UNIT-I

1. INTRODUCTION TO JAVA

1.1 History of Java:

1. Origins: Java was developed by James Gosling, Mike Sheridan, and Patrick Naughton at Sun Microsystems in the early 1990s. It was originally conceived as a language for programming consumer electronics, such as television set-top boxes.

2. Java 1.0 (1996): The first official version of Java was released to the public in 1996. It gained early popularity due to its "Write Once, Run Anywhere" mantra, which means that code written in Java can run on any platform that has a Java Virtual Machine (JVM).

3. Acquisition by Oracle: In 2010, Oracle Corporation acquired Sun Microsystems, including the rights to Java.

4. Java 8 (2014): Java 8 introduced significant changes with the introduction of lambda expressions, the Stream API, and the java.time package for date and time handling.

5. Java 9, 10, 11, and beyond: Oracle continued to release new versions of Java with various enhancements, including modularity with Java 9, and long-term support (LTS) releases every three years, starting with Java 11.

1.2 Key Features of Java:

1. Platform Independence: Java code is compiled into bytecode, which can run on any platform with a Java Virtual Machine (JVM).

2. Object-Oriented: Java is a pure object-oriented programming language, which promotes modularity and code reusability.

3. Strongly Typed: Java enforces strong data typing, which helps catch errors at compile-time.

4. Automatic Memory Management: Java manages memory automatically through its garbage collection mechanism.

5. Security: Java has built-in security features, including a robust sandbox to protect against malicious code.

6. Rich Standard Library: Java offers a comprehensive standard library that includes classes for I/O, networking, data structures, and more.

7. Multi-threading: Java supports multithreading, allowing you to create applications that can perform multiple tasks concurrently.

8. Exception Handling: Java provides robust exception handling to deal with runtime errors.

1.3 Getting Started with Java:

To get started with Java, you need to follow these steps:

1. Install Java: Download and install the Java Development Kit (JDK) from the Oracle website or an alternative distribution like OpenJDK.

2. Choose an Integrated Development Environment (IDE): Popular Java IDEs include Eclipse, IntelliJ IDEA, and NetBeans. These tools provide a user-friendly environment for writing, testing, and debugging Java code.

3. Write Your First Java Program: Create a simple Java program, typically saved with a `.java` file extension, using a text editor or your chosen IDE.

Here's a basic "Hello World" example:

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

4. Compile and Run: Use the JDK's `javac` compiler to compile your Java source code into bytecode and then use the `java` command to run the compiled program.

5. Learn the Basics: Start with the fundamentals of Java, including variables, data types, operators, control structures, and functions.

6. Explore Advanced Topics: As you become more comfortable, delve into topics like object-oriented programming, exception handling, and multithreading.

7. Practice: The best way to learn Java is through practice. Work on coding exercises, projects, and explore Java's extensive libraries to gain hands-on experience.

8. Refer to Documentation and Resources: Utilize online documentation, tutorials, and books to deepen your understanding of Java.

2.JAVA PROGRAMS

What is a Java Application?

It is basically like the Java program, and it runs on an underlying OS (operating system) that gets support from the virtual machine. Java applications are also known as application programs.

A Java application does not necessarily require a graphical user interface for executing a Java application. It can easily run without it.

Features of Java application:

- These are very similar to the Java programs.
- You can independently execute these without making use of the web browser.
- The execution of these apps requires a *main* function.
- A Java application is capable of executing various programs via the local system.
- The applications have full access to the local file systems and also networks.
- A user needs an application program when they need to perform a task directly.

What is a Java Applet?

It is a type of Java program that a user can easily embed in a web page. A Java applet works at the client-side, and it runs inside the browser (web browser). One can use the **OBJECT** or the **APPLET** tag for embedding an applet into an HTML page and host it on any web server. The applets also serve the purpose of making a website more entertaining and dynamic.

Features of Java applet:

- A Java applet is a small Java program.
- One can easily include a Java applet along with various HTML documents.
- Execution of these applets does not require any main function.
- One needs a web browser (Java-based) for the execution of the applets.
- They don't have any network access or local disk.
- These can only access the services that are browser-specific.
- These can't perform the execution of the programs from the local machines.
- A Java applet cannot establish any access to the local system.
- One requires a Java applet for performing small tasks. Conversely, it can also serve as a part of any task.

3. VARIABLE.

The Variable is a piece of memory that is typically used to store information. All the variables must be declared before use in the Java code. During the program execution, the value stored in the variable can be changed.

Variables are just the name of the reserved memory location, and all the operations done on the variables will definitely effects that memory location.

3.1 How to declare a variable in Java?

The Syntax for declaring a variable in java: **data_type variable_name;**
where, **data_type** is the type of data to be stored in this variable, and **variable_name** is the name given to the variable.

Example: int age; Here, **int** is the data type and **age** is the name of the variable where int data type allows the age variable to hold the integer value.

3.2 How to initialize a variable in Java?

The Syntax for initializing a variable: **data_type variable_name = value;**
where, **data_type** is the type of data to be stored in this variable, and **variable_name** is the name given to the variable. **value** is the initial value stored in the variable.

Example: int age = 20; Here, int is the data type and age is the name of the variable where int data type allows age variable to hold the integer value 20.

3.3 Variables Scope and Lifetime in Java:

The **scope of a variable** is the part of the program over which the variable name can be referenced. You cannot refer to a variable before its declaration. Variable scope refers to the accessibility of a variable.

The **lifetime of a variable** refers to the time in which a variable occupies a place in memory. The scope and lifetime of a variable are determined by how and where the variable is defined.

3.4 Scope vs Lifetime in Java:

Scope refers to the range of code where you can access a variable. We can divide the scope into:

1. Method body scope variable is accessible in method body only (local variables, parameters)
2. Class definition scope variable is accessible in the class definition (instance variables)

Lifetime refers to the amount of time a variable (or object) exists. We can divide lifetime into categories too:

1. Method body lifetime exists on method body lifetime entry, disappears on method body exit (local variables, parameters).
2. Class definition lifetime exists as long as the object is around (instance variables).

3.5 Naming Conventions of Variables in Java:

Variable's names are case-sensitive. The variable name **money** is not the same as Money or **MONEY**.

Variables naming cannot contain white spaces, for example, **int num ber = 100;** is invalid because the variable name has space in it.

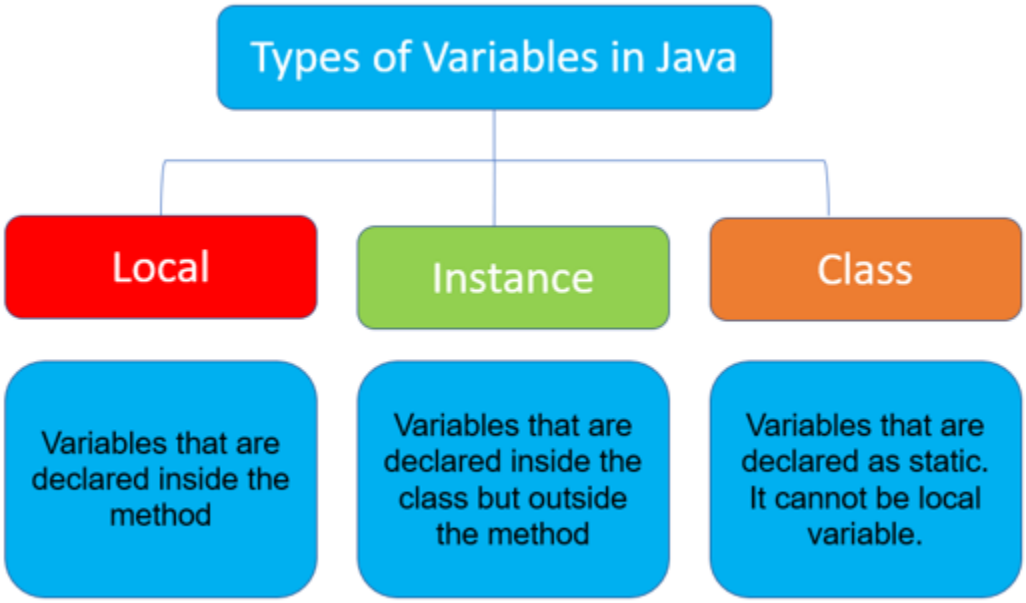
A variable name can begin with special characters such as **$** and **_**

As per the java coding standards, the variable name should begin with a lower case letter, for example, **int number;** For lengthy variables names that have more than one word do it like this: **int smallNumber; int bigNumber;** (start the second word with a capital letter).

3.6 Types of Variables in Java:

Basically, there are three types of variables in java :

1. Local Variable
2. Instance Variable
3. Static Variable



3.6.1 Local Variables in Java:

Local Variables are declared inside the method of the class. The scope of the local variable is limited to the method, which means you cannot change the value of the local variable outside the method and you cannot even access it outside the method. The initialization of the local variable is mandatory.

Scope of Local Variable*:* Within the block in which it is declared.
The lifetime of Local Variable*:* Until the control leaves the block in which it is declared.

Example to Understand Local Variables in Java:

```
class LocalVariableExample{
      public void EmployeeAge (){
      // local variable age
      int age = 30;
      age = age + 5;
        System.out.println ("Employee age is : " + age);
      }
      public static void main (String args[])
      {
        LocalVariableExample obj = new LocalVariableExample ();
        obj.EmployeeAge ();
      }
}
```

Output: Employee age is : 35

In the above code, **age** is the Local Variable to the method **EmployeeAge()**. If we declare age outside the method, it will give a compilation error.

3.6.2 Instance Variables in Java:

The instance variable is also known as a non-static variable. It is always declared in a class but outside the method, block, or constructor. The instance variable is created when an object of the class is created and destroyed when the object is destroyed. Instance Variables can only be accessed by creating the objects of the class. The initialization of the instance variable is not required, it takes the default value as 0.

Scope of Instance Variable: Throughout the class except in static methods.
The lifetime of Instance Variable: Until the object is available in the memory.

Example to Understand Instance Variables in Java:

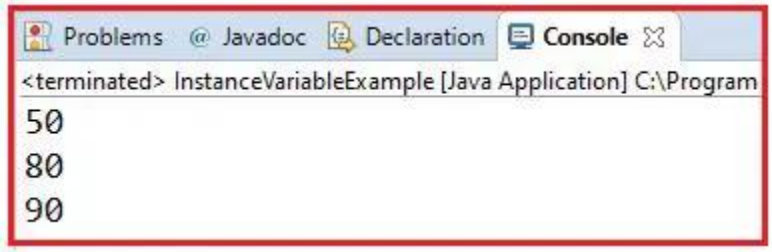
```
class Marks{
      // These variables are instance variables.
      // These variables are in a class
      // and are not inside any function
```

```
        int engMarks;
        int mathsMarks;
        int phyMarks;
}


class InstanceVariableExample{
        public static void main (String args[]){
        //Object
        Marks obj1 = new Marks ();
        obj1.engMarks = 50;
          obj1.mathsMarks = 80;
        obj1.phyMarks = 90;

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

Output:



The static variable is also known as the Class Variable. Static variables are created at the start of program execution and destroyed automatically when execution ends. These variables are declared similarly to instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block. Initialization of Static Variable is not mandatory, it's default value is 0. If we access the static variable without the class name, the 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 class_name.variable_name;

Scope of Static Variable: Throughout the class.

The lifetime of Static Variable: Until the end of the program.

Example to Understand Static Variables in Java:

```
class Employee{
      // static variable salary
      public static double salary;
```

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

Output: Harsh's average salary:1000.0

4. OPERATOR

An **operator** in Java is a symbol that is used to perform operations. Operators are the constructs that can manipulate the values of the operands. Consider the expression **2 + 3 = 5**, here **2** and **3** are **operands** and **+** is called **operator**. In this article, the goal is to get you the expertise required to get started and work with operators in Java.

There as many types of Operators in Java as follows:

- Arithmetic Operator
- Assignment Operator
- Relational Operator
- Logical Operator
- Bitwise Operator
- Unary Operator

- Shift Operator
- Ternary Operator

4.1 Arithmetic Operators in Java:

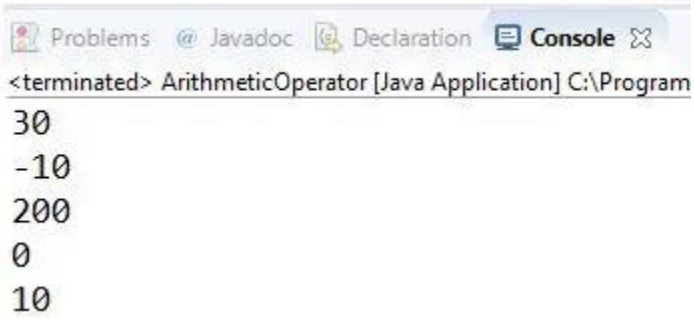
Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc. The following table shows the **Types of Arithmetic Operators** in Java:

Operator	Symbol	Description
Addition Operator	+	Adds the left operand with the right operand and returns the result.
Subtraction Operator	–	Subtracts the left operand and right operand and returns the result.
Multiplication Operator	*	It multiples the left and right operand and returns the result.
Division Operator	/	Divides the left operand with the right operand and returns the result.
Modulo Operator	%	Divides the left operand with the right operand and returns the remainder.

Example to Understand Arithmetic Operators in Java:

```
public class ArithmeticOperator
{
	public static void main (String[]args)
	{
	int A = 10;
	int B = 20;
	  System.out.println (A + B);
	  System.out.println (A - B);
	  System.out.println (A * B);
	  System.out.println (A / B);
	  System.out.println (A % B);
	}
}
```

Output:



4.2 Assignment Operators in Java:

The Java **assignment operator** is used to assign the value on its right to the operand on its left.
The following table shows the **Types of Assignment Operators** in Java.

Symbol	Description	Example

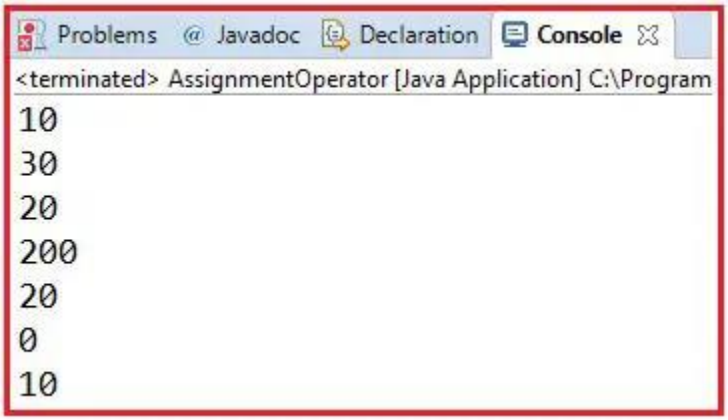
=	Assign the right operand to the left operand.	1. int a=10;
		2. int b=20;
		3. a+=4; //a=a+4 (a=10+4)
		4. b-=4; //b=b-4 (b=20-4)

+=	Adding left operand with right operand and then assigning it to variable on the left.	int a=5; a += 5; //a=a+5;
-=	subtracting left operand with right operand and then assigning it to variable on the left.	int a=5; a -= 5; //a=a-5;
*=	multiplying left operand with right operand and then assigning it to the variable on the left.	int a=5; a *= 5; //a=a*5;
/=	dividing left operand with right operand and then assigning it to a variable on the left.	int a=5; a /= 5; //a=a/5;
%=	assigning modulo of left operand with right operand and then assigning it to the variable on the left.	int a=5; a /= 5; //a=a/5;

Example to Understand Assignment Operators in Java:

```
public class AssignmentOperator
{
	public static void main (String[]args)
	{
	int a = 10;
	int b = 20;
	int c;
	  System.out.println (c = a);		// Output =10
	  System.out.println (b += a);		// Output=30
	  System.out.println (b -= a);		// Output=20
	  System.out.println (b *= a);		// Output=200
	  System.out.println (b /= a);		// Output=2
	  System.out.println (b %= a);		// Output=0
	  System.out.println (b ^= a);		// Output=0
	}
}
```

Output:



4.3 Relational Operators in Java:

The **Relational Operators in Java** are also known as Comparison Operators. It determines the relationship between two operands and returns the Boolean results, i.e. true or false after the comparison. The following table shows the **Types of Relational Operators** in Java.

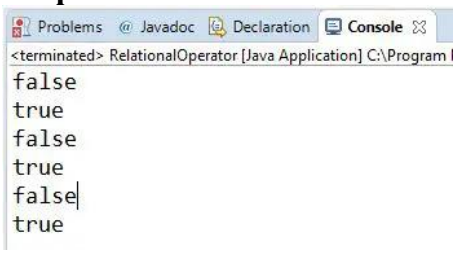
Operator	Symbol	Description	Example
Equal to	==	returns true of the left-hand side is equal to the right-hand side.	5==3 is evaluated to be false.
Not Equal to	!=	returns true of the left-hand side is not equal to the right-hand side.	5!=3 is evaluated to be true.

Less than	<	returns true of the left-hand side is less than the right-hand side.	5<3 is evaluated to false
Less than or equal to	<=	returns true of the left-hand side is less than or equal to the right-hand side.	5<=5 is evaluated to be true
Greater than	>	returns true of the left-hand side is greater than the right-hand side.	5>3 is evaluated to be true
Greater than or Equal to	>=	returns true of the left-hand side is greater than or equal to the right-hand side.	5>=5 is evaluated to be true
Instance of Operator	instanceOf	It compares an object to a specified type.	String test = "asdf"; boolean result; result = test instanceOf String; //true

Example to Understand Relational Operators in Java:

```
public class RelationalOperator
{
      public static void main (String[]args)
      {
      int a = 10;
      int b = 20;
   System.out.println (a == b);// returns false because 10 is not equal to 20
   System.out.println (a != b);// returns true because 10 is not equal to 20
   System.out.println (a > b);            // returns false
   System.out.println (a < b);            // returns true
   System.out.println (a >= b);           // returns false
   System.out.println (a <= b);           // returns true
      }
}
```

Output:



4.4 Logical Operators in Java:

The **Logical Operators in Java** are mainly used in conditional statements and loops for evaluating a condition. They are basically used with binary numbers. The following table shows the types of Logical Operators in Java.

Operator	Symbol	Description	Example
Logical OR	||	It returns true if either of the Boolean expressions is true.	false || true is evaluated to true
Logical AND	&&	It returns true if all the Boolean Expressions are true	false && true is evaluated to false.

Example to Understand Logical Operators in Java:

```
public class LogicalOperators
{
	public static void main (String[]args)
	{
	int a = 10;
	  System.out.println (a < 10 & a < 20);                //returns false
	  System.out.println (a < 10 || a < 20);               //returns true
	  System.out.println (!(a < 10 & a < 20)); //returns true
	}
}
```

Output:

4.5 Bitwise Operators in Java:

The **Bitwise Operators in Java** perform bit-by-bit processing. They can be used with any of the integer types. The following table shows the **Types of Bitwise Operators** in Java.

Operator	Symbol	Description	Example
Bitwise OR	|	It compares corresponding bits of two operands. If either of the bits is 1, it gives 1. If not, it gives 0.	int a=12, b=25; int result = a|b; //29

Bitwise AND	&	It compares corresponding bits of two operands. If either of the bits is 1, it gives 1. If either of the bits is not 1, it gives 0.	int a=12, b=25; int result = a&b; //8
Bitwise XOR	^	It compares corresponding bits of two operands. If corresponding bits are different, it gives 1. If corresponding bits are the same, it gives 0.	int a=12, b=25; int result = a^b; //21
Bitwise Complement	~	It inverts the bit pattern. It makes every 1 to 0 and every 0 to 1.	int a=35; int result = ~a; //-36

Example to Understand Bitwise Operators in Java:

```
public class BitwiseOperators
{
	public static void main (String[]args)
	{
	int a = 58;                                                    //111010
	int b = 13;                                                    //1101
	  System.out.println (a & b);              //returns 8 = 1000
	  System.out.println (a | b);              //63=111111
	  System.out.println (a ^ b);              //55=11011
	  System.out.println (~a);                 //-59
	}
}
```

Output:

4.6 Unary Operators in Java:

The **Unary Operators in Java** need only one operand. They are used to increment, decrement, or negate a value. The following table shows the **Types of Unary Operators** in Java.

Operator	Symbol	Description	Example
Unary minus	–	It is used for negating the values.	int a=5.2; int b = -a; //-5.2

Unary plus	+	It is used for giving positive values.	int a=5.2; int b = +a; //5.2
Increment Operator	++	It is used for incrementing the value by 1.	int a=5.2; int b = ++a; //6.2
Decrement Operator	—	It is used for decrementing the value by 1.	int a=5.2; int b = –a; //4.2

Logical NOT Operator	!	It is used for inverting a Boolean value	boolean a=false;
			boolean b=!a;
			//true

Note: There are basically two types of Increment Operator:

Post Increment Operator in Java:
Value is first used for computing the result and then incremented.

Example:
```
int b = 10, c = 0;
 // c=b then b=b+1
 c=b++;
 System.out.println("Value of c (b++) = " +c);
```

Output: Value of c (b++) = 10

Pre Increment Operator in Java:
Value is incremented first and then result is computed.

Example:
```
int a = 20, c = 0;
 // a = a+1 and then c = a;
 c=++a;
 System.out.println("Value of c (++a) = " +c);
```

Output: Value of c (++a) = 21
Note: There are two types of Decrement Operator:

Post Decrement Operator in Java:
Value is first used for computing the result and then decremented.

Example:
```
int e = 40, c = 0;
 // c=e then e=e-1
 c=e-;
 System.out.println("Value of c (e-) = " +c);
```

Output: Value of c (e–) = 40

Pre Decrement Operator in Java:
Value is decremented first and then result is computed.

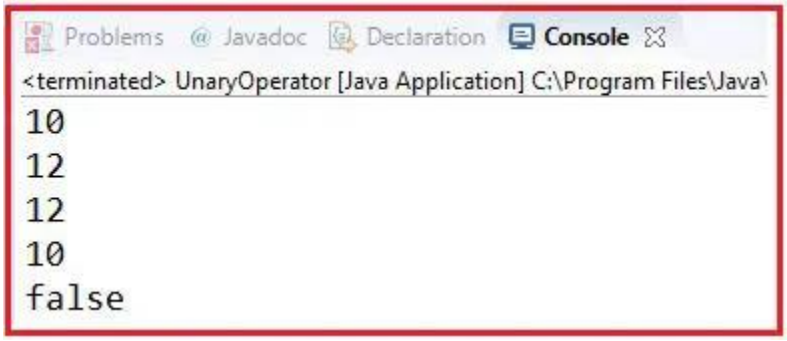
Example:
```
int d = 20, c = 0;
 // d=d-1 then c=d
 c=-d;
 System.out.println("Value of c (-d) = " +c);
```

Output: Value of c (–d) = 19

Example to Understand Unary Operators in Java:

```
public class UnaryOperator
{
      public static void main (String[]args)
      {
      int a = 10;
      boolean b = true;
        System.out.println (a++);                    //returns 11
        System.out.println (++a);
        System.out.println (a--);
        System.out.println (--a);
        System.out.println (!b); // returns false
      }
}
```

Output:



4.7 Shift Operator in Java:

The **Shift Operator in Java** is used when you're performing logical bits operations, as opposed to mathematical operations. 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.

Types of Shift Operators in Java:

Operator	Symbol	Description	Example
Left Shift Operator	<<	It is used to shift all of the bits in value to the left side of a specified number of times.	10<<2 //10*2^2=10*4=40
Right Shift Operator	>>	It is used to move the left operand's value to the right by the number of bits specified by the right operand.	10>>2 //10/2^2=10/4=2

Example to Understand Shift Operators in Java:

```
public class ShiftOperators
{
      public static void main (String[]args)
      {
      int a = 58;
        System.out.println (a << 2);          //232=11101000
        System.out.println (a >> 2);          //returns 14=1110
      System.out.println (a >>> 2);           //returns 14
      }
}
```

Output:

4.8 Ternary Operator in Java:

The **Ternary Operator in Java** is also known as the Conditional Operator. It is the shorthand of the if-else statement. It is the one-liner replacement of the if-else statement in Java. It is called ternary because it has three operands. The general format of the ternary operator 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 ':'.

Example:

```
int a=2;
 int b=5;
 int c = (a<b)?a:b;
```

Output: 2

Example to Understand Ternary Operator in Java:

```
public class TernaryOperators
{
      public static void main (String[]args)
      {
      int a = 20, b = 10, c = 30, res;
      res = ((a > b) ? (a > c) ? a : c : (b > c) ? b : c);
        System.out.println ("Max of three numbers = " + res);
      }
}
```

Output: Max of three numbers = 30

4.9 Operator Precedence and Associativity in Java

Java has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. For example, multiplication and division have higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.
Precedence and associative rules are used when dealing with hybrid equations involving more than one type of operator. In such cases, these rules determine which part of the equation to consider first as there can be many different valuations for the same equation.

4.9.1 Precedence order:

When two operators share an operand the operator with the higher *precedence* goes first. For example, 1 + 2 * 3 is treated as 1 + (2 * 3), whereas 1 * 2 + 3 is treated as (1 * 2) + 3 since multiplication has a higher precedence than addition.

4.9.2 Associativity:

When an expression has two operators with the same precedence, the expression is evaluated according to its *associativity*. For example x = y = z = 17 is treated as x = (y = (z = 17)), leaving all three variables with the value 17, since the = operator has right-to-left associativity (and an assignment statement evaluates to the value on the right hand side). On the other hand, 72 / 2 / 3 is treated as (72 / 2) / 3 since the / operator has left-to-right associativity. Some operators are not associative: for example, the expressions (x <= y <= z) and x++– are invalid.

The table below shows all Java operators from highest to lowest precedence, along with their associativity.

Level	Operator	Description	Associativity
16	[] . ()	access array element access object member parentheses	left to right
15	++ —	unary post-increment unary post-decrement	not associative
14	++ — + – ! ~	unary pre-increment unary pre-decrement unary plus unary minus unary logical NOT unary bitwise NOT	right to left
13	() new	cast object creation	right to left
12	* / %	multiplicative	left to right
11	+ – +	additive string concatenation	left to right
10	<< >> >>>	shift	left to right
9	< <= > >= instanceof	relational	not associative
8	== !=	equality	left to right
7	&	bitwise AND	left to right
6	^	bitwise XOR	left to right
5	|	bitwise OR	left to right
4	&&	logical AND	left to right
3	||	logical OR	left to right

2	?:	ternary	right to left
1	= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment	right to left

Note: Larger the number means higher the precedence.

5. DATA TYPE

The Data types are something that gives information about

1. **Size** of the memory location.
2. The **range of data** that can be stored inside that memory location
3. Possible **legal operations** that can be performed on that memory location.
4. What **types of results** come out from an expression when these types are used inside that expression.

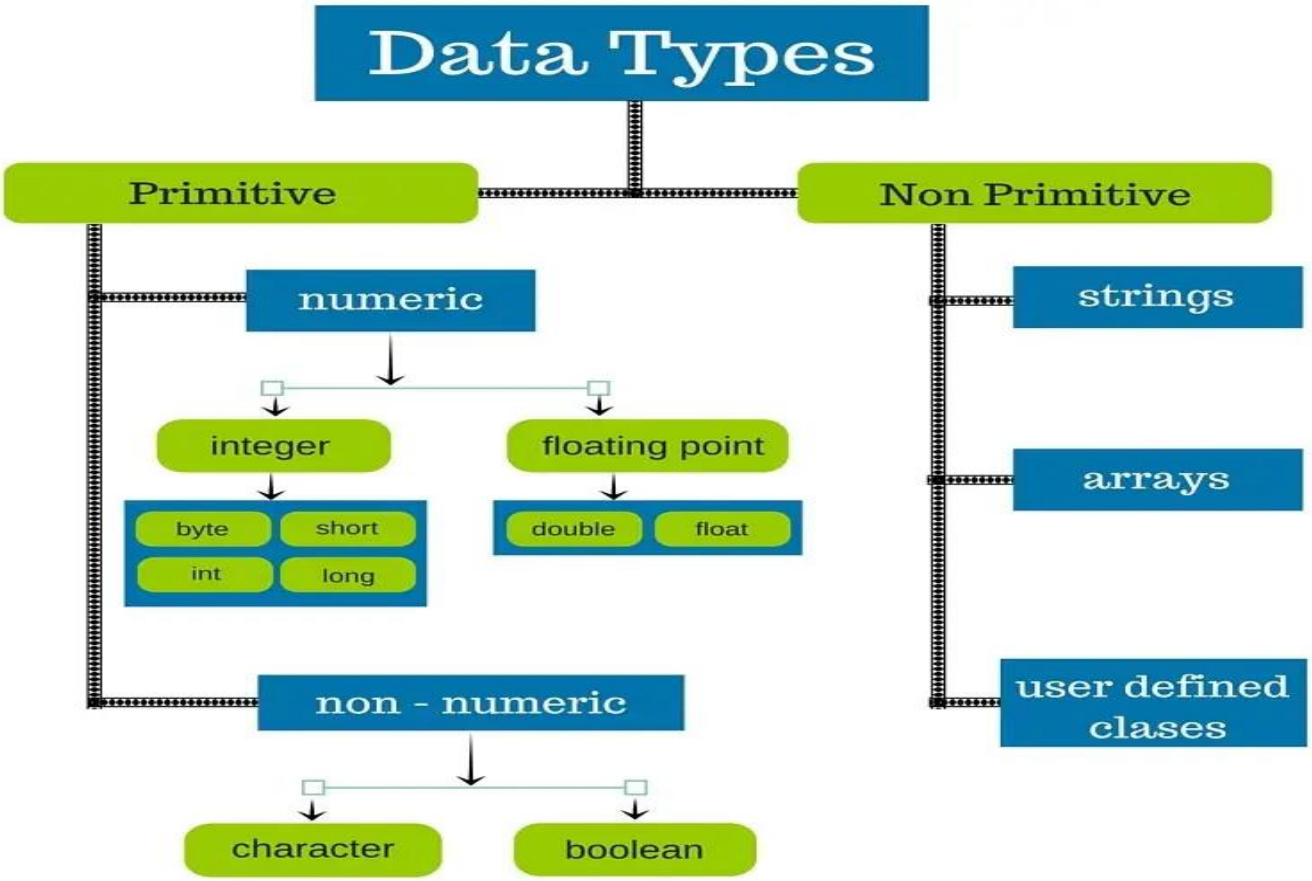
The keyword which gives all the above information is called the **data type**.

Classification of Java Data Types:

The **Data Types in Java** specifies the size and type of values that can be stored in an identifier. The **Java** language is rich in its data types. Data types in Java are classified into two types:

1. **Primitive Types:** Examples: Integer, Character, Boolean, and Floating Point.
2. **Non-primitive Types:** Examples: Classes, Interfaces, and Arrays.

For a better understanding of Java Data Types, please have a look at the following image which gives you an overview of different kinds of data types.



Note: Java is a strictly or strongly typed programming language i.e. before using the variable, first you need to declare it using data type (either primitive or non-primitive data type).

5.1 Primitive Data Types in Java:

Primitive data types are only single values, they have no special capabilities. Java defines eight primitive types of data: **byte, short, int, long, char, float, double, and boolean**. The primitive data types are also commonly referred to as **simple types**.

The primitive data types can be put in four groups:

1. **Integers**: This group includes **byte, short, int,** and **long**, which are for whole-valued signed numbers.
2. **Floating-point numbers**: This group includes **float** and **double**, which represent numbers with fractional precision.
3. **Characters**: This group includes **char**, which represents symbols in a character set, like letters and numbers.
4. **Boolean**: This group includes **boolean**, which is a special type for representing true/false values.

Integer Data Type in Java:

Java defines four integer types: **byte**, **short**, **int,** and **long**. All of these are can hold signed, positive and negative values.

Byte Data Type in Java:

The smallest integer type is a byte. Variables of type are byte is especially used when you're working with raw binary data that may not be directly compatible with java's other built-in types. It is an 8-bit signed two's complement integer and can hold a value between -128 to 127 (inclusive). That means its minimum value is -128 and the maximum value is 127 whereas its default value is 0.
As it is the smallest integer type (4 times smaller than the int data type), so you can use this byte data type instead of the int data type when the value is between -128 and 127. This is because it saved memory in large arrays where the memory savings is most required.
Examples: **byte b1 = 100; byte b2 = -100;**

Short Data Type in Java:

The short Data Type in Java can hold a value between -32,768 to 32,767 (inclusive). That means its minimum value is -32,768 and the maximum value is 32,767 whereas its default value is 0. It is a 16-bit signed two's complement integer. Like the Byte data type, the short can also be used to save memory as it is 2 times small than the int data type.
Examples: **short s1 = 5000; short s2 = -777;**

Int Data Type:

The most commonly used integer type is int. In addition to other uses, variables of type int are commonly employed to control loops and index arrays. It is a 32-bit signed two's complement integer and this data type can hold a value between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). That means its minimum value is -2,147,483,648 and the maximum value is 2,147,483,647 whereas its default value is 0. The int data type is generally used as a default data type for integral values unless there is no problem with memory.
Examples: **int i1 = 50000; int i2 = -50000;**

Long Data Type:

It is useful for those occasions where an int type is not large enough to hold the desired value. It is a 64-bit two's complement integer and this data type can hold a value between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). That means its minimum value is -9,223,372,036,854,775,808 and maximum value is 9,223,372,036,854,775,807 where as its default value is 0.
Examples: **long l1 = 666000L; long l2 = -222000L;**

Floating-Point Data Types:

Floating-point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision. Java Provides two data types to hold floating-point values. They are as follows.

1. **Float Data Type**
2. **Double Data type**
3.

Float Data type:

It specifies a single-precision 32-bit IEEE 754 floating-point, which is faster on some processors and takes half as much space as double-precision. So, it is recommended to use float instead of double data type if you want to save memory in large arrays of the floating-point numbers. Its value range is unlimited and its default value is 0.0F. This data type should not be used for precise values, such as currency.

Examples: float f1 = 123.4f; float f2 = 768.5f;

Double Data Type:
It specifies a single-precision 64-bit IEEE 754 floating-point, which is actually faster than single-precision on some modern processors that have been optimized for high-speed mathematical calculations. Its value range is also unlimited like a float data type. Its default value is 0.0d.

Example: double d1 = 345.6; double d2 = 125.7;

Char Data Type:
In Java, the data types used to store characters are **char**. Java uses Unicode to represent characters. There are no negative chars. Unicode defines a fully international character set that can represent all of the characters found in all human languages like Latin, Greek, Arabic, and many more. This data type is a single 16-bit Unicode character and its value-ranges between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).

Example: char ch = 'A';

Why does char use 2 bytes in java and what is \u0000?

This is because java uses the Unicode system instead of the ASCII code system and the \u0000 is the lowest range in the Unicode system. In our upcoming articles, we will discuss the Unicode System.

Boolean Data Type:

Java has a primitive type called **boolean**, which is used to store two possible values i.e. true or false. The default value is false. This data type in Java is basically used for simple flags that track true/false conditions.
Example: **Boolean b1= true;**

The values, ranges, and sizes of these primitive data types are shown in this table:

Data Types	Valid Values	Default Values	Size	Range	Example

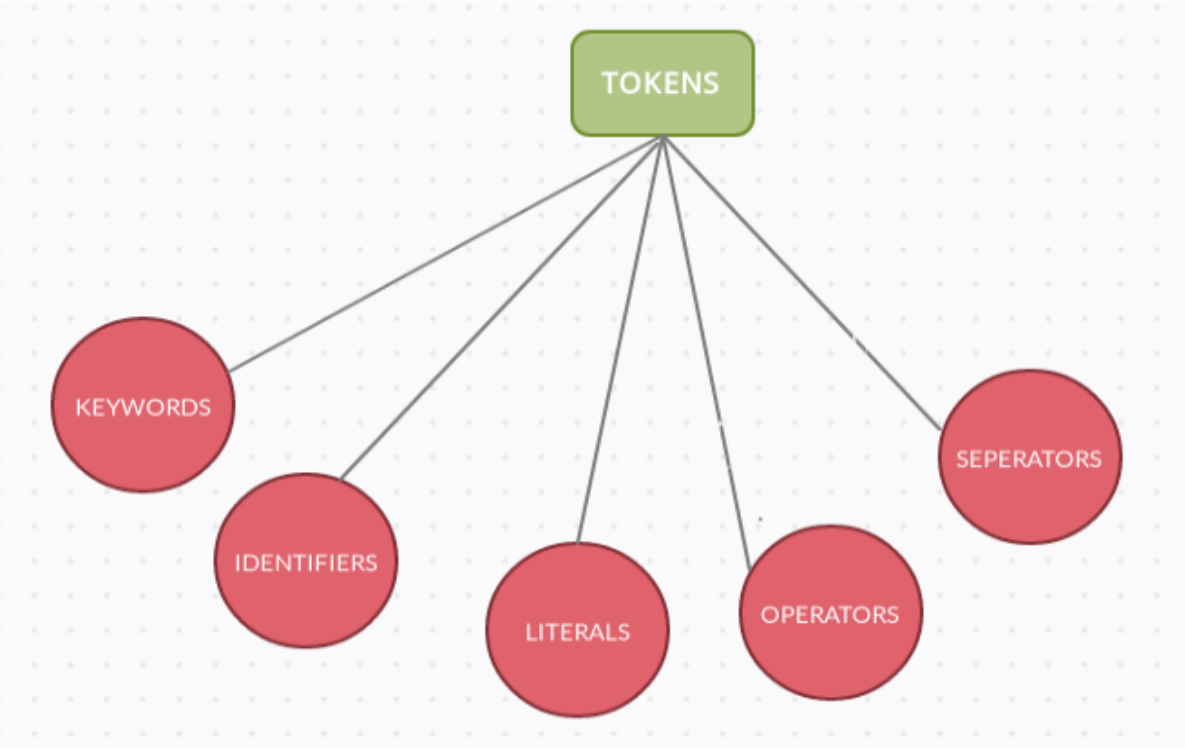
boolean	true, false	false	1 bit	–	boolean isEnabled = true;
byte	integer	0	8 bits	–	byte thiss =1;
char	unicode	\u0000	16 bits	–	char a =’a’;
short	integer	0	16 bits	[-32,768, 32,767]	short counter =1;
int	integer	0	32 bits	[-2,147,483,648, 2,147,483,647]	int I = 10;
long	integer	0	64 bits	[-9,223,372,036,854,775,808, 9,223,372,036,854,775,807]	long song = 100;
float	floating point	0.0	32 bits	–	float pi = 3.14F;
double	floating point	0.0	64 bits	–	Double team = 1e1d;

These are the data types in Java. The primitive data types are much like the C++ data types. But, unlike C++, **String is not a primitive data type. Strings in Java are objects**. The Java library has the String class and we create their objects when we deal with strings. When it comes to numeric data types, Java does not have any notion of unsigned integers. So, the numeric data types in Java can always store positive and negative values. We will talk about the non-primitive data types later.

5.2 Non Primitive Data Types :

A **non-primitive data type** is something such as an array structure or class known as the non-primitive data type.

6.TOKENS:



Tokens are the smallest individual unit of the program , In other words, they are the building blocks of a language.

Basically In Java, the program is a collection of classes and methods, while methods are a collection of various expressions and statements .

Moreover, statements and expressions are made up of tokens. Tokens in Java are the small units of code which a Java compiler uses for constructing statements and expressions.

There are five tokens in Java which are shown below:

1. Keywords

2. Identifiers

3. Literals

4. Operators

5. Special Symbols/Separators

Let's take a example of token

```
public class ABC        {
 public static void main(String args[])   {
System.out.println("Technoname");
```

```
}
}
```

In the above code , public, class, ABC, {, static, void, main, (, String, args, [,],), System, ., out, println, Technoname, etc. are the Java tokens.

The Java compiler (JDK: Java Development Kit) converts/translates this code to Java Bytecode and later this Bytecode translate to machine code with the help of JVM(Java Virtual Machine).

6.1 Keywords :

Keywords are the reserved words , that is to say these are the words which are defined by language.
For Example : int , float,double etc
All keywords in Java are in lowercase Certainly, variable name must not match with any keyword
However , there are 32 keywords in C, 63 in C++ and 50 in Java

01. abstract	02. boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. throw	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp

Keywords in Java

6.2 Identifiers

Identifiers are the name of a variable/ array / method/class/ interface / package.

For Example :

```
int n;
 int arr[];
class ABC { }
```

In above example n,arr,ABC are identifiers.

Identifier naming rules :

It can contain alphabets (a-z,A-Z), digits(0-9) or symbols(_,$)
It cannot contains space,dot and other special symbols(@,#,&)
First letter cannot be a digit
It must not match with any keyword of Java
Two identifier cannot have same name within same scope
There is no limit on length of identifier because big names are more understandable.
In conclusion, All the above points can be explained by below examples :

```
//Valid Identifiers
$technoname  //correct
_techno      //correct
techno       //correct
techno_name  //correct
tech21no     //correct

//Invalid Identifiers
techno name    //error
&techno     //error
33techno       //error
float          //error
techno/name    //error
techno's       //error
```

6.3 Literals

Literals in Java are similar to normal variables which were explained above but their values cannot be changed once assigned. In other words, literals are constant variables having fixed values.
These are defined by programmer and can belong to any data type. There are five types of literals available in Java :

1. Integer
2. Floating Point
3. Character
4. String
5. Boolean

Literal	Type
33	int
7.89	double
false, true	boolean
'A', '9', '-'	char
"technoname"	String
null	any reference type

Literals in Java

Example :

```
public class TechnoName {
public static void main(String[] args)    {
int tech1 = 33;   // Int literal
float tech2 = 33.10;    // Float literal
char tech3 = 'tech' // char literal
String tech4 = "technoname"; // String literal
boolean tech5 = true; // Boolean literal

System.out.println(tech1); //33
System.out.println(tech2); //33.10
System.out.println(tech3); //tech
System.out.println(tech4); //technoname
System.out.println(tech5); //true
 }
}
```

6.4 Operators

Basically, operators are the special symbols that tells the compiler to perform a special mathematical and non-mathematical operations.
Java provides different types of operators that can be classified based on the functionality they provide.

Moreover, Java supports eight types of operators which are as follows:
Arithmetic Operators
Assignment Operators
Relational Operators
Unary Operators
Logical Operators
Ternary Operators
Bitwise Operators
Shift Operators

Operator	Symbols
Arithmetic	+ , – , / , * , %
Unary	++ , – – , !
Assignment	= , += , -= , *= , /= , %= , ^=
Relational	==, != , < , >, <= , >=
Logical	&&, ||
Ternary	(Condition) ? (Statement1) : (Statement2);
Bitwise	& , | , ^ , ~
Shift	<< , >> , >>>

Operators in Java

6.5 Separators

The separators in Java is also known as punctuators. They have special meaning known to Java compiler and cannot be used for any other purpose .

Some of the special symbols are shown in below table :

Symbol	Description
Square *Brackets []*	These are used define array elements , single square brackets define one dimensional array while two blocks of square brackets defined two dimensional array.
Parentheses()	These indicate a function call along with function parameters
Braces{}	The opening and ending curly braces indicate the beginning and end of a block of code .
Comma (,)	This helps in separating more than one statement, value or parameter in an expression
Semi-Colon (;)	It is used to separate two statements and used at end of a statement.

Separators in Java