Introduction

We've been working only with the **int** data type so far in the course.

In this video, we'll continue to look at **int**, as well as several other primitive types.

We'll also introduce the wrapper class, a special category of data type, which offers additional functionality that primitive types don't.

Java's Primitive Types

In Java, primitive types are the most basic data types.

The eight primitive data types in Java are shown in the table below, listed by the type of data stored for each:

Whole number	**Real number (floating point or decimal)**
byte short int long	float double
Single character	**Boolean value**
char	boolean

Java's Primitive Types

Consider these types as the building blocks of data manipulation.

Remember that primitive data types are simply placeholders in memory for a value.

What actually is an integer?

An integer is a whole number, meaning it doesn't contain a fractional element, or a decimal.

What values can we store in an integer?

There's a specified range of values allowed for the **int**, which is true for most data types.

What this means is, that the allowable range of values is NOT infinite.

There's a defined minimum, and maximum value, for each numeric data type, meaning you can't assign a number bigger or smaller (outside of that range).

Using the + sign in System.out.print

The plus sign, +, when used in **System.out.print** will print different data types together as a single line of text.

In the example:

```
System.out.print("Integer Minimum Value = " + myMinIntValue);
```

We want to print a label, before a numeric integer value.

Whatever follows the plus sign in **System.out.print** here, is converted to a **String** by Java, and concatenated to the **String** before it.

This is perfectly valid syntax in Java.

Classes

So what is a class?

A class is a building block for object-oriented programming, and allows us to build custom data types. We'll be talking more about classes in future videos.

Wrapper Classes

Java uses the concept of a wrapper class, for all of its eight primitive data types.

A wrapper class provides simple operations, as well as some basic information about the primitive data type, which cannot be stored on the primitive itself.

We saw that MIN_VALUE, and MAX_VALUE, are elements of this basic information, for the int data type.

Wrapper Classes

The primitive types, and their respective wrapper classes, are shown in the table below.

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

You can see there, that in general, it's pretty easy to remember the wrapper class name, for your primitive data type. It's the same name, but with an uppercase letter at the start.

Wrapper Classes

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

The wrapper classes for char and int, Character and Integer respectively, are the only two that differ in name (other than that first capitalized letter) from their primitive types.

The Integer Wrapper Class

In the code we just reviewed, we were able to use MIN_VALUE, and MAX_VALUE, on the wrapper class Integer.

```
int myMinIntValue = Integer.MIN_VALUE;
int myMaxIntValue = Integer.MAX_VALUE;
```

To discover the minimum and maximum range of numbers, that can be stored in an int, as we saw when we printed out these values previously:

```
Integer Value Range (-2147483648 to 2147483647)
```

Overflow and Underflow in Java

If you try and put a value larger than the maximum value into an int, you'll create something called an Overflow situation.

And similarly, if you try to put a value smaller than the minimum value into an int, you cause an Underflow to occur.

These situations are also known as integer wraparounds.

Overflow and Underflow in Java

The maximum value, when it overflows, wraps around to the minimum value, and just continues processing without an error.

The minimum value, when it underflows, wraps around to the maximum value, and continues processing.

This is not usually behavior you really want, and as a developer, you need to be aware that this can happen, and choose the appropriate data type.

When will you get an overflow? When will you get an error?

An integer wraparound event, either an overflow or underflow, can occur in Java when you are using expressions that are not a simple literal value.

The Java compiler doesn't attempt to evaluate the expression to determine its value, so it DOES NOT give you an error.

When will you get an overflow? When will you get an error?

Here are two more examples that will compile, and result in an overflow. The second example may be surprising. Even though we are using numeric literals in the expression, the compiler still won't try to evaluate this expression, and the code will compile, resulting in an overflow condition.

```java
int willThisCompile = (Integer.MAX_VALUE + 1);
int willThisCompile = (2147483647 + 1);
```

If you assign a numeric literal value to a data type that is outside of the range, the compiler DOES give you an error. We looked at a similar example previously.

```java
int willThisCompile = 2147483648;
```

What does an underscore mean in a numeric literal?

In Java, you cannot put commas in a numeric literal.

For example, the following is not valid syntax.

```
int myMaxIntTest = 2,147,483,647;
```

So Java provided an alternative way to improve readability, the underscore.

```
int myMaxIntTest = 2_147_483_647;
```

You can put the underscore anywhere you might want a comma, but you can't use an underscore at the start or end of the numeric literal.