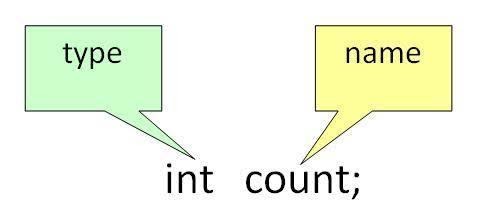
**JAVA WEEK 1 DAY 2 VARIABLES**

**What is a variable?**

* A variable is a name given to a memory location. It is the basic unit of storage in a program.
* The value stored in a variable can be changed during program execution.
* A variable is only a name given to a memory location; all the operations done on the variable effects that memory location.
* In Java, all the variables must be declared before use.

**How to declare variables?**

Variables can be declared in java as follows:



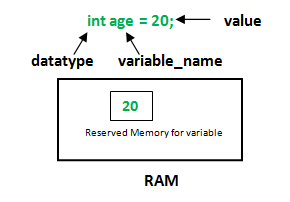
**type:** Type of data that can be stored in this variable.

**name:** Name given to the variable.

In this way, a name can only be given to a memory location. It can be assigned values in two ways:

* Variable Initialization
* Assigning value by taking input

**Declaring and initialize variables**



**datatype:** Type of data that can be stored in this variable.

**variable\_name:** Name given to the variable.

**value:** It is the initial value stored in the variable.

**Declaring variables examples**

**float** simpleInterest; - **Declaring float variable**

**int**myAge = 19; - **Declaring and Initializing integer variable**

**char**firstLetter = 'h'; - **Declaring and Initializing character variable**

**Variable Types**

**There are three types of variables in Java:**

1. **Local Variables:** A variable defined within a block or method or constructor is called local variable.

* These variables are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
* The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variables only within that block.
* Initialization of Local Variable is Mandatory.

2.**Instance Variables:** Instance variables are non-static variables and are declared in a class outside any method, constructor or block.

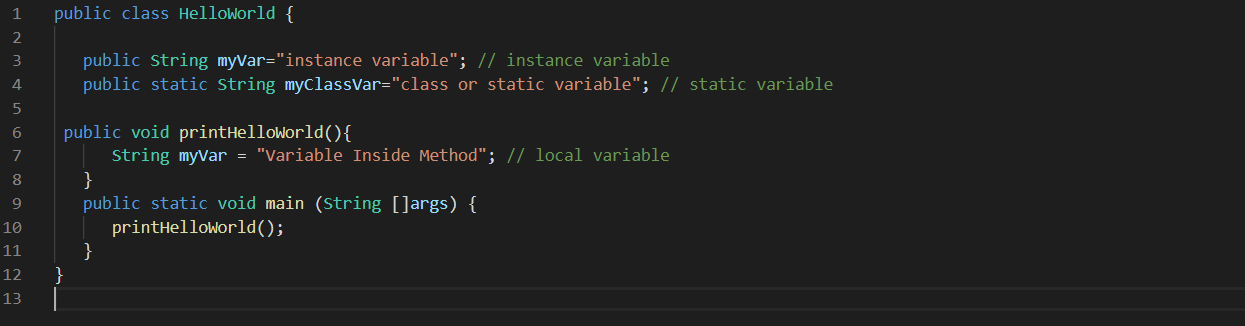
* As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
* Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.
* Initialisation of Instance Variable is not Mandatory. Its default value is 0
* Instance Variable can be accessed only by creating objects.

3. **Static Variables:** Static variables are also known as Class variables.

* These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
* Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
* Static variables are created at the start of program execution and destroyed automatically when execution ends.
* Initialization of Static Variable is not Mandatory. Its default value is 0
* If we access the static variable like Instance variable (through an object), the compiler will show the warning message and it won’t halt the program. The compiler will replace the object name to class name automatically.
* If we access the static variable without the class name, Compiler will automatically append the class name.
* To access static variables, we need not create an object of that class, we can simply access the variable as

**Instance variable Vs Static variable**

* Each object will have its **own copy** of instance variable whereas We can only have **one copy** of a static variable per class irrespective of how many objects we create.
* Changes made in an instance variable using one object will **not be reflected** in other objects as each object has its own copy of instance variable. In case of static, changes will**be reflected** in other objects as static variables are common to all object of a class.
* We can access instance variables through **object references** and Static Variables can be accessed directly using **class name.**



All variables names should follow below general rules:

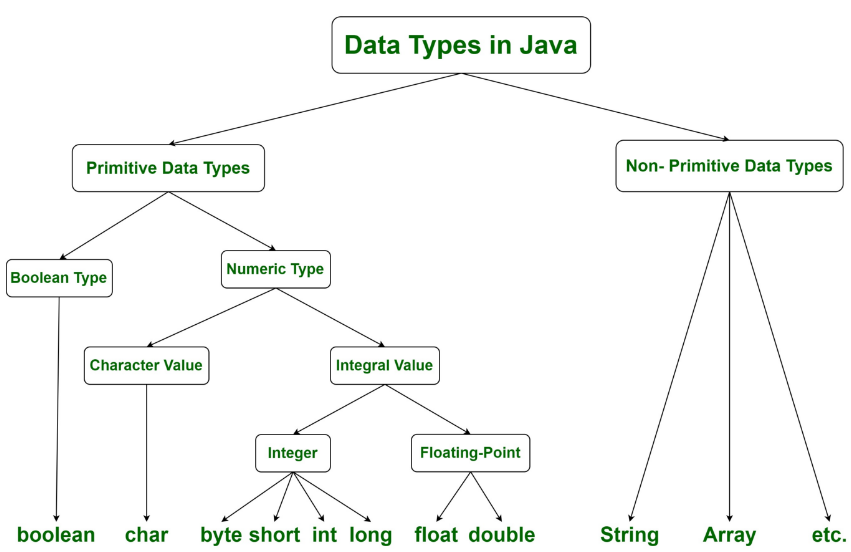
* should begin with a letter (A to Z or a to z), currency character ($) or an underscore (\_).
* After the first character, variables can have any combination of characters.
* key word cannot be used as an identifier
* Most importantly, variables are case sensitive.
* Examples of legal variables: age, $salary, \_value, \_\_1\_value.
* Examples of illegal variables: 123abc, -salary.

**DATA TYPES IN JAVA**

**Data type** defines the values that a variable can take, for example if a variable has **int** data type, it can only take**integer** values. Java is a statically typed language; the data type of a variable is known at compile time. This means that you must specify the type of the variable (Declare the variable) before you can use it. In java we have two categories of data type:

**1) Primitive data types**

**2)  Non-primitive data types – Arrays and Strings are non-primitive data types**, we will discuss:



**The above diagram illustrates how data type classification in Java**

**Primitive data types** **-** A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

|  |  |  |
| --- | --- | --- |
| **Data** **Type** | **Size** | **Description** |
| byte | 1 byte | Stores whole numbers from -128 to 127 |
| short | 2 bytes | Stores whole numbers from -32,768 to 32,767 |
| int | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| boolean | 1 bit | Stores true or false values |
| char | 2 bytes | Stores a single character/letter or ASCII values |

**Non-Primitive data types** **-** These are the datatypes which have instances like objects. Hence, they are called reference variables. They are primarily classes, arrays, strings or interfaces.

**JAVA OPERATORS**

An operator is a character that represents an action, for example - is an arithmetic operator that represents subtraction.

Java operators can be classified into:

* Arithmetic Operators
* Relational Operators
* Bitwise Operators
* Logical Operators
* Assignment Operators
* Misc Operators

#### ****Arithmetic operators****

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators −Assume integer variable A holds 10 and variable B holds 20, then

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + (Addition) | Adds values on either side of the operator. | A + B will give 30 |
| - (Subtraction) | Subtracts right-hand operand from left-hand operand. | A - B will give -10 |
| \* (Multiplication) | Multiplies values on either side of the operator. | A \* B will give 200 |
| / (Division) | Divides left-hand operand by right-hand operand. | B / A will give 2 |
| % (Modulus) | Divides left-hand operand by right-hand operand and returns remainder. | B % A will give 0 |
| ++ (Increment) | Increases the value of operand by 1. | B++ gives 21 |
| -- (Decrement) | Decreases the value of operand by 1. | B-- gives 19 |

**Relational operators –** the table below show the relational operators supported by java.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == (equal to) | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != (not equal to) | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > (greater than) | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < (less than) | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= (greater than or equal to) | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= (less than or equal to) | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

**Bitwise operators –** works on bits and performs bit-by-bit operation, assume if a = 60 and b = 13; now in binary format they will be as follows:

a = 0011 1100

b = 0000 1101

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & (bitwise and) | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) will give 12 which is 0000 1100 |
| | (bitwise or) | Binary OR Operator copies a bit if it exists in either operand. | (A | B) will give 61 which is 0011 1101 |
| ^ (bitwise XOR) | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49 which is 0011 0001 |
| ~ (bitwise compliment) | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number. |
| << (left shift) | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 will give 240 which is 1111 0000 |
| >> (right shift) | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 1111 |
| >>> (zero fill right shift) | Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros. | A >>>2 will give 15 which is 0000 1111 |

**Logical operators -** The following table lists the logical operators −Assume Boolean variables A holds true and variable B holds false, then:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && (logical and) | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false |
| || (logical or) | Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. | (A || B) is true |
| ! (logical not) | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true |

#### ****The Assignment Operator****

Following are the assignment operator supported by Java language −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Simple assignment operator. Assigns values from right side operands to left side operand. | C = A + B will assign value of A + B into C |

**JAVA CLASSES AND OBJECTS**

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

A Class is like an object constructor, or a "blueprint” or a factory for creating objects. This means without a class no object can be created.

#### ****What is a class in Java?****

Class − A class can be defined as a template/blueprint that describes the behaviour/state that the object of its type support.

**Creating a class**

To create a class, use the keyword class

A class is made up of:

**Constructor**

A constructor is a method called when creating an object from a class e.g public Person()in the example. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class. Some constructors may accept parameter and some may not.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

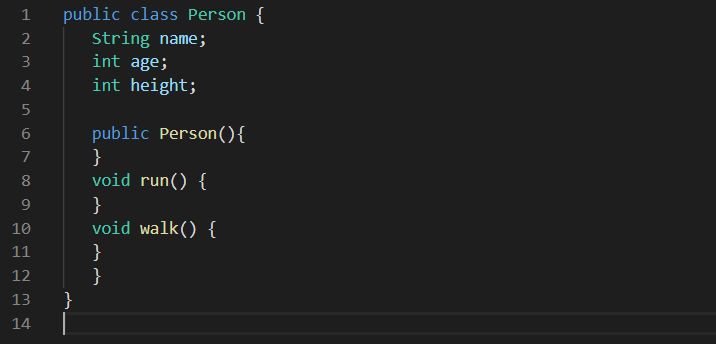
**Methods**

A class can have any number of methods to access the value of various kinds of methods. In the example given below run, walk and sleep are methods

**Variables/Properties**

Usually these are attributes the describe objects that a created from that class. In the below example name, age and height are variables

**Example of class:**



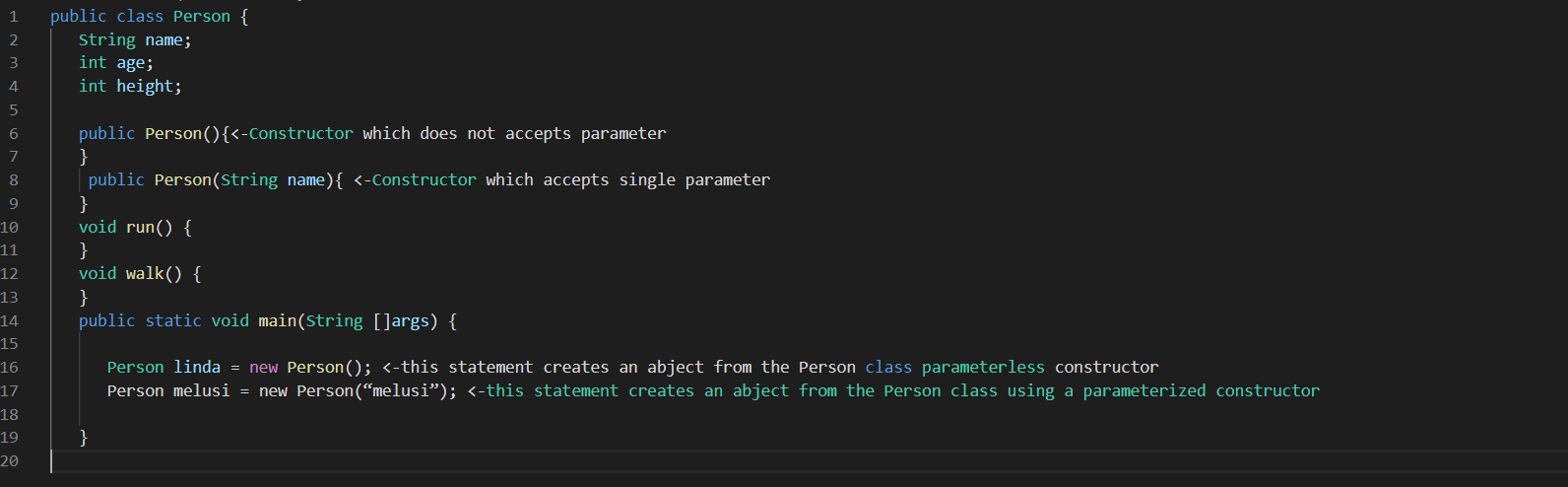
#### ****What is an Object in Java?****

Object is an instance of a class, i.e. they are created from classes. Objects have states/attributes and behaviours. Example: A dog has states - colour, name, breed as well as behaviours – wagging the tail, barking, eating.

**Creating an object**

In Java, an object is created from a class. We have already created the class named Person, so now we can use this to create objects.

To create an object of Person, specify the class name, followed by the object name, and use the keyword new:  and call the constructor to create the object for us ,as shown below:



### Daily Notes - Java Classes and Objects