Variables

Variables give us a way to store pieces of data, like a numerical value or a word of text. Every variable has a unique name and a specific type of thing it can hold. We can think of them like a bucket that can only hold one type of building block: no other types of blocks will fit inside, and it can only hold one block at a time, so if you put a new block into it the block that is already there will be lost. At any time, however, you can find out what block it’s holding and use that information.

Another, slightly more accurate way of thinking about variables is to imagine a lined sheet of paper where we write a different value on each line. We can change the value on a particular line, but we have to erase what’s already there first. We can look at a particular “variable,” the eighth line, for instance, and read its value.

In reality, a variable is a particular section of memory in the computer. Data can be written, or stored, in that section of memory, and can also be read, or retrieved. For example, if we have a variable named myValue that is storing the number 5, calculating myValue + 2 will give us a result of 7.

In Java and C we create variables with a line of code like this:

type variableName;

This is called **declaring** the variable. When writing a variable declaration we always have to provide both the type and a name. In C, all variables must be declared at the beginning of the code section. Java allows you to declare variables at any point in your code.

# Variable Names

The name of the variable, called the **identifier**, must follow some basic rules in both Java and C:

* Identifiers must contain only letters, numbers, and underscores.
* They must begin with a letter or underscore, not a number.
* Identifiers are case sensitive, so upper- and lower-case letters are considered to be different. For example, variable, VARIABLE, and Variable are all different identifiers.
* Certain keywords that are used by the programming language can’t be used as identifiers. The word float is the type for variables that can hold floating-point numbers, so we can’t declare a variable with the name float. However, because identifiers are case sensitive, we would be able to name it Float.

|  |  |  |
| --- | --- | --- |
| Valid | Invalid | Invalid Reason |
| foo1 | 1foo | Doesn't start with a letter or underscore |
| \_foo | $foo | 1. Doesn't start with a letter or underscore 2. $ is an illegal character |
| foo | foo$ | $ is an illegal character |
| valid\_name | invalid-name | - is an illegal character |
| a\_very\_long\_valid\_name | foo bar | Can't have spaces in identifier names |
| Int | int | int is a keyword |

It is always a good idea to use descriptive names for your variables. Descriptive variable names can make a big difference in how understandable your code is. Consider the following examples, and how much clearer calculations can be when you use appropriate names.

|  |  |
| --- | --- |
| Good Identifier Names | Bad Identifier Names |
| int rate;  int time;  int distance;  rate = 60;  time = 20;  distance = rate \* time; | int x; int y; int z;  x = 60;  y = 20;  z = x \* y; |
| float PI = 3.1416F;  float radius = 5.25F;  float volume;  volume = 4.0 / 3.0 \* PI \* radius \*  radius \* radius; | float x = 3.1416F;  float id1 = 5.25F;  float id2;  id2 = 4.0 / 3.0 \* x \* id1 \* id1 \*  id1; |

A common convention is to make variable names lowercase, with uppercase letters used to separate them into multiple words, such as playerHealth or enemyAttackTime. This style is called camel case, or camelCase.

# Variable Types

The type of the variable specifies what sort of values can be stored in that variable’s memory, as well as what operations can be applied to the variable.

## Integral Types

An integral type allows us to represent an integer. An integer is a number that does not have a fractional or decimal part. For example, the numbers 2, 7, 18, and -5 are integers, but 1/3 and 1.5 are not.

In both Java and C there are four data types that can store negative and positive integers. Each of these types uses a different amount of memory, which means that they each have a different range of values that they can represent.

In C, there are additional integral types that only represent positive integers. These are called **unsigned** types, while the integral types that can represent both positive and negative integers are **signed**.

### Java

|  |  |  |
| --- | --- | --- |
| Data Type | Size | Value Range |
| byte | 1 byte | -128 to 127 |
| short | 2 bytes | -32,768 to 32,767 |
| int | 4 bytes | -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |

### C

|  |  |  |
| --- | --- | --- |
| Data Type | Size | Value Range |
| char | 1 byte | -128 to 127 |
| short | 2 bytes | -32,768 to 32,767 |
| int | 4 bytes | -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| unsigned char | 1 byte | 0 to 255 |
| unsigned short | 2 bytes | 0 to 65,535 |
| unsigned int | 4 bytes | 0 to 4,294,967,295 |
| unsigned long | 8 bytes | 0 to 18,446,744,073,709,551,615 |

## Floating Point Types

A floating point type can store any number, positive or negative. There are two types in both Java and C that can represent floating point numbers, with different precision. The precision value in the table below means that if you try to store a number with more than that many digits, such as trying to store 4.12758234958 in a float variable, any digits past the precision value will not be stored accurately.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Type | Size | Value Range | Precision |
| float | 4 bytes | 1.1754x10-38 to 3.4028x1038 | 6 digits |
| double | 8 bytes | 2.2250x10-308 to 1.7976x10308 | 15 digits |

Numbers that are supposed to be floats must be written with an f after the number, such as 3.14f. Without the f, the number is assumed to be a double.

## Other Types

Java has a type called boolean which represents either true or false. C does not have a built-in Boolean type, so in that case 0 is used to represent false while any other value (usually 1) represents true.

Java also has a type called char which, unlike the C char type, is used only to represent single letters.

## Mixing Types

So what happens if you have a variable of one type and you try to use it to store a different type? This depends on the types. In general, larger integral or floating point types are able to store smaller integral or floating point types. For instance, a long can store the value of an int, and a double can store the value of a float.

However, an int simply isn’t big enough to store all the data of a long. If you look at the tables above, you can see that the memory used by an int is only half the size of a long. Some compilers will warn you when you try to store a larger value in a smaller integer: depending on the value, you may end up with an unexpected result.

Integral types obviously can’t keep the decimal part of a floating point type, and floating point types can’t store the full range of integers. Unless you have a specific reason, it’s a good idea not to mix these types either.

# Initializing Variables

When we declare a variable, it has no starting value. We haven’t written anything to the variable’s memory yet so it essentially contains random garbage. This can be a significant source of bugs, because if we try and read from the variable we can get unpredictable results.

int foo; /\* foo is declared, it contains no value and cannot be read \*/

foo = 0; /\* foo is now initialized and can be read from \*/

We can both declare and initialize a variable on the same line. It is a good practice to always initialize your variables, even if you will be setting their value later, to avoid potential issues with uninitialized memory.

int foo = 0; /\* foo is declared and initialized, can be read from \*/