The Integer and Double Classes

Learn how the Integer and Double classes are different than primitive data types.

In this article, we're going to take a look at the `Integer` and `Double` classes. But wait, aren't `int`s and `double`s primitive data types? Yes, they are, however, but Java contains wrapper classes named `Integer` and `Double` which store instances of the primitive data inside of them.

The main purpose of these wrapper classes is to convert between primitive data and objects. This is needed because it allows us to use the wrapper class methods to manipulate the data, such as converting from one data type to another, or converting primitive data into object form to be used by certain data structures or methods in Java.

Let's take a look at an example using the ArrayList class.

If you haven't seen the ArrayList class before, you can think of it as a place to store a list of objects. ArrayLists (as well as other Java data structures) only accept *objects* as input, not primitive data. This is why we need the `Integer` and `Double` classes — we can't use `int`s and `double`s with ArrayLists. There are many methods in this class used to add, remove, insert, and manipulate the contained data in different ways. This is a very useful class that you can learn much more about it later in the AP Computer Science path.

In this example, we are storing integer data into an ArrayList. An ArrayList can't store `int` data — they *have* to hold objects. To get around that, we can create an ArrayList of `Integer` objects and let Java convert the primitive data to an `Integer` object automatically.

```
ArrayList<Integer> myArrayList = new ArrayList<>();
myArrayList.add(5);
int num = myArrayList.get(0); // Asking for the first (technically "zero-th") number in
myArrayList. num should now hold 5.
```

This example shows how the ArrayList uses the `Integer` wrapper class in order to convert any `int` data it receives into `Integer` objects. When `5` is added to `myArrayList`, the primitive data is automatically converted to an `Integer` object before being stored into the ArrayList. This is called **autoboxing**. Similarly, when the element is accessed, the `Integer` stored in an the ArrayList is automatically converted from an object of the `Integer` wrapper class to primitive data again. This lets us store the primitive `int` value in the `int` variable named `num`. This is called **unboxing** and can be seen in the last line of the code block above.

This type of conversion to and from wrapper classes applies to all wrapper class types: `Integer`, `Double`, `Float`, `Long`, `Short`, `Boolean`, `Byte`, and `Character`, although this article will focus on the `Integer` and `Double` wrapper classes.

This conversion also takes place when initializing an instance of a wrapper class object. For example, if we were to create a new `Double` object, then the primitive data passed to the constructor will be autoboxed. We can also assign the primitive value directly or use the static `valueOf()` method. Here are a few examples:

```
Double wrapper1 = 23.456;
Integer wrapper2 = 3;
Double wrapper3 = new Double(13.57);
Integer wrapper4 = new Integer(7);
Double wrapper5 = Double.valueOf(30.59);
Integer wrapper6 = Integer.valueOf(15);
```

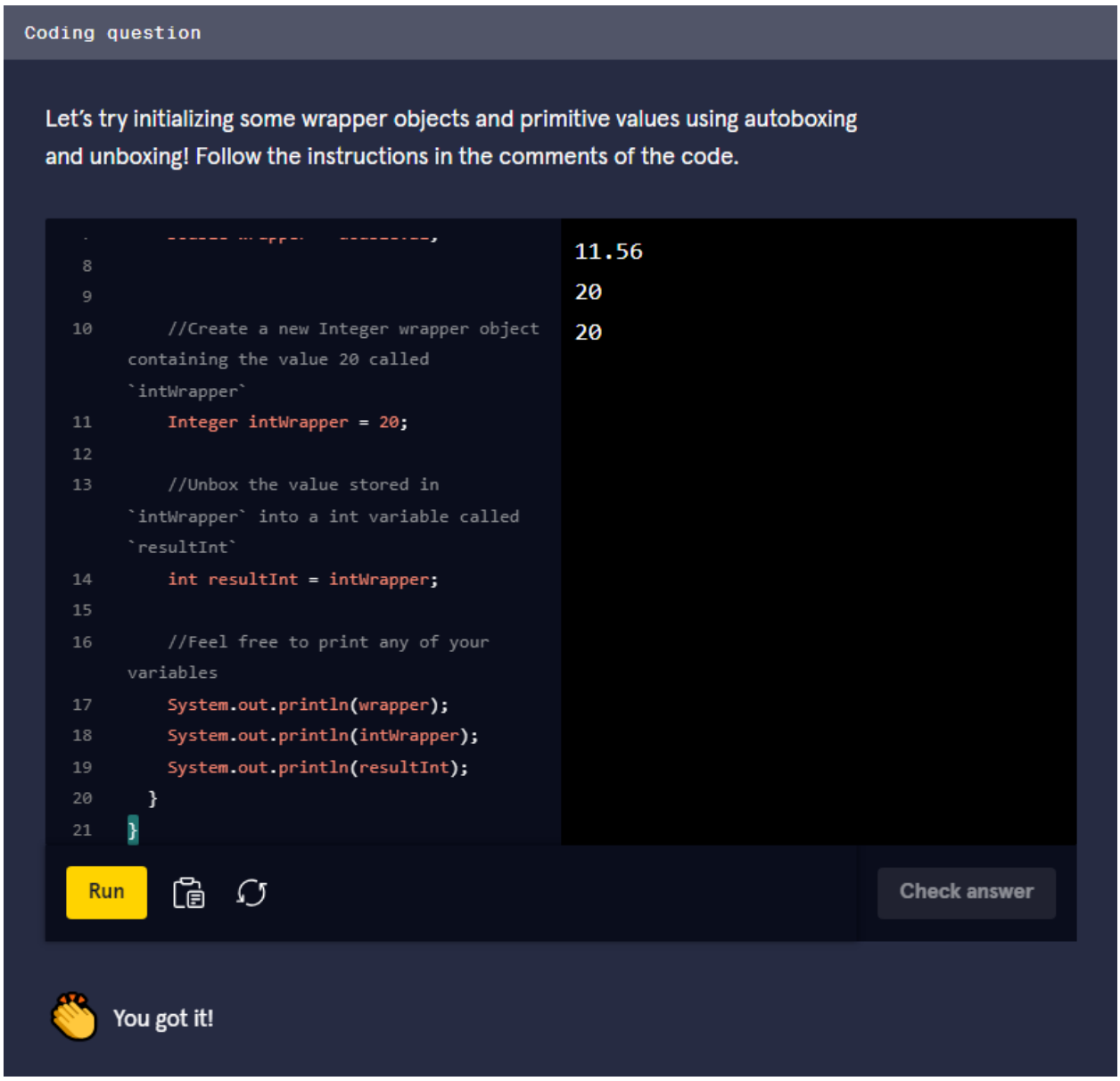
Each of these examples causes the compiler to autobox the primitive value into an object of the correct wrapper class. Notice how we can simply initialize the wrapper object to equal a primitive value and the compiler will still perform autoboxing.

Here are some examples where we unbox the values:

```
// Autoboxing into Integers
Integer wrapper = 3;
Integer wrapper2 = Integer.valueOf(15);

// Unboxing back to ints
int primitiveInt = wrapper;
int primitiveInt2 = wrapper2.intValue();
```

As you can see, the easiest way to unbox or autobox values is to simply assign a primitive value to a wrapper object, or to assign a wrapper object to a primitive variable. Java makes this conversion easy for you — the only thing you need to remember is to use the `Integer` and `Double` classes when you're working with a data structure that requires objects.

These previous examples also showed some of the methods that are contained within the wrapper classes. Each wrapper class contains methods for converting to the other data types as well as other useful methods and constants.




```java
public class AutoboxingAndUnboxing {
  public static void main(String[] args) {

    double doubleVal = 11.56;

    //Autobox the value of `doubleVal` into a Double wrapper object called `wrapper`
    Double wrapper = doubleVal;


    //Create a new Integer wrapper object containing the value 20 called `intWrapper`
    Integer intWrapper = 20;
```

```
    //Unbox the value stored in `intWrapper` into a int variable called `resultIn
t`
    int resultInt = intWrapper;

    //Feel free to print any of your variables
    System.out.println(wrapper);
    System.out.println(intWrapper);
    System.out.println(resultInt);
  }
}
```

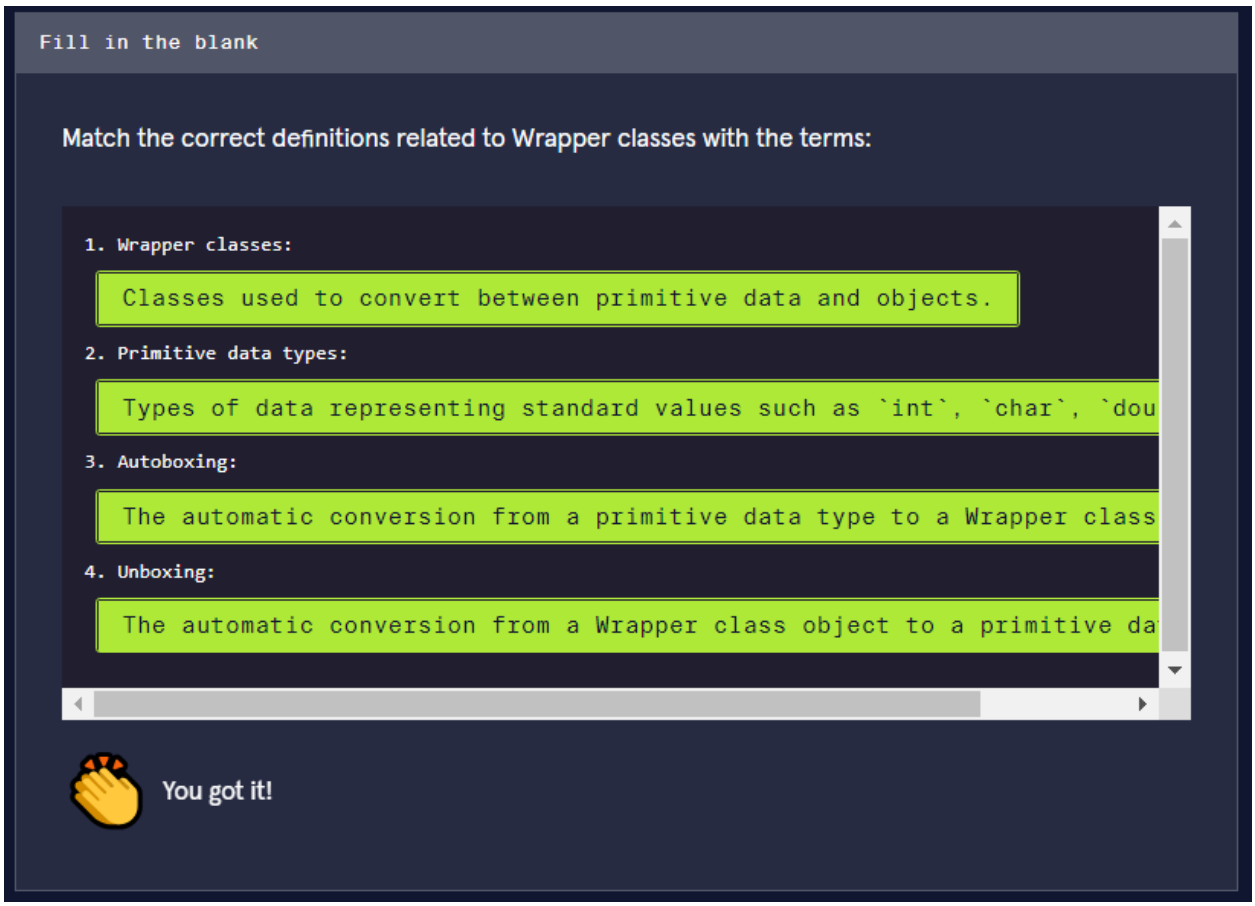
Autoboxing and unboxing are also used when passing values to methods. This is because, if a method is expecting a primitive data type in its input parameters, but an object of a wrapper class is passed instead, the java compiler will automatically unbox the wrapper object to access the primitive data contained within it. Likewise, if the method input parameters expect a wrapper object instead of a primitive value, the value will be autoboxed by the compiler.

Let's look at an example of autoboxing and unboxing in methods:

```
public class AutoboxingAndUnboxing{

  public static int acceptWrapperObj(Integer intVal) {
    System.out.println("Value of Integer wrapper object: " + intVal.toString());
    int toReturn = intVal;
    return toReturn;
  }

  public static Double acceptPrimitiveDouble(double doubleVal) {
    System.out.println("Value of primitive double: " + doubleVal);
    Double toReturn = Double.valueOf(doubleVal);
    return toReturn;
  }

  public static void main(String[] args) {
    int testInt = 5;
    int resultInt = acceptWrapperObj(testInt); // autobox the primitive value

    Double testDoubleObj = Double.valueOf(3.14);
    Double resultDouble = acceptPrimitiveDouble(testDoubleObj); // unbox the wrapper object
  }
}
```

As you can see in this example, the `acceptWrapperObj` method has an `Integer` object as a parameter, but we can pass a primitive value to it. The primitive value is autoboxed when it is passed to the method. Likewise, we can see an example of unboxing in the `acceptPrimitiveDouble` method. The method only accepts a primitive `double` as a parameter, but we can pass a `Double` object since it will automatically be unboxed.

Let's look at the `Integer` and `Double` classes in more detail after a brief quiz.



The `Integer` and `Double` classes are both located in the `java.lang` package along with wrapper classes for the other primitive data types.

The `Integer` class contains both static and non-static methods which can be used for comparisons, conversions, and calculations. We will be looking at some of the main ones, but you can see a list of all of them using by viewing the java API documentation: Java API: Integer Class

In order to get the value which is stored in the `Integer` wrapper object, we can use `.intValue()` and to initially set the value of the wrapper object, we can use either of the constructors. The first constructor accepts a primitive `int` value which is autoboxed to a wrapper object, while the second constructor accepts a `String` value. The string is parsed and converted into an `Integer` value. Let's look at some of the methods from the `Integer` class being used in code!

```
Integer wrapperInt = new Integer(5);
Integer wrapperInt2 = Integer.valueOf(230);
Integer wrapperFromStr = new Integer("100");

int primitiveIntFromWrapper = Integer.parseInt("150");
int primitiveIntFromWrapper2 = wrapperFromStr.intValue();
double convertedValue = wrapperInt.doubleValue();
```

Additionally, the `Integer` class also has some static fields which we can use. The ones we will be looking at are the `MAX_VALUE` and `MIN_VALUE` fields. Calling `Integer.MAX_VALUE` will return the largest value which an integer can store (2^{31}-1) while `Integer.MIN_VALUE` returns the smallest value which an integer can store (-2^{31}).

Here is what they look like when being stored in `Integer` objects.

```
Integer wrapperIntMax = Integer.MAX_VALUE;
Integer wrapperIntMin = Integer.MIN_VALUE;
```

The `Double` class has similarities to the `Integer` class in how it also has two constructors, one accepting a primitive double and the other accepting a `String`. The `Double` class also includes useful static and non-static methods as well as static fields. We will go over the main ones you will use, although a list of all of them can also be found in the API documentation: Java API: Double Class

In addition to having the `MAX_VALUE` and `MIN_VALUE` static fields, two additional important fields are `POSITIVE_INFINITY` and `NEGATIVE_INFINITY`. These allow infinity and negative infinity to be stored in code through a `double` data type or `Double` object.

The `Double` class also contains many methods for converting to other data types as well as tests to see if the stored value is infinitely large or not. To retrieve the value from the wrapper object, we can use `.doubleValue()` and to convert from a `String` we can use the `.valueOf()` or `.parseDouble()` static methods.

Here are some examples using the `Double` class:

```
Double wrapperDouble = new Double(10.34);
Double wrapperDoubleFromStr = new Double("20.40");
Double inf = Double.POSITIVE_INFINITY;
Double negInf = Double.NEGATIVE_INFINITY;

if(Double.isInfinite(inf + 1)) {
    System.out.println("Infinite");
}

double value = wrapperDouble.doubleValue();
float convertedVal = wrapperDoubleFromStr.floatValue();
```

Now that you have learned what wrapper classes are, how they work, and you have seen some examples, let's try a coding problem!

Coding question

Here's the final challenge! Let's experiment with the `Integer` and `Double` classes while using some of their methods! We will use the `Double` class `POSITIVE_INFINITY` static field and the `intValue()` method to try and store an infinitely large the value into an integer. We will then compare the result to the `Integer` class `MAX_VALUE` static field to see if we have maxed out the integer's value during the conversion. We will print out `"True"` if we find that the result of the conversion is the maximum value an integer can hold.

```
1  public class WrapperExperiment {
2    public static void main(String[] args) {
3
4      //Store the `POSITIVE_INFINITY` static field from the `Double` class into a `Double` wrapper object
5  Double inf = Double.POSITIVE_INFINITY;
6
7      //Unbox the infinitely large `Double` into an `int`
8    int result = inf.intValue();
9
10     //Modify the if condition to test if the integer value is equal to the `MAX_VALUE` static field from the `Integer` class
11
12     if(result == Integer.MAX_VALUE) {
13       System.out.print("True");
```

```
True
```

Run

Check answer

You got it!

```java
public class WrapperExperiment {
  public static void main(String[] args) {

    //Store the `POSITIVE_INFINITY` static field from the `Double` class into a `Double` wrapper object
Double inf = Double.POSITIVE_INFINITY;

    //Unbox the infinitely large `Double` into an `int`
  int result = inf.intValue();

    //Modify the if condition to test if the integer value is equal to the `MAX_VALUE` static field from the `Integer` class
```

```
    if(result == Integer.MAX_VALUE) {
      System.out.print("True");
    }

  }
}
```

Great work! Here are some of the main takeaways from this article:

- The `Integer` class and `Double` class are part of the `java.lang` package. They are used as a wrapper class to allow primitive `int` and `double` values to be used by methods that require Objects.
- Autoboxing is the automatic conversion that the Java compiler makes between primitive types and their corresponding object wrapper classes. This includes converting an `int` to an `Integer` and a `double` to a `Double`.
- Unboxing is the automatic conversion that the Java compiler makes from the wrapper class to the primitive type. This includes converting an `Integer` to an `int` and a `Double` to a `double`.
- The `Integer` and `Double` classes have important static variables named `Integer.MAX_VALUE`, `Integer.MIN_VALUE`, `Double.POSITIVE_INFINITY`, and `Double.NEGATIVE_INFINITY`. These can be used to find the largest and smallest `Integer` values that Java can store as well as represent the concept of infinity.