Introduction

In the last video, we were introduced to three new data types – the byte, the short, and the long.

In this video, we'll be using these additional types in some basic arithmetic.

We've already done some math, using integers, but now, we'll be using these other data types.

We'll finish the video with a discussion about casting, which is a way to get Java to treat a variable of one type like a different data type.

We'll talk about when, and why, casting is sometimes necessary in Java code.

Rules for declaring multiple variables in one statement

- You cannot declare variables with different data types in a single statement.
- If you declare multiple variables of the same data type in a single statement, you must specify the data type only once before any variable names.

Assigning expressions to variables with data types that don't match

The Java compiler does not attempt to evaluate the value, in a variable, when it's used in a calculation, so it doesn't know if the value fits, and throws an error.

```
byte myNewByteValue = (myMinByteValue / 2);
```

If your calculation uses literal values, Java can figure out the end result at compile time, and whether it fits into the variable, and won't throw an error if it does.

```
byte myNewByteValue = (-128 / 2);
```

In both examples, an int result is being returned from the calculation, but in the second example, Java knows the returned value can fit into a byte.

Casting in Java

Casting means to treat or convert a number, from one type to another. We put the type we want the number to be, in parentheses like this:

```
(byte) (myMinByteValue / 2);
```

Other languages have casting too, this is common practice and not just a Java thing.

What does it mean when Java defaults the data type to an int?

So, what effect does int, being the default value, have on our code?

Looking at the scenarios we just looked at in summary, we know the following:

This statement works because the result is an int, and assigning it to an int variable is fine.

```
int myTotal = (myMinIntValue / 2);
```

This statement doesn't work, because the expression (myMinShortValue / 2) is an int, and an int can't be assigned to a short, because the compiler won't guess the result.

```
short myNewShortValue = (myMinShortValue / 2);
```

What does it mean when Java defaults the data type to an int?

This statement works, because the result of (-128 / 2) is an int, but when calculations use only literal values, the compiler can determine the result immediately, and knows the value fits into a short.

```
short myNewShortValue = (-128 / 2);
```

And finally, this code works because we tell the compiler we know what we're doing by using this cast, and the compiler doesn't give an error.

```
short myNewShortValue = (short) (myMinShortValue / 2);
```

What does it mean when Java defaults the data type to an int?

```
int myTotal = (myMinIntValue / 2);
```