Variables and Types

Logistics

- HW00: **Due Wednesday January 25th, 2023 @ 11:59pm ET**
- Recitations start next week
- Office Hours have somewhat started this week, the regular office hour schedule should be out next week

Learning Objectives

- To be familiar with primitive data types
- To be able to write expressions using primitive data types
- To know what a variable is
- To be able to declare variables
- To be able to solve problems using primitive type variables

Overview

- One role of a computer program is to model and manipulate real or imaginary world entities. To do this, the computer must store some **data** to model these entities
- In this module, we will learn how to represent the properties (or attributes) of the entities that our program will manipulate
- Example:
    - Entity: **`student`**
    - Properties: **`name, age, height, etc.`**

Data

- **Data** is a piece of information. We use data to model entities & solve problems

- All data (in Java) has a **data type**
  - Defines the set of possible values a piece of data can have
  - Defines the possible operations that can be performed on that data

- Two types of data types in Java
  - Primitive types
  - Object types

Object types

More on these in a future lecture

- They hold a reference to an object
- **`String`** is an object type and is associated with text (sequence of characters) values (e.g. **`"Andrew"`**)

Primitive types

- **`int`**: stores whole numbers (positive or negative) like **`3`**, **`-5`**, **`19000`**
  - “int” is short for Integer
- **`double`**: stores decimal numbers (positive or negative) like **`3.5`**, **`-5.1`**, **`19000.1`**
  - Note: not infinitely precise. Computers are physical and finite.
- **`boolean`**: stores Boolean values, either **`true`** or **`false`**

There are others we will introduce later

Operations on `int`

Type of operand 1	Operator	Type of operand 2	Type of result	Example	Example Result
`int`	`+`	`int`	`int`	`3 + 5`	`8`
`int`	`-`	`int`	`int`	`4 - 6`	`-2`
`int`	`*`	`int`	`int`	`2 * 3`	`6`
`int`	`/`	`int`	`int` 	`3 / 2`	`1`

The modulo (%) operator

- The mod operator **`(x % y)`** returns the remainder after you divide x (first number) by y (second number)

`5 % 2` ➔ `1`

`4 % 2` ➔ `0`

Operations on `double`

Type of operand 1	Operator	Type of operand 2	Type of result	Example	Example Result
`double`	`+`	`double`	`double`	`3.5 + 5.5`	`9.0`
`double`	`-`	`double`	`double`	`4.0 - 6.0`	`-2.0`
`double`	`*`	`double`	`double`	`2.5 * 1.0`	`2.5`
`double`	`/`	`double`	`double`	`3.0 / 2.0`	`1.5`

Operations on `double` and `int`

- When one of the operand is of type **`double`**, the results is of type **`double`**

Type of operand 1	Operator	Type of operand 2	Type of result	Example	Example Result
`double`	`+`	`int`	`double`	`3.5 + 5`	`8.5`
`int`	`-`	`double`	`double`	`4 - 6.0`	`-2.0`
`double`	`*`	`int`	`double`	`2.5 * 1`	`2.5`
`double`	`/`	`int`	`double`	`3.0 / 2`	`1.5`

Operations on `boolean`

	Type of operand 1	Operator	Type of operand 2	Type of result	Example	Example Result
"and"	`boolean`	`&&`	`boolean`	`boolean`	`true && false`	`false`
"or"	`boolean`	`||`	`boolean`	`boolean`	`true || false`	`true`
"not"	`boolean`	`!`	`N/A`	`boolean`	`!true`	`false`

Comparison: Equality

- The `==` operator is used to check for equality.
- The result is a **`boolean`** value (`true` or `false`)

```
4 == 5;  // evaluates to false
```

- The result of the comparison can be printed

```
System.out.print(4 == 5);    // prints false
```

Comparison: Inequality

- The `!=` operator is used to check for inequality (not equals).
- The result is a **`boolean`** value (**`true`** or **`false`**)

```
4 != 5;  // evaluates to true
```

- The result of the comparison can be printed

```
System.out.print(4 != 5);    // prints true
```

Comparison: Others

- For types like **`int`** and **`double`**, we can also perform other comparisons
- The result is a **`boolean`** value (**`true`** or **`false`**);

Operator Name	Syntax	Example	Example Output
Less than	`<`	`5 < 6`	`true`
Less than or equal to	`<=`	`5 <= 5`	`true`
Greater than	`>`	`2 > 3`	`false`
Greater than or equal to	`>=`	`5 >= 1`	`true`

Operator Chaining & Priority

- You can chain multiple operators together in one line:

```
110 + 120 + 160 + 121 + 240
```

- Sometimes the order of operations is unclear. Example:

```
110 + 120 * 2 == 2
```

- To avoid confusion, use parenthesis to specify the order of operations:

```
(110 + (120 * 2)) == 2
```

Does the same thing as the above without parenthesis.
Parenthesis are still recommended for general use.

Expressions

- A sequence of ***operators*** and their ***operands*** (values to act on) that specifies a computation
- Examples:
  - `1 + 2 + 3`
  - `240 != 240`
  - `(-4 + (4 * 4 - 4 * 1 * 6)) / (2 * 6) >= 0`
  - `3.14 * 6.02 - 1000.00`
  - `!false && true == false`

Live Coding DEMO (Part 1)

LeapYear.java

Program that will determine if a year is a leap year.

Variables

- Variables are a **portion of computer memory** used to store a value (data).
  - Allows us to store data and the result of computations for later usage.
  - A way for the computer to “remember” data.
- Every variable has a **name** that we can use to refer to the variable
- Every variable has a **data type** that defines which data can be stored in that variable

Variable declaration

- Creates a variable
- Associates a variable to a type
- The type determines how space (bits) the computer will use to store the value associated with the variable.
- Examples

```
// declaring the variable name
String name;
// declaring the variable age
int age;
```

Variable initialization

- Assigns a value to a variable: using the **=** sign
- The value and the type of the variable must be compatible
- Examples

```
// declaring and initializing the variable name
String name = "Malcom";
```

variable ← value

Always surround a String value with "

One line: declaration + initialization

```
// declaring the variable age
int age;
age = 14;
```

variable ← value

Two steps: declaration then initialization

```
boolean is_taking_CIS110 = true;
```

Operations on variables

- Assignment statement initializes or changes the value of a variable previously declared
- Operators can be applied to values to perform computation
    - Such as Mathematical operators applied to **`int`** and **`double`** values
- Example:

```
int x = 1100;

x = 2400 + 1400;
```

Variables in Expressions

- Variables can be named in expressions, which will use the value stored in the variable as part of the computation:

Example:

```
int x = 12;
int y = x * 30;  // results in y being 360
int z = 20 + y;  // z equals 380
x = x + 1;       // x equals 13
```

Compound Assignment Operators

- Shortcuts that do a math operation and assignment in one step

+ shortcuts	- shortcuts	* shortcut	/ shortcut	% shortcut
x = x + 1;	x = x - 1;	x = x * 2;	x = x / 2;	x = x % 2;
x += 1;	x -= 1;	x *= 2;	x /= 2;	x %= 2;
x++;	x- -;			

Last row only works for adding/subtracting **1**

Second row only works for operating on the same value being assigned into.

Printing a variable

- Put the variable name without the quotes in the print command

```
String name = "Maya";
System.out.print(name);
```

Prints **Maya**

- Use the + operator to append the value of a variable to a text in the print command

```
System.out.print("Name of the student: " + name);
```

Prints Name of the student: **Maya**

Operator Type Errors

- Sometimes mixing variable types and values will result in compiler errors:

```
// Wrong value for the specified variable type
int pi = 3.14159;
double x = true;

// Using operators with incompatible/mismatching types
int y = 1 + false;
boolean z = 110 && 120;
```

Casting

- Type casting converts a variable from one type to another
- `(int)` is used to cast a value to **`int`**

`(int) 3.5` ➔ `3`

- `(double)` is used to cast a value to **`double`**

`(double) 3/2` ➔ `1.5`

- Casting applies only to the closest operand

`(double) 3/2` is the same as **`((double) 3)/2`** or **`3.0/2`**

Live Coding DEMO (Part 2) w/ Variables!

LeapYear.java

Program that will determine if a year is a leap year.