applies unboxing when an object of a wrapper class is:

- Passed as a parameter to a method that expects a value of the corresponding primitive type.

- Assigned to a variable of the corresponding primitive type.

The `Unboxing` example shows how this works:

```
import java.util.ArrayList;
import java.util.List;

public class Unboxing {

    public static void main(String[] args) {
        Integer i = Integer.valueOf(-8);

        // 1. Unboxing through method invocation
        int absVal = absoluteValue(i);
        System.out.println("absolute value of " + i + " = " + absVal);

        List<Double> doubles = new ArrayList<>();
        doubles.add(3.1416);    // Π is autoboxed through method invocation.

        // 2. Unboxing through assignment
        double pi = doubles.get(0);
        System.out.println("pi = " + pi);
    }

    public static int absoluteValue(int i) {
        return (i < 0) ? -i : i;
    }
}
```

The program prints the following:

```
absolute value of -8 = 8
pi = 3.1416
```

Autoboxing and unboxing lets developers write cleaner code, making it easier to read. The following table lists the primitive types and their corresponding wrapper classes, which are used by the Java compiler for autoboxing and unboxing:

Primitive type	Wrapper class
boolean	Boolean
byte	Byte
char	Character

Primitive type	Wrapper class
float	Float
int	Integer
long	Long
short	Short
double	Double

Last update: *September 14, 2021*

String Builders ← **Autoboxing and Unboxing** → That's the end of the series!

Home > Tutorials > Numbers and Strings > Autoboxing and Unboxing