codecademy

Variables

`boolean` Data Type

In Java, the `boolean` primitive data type is used to store a value, which can be either `true` or `false`.

```
boolean result = true;
boolean isMarried = false;
```

Strings

A String in Java is a Object that holds multiple characters. It is not a primitive datatype.
A String can be created by placing characters between a pair of double quotes (").
To compare Strings, the `equals()` method must be used instead of the primitive equality comparator `==`.

```
// Creating a String variable
String name = "Bob";

// The following will print "false"
because strings are case-sensitive
System.out.println(name.equals("bob"));
```

`int` Data Type

In Java, the `int` datatype is used to store integer values. This means that it can store all positive and negative whole numbers and zero.

```
int num1 = 10;   // positive value
int num2 = -5;   // negative value
int num3 = 0;    // zero value
int num4 = 12.5; // not allowed
```

`char` Data Type

In Java, `char` is used to store a single character. The character must be enclosed in single quotes.

```
char answer = 'y';
```

Primitive Data Types

Java's most basic data types are known as *primitive data types* and are in the system by default.
The available types are as follows:

- `int`
- `char`
- `boolean`
- `byte`
- `long`
- `short`
- `double`
- `float`

`null` is another, but it can only ever store the value `null`.

```
int age = 28;

char grade = 'A';

boolean late = true;

byte b = 20;

long num1 = 1234567;

short no = 10;
```

```
float k = (float)12.5;

double pi = 3.14;
```

Static Typing

In Java, the type of a variable is checked at compile time. This is known as *static typing*. It has the advantage of catching the errors at compile time rather than at execution time.
Variables must be declared with the appropriate data type or the program will not compile.

```
int i = 10;          // type is int
char ch = 'a';       // type is char

j = 20;              // won't compile, no
type is given
char name = "Lil";   // won't compile,
wrong data type
```

final Keyword

The value of a variable cannot be changed if the variable was declared using the `final` keyword. Note that the variable must be given a value when it is declared as `final`. `final` variables cannot be changed; any attempts at doing so will result in an error message.

```
// Value cannot be changed:
final double PI = 3.14;
```

double Data Type

The `double` primitive type is used to hold decimal values.

```
double PI = 3.14;
double price = 5.75;
```

Math Operations

Basic math operations can be applied to `int`, `double` and `float` data types:

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division
- `%` modulo (yields the remainder)

These operations are not supported for other data types.

```
int a = 20;
int b = 10;

int result;

result = a + b;  // 30

result = a - b;  // 10

result = a * b;  // 200

result = a / b;  // 2

result = a % b;  // 0
```

Comparison Operators

Comparison operators can be used to compare two values:

- `>` greater than
- `<` less than
- `>=` greater than or equal to
- `<=` less than or equal to
- `==` equal to
- `!=` not equal to

They are supported for primitive data types and the result of a comparison is a boolean value `true` or `false`.

```
int a = 5;
int b = 3;

boolean result = a > b;
// result now holds the boolean value
true
```

Compound Assignment Operators

Compound assignment operators can be used to change and reassign the value of a variable using one line of code. Compound assignment operators include `+=`, `-=`, `*=`, `/=`, and `%=`.

```
int number = 5;

number += 3; // Value is now 8
number -= 4; // Value is now 4
number *= 6; // Value is now 24
number /= 2; // Value is now 12
number %= 7; // Value is now 5
```

Increment and Decrement Operators

The increment operator, (`++`), can increase the value of a number-based variable by `1` while the decrement operator, (`--`), can decrease the value of a variable by `1`.

```
int numApples = 5;
numApples++; // Value is now 6

int numOranges = 5;
numOranges--; // Value is now 4
```

Order of Operations

The order in which an expression with multiple operators is evaluated is determined by the order of operations: parentheses → multiplication → division → modulo → addition → subtraction.