Day 2: Java Basics

Download

Agenda

- Variables
- Primitive Data Types
- Naming Conventions
- Brief Intro to Strings
- Output
- Lab 1: Printing Data

Download

Variables

Download

In order to create a variable in Java, we must follow this format.

We can declare a variable to be set/initialized later.

We must then initialize it with a value later.

We can also combine it into a single line as seen in the third case.

Variable Declaration:

```
<type> <variable name>;
```

Variable Initialization:

```
<variable name> = <value>;
```

Combined:

```
<type> <variable name> = <value>;
```

Download

Java is a semi-colon language, so we must add a semi-colon at the end of the line to denote the line has ended.

Variable Declaration:

```
<type> <variable name>;
```

Variable Initialization:

```
<variable name> = <value>;
```

Combined:

```
<type> <variable name> = <value>;
```

Download

What do I mean by <type>?

Variable Declaration:

```
<type> <variable name>;
```

Variable Initialization:

```
<variable name> = <value>;
```

Combined:

```
<type> <variable name> = <value>;
```

Download

Data Types

Download

Java has 8 primary "primitive" data types. These are shown on the right:

These data types represent the way data is stored in Java. They are required when defining variables and later functions.

1. byte
2. short
3. char
4. int
5. long
6. float
7. double
8. long
9. boolean

Download

These primitive data types don't have methods attached to them. They exist on their own and simply represent some value, but we don't have functions that connect to the type.

1. byte
2. short
3. char
4. int
5. long
6. float
7. double
8. long
9. boolean

For example, with a "char", we cannot use a ".toUpperCase()" on it.

This will make more sense when we talk more about Strings.

1. byte
2. short
3. char
4. int
5. long
6. float
7. double
8. long
9. boolean

Download

Byte

A byte represents an 8-bit signed **two's complement** integer storing values from -128 to 127 (-2^8 to 2^8-1)

Example of a byte variable:

```
byte x = 120;
```

OR

```
byte x = -52;
```

Two's Complement?

Remember binary from 110?

A Two's Complement means the first bit in the binary value defines whether the value is positive or negative.

Download

Binary

Let's review binary quickly first, then we can see this Two's Complement in action!

Let's say we have the binary value of 110.

What is the decimal of that?

Download

Binary

Let's review binary quickly first, then we can see this Two's Complement in action!

Let's say we have the binary value of 110.

What is the decimal of that?

Remember, binary works with powers of 2.

Download

Binary

Let's review binary quickly first, then we can see this Two's Complement in action!

Let's say we have the binary value of 110.

What is the decimal of that?

Remember, binary works with powers of 2.

$$110 = 1 * 2^{2} + 1 * 2^{1} + 0 * 2^{0} = 6$$

Negative binary?

$$110 = 1 * 2^{2} + 1 * 2^{1} + 0 * 2^{0} = 6$$

Now, without using a negative sign, how can we turn this into a negative value?

Turn and talk with the person next to you. Think of a way we could turn this into a negative number.

Hint: it doesn't have to evaluate to 6 in decimal, it could give us -2.

Negative binary?

$$110 = 1 * 2^{2} + 1 * 2^{1} + 0 * 2^{0} = 6$$

The way Java does it uses the first bit in the binary string as an indicator of positive or negative.

If the first bit is a 1, it means the value is a negative number. If the bit is a 0, it means the value is a positive number.

So, actually, the way Java works with binary, 110 doesn't give us 6, it actually gives us -2. This is because the leading 1 denotes a negative number. We then take the value after, being 10, or $1 * 2^{1} + 0 * 2^{0} = 2$. We can then add the negative back, so we get -2.

Download

Two's Complement

So why is it called a two's complement?

If we invert each binary bit and add 1 to the binary, we would get the negative complement.

Take 0110 being decimal 6. To get the negative equivalent, we would flip each binary value to get 1001, then add 1 to it, giving us 1010.

When we evaluate it, the leading 1 serves as the negative indicator, but it's included in the calculation.

So we get $-(1 * 2^3) + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = -6$

So what is the largest binary value with this system?

Well that depends on the size of the type. For a byte, we would have 1000 0000.
This gives us

$$-(1 * 2^7) + 0 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 0 * 2^0 = -128$$

this means the biggest positive value is 127, as we would have 0111 1111.

So what about 1111 1111?

Download

So what is the largest binary value with this system?

Well that depends on the size of the type. For a byte, we would have 1000 0000.
This gives us

$$-(1 * 2^7) + 0 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 0 * 2^0 = -128$$

this means the biggest positive value is 127, as we would have 0111 1111.

So what about 1111 1111?

Well, that gives us -1. Why?

Fun fact: Did anyone play the game Civilizations?

Fun fact: Did anyone play the game Civilizations?

This signed value is why Gandhi in the game is so ruthless. In a signed value, when you go from 0000 0000 and subtract 1 to get -1, you would be at 1111 1111. However, if it's unsigned, meaning the leading bit doesn't denote a signed value, we would instead get 256, which is a lot larger than -1.

Download

Back to data types

Download

short

Our next type, in no particular order, is a short.

This is a 16-bit signed two's complement integer storing values from -32768 to 32767 (-2^{16} to $2^{16}-1$)

```
short age = 25;

short q = -10382;
```

Download

int

Our next type, in no particular order, is an int.

This is a 32-bit signed two's complement integer storing values from -2,147,483,648 to 2,147,483,647 (-2^{31} to $2^{31}-1$)

```
int age = 32;
```

```
int x = -38192371;
```

Download

long

Our next type, in no particular order, is a long.

This is a 64-bit signed two's complement integer storing values from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (-2^{63} to $2^{63}-1$)

You may have to add an "L" at the end of the number to denote to Java that the number is a long and not an integer.

```
long currentTime = 1756082145L;
```

Download

float

Next up, we have a float.

A float is a 32-bit floating point data type. It's a less precise floating point value due to how it is represented in its binary form.

It is good if we need smaller decimal values, slightly less precision, and to compute values faster. It has a smaller memory footprint.

```
float a = 3.141f;
```

//Float values must end with an "f" to denote it is a float and not a double.

Download

double

Next up, we have a double.

A double is a 64-bit floating point data type. It's a more precise floating point value, due to having more binary bits to use.

It's also the default type for decimals in Java.

Due to the higher bit usage, it uses more memory, but that means it's more precise.

```
double a = 3.141;
```

//Note: Does not require a "d" after, unlike a float that requires an "f"

Download

boolean

Now, this is the smallest type, a boolean.

This represents the value of `true` and `false`. Unlike Python, the literals of true and false are lowercase.

It also represents a single bit of information as it doesn't have to occupy any additional space.

```
boolean isStudent = true;

boolean hasGraduated = false;
```

char

The last type for our primitive types is a char.

A char contains a single 16-bit character. This is a single character and requires the use of single quotes surrounded the character.

```
char playAgain = 'n';

char keepGoing = 'y';
```

Download

Naming Conventions

A quick note about Java's Naming Conventions!

Download

In Java, we use Camel Casing for variables and functions (methods).

Classes, which we'll touch upon later in the semester, use Pascal Casing.

Constants and enums are all in Capital/Uppercase Letters.

First character must be a letter, numbers can follow after. Underscores are okay, but not recommended due to naming conventions.

Download

Camel Casing

First letter of each word after the first word is capitalized.

```
aReallyLongVariableName
```

No spaces since Java sees a space as a separator, not a part of the variable.

Download

Pascal Casing

First letter of each word is capitalized, including the first character of the firstWord.

```
AReallyLongClassName
```

Again, no spaces due to how Java reads spaces as separators of code.

Download

Upper Case

This one is pretty self explanatory

```
ACONSTANTVALUE
```

This one can also have underscores to make it easier to read.

```
A_CONSTANT_VALUE
```

These naming conventions are there to help us create code that follows a standard convention. Please stick to these conventions. They are ways that allow us to create names for variables, functions (methods), classes (later), and constants with multiple words, while maintaining a degree of readability.

We can't have spaces in these names, so these help us identify different words.

Download

Back to data types

Sorta...

Download

Did we miss one?

Download

What about Strings?

In Java, a String isn't a "primitive" data type.

This is because there are more things we can do with strings.

We will look and dive into Strings later in the semester, but here's the basic gist of them.

A String is a data type. However, it's what we call a **Object Data Type**

To use a String, simply surround the text with double quotes and define the type as a String.

String can contain any character, as long as they are enclosed in the double quotes.

```
String name = "Edward";

String courseName = "Foundations of
Program Design"

String course = "CS111 Foundations
of Program Design"
```

Output

Download

Last main point for today's lecture.

We can output all of these variables and values using our trust `"System.out.println()"` that we saw last class.

With println, we can pass in variables or values to print out. Unlike the Python print, we cannot pass in a comma to separate values. Instead, we use the + operator to concatenate/combine values.

When a string is present, the values will be concatenated.

```
int age = 25;

System.out.println("Age:" + age);

//This works, outputting "Age: 25"
```

When we run the code to the right, we don't get any errors as it can concatenate (add) the values together, despite age being a number.

```
int age = 25;

System.out.println("Age:" + age);

//This works, outputting "Age: 25"
```

A downside to this is that once it begins concatenating, it will naturally concatenate, unless parentheses are added to ensure we stop concatenating.

```
System.out.println("Age:" + 25 +
2);
```

//Above produces: "Age: 252"

//If we want to add 2, we would have to do the following:

```
System.out.println("Age:"+(25+2));
```

//This produces: "Age: 27" as it does the arithmetic first.

If we don't want to print the new line character after, we can use `System.out.print();`

This is helpful when we want to print different values on the same line and not use a single print statement.

More useful when we touch upon loops and printing out patterns of text.

```
System.out.print("Hello World");

//This won't print a new line after


System.out.print("Welcome ");

System.out.print(name);

System.out.print("!");
```

Download

Lab 1: Printing Data

Download

Lab 1: Printing Data

For the remainder of class, we will work on Lab 1. This is a simple exercise on using variables and print statements.

When you finish it, please submit it by pushing your code to GitHub and submitting the link to Canvas.