Week 2: Computer Science 1

Variables

Storing data

A computer's memory is like a giant filing cabinet. Each location in memory has a numbered address. The address is like a file folder in the filing cabinet. The address is used to reference the location in memory.

A computer's memory locations are sequential, meaning they are numbered in order. Our programs can reference these locations by their address. The address is like a file folder in the filing cabinet.

Memory locations

1001	1002	1003	1004	1005	...	...

It would be cumbersome to remember the numbered addresses in memory. Instead, we use named **variable** to reference memory locations. A variable is like a label on a file folder. The label is used to reference the file folder in the filing cabinet.

Variables and **constants** are used to store information to be referenced by a computer program. They are named locations to reference a computer's memory locations.

- **Variables** are values that can change after being assigned a value.
- **Constants** are values that do not change after being assigned a value. (We will discuss later)

Each location in memory can only store a set number of bits usually a byte (8 bits). We can use these bits to represent different values. The number of bits determines the range of values that can be stored in a memory location.

These are the most common bit sizes:

Bit Size	Range of Values	Range of Vales in powers of 2
8 bits	0 to 255	0 to 2^8
16 bits	0 to 65,535	0 to 2^16
32 bits	0 to 4,294,967,295	0 to 2^32
64 bits	0 to 18,446,744,073,709,551,615	0 to 2^64

Their are many types of information that we want to store in our computers memory. Numbers, letters, words, sentences, and more. Each **type** of information has a different size and range of values.

To store different **types** of information, we use different **types** of variables.

Variable Types

Type	Size (bits)	Range of Values
byte	8 bits	-128 to 127
short	16 bits	-32,768 to 32,767
int	32 bits	-2,147,483,648 to 2,147,483,647
float	32 bits	-3.4E38 to 3.4E38
double	64 bits	-1.7E308 to 1.7E308
char	8 bits	0 to 255
boolean	8 bits	true or false

Note on E notation. 3.4E38 is 3.4 x 10^38

Whole Numbers

A **short**, **int**, and **long** are all used to store integers.

0,1,2,3,4,5,6,7,8,9,10, . . .

The difference between them is the size of the number they can store. We will mainly use int's in this class.

If you store a value outside the range of the variable type, you will **overflow** the variable. This means the value will wrap around to the other end of the range. Be careful when using variables near the limits of their range.

To create variables we follow the following steps: initialize, declare, and assign.

- **Declare**: Tell the computer the name and type of the variable.
- **Initialize**: The first time you assign a value to a variable.
- **Assign**: Give the variable a value.

```
// Declare a variable named age of type int
int age;

// Initialize age to the value of 10.
age = 10;

// Assign the value of 20 to age.
age = 20;

// Declare and initialize with one statement.
int x = 15;
```

The **equals sign (=)** is called the **assignment operator**. It assigns the value on the right to the variable on the left.

If we put 10 into a variable named age, then the contents of age becomes 10. We can represent this in a diagram.

The diagram is not exactly correct since a computer stores numbers in binary but for simplicity we will use decimal numbers.

Now let's assign the value of 20 to age.

Is the following code ok?

```
// Declare a variable named age of type int
int age;

// Initialize age to the value of 10.
age = 10;

// Declare and initialize age to the value of 10.
int age = 20;
```

This is not allowed. You cannot declare a (local) variable of the same name twice. You can only declare a variable once. You can assign a value to a variable as many times as you want. We will discuss local variables later.

Decimal Numbers

Float and **double** are used to store decimal numbers. The difference from integers is the addition of a decimal place, the size of the number they can store, and the precision of the number. A double is more precise than a float. The default type for decimal numbers is a double. To use a float you must add an f to the end of the number.

0.0, 0.1, 0.2, 0.3, 0.4, 0.5, . . .

```
// Declare a variable named price of type float
float price;

// Initialize price to the value of 10.99.
price = 10.99f;

// Assign the value of 20.99 to price.
price = 20.99f;

// Declare and initialize with one statement.
double cost = 15.99;
```

True or False

A **boolean** is used to store a true or false value. A boolean can only store one of two values: true or false.

```
// Declare a variable named isRaining of type boolean
boolean isRaining;

// Initialize isRaining to the value of true.
isRaining = true;

// Assign the value of false to isRaining.
isRaining = false;

// Declare and initialize with one statement.
boolean isSunny = true;
```

Characters

A **char** is used to store a single character. A character is a letter, number, or symbol.

'A' to 'Z', 'a' to 'z', '0' to '9', '!', '@', '#', '$', '%', '^', '&', '*', '(', ')', '-', '+', '=', ' '

Use single quotes (' ') to indicate a character.

Remember that computers only can store numbers. To store a character, the computer uses a number to represent the character. The number is called the **ASCII** code.

```
// Declare a variable named letter of type char

char letter;

// Initialize letter to the value of 'A'.
letter = 'A';

// Assign the value of 'B' to letter.
letter = 'B';

// Declare and initialize with one statement.
char grade = 'A';
```

Words

A **String** is used to store a sequence of characters. A String is a sequence of characters enclosed in double quotes (" "). Strings are not a primitive type but are a class. We will discuss classes later. Notice how the String is capitalized which sets it apart from the other types.

"Hello", "World", "CS1", "Java", "Programming"

```
// Declare a variable named name of type String
String name;

// Initialize name to the value of "John".
name = "John";

// Assign the value of "Jane" to name.
name = "Jane";

// Declare and initialize with one statement.
String firstName = "Michael";
```

ASCII (American Standard Code for Information Interchange) is a character encoding standard. ASCII codes represent text in computers. Each character is assigned a number from 0 to 127. The ASCII table is a list of all the characters and their corresponding numbers.

https://www.ascii-code.com/

Let's check the numbers for the characters 'A' and 'a' in the ASCII table.

```
// Let's create a char variable and assign it the value 'A'
char letter = 'A';
System.out.println(letter);
System.out.println((int) letter);

// Let's reassign the variable to the value 'a'
letter = 'a';
System.out.println(letter);
System.out.println((int) letter);
```

```
System.out.println((int) 'A');
```

The (int) is called a **cast**. It converts the character 'A' to an integer. The output of the above code is 65. The ASCII code for 'A' is 65.

Casting is a way to convert one type of variable to another type. Each one of the variable types has a specific size of memory it uses so only certain types can be cast to other types.

Smaller types can be cast to larger types but larger types cannot be cast to smaller types without losing information.

There are certain practices you should follow when naming variables.

- Variable names should be descriptive.
- Variables should start with a lower case letter.
- Internal words start with capital letters (camel case) or are separated by an underscore "_" (snake case). No blank spaces.
- Variable names should not start with underscore "_" or dollar sign "$" characters, even though both are allowed.

```
// Good variable names
int age;
// camel case
int numberOfStudents;
// snake case
int number_of_students;
```

There are certain **reserve words** that cannot be used as variable names. These are words that have a special meaning in Java.

Reserved Words

abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, enum, extends, final, finally, float, for, goto, if, implements, import, instanceof, int, interface, long, native, new, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, try, void, volatile, while

Math Operators

Symbols like "+", "-", "*", "/", and "%" are called **operators**. They are used to perform operations on variables and values. The following table shows the math operators in Java.

Operator	Description	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y

The **addition** operator (+) adds two values together.

```
int x = 5;
int y = 10;
int z = x + y;
System.out.println(z);
// Output: 15
```

The **addition** operator (+) can also be used to concatenate two strings together.

Concatenate means to join together.

```
String firstName = "Michael";
String lastName = "Curry";
System.out.println(firstName + " " + lastName);
// Output: Michael Curry
```

The **addition** operator (+) can have two different meanings depending on the types of the variables. This is called **overloading** an operator.

```
System.out.println("11" + "7");
// Output: 117
System.out.println(11 + 7);
// Output: 18
System.out.println(11 + "7");
// Output: 117
```

In the last print statement we have an int and a String. The int is converted to a String and then concatenated with the other String.

The **subtraction** operator (-) subtracts one value from another.

```
int x = 10;
int y = 5;
int z = x - y;
System.out.println(z);
// Output: 5
```

Can you subtract two Strings?

No, you cannot subtract two strings in Java. The subtraction operator (-) is not defined for strings.

The **multiplication** operator (*) multiplies two values together.

```
int x = 10;
int y = 5;
int z = x * y;
System.out.println(z);
// Output: 50
```

We must always explicitly use the multiplication operator (*) when multiplying two values together. The multiplication operator (*) is not implied.

```
System.out.println((2)(4)); // illegal missing *
System.out.println(2x4); // illegal cannot use x
```

The **division** operator (/) divides one value by another.

```
int x = 5;
int y = 2;
int z = x / y;
System.out.println(z);
// Output: 2
```

If the two variables are integers the division operator (/) performs integer division. The result is an integer. The fractional part of the result is truncated (removed).

```
System.out.println(5.0 / 2.0);
// Output: 2.5
```

If at least one of the variables is a double or float, the division operator (/) performs floating point division. The result is a double or float

The **modulus** operator (%) returns the remainder of a division operation.

```
int x = 5;
int y = 2;
int z = x % y;
System.out.println(z);
// Output: 1
```

Within a String all math operators are treated as characters. They are not treated as operators.

```
System.out.println("2 + 2 = 4");
// Output: 2 + 2 = 4
```

The rules of **precedence** determine the order in which operators are evaluated. Operators with higher precedence are evaluated first. Operators with the same precedence are evaluated from left to right. These are the same rules that are used in math.

```
System.out.println(2 + 3 * 4);
// Output: 14
System.out.println((2 + 3) * 4);
// Output: 20
System.out.println(10 - 5 - 3);
// Output: 2
System.out.println(10 - (5 - 3));
// Output: 8
```

Their is two more sets math operators that we will discuss. The increment operator: (++) and (+=). The decrement operator: (--) and (-=). The increment operator adds one to a variable and the decrement operator subtracts one from a variable.

```java
int y = 5;
y = y + 1;
System.out.println(y);
// Output: 6

int x = 5;
x++;
System.out.println(x);
// Output: 6

int z = 5;
z += 1;
System.out.println(z);
// Output: 6
```

Let's explain the following lines of code and review the rules of types, variables, operators, precedence, and casting.

```
int x; // Declare a variable named x of type int
x = 7; // Initialize x to the value of 7
System.out.println(x); // Output: 7
System.out.println(x + 4); // Output: 11
System.out.println(x/2); // Output: 3
System.out.println((x + 4) * (x / 2)); // Output: 33
System.out.println("x"); // Output: x
x++; // Increment x by 1
System.out.println(x); // Output: 8
System.out.println("x = " + x); // Output: x = 8
x--; // Decrement x by 1
System.out.println(x); // Output: 7
```

Notice the line "System.out.println("x = " + x);" The plus sign (+) is used to concatenate the String "x = " with the value of x. The value of x is converted to a String and then concatenated with the other String.

Relational Operators

Relational operators are used to compare two values. They return a boolean value of true or false. The following table shows the relational operators in Java.

Operator	Description	Example
==	Equal	x == y
!=	Not Equal	x != y
>	Greater	x > y
<	Less	x < y
>=	Greater or Equal	x >= y
<=	Less or Equal	x <= y

```java
boolean b;
b = 2 < 3;
System.out.println(b);
// Output: true

int x = 2;
int y = 3;
System.out.println(x < y);
// Output: true

System.out.println(x == y);
// Output: false

System.out.println(x != y);
// Output: true

System.out.println(1 < x > 5);
// Illegal! cannot chain relational operators
```

Notice the line "System.out.println(1 < x > 5);" The relational operators (<, >) cannot be chained together. You must use two separate relational operators to compare three values. To use more than two relational operators, you must use logical operators.

Logical Operators

Logical operators are used to combine two or more relational expressions. They return a boolean value of true or false. The following table shows the logical operators in Java.

Operator	Description	Example
&&	And	x && y
||	Or	x || y
!	Not	!x

The operator && is called the **and** operator. It returns true if both expressions are true.

The operator || is called the **or** operator. It returns true if either expression is true.

The operator ! is called the **not** operator. It returns true if the expression is false.

We can use a **truth table** to show all the possible values of a logical expressions: q && p and p || p.

p	q	p && q
true	true	true
true	false	false
false	true	false
false	false	false

p	q	p || q
true	true	true
true	false	true
false	true	true
false	false	false

The ! operator is called the **not** operator. It returns true if the expression is false. It only works on one expression.

p	!p
true	false
false	true

What does the following print?

```
boolean p = true;
boolean q = false;
boolean r;

r = p && q;
System.out.println(r);
```

```java
boolean p = true;
boolean q = false;
boolean r;

r = p && q;
System.out.println(r);
// Output: false
```

The expression p && q is false because p is true and q is false. Both expressions must be true for the && operator to return true.

What does the following print?

```
boolean p = true;
boolean q = false;
boolean r;

r = p || q;
System.out.println(r);
```

```java
boolean p = true;
boolean q = false;
boolean r;

r = p || q;
System.out.println(r);
// Output: true
```

The expression p || q is true because p is true. Only one expression must be true for the || operator to return true.

What does the following print?

```
boolean p = true;

System.out.println(!p);
```

```java
boolean p = true;

System.out.println(!p);
// Output: false
```

The expression !p is false because p is true. The ! operator returns the opposite of the expression.

Printing to the Screen

Let's look at the three different ways to print output to the screen.

System.out.println() is one of the methods to print output to the screen. It prints the string inside the parentheses to the screen and then moves the cursor to the next line.

System.out.print() is another method to print output to the screen. It prints the string inside the parentheses to the screen and leaves the cursor on the same line.

System.out.printf() is another method to print output to the screen. It prints the string inside the parentheses to the screen and leaves the cursor on the same line. It also allows you to format the output.

System.out.printf()

```
String name = "Class";
int age = 100;
char grade = 'A';

System.out.printf("Hello, %s! %n You should get %d on Test 1 which is an %c.", name, age, grade);
// Hello, Class!
// You should get 100 on Test 1 which is an A.
```

The %s is a placeholder for a String. The %d is a placeholder for an integer. The %f is a placeholder for a float or double. The %c is a placeholder for a char. The %b is a placeholder for a boolean. The %n is a placeholder for a new line.

Exercises

Do you think the following is okay/possible?

```
char c;
char c2 = c;

int x = 5;
int x = 7;

int y = 123;
double z = y;

double a = 2.34;
int b = a;
```

```
char c;
char c2 = c;
// No, c is not initialized.

int x = 5;
int x = 7;
// No, you cannot declare a variable twice.

int y = 123;
double z = y;
// Yes, you can assign a smaller type to a larger type.

double a = 2.34;
int b = a;
// No, you cannot assign a larger type to a smaller type.
```

Which of the following are allowed?

```
int x = 2.3;

double y = 4;

int a = ‘D’;
```

```
int x = 2.3;
// No, you cannot assign a double to an int.

double y = 4;
// Yes, you can assign an int to a double.

int a = ‘D’;
// Yes, you can assign a char to an int.
```

What do the following print?

```
int x = 5;
int y = 2;
System.out.println((x + y) * 2 + (x - y));

System.out.println(x < y && x <= y);

System.out.println(x < y || x <= y);

System.out.println(x < y || x > y);

System.out.println(!(x < y || x <= y));

System.out.println("Is x a String? " + x);
```

```java
int x = 5;
int y = 2;
System.out.println((x + y) * 2 + (x - y));
// Output: 17

System.out.println(x < y && x <= y);
// Output: false

System.out.println(x < y || x <= y);
// Output: false

System.out.println(x < y || x > y);
// Output: true

System.out.println(!(x < y || x <= y));
// Output: true

System.out.println("Is x a String? " + x);
// Output: Is x a String? 5
// x is cast to a String.
```