**Java**

**Java:**

* + Java is a popular programming language.
  + It's used to create software and apps.
  + Known for its "write once, run anywhere" feature.
  + Java is not a fully object-oriented language as it supports primitive data types like int, byte, long, short, etc., which are not objects.

**Primitive Vs Non-primitive:**

Primitive Data Types: A primitive data type is pre-defined by the programming language. The size and type of variable values are specified, and it has no additional methods.

Non-Primitive Data Types: These data types are not actually defined by the programming language but are created by the programmer.

**Basic Structure:**

* + Code is written in classes.
  + Each program has a **main** method to start from.
  + Statements end with a semicolon **;**.

**Variables:**

* + Hold data like numbers or text.
  + A variable is nothing but a memory location name for the data.
  + Declare with a type and a name.
  + Example: **int age = 25;**
  + In Java, variables can be classified into three main types: local variables, instance variables, and static variables

difference between object and variable in Java?

Simple variable can only hold one value (string, number, boolean etc). Object can hold pairs of variables and values.

**Data Types:**

A category that specifies the kind of data a variable can hold in a programming language.

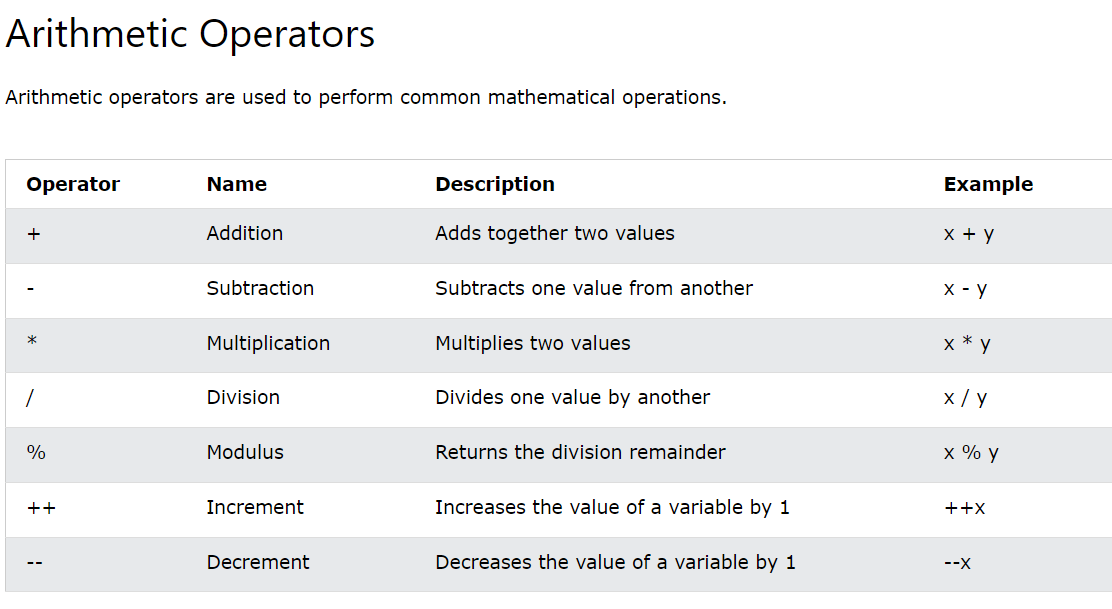
* + **int** for whole numbers.
  + **double** for decimal numbers.
  + **boolean** for true/false values.
  + **String** for text.

**Operators:**

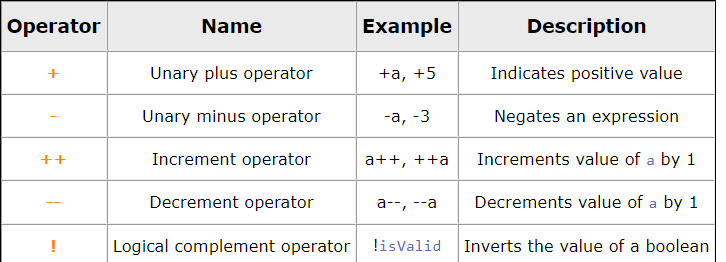
Operators are used to perform operations on variables and values.

Types of Operators:

* Arithmetic operators

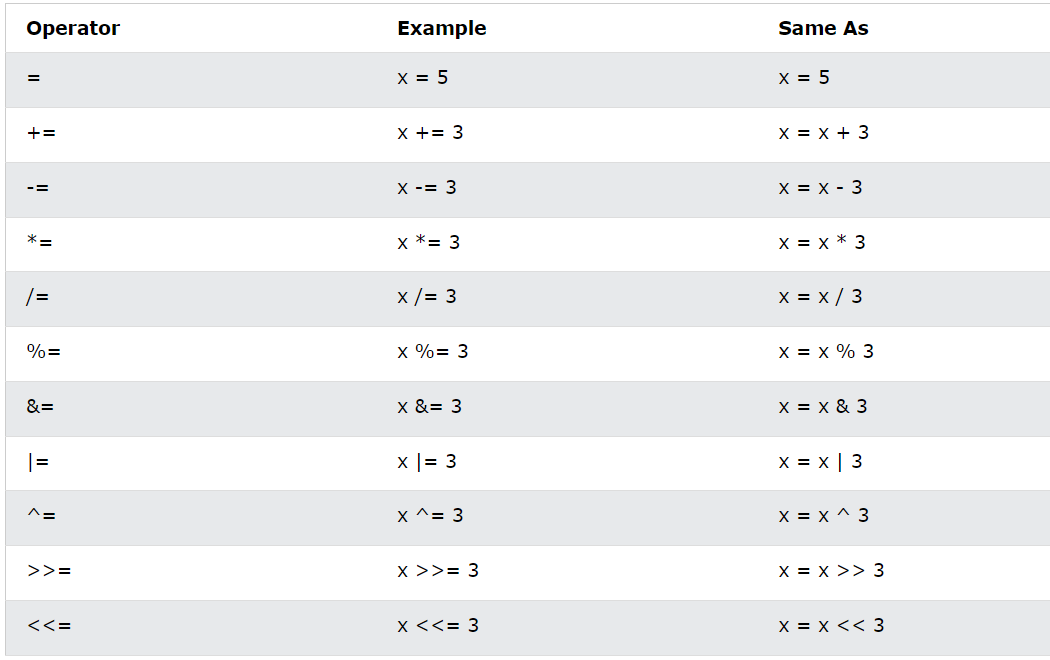


* Unary Operators:



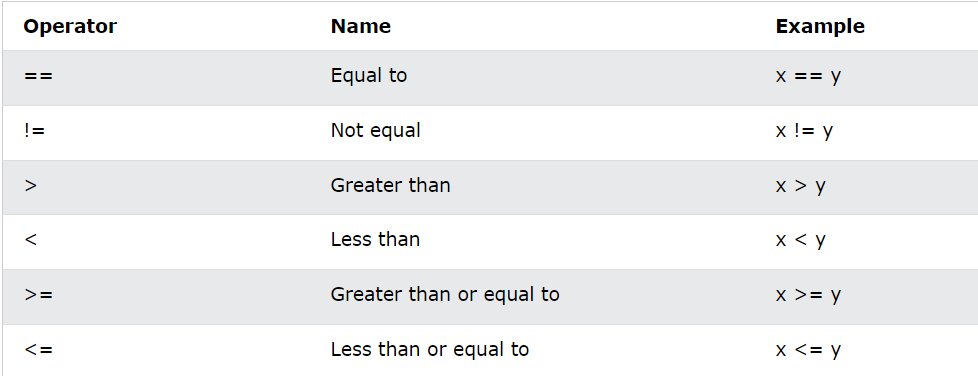
* Assignment operators

Assignment operators are used to assign values to variables.

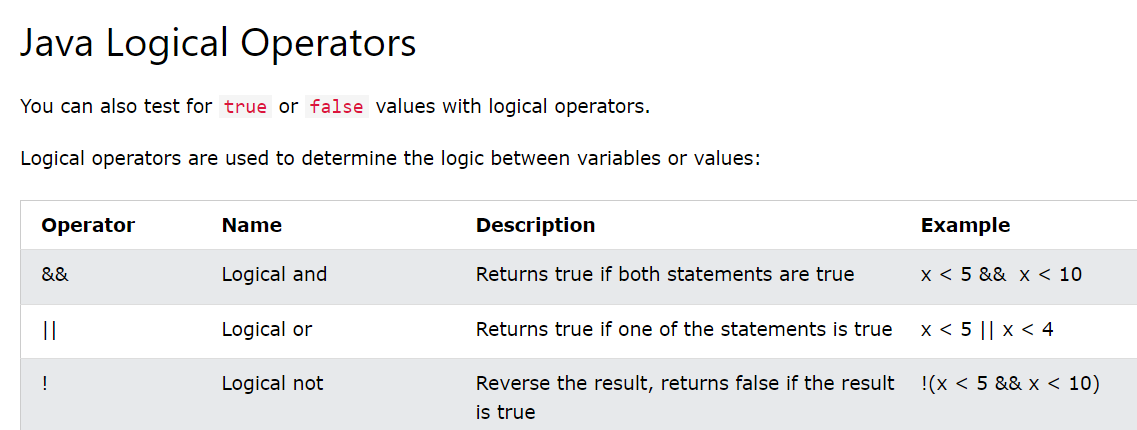


* Comparison operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.



* Logical operators



* Bitwise operators

Comments:

Comments in Java are the statements that are not executed by the compiler and interpreter.

Single line comment: //

Multi line comment: /\* \*/

# Input in java:

The input is the data that we give to the program.

import java.util.Scanner;

public class Input{

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter two numbers : ");

        var a = sc.nextInt();

        int b = sc.nextInt();

        String p = a>b ?

        a+" is greater" :

        b+" is greater";

        System.out.println(p);

        sc.close();

    }

}

Closing the **Scanner** in Java is necessary to release system resources it uses for reading input. This helps prevent memory waste and potential program issues.

Use **scanner.close()** after you're done to keep your program efficient and error-free.

The next() method uses whitespace as the default delimiter.

Type Conversion:

Type conversion means we convert one type of data to the other type.

e.g., int to float, float to int

Also known as widening conversion and implicit conversion.

We can convert int to float but we cannot convert int to bool because they are different data types with incompatible value ranges and meanings.(It is not possible for the compiler)

At the time of conversion destination data type size must be greater than source data type.

e.g., int -> long (is possible)

long -> int (not possible) because long size 8 and int is 4 so 8 byte data cannot fit in 4 byte.



// Reverse of this is not possible

If we do this then this error will come:

error: incompatible types: possible lossy conversion from long to int

        int b = a;

                ^

type conversion vs type casting in java?

In type casting, a data type is converted into another data type by a programmer using casting operator. Whereas in type conversion, a data type is converted into another data type by a compiler.

Float value assigning:

When we try to assign the value 4.5 to a float variable, it can lead to an error. This happens because 4.5 is considered a "double" by default, and we can't put a double into a float directly.

float a = 4.5;

For a float assignment, we must show the compiler that it's a float value. To do this, we add 'f' to the end of the value:

float a = 4.5f; // Note the 'f' suffix to indicate a float literal

This way, the compiler understands that 4.5 should be treated as a float value and assigns it correctly to the float variable "a".

Type Casting:

Type casting is a way of converting data from one data type to another data type.

Type casting in Java is performed by the programmer forcefully to convert one data type to another.

However, it's important to note that during this process, data may be lost if the target type can't **handle** the full range of the source type.

Type Casting Method:

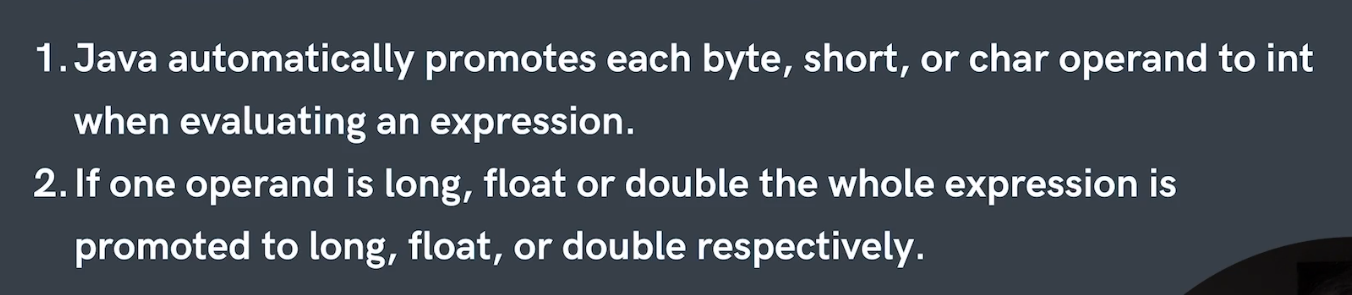
float a = 4.598f;

int b =(int) a;

System.out.println(b); // output = 4

It is also known as narrowing and explicit conversion.

Type promotion In Expression:



1) In this Example, during the expression evaluation, byte, char, and short values are automatically converted to int. This precaution prevents errors from occurring.

byte a = 15;

char b = 'b';

short c = 45;

int result = a+b+c;

System.out.println(result);

2) At the time of expression evaluation all data types automatically converted into biggest data type in expression.

int a = 45;

        byte b = 1;

        long l = 56;

        double d = 45.22;

    /\*  long result = a+b+l+d; -> it gives error because the result is double

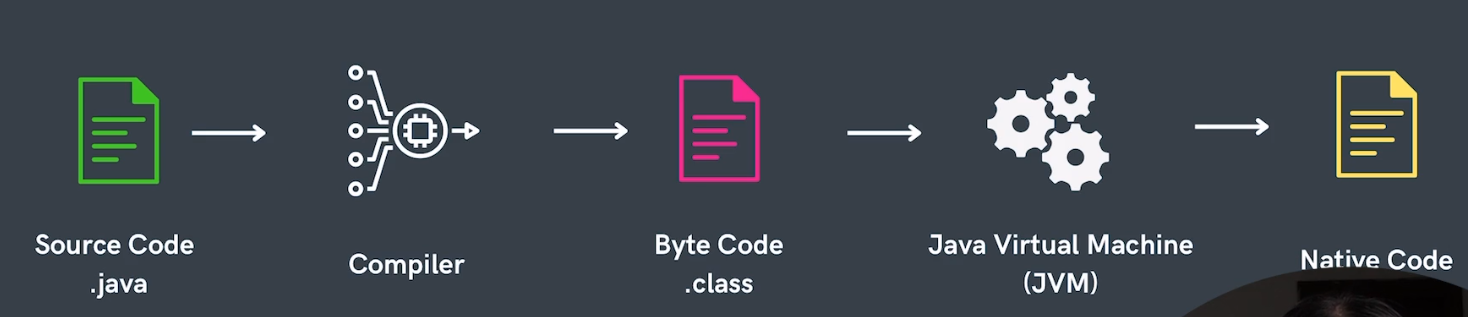
         and we cannot store it in long \*/

        double result = a+b+l+d;

        System.out.println(result);

The system ensures compatibility by promoting all values to a common data type to ensure that the expression is evaluated correctly.

How is our code running?



1. **Writing Code:** You write instructions in Java to tell the computer what you want it to do. This code is like a recipe.
2. **Compiling:** Before the computer can understand your code, it needs to be translated into a language it can read. This process is called compilation.
3. **Creating Files:** After compilation, your code becomes a file that the computer can understand. This file has a special name with ".class" at the end.
4. **Java Virtual Machine (JVM):** When you want to run your program, a special program called the Java Virtual Machine (JVM) comes into play. It reads and executes your ".class" files.
5. **Interpreting Code:** The JVM reads the instructions in your ".class" files one by one, like following the steps in a recipe. It knows the Java language and can understand what each instruction means.
6. **Output and Actions:** As the JVM reads your instructions, it performs actions like calculations, displaying text, or interacting with users.
7. **Results:** Your code's instructions are carried out step by step, and you see the results on your screen or get other outcomes depending on what your code does.

In short, you write code, the compiler makes it understandable for the computer, and the JVM interprets and executes the instructions, making your program run and produce results.

Note: If there is errors in the code then it is detected during the compilation stage, and the compiler informs us about them.

why java is called portable language?

Java is called a portable language because code written in Java can run on different computer systems without modification.

Ternary Operators:

variable = condition ? (if true) : else;



boolean c = (5>2) ? true : false;

Function:

A block of code that performs a specific task, making it easier to organize, reuse, and manage code in a program.

public class Basic {

    static void greet(String name) {  // Function with parameter 'name'

        System.out.println("Good Morning, " + name + "!!");  // Printing a greeting

    }

    public static void main(String[] args) {

        greet("Sumit");  // Calling greet function with argument "Sumit"

        greet("Saurabh");  // Calling greet function with argument "Saurabh"

    }

}

Function Overloading:

Function overloading is a feature in programming where multiple functions with the same name exist in a class, but they differ in the number or types of their parameters (Same Name, Different Parameters).

public class FunctionOverloading {

    static void greet(String name) {

        System.out.println("Good Morning, " + name + "!!");

    }

    static void greet(String name , String name2) {

        System.out.println("Good Morning, " + name + " and "+name2+"!!");

    }

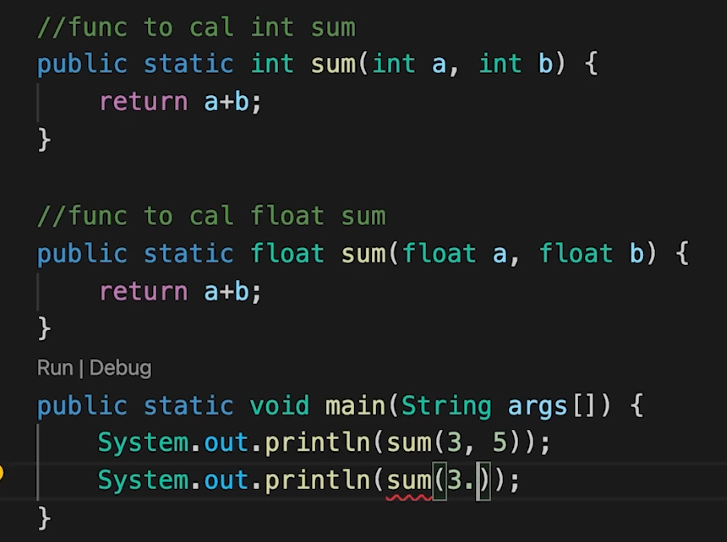
    public static void main(String[] args) {

        greet("Sumit");

        greet("Sumit","Saurabh");

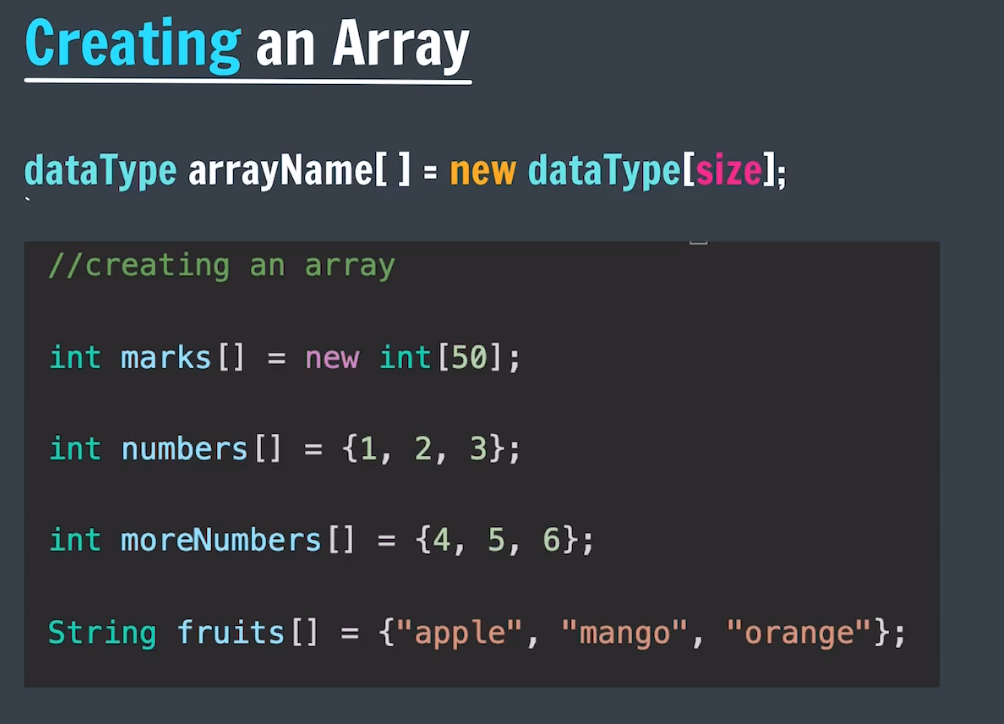
    }

}



Array:

Array is collection of similar elements.



Note:

-∞ == Integer.MIN\_VALUE

∞ == Integer.MAX\_VALUE

**Shallow Copy:**

* Copies references to elements, not the elements themselves.
* Changes to copied elements affect the original.
* Often done with copy constructors or **clone()**.

**Deep Copy:**

* Creates independent duplicates of objects, including their elements.
* Changes to copied elements don't affect the original.
* Usually done manually or with serialization.

when you assign one array to another in Java like **arr = marks**, you are not creating a new copy of the array; you're creating another reference to the same array. Any changes made through one reference will also affect the other because they both point to the same underlying data.

*Arrays Useful Functions:*

* Arrays.fill(arrayName , value) this is used to fill all array with particular value

import java.util.Arrays;

class RemoveDuplicateFromString{

    public static void main(String[] args){

        boolean arr[] = new boolean[26];

        Arrays.fill(arr , false);

    }

}

Strings:

In java strings are **immutable** means once we crate a string we cannot change its data.

Methods in Strings:

class StringMethods{

    public static void main(String[] args){

        String s = "SuMit";

        System.out.println("Length is : " + s.length());

        System.out.println("lower case is : " + s.toLowerCase());

        System.out.println("upper case is : " + s.toUpperCase());

        // str.trim() removes all leading and trailing spaces

        String nontrimmed = "              str              ";

        System.out.println("After trim : " + nontrimmed.trim());

        // This will print substring from index to index or from index

        // In this start index is included but end undex is excluded

        System.out.println("sub string is : " + s.substring(1 , 3));

        // this will return string from start index to end

        System.out.println("sub string is : " + s.substring(1));

        // str.replace(old char:'' , new char:''); used to replace old char to new char

        System.out.println("After replace is : " + s.replace('S' , 'c'));

    // str.startsWith(String:"") return boolean check wheather start with entered string or not

        System.out.println("Is start with this : " + s.startsWith("S"));

        // str.endsWith(String:"") return boolean check wheather end with entered string or not

        System.out.println("Is end with this : " + s.endsWith("g"));

        // str.charAt(index:5) most imp method used to get particular index character in String

        System.out.println("at index : " + s.charAt(2));

        String str = "My name is Sumit. Sumit Satre";

    // str.indexOf(String:"")method is used to find the index of a substring within a string

    // if not not present return -1

        System.out.println("index of : " + str.indexOf("it" ));

        // Starts finding from index 15

        System.out.println("index of : " + str.indexOf("it" , 15));

        // find at last

        System.out.println("index of : " + str.lastIndexOf("it"));

        // This find upto index 5

        System.out.println("index of : " + str.lastIndexOf("it" , 5));

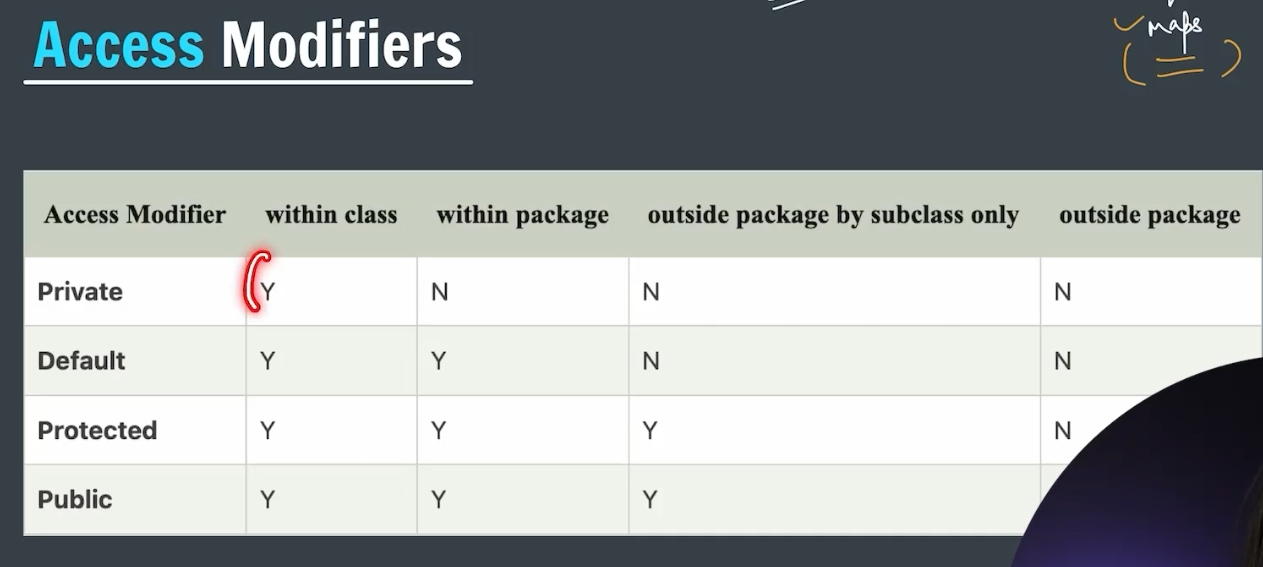
        System.out.println("is equal : " + s.equals("Sumit"));  // true == SuMit

        System.out.println("is equal : " + s.equalsIgnoreCase("sumit"));

    }

}

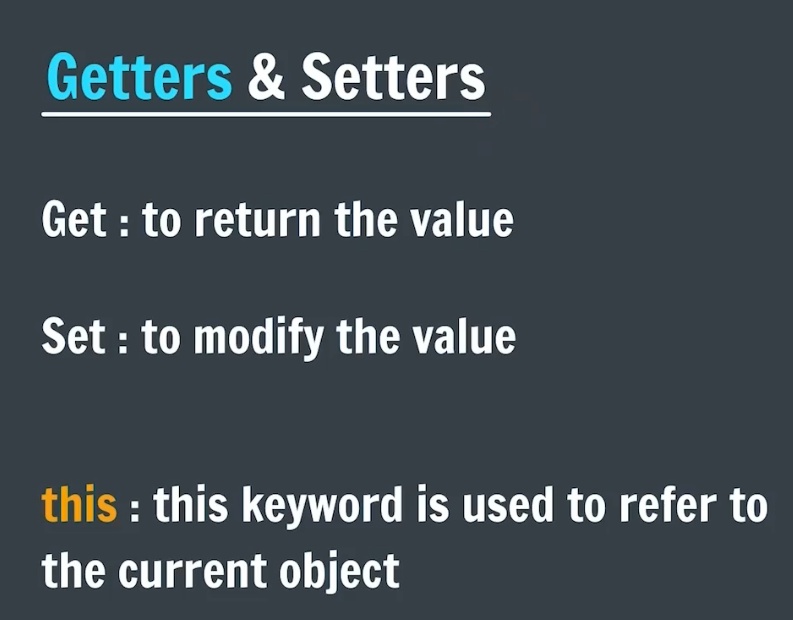
**OOP:**



Getters And Setters:

Methods used to access and modify private data and functions in a class.

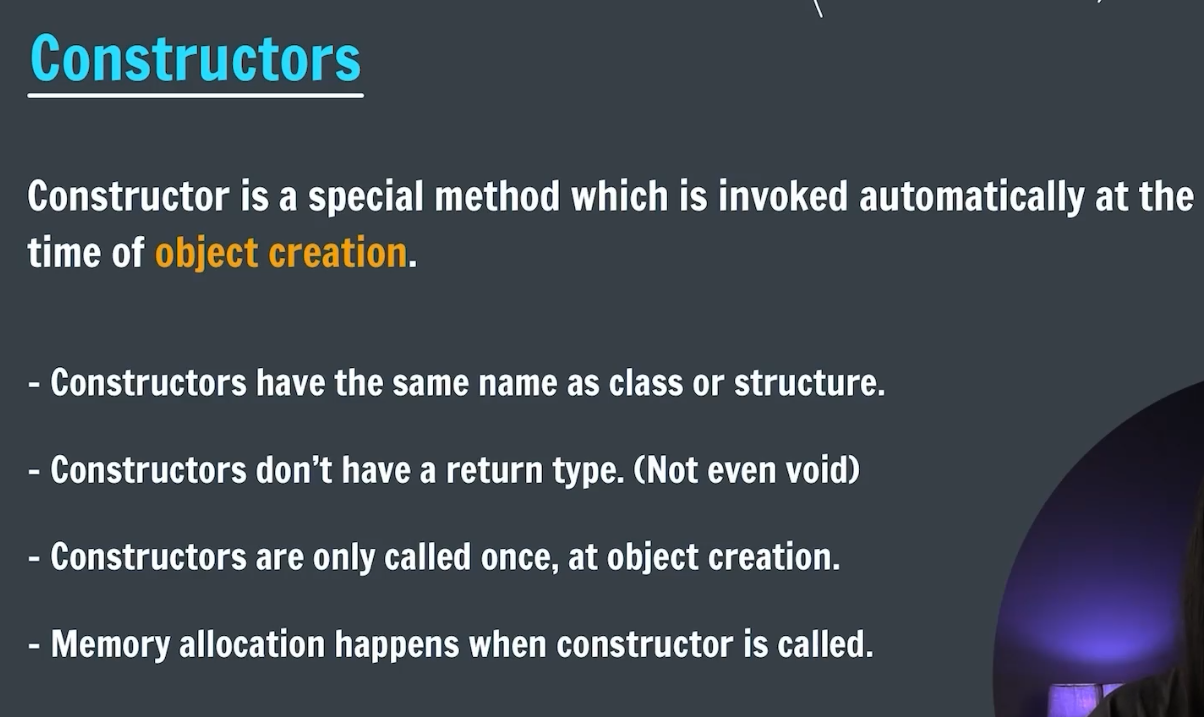
* **Accessing Private Data:** Since private data/functions can't be directly accessed, getters provide a way to retrieve them.
* **Modifying Private Data:** Similarly, setters allow changing private data safely.



**"this" Keyword in Java Classes:**

* **Purpose:** The "this" keyword in Java refers to the current object in memory.
* **Usage:** It is primarily used for:
  + **Accessing Object Members:** "this" is used to access and work with attributes and methods of the current object.
  + **Resolving Ambiguity:** When a local variable has the same name as an instance variable, "this" helps distinguish between them.
* **Example:** If you have a class with an attribute **name** and a method **setName**, you can use "this" to differentiate between the instance variable and method parameter: **this.name = name;**.
* **Clarity:** "this" improves code readability by explicitly indicating that you're working with the current object.

**Constuctors:**



**Problem:** You want to create a **Student** object in your Java code, but you haven't defined a way to create one without providing initial information. This causes an error because Java needs a method to create an empty **Student** object.

**Solution:** To solve the problem, add a simple constructor to the **Student** class that doesn't require any initial information. This way, you can create a **Student** object without specifying details right away. Here's a quick example:

**Destructors:**

In Java, there are no destructors in the same sense as in some other programming languages like C++ or C#. In those languages, destructors are used to clean up resources or perform actions when an object is being destroyed or goes out of scope.

In Java, object destruction and memory management are handled by the Java Virtual Machine (JVM) through a process called garbage collection. Developers do not have direct control over when an object is destroyed or memory is reclaimed; instead, the JVM manages this automatically.

**Inheritance in Java:**

In Java, inheritance is the mechanism through which one class inherits all the properties and behaviors of another class automatically.

* **Single Inheritance:** In Java, a class can inherit from only one base class.
* **Hierarchical Inheritance:** This occurs when multiple derived classes inherit from a single base class.(1 parent )
* **Multilevel Inheritance:** In Java, it's possible to derive a class from another derived class, creating a chain of inheritance (e.g., A -> B -> C).
* **Hybrid Inheritance:** This term describes a combination of multiple inheritance and multilevel inheritance, although pure multiple inheritance with classes is not supported in Java.

**Polymorphism in Java:**

Polymorphism in Java is the ability of different objects to respond to the same method or message in unique ways based on their individual implementations.

**Types of Polymorphism:**

* + **Compile-time (Static) Polymorphism:** Also known as method overloading, it occurs when multiple methods in the same class have the same name but different parameters (i.e., different method signatures).
  + **Run-time (Dynamic) Polymorphism:** Also known as method overriding, it happens when a subclass provides a specific implementation for a method that is already defined in its superclass. The method in the subclass must have the same name, return type, and parameters as the one in the superclass.

**Example of Compile-time Polymorphism:**

class MathOperations {

    int add(int a, int b) {

        return a + b;

    }

    double add(double a, double b) {

        return a + b;

    }

}

Here, the **add** method is overloaded with different parameter types.

class Animal {

    void makeSound() {

        System.out.println("Some sound");

    }

}

class Dog extends Animal {

    @Override

    void makeSound() {

        System.out.println("Woof!");

    }

}

When method overriding happens between child and parent class priority is given to the child class method and it will run.

**Abstraction:**

Hiding essential details and showing only essential details to the use is called as the abstraction.

We cannot create an object of an abstract class.

It can have constructor, abstract and non-abstract methods.

**Static Keyword:**

Static keyword n a java is used to share same variable or method of a given class.

We can make method , properties , blocks and nested class static.

Static block Vs non-static block:

* Static blocks are associated with the class itself and run only once when the class is loaded, regardless of how many objects of that class are created.(only runs once for class regardless how many objects we create)
* Instance blocks are associated with instances of the class and run every time a new object is created, just before the constructor is invoked for that object.

**Abstract Class:**

1. An abstract class can have both abstract (methods without implementation) and concrete (methods with implementation) methods.
2. It can also have instance variables (fields).
3. It allows constructors.
4. You can extend only one abstract class.
5. It supports the concept of code reusability through inheritance.
6. It may have access modifiers for its methods (public, protected, private, etc.).
7. It can have method implementations that provide a default behavior.

**Interface:**

1. An interface can only have abstract methods (methods without implementation) by default, but in Java 8 and later, it can have default and static methods with implementations.
2. It can't have instance variables (fields), only constants (public static final fields).
3. It doesn't allow constructors.
4. You can implement multiple interfaces in a single class.
5. It supports the concept of multiple inheritance by allowing a class to implement multiple interfaces.
6. All interface methods are implicitly public and abstract (except for default and static methods).
7. It does not provide method implementations; it defines a contract that implementing classes must adhere to.

Shortcut Imp:

To get the efficient XOR of 1 to n elements in Java, you can use the following steps:

1. Find the remainder of n when it is divided by 4.
2. If the remainder is 0, then the XOR of all numbers from 1 to n is n.
3. If the remainder is 1, then the XOR of all numbers from 1 to n is 1.
4. If the remainder is 2, then the XOR of all numbers from 1 to n is n + 1.
5. If the remainder is 3, then the XOR of all numbers from 1 to n is 0.

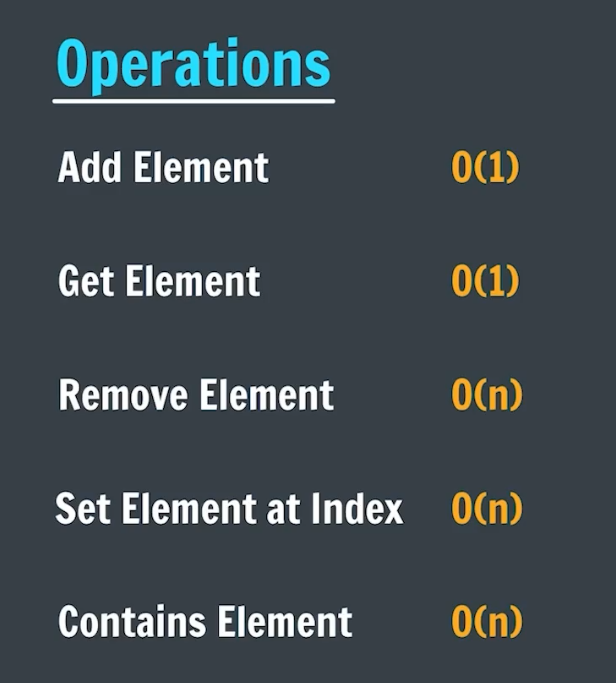
Here is an example of how to use this approach in Java:

public class XorCalculator {  
  
 public int getXor(int n) {  
 int remainder = n % 4;  
 switch (remainder) {  
 case 0:  
 return n;  
 case 1:  
 return 1;  
 case 2:  
 return n + 1;  
 case 3:  
 return 0;  
 default:  
 throw new IllegalArgumentException("Invalid input n: " + n);  
 }  
 }  
}

This algorithm is efficient because it only requires a single modulo operation and a simple switch statement.

ArrayList:



****

How to create ArrayList?

import java.util.ArrayList;

class CreateArraylist{

    public static void main(String [] args){

        ArrayList<Integer> list = new ArrayList<>();

        ArrayList<Boolean> list2 = new ArrayList<>();

        ArrayList<Float> list3 = new ArrayList<>();

        ArrayList<String> list4 = new ArrayList<>();

        ArrayList<Long> list5 = new ArrayList<>();

        ArrayList<Short> list6 = new ArrayList<>();

        ArrayList<Double> list7 = new ArrayList<>();

    }

}

**Two Pointer Approach:**

Bitwise Operators:

**Linked List:**

In Java, both instance variables and reference variables are used within classes, but they serve different purposes and have different characteristics. Here's the difference between instance variables and reference variables:

1. **Instance Variables**:
   * Instance variables are declared within a class but outside any method, constructor, or block.
   * They are also known as member variables or fields.
   * Each object (instance) of the class has its own copy of instance variables, which are distinct from those of other objects of the same class.
   * Instance variables define the state of each object and hold specific data unique to each object.
   * They are accessed using the dot notation (**objectName.variableName**) from within instance methods or constructors of the class or by using the **this** keyword to refer to the current object's instance variables.
2. **Reference Variables**:
   * Reference variables are variables that hold references (memory addresses) to objects.
   * They are declared with the class name followed by the variable name and can hold references to objects of that class or its subclasses.
   * Unlike instance variables, reference variables are not part of an object's state; instead, they store the address (reference) of an object in memory.
   * Reference variables are used to access and manipulate objects by calling their methods or accessing their instance variables.
   * When an object is created using the **new** keyword, a reference variable is used to store the memory address of the newly created object.

In summary, instance variables are part of the state of an object and hold specific data for each object, whereas reference variables store memory addresses (references) to objects and are used to access and manipulate objects in Java.