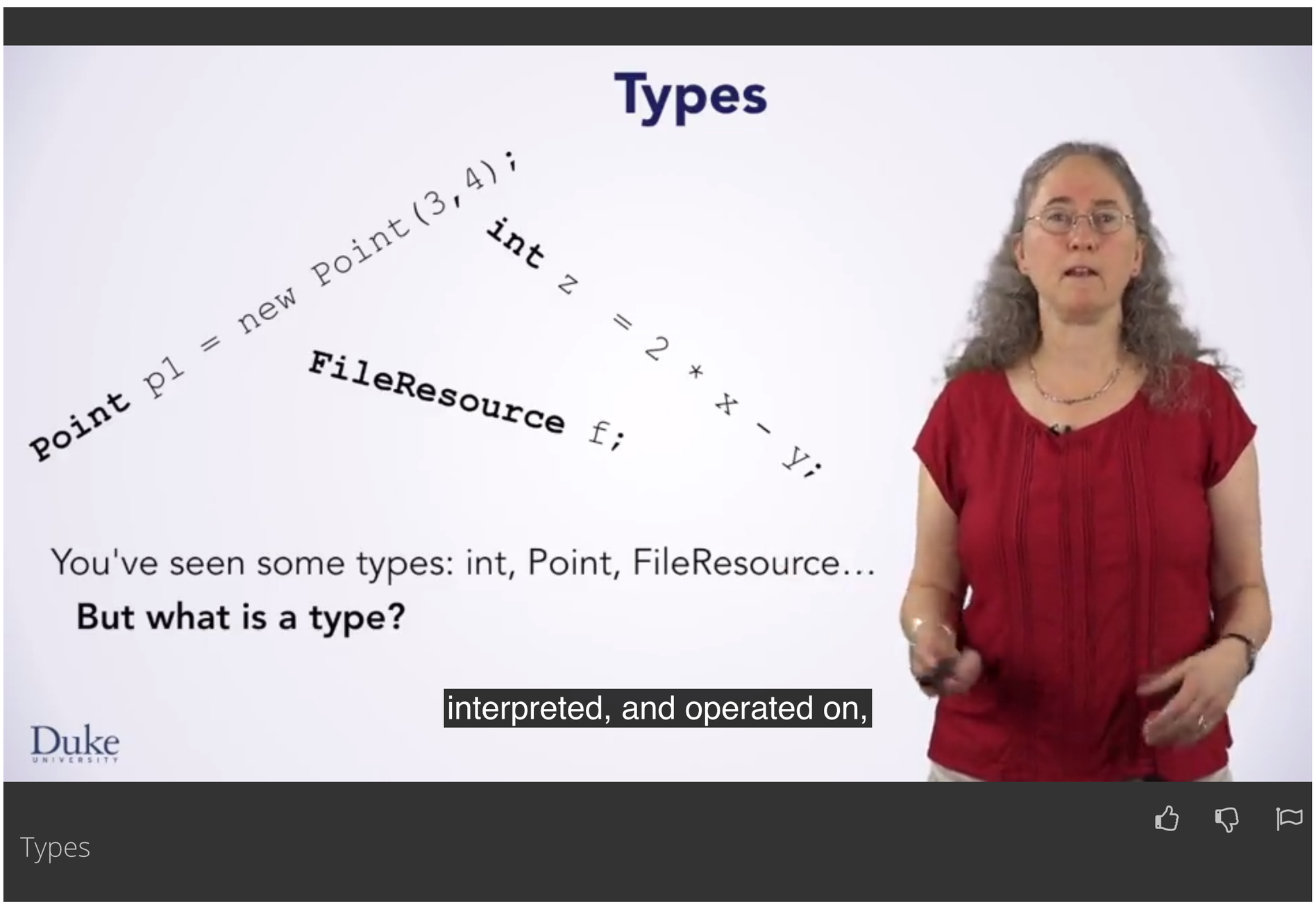

Types

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript English

0:02
At this point, you have seen a variety of types such as Int, Point and FileResource. But, what exactly is a type? A type specifies how data should be represented, interpreted, and operated on, as well as, what operations you can do with it. One important rule of computing is that, everything is a number. If you didn't learn about the everything is a number principle, and whatever intro courses brought you here, we'll give you some links to videos about it. Specifically, this means that everything is stored in the computer's memory as bits, ones and zeroes. But, not all numbers mean the same thing. Some numbers might mean plane numbers, while others might mean letters, and others might mean the locations of data in the computer's memory. So, the type of a value specifies how to interpret those numbers. It tells Java how to assign meaning to the ones and zeros stored in memory. The type also specifies what operations you can perform on the data. The type tells Java not only what you can do but how it should be done. Let's talk about both of these points in more detail. We just said that the type tells Java how to interpret the ones and zeroes in memory. Let's talk about that a little bit more, huh? On the left, we have the conceptual representation of the program state that we have been working with. There's a box for a variable called x. On the right, we have a bunch of bits from the computer's memory. The blue ones correspond to x. But what do they mean for the value of x? Well, if x is an int, then these bits would mean it has a value of 1234567890. We're not going into the details of how you can figure that out here, nor do you need to know it for beginning Java programming. But, if you take a computer organization class, you will learn a lot more about how data is represented. The point I want to make here is that, if x had a different type, like float, then the same bits would have different meaning. These same ones and zeros would mean 1228890.25. And if x were a string, then the bits would be the location in the computer's memory of the actual string object, which would have a sequence of characters. Those would also be stored with a bunch of bits that would be interpreted as letters, since your type would be char. We also said that the type tells us what operations we can do and how they are done. Consider this simple bit of code, x + y. Is it legal? And if so, what does it do? To answer this, you need to know the types of both x and y. If x or y are both ints, then this code is legal and performs integer arithmetic. If x and y are both strings, then this code is also legal but perform string concatenation. It makes a string with the letters of x, first, then the letters of y, immediately after. Notice that, even though the plus operation is legal for two different types, we may perform that operation differently for one type than for the other. If x and y are both points, then this code is not legal. While we're talking about types, you might be wondering what you'd do if you need to convert between types. The answer is that, it depends. For some type conversions, they can happen implicitly. If you have an int and you need to convert it to a double precision floating point number, the compiler will just automatically insert the conversion for you without complaint. The general rule is that, you can use implicit conversion whatever the compiler will consider it safe. Here, we are turning 3 into 3.0, which is not a problem. Note that the compiler does not consider the values, only the types, when deciding if an implicit conversion is okay. For some type conversions, you can explicitly cast. This means, that you tell the compiler that even though what you're doing is questionable, you are sure you want to do it. Here, we are turning the double 3.14 into an int, which will discard the fractional part, leaving us with x = 3. The compiler wants to be sure that we meant to do this, so we explicitly cast by writing int in parentheses. Other conversions require calling methods to calculate out the converted value. For example, if we have this string "3", and want to turn it into an integer, we can't just directly cast it because the conversion is actually somewhat complicated. Instead, we have to call a method like Integer.parseInt which will perform the conversion. The last thing we will mention about types is that, there are two major categories of types in Java, primitives and objects. There are eight primitive types: int, double; char, boolean, long, float; byte, short. We'll primarily use the first four of these. Variables are primitive types hold their value directly in their box. Primitive types don't have methods, so you can't do dot method call on a primitive, and they can't be null. Although, each primitive type has an associated wrapper class, which gives you an object to hold that primitive. Everything else is an object type. Some are built in the Java, like string. Others are part of libraries that you might use, like point or fileresource. And yet, others are classes you will create yourself. Whenever you make a class, the class you make is its own new type. Unlike primitives, the value of variables of object types is an arrow pointing at the object. This arrow is called a reference. You can invoke methods on the object with dot method name, and the reference can be null. Meaning, it does not refer to any object. If you do == on two objects, you're checking to see if the arrows point at exactly the same object. Okay, that's the basics of types. We know it's a lot to absorb at once. But, you'll get better with these ideas as you practice more Java.

Downloads

Lecture Video mp4

Subtitles (English) WebVTT

Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?