Types in Java

- In order to interpret the memory that is set aside for a variable correctly, we must know what type of data it’s supposed to be.

- Primitive type categories: Integers, floating point numbers, characters, Boolean values

Integers:

Whole numbers, positive, negative, and zero. The difference in the types have to do with the size an memory space the integer takes up.

byte is smallest: It takes up one byte = 8 bits.

Minimum: -128

Maximum: 127

short is next: It takes up 2 bytes = 16 bits.

Minimum: -32,768

Maximum: 32,767

int: 4 bytes = 32 bits

Minimum: -2^31 (signed)

Maximum: (2^31) -1

long: 8 bytes = 64 bits

Minimum: -2^63 (signed)

Maximum: (2^63) -1

-We can always assign narrow types to wider types, but we cannot assign wider types to narrower types.

For example, let’s say we have two declared variables.

*int numStudents = 20;*

*byte numLives = 3;*

If we try to assign,

*numLives = numStudents*

We won’t be able to. Why? Because the type of variable *numStudents,* a num, is wider than the type of variable *numLives* is, which is a byte. Even though 20 is a valid value to assign to a byte, *numStudents* has already been declared as num type, a wider data type than byte. Therefore, *numLives* cannot be assigned *numStudents.* If we assigned *numStudents* the value of *numLives*, this would be valid.

- There is a fix for this called “Casting”.

*numLives = (byte) numStudents;*

\*However, if numStudents was assigned the value of 200, this would not work because the maximum value of a byte is 127.

We can assign an integer literal to any variable it will fit.

During the assignment of a long data type, at the end of the number and before the semi-colon, we must type an L.

*Long veryLongNumber = 6000000000000000L;*

We must do this because it was an invalid integer with all of those zeros. But it is a valid long value once we add the L at the end of the line.

Given these variable declarations, which of the following assignments is NOT valid?

1. short1 = int1;

We cannot assign int to short because int is a wider data type than short.

Floating Points

-Kind of like real numbers- sort of. They do accept values with decimal points.

-What we have is something like scientific notation (exponential notation). We have an exponent 10 to some exponent times a value which is called the mantissa.

1.563 x 10^12

The idea of floating point numbers is that instead of storing a binary representation of the whole number, we’re going to store a mantissa and an exponent.

- We have two floating point types:

float (floating point number) : Size is 4 bytes

double (double precision floating point number): size is 8 bytes.

-Floating point numbers cannot always be accurate. If we think about the fraction 1/3 in decimal form, this equals 0.33333… and so on. If we think about this decimal value as a finite amount of digits, this will be inaccurate because we’ll be missing the rest of the actual value which keeps going on forever.

- Similarly, we have a problem like this with our floating point numbers. The mantissa only contains a certain number of bits.

Note: what is inaccurate in base 2 (in the computer) is not always inaccurate in base 10.

As we observed with integers, narrower data types cannot be assigned values of wider data types. Float and double are both wider than all integer data types.

Similar to declaring a value to long with an “L”at the end, numeric constants with a decimal point in Java are interpreted as doubles unless followed by “f” after the value.

19.5 is a double.

19.5F is a float.

- We can write very large or very small numbers (exponential or scientific form) in code.

3.8E14 = 3.8 x 10^(14)

2.3E-20 = 2.3 x 10^(-20)

- If we assign a float and double variable to the same value, float will give a different answer than double. This is because a double will have a larger range of minimum and maximum values (higher position). This demonstrates that floats and doubles don’t have the best accuracy (although double is more accurate. Using code, we can compare these values.

*System.out.println(floatNum==doubleNum)*

What prints from this is: false. This is because these two numbers are not interpreted the same as different data types although they have the same assigned value.

Let’s go back to the top and declare 2 long data types.

*Long veryLargeNumber = 6000000000000L*

*long large2 = veryLargeNumber;*

Then, we’ll compare these two.

*System.out.println(large2==veryLargeNumber)*

The result is: true.

We have to be very careful when it comes to our floating point numbers because when we compare their values, the output is false.

Take note of the output that if we try to cast a float variable to an int:

*int numStudents = 20;*

*float floatNum = 3.14f;*

*numStudents = (int) floatNum;*

*System.out.print(“numStudents: “)*

*System.out.println(numStudents)*

Run and output

numStudents: 3

What happened when we cast a floating point to an int (or any integer value), we are just going to throw away everything after the decimal point. Even if it was 3.99, the output would not round up, it would truncate.

True or false: For applications where accuracy is very important, it is better to store monetary values in integers (number of cents) rather than in floating point numbers.

1. True

Characters

*char* is a Unicode representation. Unicode characters are generally 2 bytes. Character literals are enclosed in single quotes: ‘A’.

Special characters:

‘\n’ : newline

‘\t’ : tab

‘\\’ : backslash

‘\’’ : single quote

‘\” ‘ : double quote

A note on Strings

We usually deal with more than one character at a time, (although we do have uses for that.) but it is less common than using multiple characters in what is called a String.

Strings are made up of characters. Understanding what characters are is important because Strings hold chars.

String literals are enclosed in double quotes.

char data types use single quotes; Strings are enclosed in double quotes.

‘a’ does not equal “a”.

Boolean

Possible values of true and false.

Useful when making decisions in programs.