Java Programming

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

Java is;

- **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

History of Java

James Gosling initiated Java language project in June 1991 for use in one of his many settop box projects. The language, initially called ‘Oak’ after an oak tree that

stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised Write Once, Run Anywhere (WORA), providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May, 2007, Sun finished the process, making all of Java's core code free and opensource, aside from a small portion of code to which Sun did not hold the copyright.

Local Environment Setup

Java Basic Syntax

When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods, and instance variables mean.

- **Object** - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.
- **Class** - A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.
- **Methods** - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** - Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Basic Syntax

i. Case Sensitivity - Java is case sensitive, which means identifier Helloand hello would have different meaning in Java.

ii. Class Names - For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

Example: class MyFirstJavaClass

iii. Method Names - All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

Example: public void myMethodName()

iv. Program File Name - Name of the program file should exactly match the class name. When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'

v. public static void main(String args[]) - Java program processing starts from the main() method which is a mandatory part of every Java program.

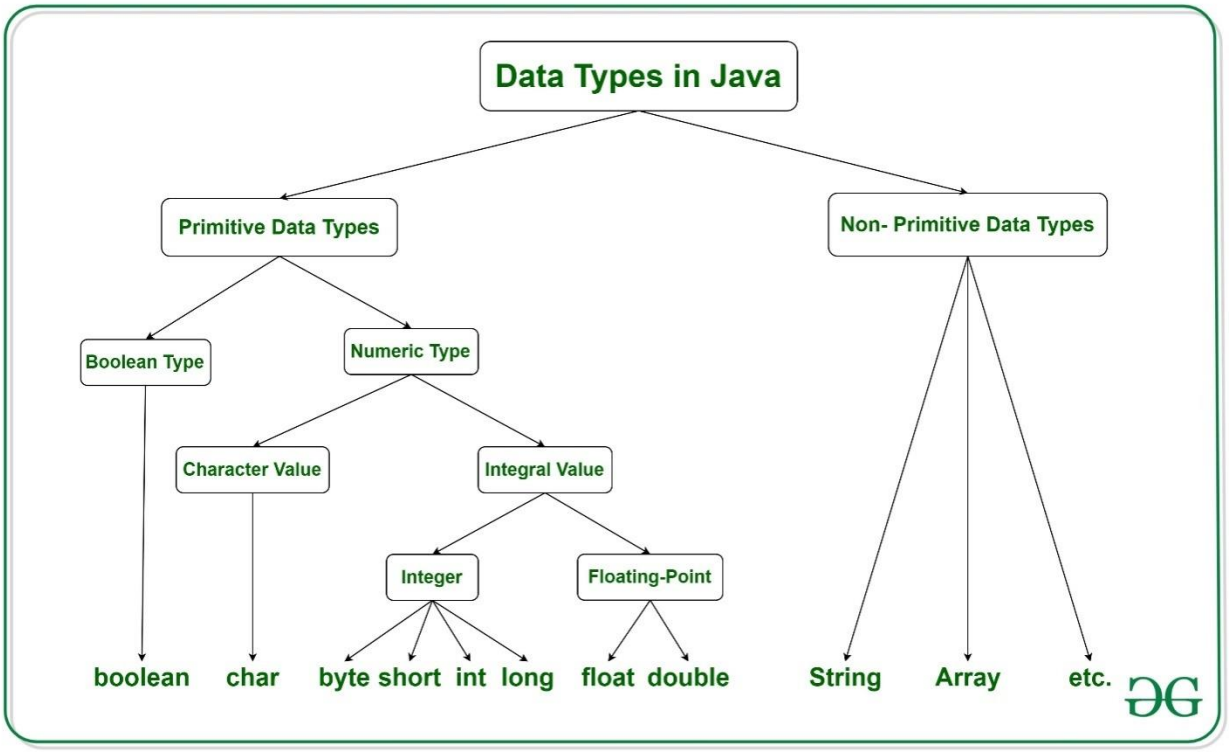
Datatypes in Java

Data types are different sizes and values that can be stored in the variable that is made as per convenience and circumstances to cover up all test cases. There are majorly two types of languages that are as follows:

1. First, one is a **Statically typed language** where each variable and expression type is already known at compile time. Once a variable is declared to be of a certain data type, it cannot hold values of other data types. For example C, C++, Java.
2. The other is **Dynamically typed languages.** These languages can receive different data types over time. For example Ruby, Python

Java is **statically typed and also a strongly typed language** because, in Java, each type of data (such as integer, character, hexadecimal, packed

decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.



Java has two categories in which data types are segregated

1. **Primitive Data Type:** such as Boolean, char, int, short, byte, long, float, and double
2. **Non-Primitive Data Type or Object Data type:** such as String, Array, etc.

Types of Primitive Data Types

Primitive data are only single values and have no special capabilities. There are **8 primitive data types.** They are depicted below in tabular format below as follows:

TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\','\n',' β'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

Non-Primitive Data Type or Reference Data Types

The **Reference Data Types** will contain a memory address of variable values because the reference types won't store the variable value directly in memory. They are strings, objects, arrays, etc.

A: Strings

Strings are defined as an array of characters. The difference between a character array and a string in Java is, that the string is designed to hold a sequence of characters in a single variable whereas, a character array is a collection of separate char type entities. Unlike C/C++, Java strings are not terminated with a null character.

Syntax: Declaring a string

```
<String Type><string variable> = "<sequence_of_string>";
```

Example:

```
// Declare String without using new operator

String s = "GeeksforGeeks";


// declare String using new operator
```

```
String s1 = new String ("GeeksforGeeks");
```

B: Class

A class is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers**: A class can be public or has default access. Refer to access specifies for classes or interfaces in Java
2. **Class name:** The name should begin with an initial letter (capitalized by convention).
3. **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. **Body:** The class body is surrounded by braces, { }.

C: Object

It is a basic unit of Object-Oriented Programming and represents real-life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

1. **State**: It is represented by the attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by the methods of an object. It also reflects the response of an object to other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

JAVA IDENTIFIERS

All Java components require names. Names used for classes, variables, and methods are called **identifiers**.

In Java, there are several points to remember about identifiers. They are as follows:

All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).

i. After the first character, identifiers can have any combination of characters.

ii. A key word OR reserved words cannot be used as an identifier.

For example “int while = 20;” is an invalid statement as while is a reserved word. There are **53** reserved words in Java.

iii. Most importantly, identifiers are case sensitive.

Examples of valid identifiers:

```
MyVariable
MYVARIABLE
myvariable
x
i
x1
i1
_myvariable
$myvariable
sum_of_array
geeks123
```

Examples of invalid identifiers:

```
 My Variable          // contains a space
123geeks             // Begins with a digit
a+c                  // plus sign is not an alphanumeric character
variable-2           // hyphen is not an alphanumeric character
sum_&_difference     // ampersand is not an alphanumeric character
```

In programming languages, identifiers are used for identification purposes. In Java, an identifier can be a class name, method name, variable name, or label. For example:

```
public class Test{

    public static void main(String[] args){

        int a = 20;

    }

}
```

In the above java code, we have 5 identifiers namely :

- **Test**: class name.
- **main**: method name.
- **String**: predefined class name.
- **args**: variable name.
- **a**: variable name.

OPERATORS IN JAVA

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are:

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators
6. Ternary Operator

1. Arithmetic Operators: They are used to perform simple arithmetic operations on primitive data types.

- *** :** Multiplication
- **/ :** Division
- **% :** Modulo
- **+ :** Addition
- **– :** Subtraction

2. Unary Operators: Unary operators need only one operand. They are used to increment, decrement or negate a value.

- **–: Unary minus**, used for negating the values.
- **+: Unary plus** indicates the positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is the byte, char, or short. This is called unary numeric promotion.
- **++: Increment operator**, used for incrementing the value by 1. There are two varieties of increment operators.
    - **Post-Increment:** Value is first used for computing the result and then incremented.
    - **Pre-Increment:** Value is incremented first, and then the result is computed.
- **—: Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operators.

- **Post-decrement:** Value is first used for computing the result and then decremented.
- **Pre-Decrement:** Value is decremented first, and then the result is computed.

- **! : Logical not operator**, used for inverting a Boolean value.
-

3. Assignment Operator: '=' Assignment operator is used to assign a value to any variable. It has a right to left associativity, i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.
The general format of the assignment operator is:

```
variable = value;
```

In many cases, the assignment operator can be combined with other operators to build a shorter version of the statement called a **Compound Statement**. For example, instead of a **=** a+5, we can write a **+=** 5.

- **+=**, for adding left operand with right operand and then assigning it to the variable on the left.
- **-=**, for subtracting right operand from left operand and then assigning it to the variable on the left.
- ***=**, for multiplying left operand with right operand and then assigning it to the variable on the left.
- **/=**, for dividing left operand by right operand and then assigning it to the variable on the left.
- **%=**, for assigning modulo of left operand by right operand and then assigning it to the variable on the left.
-

4. Relational Operators: These operators are used to check for relations like equality, greater than, less than. They return Boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements. The general format is,

```
variable relation operator value
```

Some of the relational operators are-

- **==, Equal to:** returns true if the left-hand side is equal to the right-hand side.
- **!=, Not Equal to:** returns true if the left-hand side is not equal to the right-hand side.
- **<, less than:** returns true if the left-hand side is less than the right-hand side.
- **<=, less than or equal to** returns true if the left-hand side is less than or equal to the right-hand side.

- **>, Greater than:** returns true if the left-hand side is greater than the right-hand side.
- **>=, Greater than or equal to:** returns true if the left-hand side is greater than or equal to the right-hand side.

5. Logical Operators: These operators are used to perform "logical AND" and "logical OR" operations, i.e., the function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e., it has a short-circuiting effect. Used extensively to test for several conditions for making a decision. Java also have "Logical NOT", it returns true when condition is false and vice-versa

Conditional operators are:

- **&&, Logical AND:** returns true when both conditions are true.
- **||, Logical OR:** returns true if at least one condition is true.
- **! , Logical NOT:** returns true when condition is false and vice-versa

6. Ternary operator: Ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name ternary.
The general format is:

```
condition ? if true : if false
```

The above statement means that if the condition evaluates to true, then execute the statements after the '?' else execute the statements after the ':.'

```
// Java program to illustrate
// max of three numbers using
// ternary operator.
publicclassoperators {
    publicstaticvoidmain(String[] args)
    {
        inta = 20, b = 10, c = 30, result;

        // result holds max of three
        // numbers
        result
            = ((a > b) ? (a > c) ? a : c : (b > c) ? b : c);
        System.out.println("Max of three numbers = "
                           + result);
    }
}
```

Output:

```
Max of three numbers = 30
```

JAVA MODIFIERS

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers:

i. Access Modifiers: default, public , protected, private

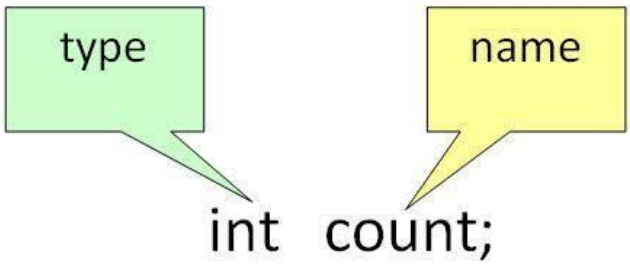
ii. Non-access Modifiers: final, abstract, strictfp

JAVA VARIABLES

Variable in Java is a data container that saves the data values during Java program execution. Every variable is assigned a data type that designates the type and quantity of value it can hold. Variable is a memory location name of the data.
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?



From the image, it can be easily perceived that while declaring a variable, we need to take care of two things that are:

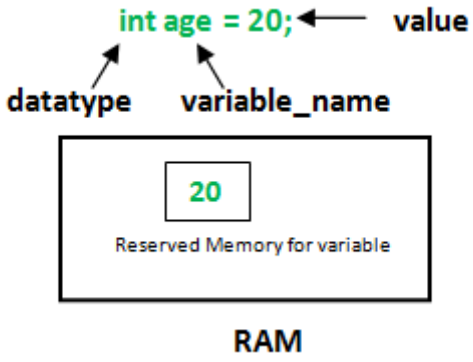
1. Datatype: Type of data that can be stored in this variable.
2. Dataname: Name was given to the variable.
In this way, a name can only be given to a memory location. It can be assigned values in two ways:

- Variable Initialization
- Assigning value by taking input

How to initialize variables?

It can be perceived with the help of 3 components that are as follows:

- **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.



Illustrations:

```
float simpleInterest;

// Declaring float variable
```

```
int time = 10, speed = 20;

// Declaring and Initializing integer variable
```

```
char var = 'h';

// Declaring and Initializing character
variable
```

Following are the types of variables in Java:

i. Local Variables

A variable defined within a block or method or constructor is called a local variable.

- These variables are created when the block is 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 the local variable is mandatory before using it in the defined scope.

Example

```
importjava.io.*;

classGFG {
    publicstaticvoidmain(String[] args)
    {
        intvar = 10; // Declared a Local Variable
        // This variable is local to this main method only
        System.out.println("Local Variable: "+ var);
    }
}
```

Output

```
Local Variable: 10
```

ii. Class Variables (Static Variables)

Class variables are variables declared within a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

```
public class Dog{
    String breed;
    int age;
    String color;
    void barking(){
    }
    void hungry(){
    }
    void sleeping(){
    }
}
```

```
importjava.io.*;

classGFG {

  publicstaticString geek = "Shubham Jain";   //Declared static variable

    publicstaticvoidmain (String[] args) {

      //geek variable can be accessed withod object creation
      //Displaying O/P
      //GFG.geek --> using the static variable
        System.out.println("Geek Name is : "+GFG.geek);
    }
}
```

iii. Instance Variables (Non-static Variables)

Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

Example.

```
importjava.io.*;

classGFG {

    publicString geek; // Declared Instance Variable

    publicGFG(){ // Default Constructor

        this.geek = "Shubham Jain"; // initializing Instance Variable
    }
//Main Method
    publicstaticvoidmain(String[] args){

        // Object Creation
        GFG name = newGFG();
        // Displaying O/P
        System.out.println("Geek name is: "+ name.geek);
    }
}
```

JAVA KEYWORDS OR RESERVED WORDS

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

abstract assert boolean break byte case catch char ass 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 strictfpsuper switch synchronized this throwthrows
transient try void volatile while.

Comments in Java

Java supports single-line and multi-line comments very similar to C and C++. All characters available inside any comment are ignored by Java compiler.

```
public class MyFirstJavaProgram{
    /* This is my first java program.
    * This will print 'Hello World' as the output
    * This is an example of multi-line comments.
    */
    public static void main(String []args){
        // This is an example of single line comment
        /* This is also an example of single line comment. */
        System.out.println("Hello World");
    }
}
```

JAVA – OBJECTS & CLASSES

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Parsing

In this chapter, we will look into the concepts - Classes and Objects.

- Object - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.
- Class - A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

Objects in Java

- Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.
- If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.
- If you compare the software object with a real-world object, they have very similar characteristics.

- Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.
- So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Constructors

When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class. 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.

Following is an example of a constructor:

```
public class Puppy{
public Puppy(){
}

public Puppy(String name){
// This constructor has one parameter, name.
}
}
```

Creating an Object

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class:

- Declaration: A variable declaration with a variable name with an object type.
- Instantiation: The 'new' keyword is used to create the object.
- Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Following is an example of creating an object:

```
public class Puppy{
        public Puppy(String name){
                // This constructor has one parameter, name.
                System.out.println("Passed Name is :" + name );
        }
        public static void main(String []args){
                // Following statement would create an object myPuppy
                Puppy myPuppy = new Puppy( "tommy" );
        }
}
```

If we compile and run the above program, then it will produce the following result:

```
Passed Name is :tommy
```

AccessingInstance Variables and Methods

Instance variables and methods are accessed via created objects. To access an instance variable, following is the fully qualified path:

```
/*First create an object */
ObjectReference = new Constructor();
/* Now call a variable as follows */
ObjectReference.variableName;
/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

Example:

This example explains how to access instance variables and methods of a class.

```
public class Puppy{
int puppyAge;

public Puppy(String name){
// This constructor has one parameter, name.
System.out.println("Name chosen is :" + name );
}
public void setAge( int age ){
puppyAge = age;
}
public int getAge( ){
System.out.println("Puppy's age is :" + puppyAge );
return puppyAge;
}
public static void main(String []args){
/* Object creation */
Puppy myPuppy = new Puppy( "tommy" );
```

```
/* Call class method to set puppy's age */
myPuppy.setAge( 2 );
/* Call another class method to get puppy's age */
myPuppy.getAge( );

/* You can access instance variable as follows as well */
System.out.println("Variable Value :" + myPuppy.puppyAge );
}
}
```

If we compile and run the above program, then it will produce the following result:

```
Name chosen is :tommy
Puppy's age is :2
Variable Value :2
```