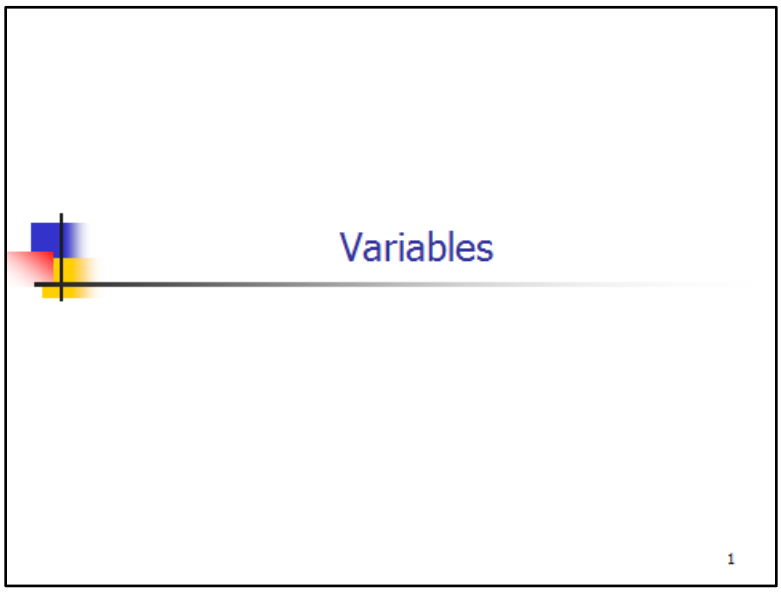
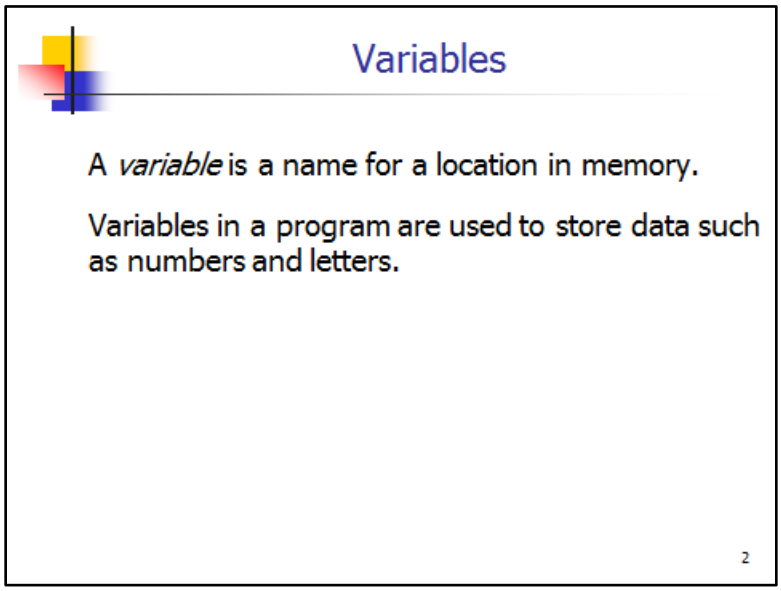



In this section of today's class, you will learn how to use *variables* to put data into your program and how to access and modify the data.

You will also learn how to tell the Java compiler that you want to make the value that you put into a variable permanent, so that it can never change as the program runs.

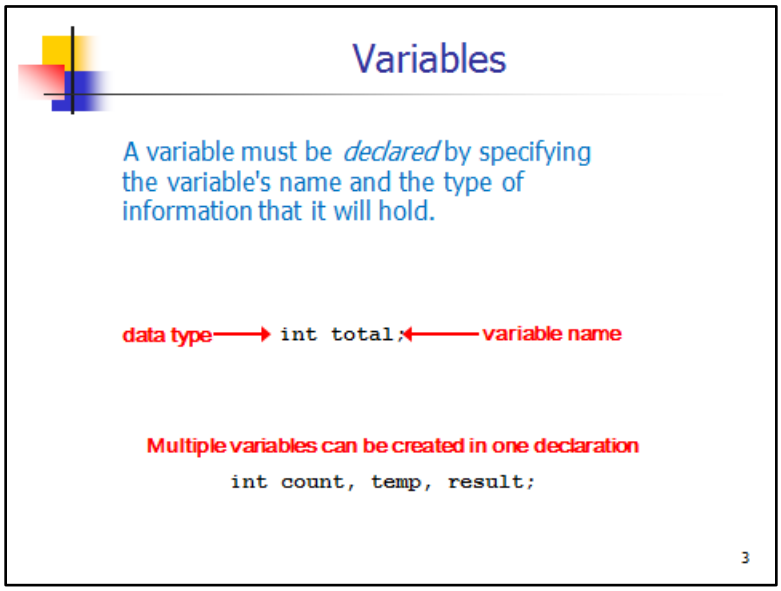



Most computer programs need to manipulate data. The computer hardware refers to memory in terms of numerical *addresses*, similar to post office boxes. Boxes in a *very large* post office.

The Java language does not let us refer directly to hardware addresses. Instead, it allows us to tell the compiler we want a place to store a certain amount of data of a certain kind, and to give that place a *name*. Then when we want to store data there, or to retrieve the data, we use the name, rather than a numerical address. We refer to this name as a *variable* because the program can change the contents.

The compiler determines the address where the data will be stored, and remembers the association between the address and the name that we specified. When we want to store data or retrieve it, we use the name.

We typically use the word "variable" to refer to *contents* of the memory location with that name. When we speak of setting or changing a variable, we mean the data, not the name.

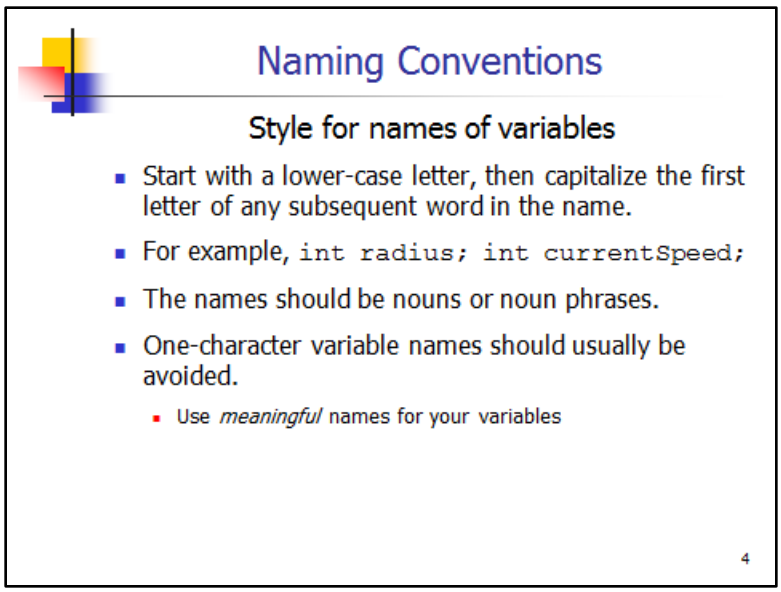



In the computer's memory, all data is binary, 1's and 0's. A given string of 1's and 0's might represent a letter, an integer, or a number with a fractional part.

Numbers that can have a fractional parts are called *floating point* numbers. Floating point numbers are similar to scientific notation, where we specify a value such as 1.2345 and a power of 10 to multiply that number by. In Java a floating point number is represented as and integer and a power of 2, which can be either positive or negative. A positive power of 2 moves the decimal point to the right. A negative power of 2 moves the decimal point to the left.

We normally don't need to be concerned with the internal representation of floating point numbers. In our source code, we write floating point values in normal decimal notation, such as 12.5014.

We we need a place to store a number or a letter, we have to tell the compiler which form of data we want to store and give the compiler a name for the data. The statement that does this is called a declaration. To declare an integer variable, we use a declaration like those shown in this slide. Note that a single declaration can create multiple variables.



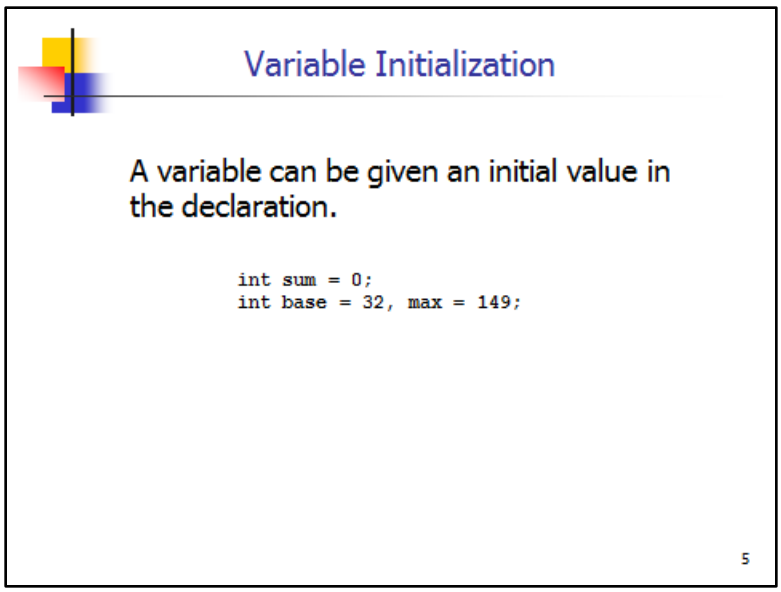

It is very important to use meaningful names for your variables, so that someone reading your program will know how you intend for the variable to be used.

For example, use the name radius, rather than the name r, for a variable that will be used to hold the radius of a circle. Don't be stingy with letters for your names, and don't use abbreviations that would not be obvious to a reader, such as rad for radius. But don't go overboard either with very long names, such as radiusOfSmallestCircleThatCanEncompassTheCityOfTampa. If you need to provide this information, use a comment.

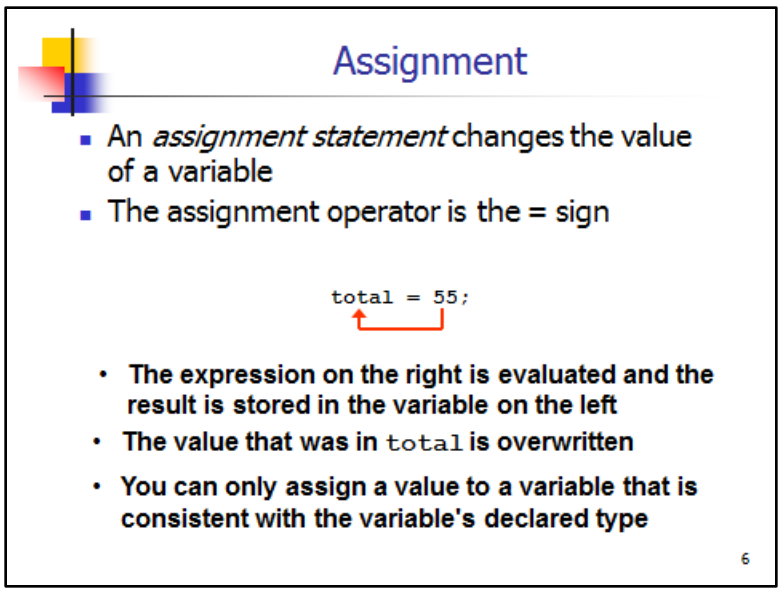
Programmers generally follow a convention, such as the one shown in this slide, for naming their variables. This doesn't matter to the compiler. It is mainly a matter of aesthetics. Follow the same convention throughout your program, in order to give it a clean uniform appearance.

One convention is to use all lower case letters, except when a variable is composed of more than one word, capitalize the first letter of each word after the first. This is often called "camel case." The name currentSpeed in this slide is an example.

Another convention is to separate the words with underscore characters. It doesn't matter which convention you follow, but you should try to be consistent throughout a program.




Variables can be given an initial value in the declaration. This is generally a good idea. If you try use the contents of a variable that has not been given a value, you will get an error. If you don't know what the value should be when you are writing the program, initialize it to 0, as shown for the variable sum in this slide.

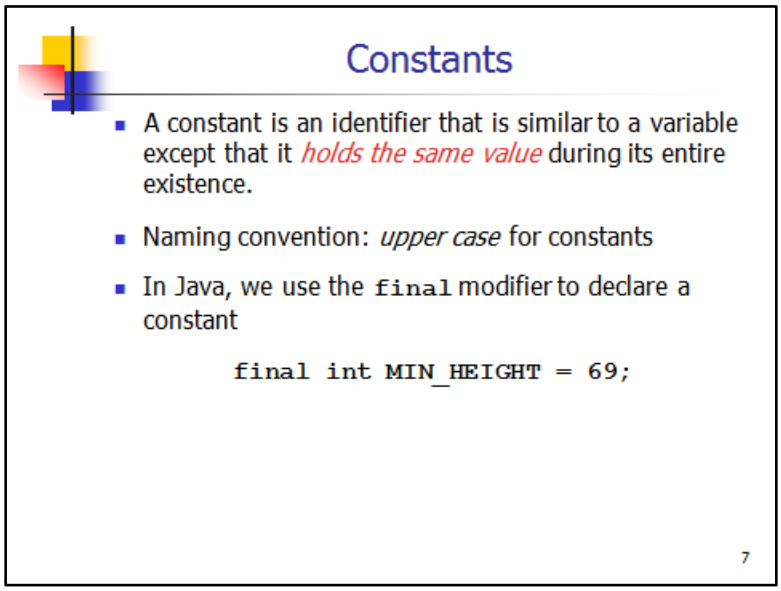



To set the value of a variable, we use the equal sign operator, as shown in this slide. This is referred to as an assignment statement. We are assigning the value on the right side of the equal sign to the variable whose name on on the left side.

When the assignment statement shown on this slide is executed, the contents of the variable total are overwritten with the value 55. The previous contents are lost forever. There is no "undo" statement in Java.

The value on the right side must be a valid value for the type of the variable on the left. If sum is declared as an int and you try to assign the value 55.5, you will get a compile time error.

Because Java forces you to use values consistent with the type, we say that Java is a strongly typed language. Sometimes this seems like a nuisence, but overall it helps prevent a lot of errors.

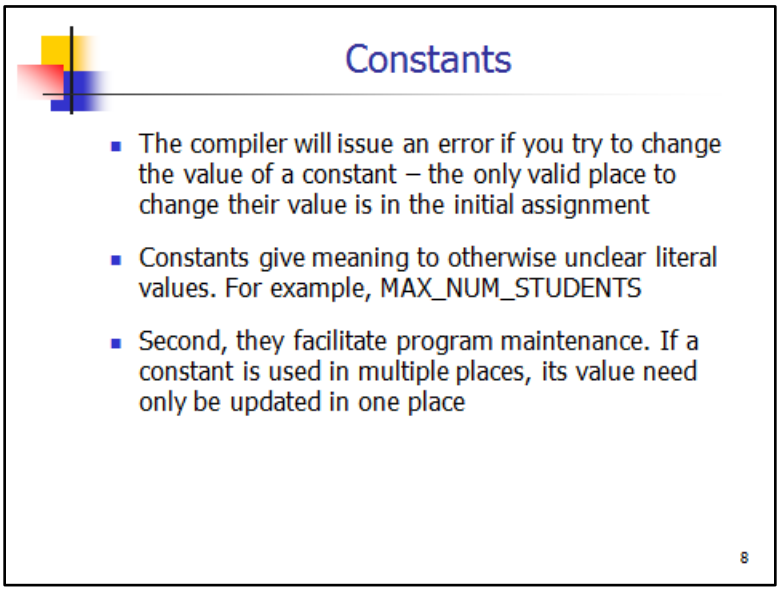



Sometimes we want to store a value that can never change. We refer to such variables as *constants*. If the idea of a constant variable seems a little strange to you, you are not alone. But we are stuck with the terminlogy. A variable represents a storage location in memory, and a storage location that cannot be changed after it is initialized is called a constant.

In Java, we tell the compiler that we want a variable to be a constant by writing the word final in front of the declaration, as shown in this slide. Because a constant cannot be changed, it must be initialized in the declaration.

By convention, we write the names of constants as all upper case, as shown in this slide.

You never *have* to make a variable final. But if you intend for a variable to be a constant, it is a good idea to declare it as such. This informs anyone reading the program of your intent, and it prevents you (or anyone else) from forgetting that it is intended to be constant and writing code to change the value.




When your program needs to use a fixed value, you *could* simple write the value as a number in statements that need it. This is generally a bad idea. Declaring a constant with a meaningful name makes your intention clear to anyone reading the program.

Even more important, if you need to change the value, you can change just the one statement that is the declaration. If you used the number 75 for the maximum number of students in a class, and later need to increase the limit to 100, you would have to track down all the places that you used the number 75 for the maximum number of students and change the 75 to 100. And it would not be safe to simply search for all instances of 75 in the program and change them to 100. You might have used 75 for other purposes as well. It is a lot easier and less error prone to update the declaration.