Java Basics

- Java is one of the most popular and widely used programming language and platform.
- A platform is an environment that helps to develop and run programs written in any programming language.
- Java is fast, reliable and secure.
- Java is used in every nook and corner from desktop to web applications, scientific supercomputers to gaming consoles, cell phones to the Internet.

Java Environment

JDK (Java Development Kit):

- Intended for software developers.
- Includes development tools such as the Java compiler, Javadoc, Jar and a debugger.

JRE (Java Runtime Environment):

- JRE contains the parts of the Java libraries required to run Java programs.
- JRE can be view as a subset of JDK.
- Intended for end users.

JVM (JVM (Java Virtual Machine):

- It is an abstract machine.
- It is a specification that provides runtime environment in which java bytecode can be executed.
- JVMs are available for many hardware and software platforms.

Java Basic Syntax

Simplest HelloWorld Program.

```
class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello, World");
    }
}
```

1. Class definition:

- Below line uses the keyword class to declare that a new class is being defined.

```
class HelloWorld
```

- HelloWorld is an identifier that is the name of the class.
- The entire class definition, including all of its members, will be between the braces {} .

2. main() method:

- In Java programming language, every application must contain a `main` method whose signature is:

```
public static void main(String[] args)
```

- **public:** So that JVM can execute the method from anywhere.
- **static:** Main method is to be called without object.
- The modifiers public and static can be written in either order.
- **void:** The main method doesn't return anything.
- **main():** Name configured in the JVM.
- **String[]:** The main method accepts a single argument: an array of elements of type String.

Note: Like in C/C++, main method is the entry point of application and will subsequently invoke all the other methods required by program.

3. println() method:

- This line outputs the string "Hello, World" followed by a new line on the screen.

```
System.out.println("Hello, World");
```

- Output is actually accomplished by the ***built-in println( )*** method.

- **System** is a predefined class that provides access to the system.
- **out** is the variable of type output stream that is connected to the console.

Notes:

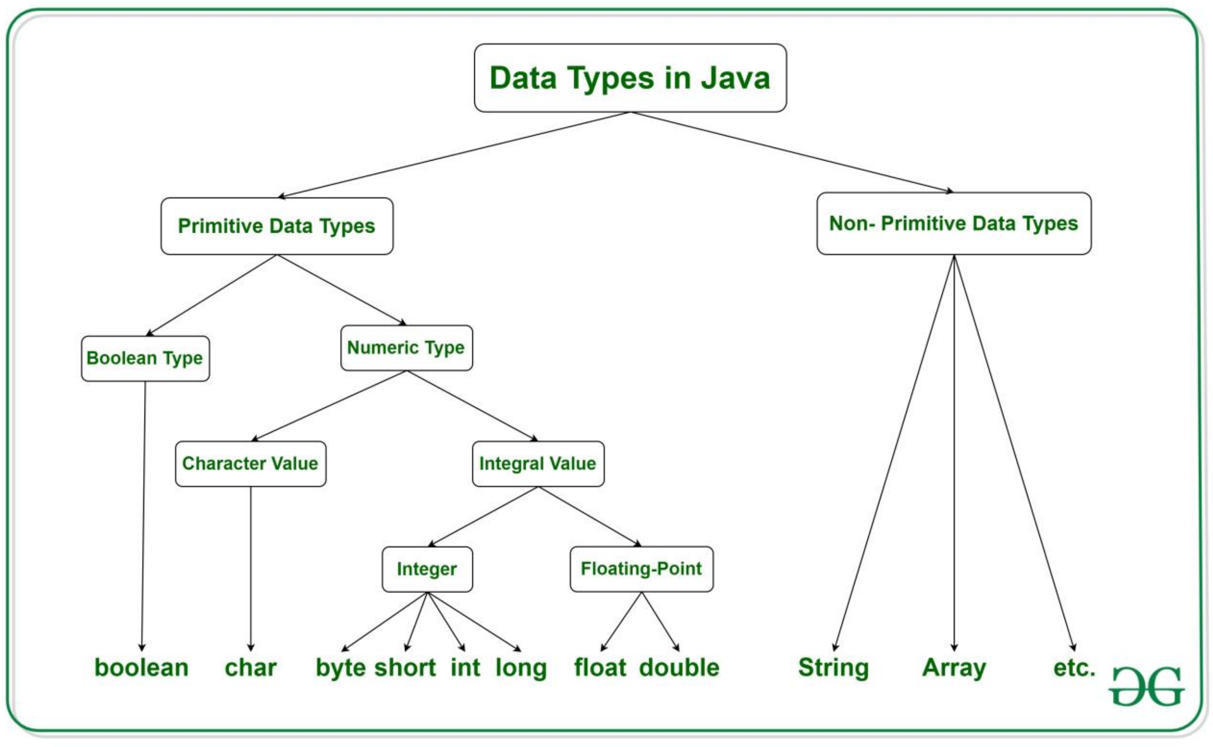
- The name of the class defined by the program HelloWorld should be same as name of file(HelloWorld.java).
- In Java, all codes must reside inside a class and there is at most one public class which contain main() method.
- By convention, the name of the main class(class which contain main method) should match the name of the file that holds the program.

Java Comments

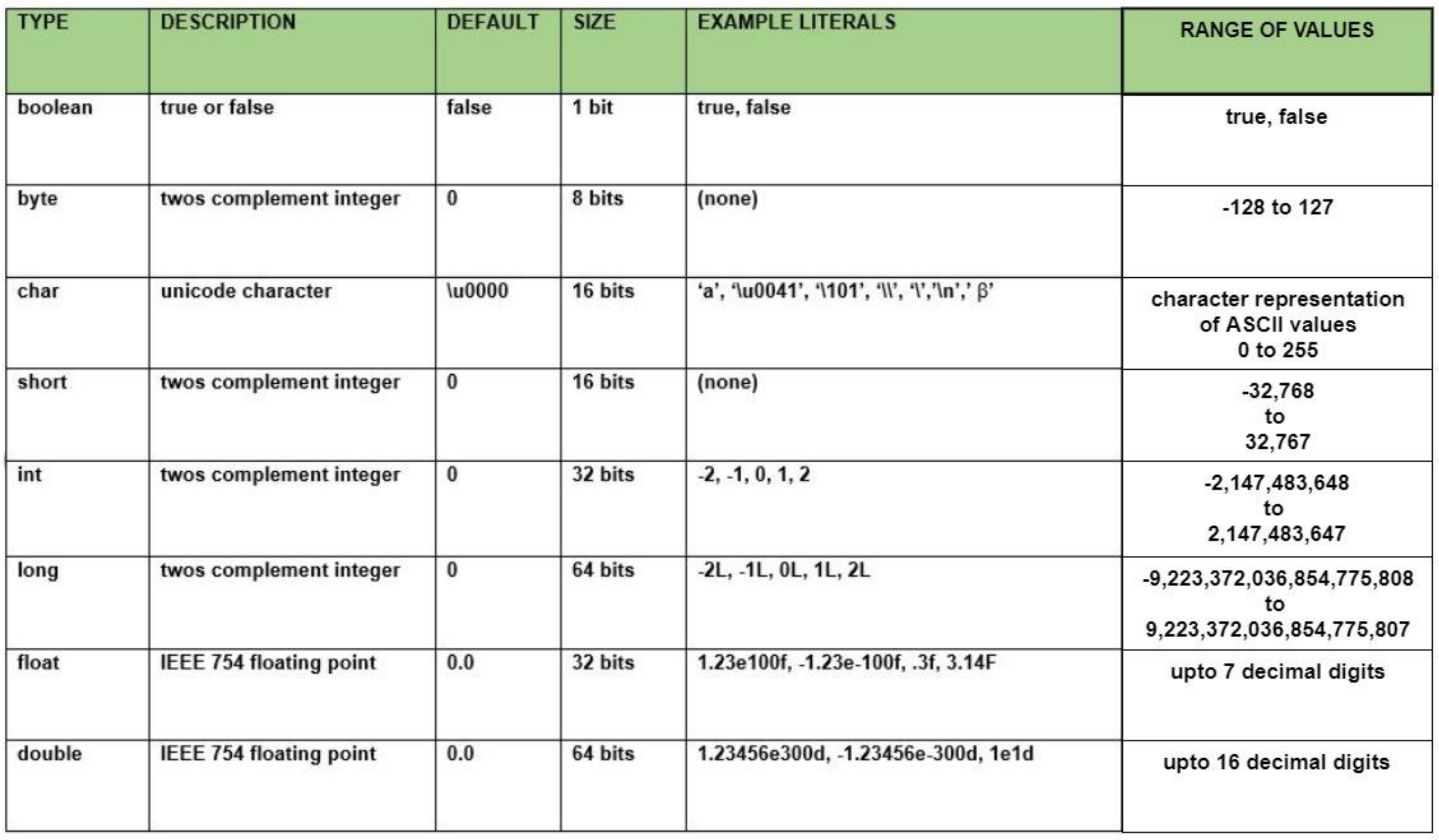
- In a program, comments take part in making the program become more human readable.
- By placing the detail of code involved and proper use of comments makes maintenance easier and finding bugs easily.
- Comments are ignored by the compiler while compiling a code.
- In Java there are **3 types of comments**:
    1. ***Single – line*** comments.
    2. ***Multi – line*** comments.
    3. ***Documentation*** comments.

Java Data Types

- There are majorly two types of languages.
    - **Statically typed language:**
        - 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.
        - ***Example:*** C, C++, Java.
    - **Dynamically typed languages:**
        - These languages can receive different data types over time.
        - ***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.



Primitive Data Types



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

Notes:

- In Java SE 8 and later, we can use the int data type to represent an unsigned 32-bit integer, which has value in the range [0, 2^{32}-1]. Use the Integer class to use int data type as an unsigned integer.
- As above we can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of 2^{64}-1. The Long class also contains methods like compareUnsigned, divideUnsigned etc to support arithmetic operations for unsigned long.
- Both float and double data types were designed especially for scientific calculations, where approximation errors are acceptable. If accuracy is the most prior concern then, recommended not to use them and use ***BigDecimal*** class instead. For details: ***Rounding off errors in Java***.

Non-Primitive / Reference Data Types

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

Strings:

- ***Strings*** are defined as an array of characters.
- The difference between a character array and a string is the string is terminated with a special character **‘\0’**.

Basic Syntax:
```
<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");
```

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:
  i. **Modifiers:** A class can be public or has default access (Refer this for details).
  ii. **Class name:** The name should begin with a initial letter (capitalized by convention).
  iii. **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.
  iv. **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.
  v. **Body:** The class body surrounded by braces, { }.

Object:

- **Object**is a basic unit of Object-Oriented Programming and represents the real-life entities.
- A typical Java program creates many objects, which as we know, interact by invoking methods.
- An object consists of :
    i. **State:** It is represented by ***attributes*** of an object. It also reflects the ***properties*** of an object.
    ii. **Behavior:** It is represented by ***methods*** of an object. It also reflects the ***response*** of an object with other objects.
    iii. **Identity** : It gives a ***unique name*** to an object and enables one object to ***interact*** with other objects.

Interface:

- Like a class, an **Interface** can have methods and variables.
- But the methods declared in an interface are by default ***abstract*** (only method signature, no body).
- Interfaces specify ***what a class must do and not how***. It is the blueprint of the class.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
- A Java library example is, ***Comparator Interface***, if a class implements this interface, then it can be used to sort a collection.

Array:

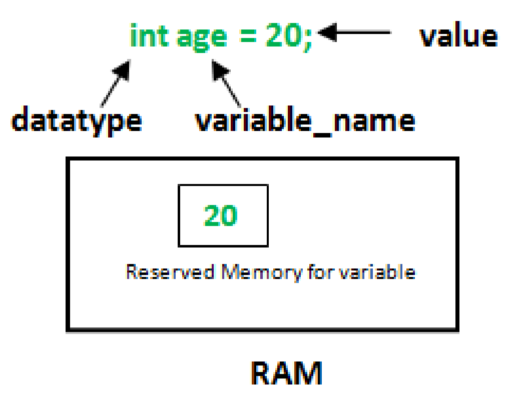
- ***Array*** is a ***group of like-typed variables*** that are referred to by a common name.
- Arrays in Java work differently than they do in C/C++.
- Following are some important point about Java arrays.
    - In Java all arrays are dynamically allocated. (discussed below).
    - Since arrays are objects in Java, we can find their length using member length.
    - This is different from C/C++ where we find length using sizeof.
    - A Java array variable can also be declared like other variables with [] after the data type.
    - The variables in the array are ordered and have an index beginning from 0.
    - Java array can be also be used as a static field, a local variable or a method parameter.
    - The **size** of an array must be specified by an int value and not long or short.
    - The direct superclass of an array type is ***Object***.
    - Every array type implements the interfaces ***Cloneable*** and ***java.io.Serializable***.

Java Variables

What is a Variable ?

- A variable is a name given to a memory location and is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution.
- It 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 ?



- **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.

Examples:
```
float simpleInterest; //Declaring float variable
int time = 10, speed = 20; //Declaring and Initializing integer variable
char var = 'h'; // Declaring and Initializing character variable
```

Types of Variables

1. Local Variables:

- A variable defined within a block or method or constructor is called local variable.
- These variable 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 variable only within that block.
- ***Initilisation*** of Local Variable is ***Mandatory***.

Example:
```
public class StudentDetails {
    public void StudentAge() {
        // local variable age
        int age = 0;
        age = age + 5;
        System.out.println("Student age is : " + age);
    }

    public static void main(String args[]) {
        StudentDetails obj = new StudentDetails();
        obj.StudentAge();
    }
}
```
Output:
```
Student age is : 5
```
Here, the variable age is a local variable to the function StudentAge(). If we use the variable age outside StudentAge() function, the compiler will produce an error

2. Instance Variables:

- Instance variables are ***non-static variables*** and are ***declared in a class outside any method, constructor or block***.
- 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.
- ***Initilisation*** of Instance Variable is ***NOT Mandatory***. Its default value is 0.
- Instance Variable can be accessed only by creating objects.
- In case we have multiple objects as in the below program, ***each object will have its own copies of instance variables***.

Examples:
```
import java.io.*;
class Marks {
    // These variables are instance variables, they are in a class and are not inside any function
    int engMarks;
    int mathsMarks;
}

class MarksDemo {
    public static void main(String args[]) {
        // first object
        Marks obj1 = new Marks();
        obj1.engMarks = 50;
        obj1.mathsMarks = 80;

        // second object
        Marks obj2 = new Marks();
        obj2.engMarks = 80;
        obj2.mathsMarks = 60;

        // displaying marks for first object
        System.out.println("Marks for first object:");
        System.out.println(obj1.engMarks);
        System.out.println(obj1.mathsMarks);

        // displaying marks for second object
        System.out.println("Marks for second object:");
        System.out.println(obj2.engMarks);
        System.out.println(obj2.mathsMarks);
    }
}
```
Output:
```
Marks for first object:
50
80
Marks for second object:
80
60
```

3. Static Variables

- Static variables are also Class variables and 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.
- ***Initilisation*** 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.
- No need to create an object of that class too access static variables, can simply access as `class_name.variable_name;`

Example:

```
import java.io.*;
class Emp {
  // static variable salary
  public static double salary;
  public static String name = "Harsh";
}

public class EmpDemo {
  public static void main(String args[]) {
    // accessing static variable without object
    Emp.salary = 1000;
    System.out.println(Emp.name + "'s average salary:"+ Emp.salary);
  }
}
```

Output:

```
Harsh's average salary:1000.0
```

Java Keywords

What are Java Keywords ?

- ***Reserved words*** in a language that are used for some internal process or represent some predefined actions.
- These words are therefore not allowed to use as a variable names or objects, using these will result into a ***compile time error***.

List of reserved words or keywords

1. **abstract** -Specifies that a class or method will be implemented later, in a subclass
2. **assert** -Assert describes a predicate (a true–false statement) placed in a Java program to indicate that the developer thinks that the predicate is always true at that place. If an assertion evaluates to false at run-time, an assertion failure results, which typically causes execution to abort.
3. **boolean –** A data type that can hold True and False values only
4. **break –** A control statement for breaking out of loops
5. **byte –** A data type that can hold 8-bit data values
6. **case –** Used in switch statements to mark blocks of text
7. **catch –** Catches exceptions generated by try statements
8. **char –** A data type that can hold unsigned 16-bit Unicode characters
9. **class** -Declares a new class
10. **continue** -Sends control back outside a loop
11. **default** -Specifies the default block of code in a switch statement
12. **do** -Starts a do-while loop
13. **double –** A data type that can hold 64-bit floating-point numbers
14. **else –** Indicates alternative branches in an if statement
15. **enum –** A Java keyword used to declare an enumerated type. Enumerations extend the base class.
16. **extends** -Indicates that a class is derived from another class or interface
17. **final** -Indicates that a variable holds a constant value or that a method will not be overridden
18. **finally** -Indicates a block of code in a try-catch structure that will always be executed
19. **float** -A data type that holds a 32-bit floating-point number
20. **for** -Used to start a for loop
21. **if** -Tests a true/false expression and branches accordingly
22. **implements** -Specifies that a class implements an interface
23. **import** -References other classes
24. **instanceof** -Indicates whether an object is an instance of a specific class or implements an interface
25. **int –** A data type that can hold a 32-bit signed integer
26. **interface –** Declares an interface
27. **long –** A data type that holds a 64-bit integer

28. **native** -Specifies that a method is implemented with native (platform-specific) code
29. **new** – Creates new objects
30. **null** -Indicates that a reference does not refer to anything
31. **package** – Declares a Java package
32. **private** -An access specifier indicating that a method or variable may be accessed only in the class it's declared in
33. **protected** – An access specifier indicating that a method or variable may only be accessed in the class it's declared in (or a subclass of the class it's declared in or other classes in the same package)
34. **public** – An access specifier used for classes, interfaces, methods, and variables indicating that an item is accessible throughout the application (or where the class that defines it is accessible)
35. **return** -Sends control and possibly a return value back from a called method
36. **short** – A data type that can hold a 16-bit integer
37. **static** -Indicates that a variable or method is a class method (rather than being limited to one particular object)
38. **strictfp** – A Java keyword used to restrict the precision and rounding of floating point calculations to ensure portability.
39. **super** – Refers to a class's base class (used in a method or class constructor)
40. **switch** -A statement that executes code based on a test value
41. **synchronized** -Specifies critical sections or methods in multithreaded code
42. **this** -Refers to the current object in a method or constructor
43. **throw** – Creates an exception
44. **throws** -Indicates what exceptions may be thrown by a method
45. **transient** -Specifies that a variable is not part of an object's persistent state
46. **try** -Starts a block of code that will be tested for exceptions
47. **void** -Specifies that a method does not have a return value
48. **volatile** -Indicates that a variable may change asynchronously
49. **while** -Starts a while loop

Reserved for future

Some keywords are reserved, even they are not currently in use.

- **const** -Reserved for future use
- **goto** – Reserved for future use

Literals

They look like keywords, but in actual they are **literals** and still can't be used as identifiers in a program

- **true**
- **false**
- **null**

Java Operators

What are Java Operators ?

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

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:** used for giving positive values. Only used when deliberately converting a negative value to positive.
- **++ Increment operator:** used for incrementing the value by 1. There are two varieties of increment operator.
  - **Post-Increment :** Value is first used for computing the result and then incremented.
  - **Pre-Increment :** Value is incremented first and then result is computed.
- **— Decrement operator:** used for decrementing the value by 1. There are two varieties of decrement operator.

    - **Post-decrement :** Value is first used for computing the result and then decremented.
    - **Pre-Decrement :** Value is decremented first and then 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 right hand side of 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.
- General format of assignment operator is,

```
variable = value;
```

- In many cases assignment operator can be combined with other operators to build a shorter version of statement called **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 variable on the left.
    - **-=** for subtracting left operand with right operand and then assigning it to variable on the left.
    - ***=** for multiplying left operand with right operand and then assigning it to variable on the left.
    - **/=** for dividing left operand with right operand and then assigning it to variable on the left.
    - **%=** for assigning modulo of left operand with right operand and then assigning it to variable on the left.

4. Relational Operators :

- These operators are used to check for relations like equality, greater than, less than.
- They return boolean result after the comparison and are extensively used in looping statements as well as conditional if else statements.
- General format is:

```
variable relation_operator value
```

- Some of the relational operators are:
    - **== Equal to :** returns true of left hand side is equal to right hand side.
    - **!= Not Equal to :** returns true of left hand side is not equal to right hand side.
    - **< less than :** returns true of left hand side is less than right hand side.
    - **<= less than or equal to :** returns true of left hand side is less than or equal to right hand side.
    - **> Greater than :** returns true of left hand side is greater than right hand side.
    - **>= Greater than or equal to :** returns true of left hand side is greater than or equal to right hand side.

5. Logical Operators :

- These operators are used to perform “logical AND” and “logical OR” operation.
- 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.
- Conditional operators are:
    - **&& Logical AND :** returns true when both conditions are true.
    - **|| Logical OR :** returns true if at least one condition is true.

6. Ternary operator :

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

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

7. Bitwise Operators :

- These operators are used to perform manipulation of individual bits of a number.
- They can be used with any of the integer types.
- They are used when performing update and query operations of Binary indexed tree.
    - **& Bitwise AND operator :** returns bit by bit AND of input values.
    - **| Bitwise OR operator :** returns bit by bit OR of input values.
    - **^ Bitwise XOR operator :** returns bit by bit XOR of input values.
    - **~ Bitwise Complement Operator :** This is a unary operator which returns the one’s compliment representation of the input value, i.e. with all bits inversed.

8. Shift Operators :

- These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively.
- They can be used when we have to multiply or divide a number by two.
    - **« Left shift operator:** shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.
    - **» Signed Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of initial number. Similar effect as of dividing the number with some power of two.
    - **»> Unsigned Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.
- General Format:

```
 number shift_op number_of_places_to_shift;
```

9. Instance of operator

- Used for type checking. It can be used to test if an object is an instance of a class, a subclass or an interface.
- General format :

```
object instance of class/subclass/interface
```

Example:

```
// Java program to illustrate instance of operator
class Operators {
    public static void main(String[] args) {
        Person obj1 = new Person();
        Person obj2 = new Boy();

        // As obj is of type person, it is not an instance of Boy or interface
        System.out.println("obj1 instanceof Person: " + (obj1 instanceof Person));
        System.out.println("obj1 instanceof Boy: " + (obj1 instanceof Boy));
        System.out.println("obj1 instanceof MyInterface: " + (obj1 instanceof MyInterface));

        // Since obj2 is of type boy, whose parent class is person and it implements
        // the interface Myinterface, it is instance of all of these classes
        System.out.println("obj2 instanceof Person: " + (obj2 instanceof Person));
        System.out.println("obj2 instanceof Boy: " + (obj2 instanceof Boy));
        System.out.println("obj2 instanceof MyInterface: " + (obj2 instanceof MyInterface));
    }
}

class Person {
}

class Boy extends Person implements MyInterface {
}

interface MyInterface {
}
```

Output:

```
obj1 instanceof Person: true
obj1 instanceof Boy: false
obj1 instanceof MyInterface: false
obj2 instanceof Person: true
obj2 instanceof Boy: true
obj2 instanceof MyInterface: true
```

Java Control Statements - Decision Making

What are Java Control Statements ?

- A programming language uses control statements to control the flow of execution of program based on certain conditions.
- These are used to cause the flow of execution to advance and branch based on changes to the state of a program.
- These statements allows to control the flow of your program's execution based upon conditions known only during run time.

1. if

- Used to decide whether a certain statement or block of statements will be executed or not.

```
if(condition) {
   // Statements to execute if
   // condition is true
}
```

2. if-else

- We can use the else statement with if statement to execute a block of code when the condition is false.

```
if (condition){
    // Executes this block if condition is true
} else {
    // Executes this block if condition is false
}
```

3. nested-if

- It is an if statement that is the target of another if or else.

```
if (condition1) {
   // Executes when condition1 is true
   if (condition2) {
      // Executes when condition2 is true
   }
}
```

4. if-else-if ladder

- The if statements are executed from the top down.
- As soon as one if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed.

```
if (condition1) {
    // statement;
} else if (condition2) {
   // statement;
} else if (condition3) {
   // statement;
}
.
.
.
else {
   // statement;
}
```

5. Switch-case

- The switch statement is a multiway branch statement.
- It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

```
switch (expression){
  case value1:
    statement1;
    break;
  case value2:
    statement2;
    break;
  .
  .
  case valueN:
    statementN;
    break;
  default:
    statementDefault;
}
```

- Expression can be of type byte, short, int, char, String or an enumeration.
- Dulplicate case values are not allowed.
- The default statement is optional.
- The break statement is used inside the switch to terminate a statement sequence.
- The break statement is optional. If omitted, execution will continue on into the next case.

6. Jump Statements

- Java supports 3 jump statements:
  - **break**
  - **continue**
  - **return**.
- These three statements transfer control to other part of the program.

Java Loops

What are Java Loops ?

- Facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.
- Java provides three ways for executing the loops.
- While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

1. while loop:

- Allows code to be executed repeatedly based on a given Boolean condition.
- The while loop can be thought of as a repeating if statement.

```
while (boolean condition) {
   // loop statements...
}
```

2. For Loop:

- Provides a concise way of writing the loop structure.
- Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line.
- Thereby providing a shorter, easy to debug structure of looping.

```
for (initialization condition; testing condition; increment/decrement){
    // statement(s)
}
```

Enhanced For Loop

- Java also includes another version of for loop introduced in Java 5.
- Enhanced for loop provides a simpler way to iterate through the elements of a collection or array.
- It is inflexible and should be used only when there is a need to iterate through elements in a sequential manner without caring index.
- **Important Points:**
- The object/variable is immutable when enhanced for loop is used.
- It ensures that the values in the array can not be modified, so it can be said as a read-only loop.
    - Here we can’t update the values as opposed to other loops where values can be modified.

```
for (T element : Collection obj or array) {
    // statement(s)
}
```

Example:

```
String array[] = { "Ron", "Harry", "Hermoine" };

for (String x : array) {
    System.out.println(x);
}
```

Java For-Each loop

```
arrList.forEach((e) -> print(e));
```

3. do while loop

- Similar to while loop with only difference that it checks for condition after executing the statements.
- An example of ***Exit Control Loop.***

```
do {
    // statements..
} while (condition);
```