JAVA

INTRODUCTION

Introduction

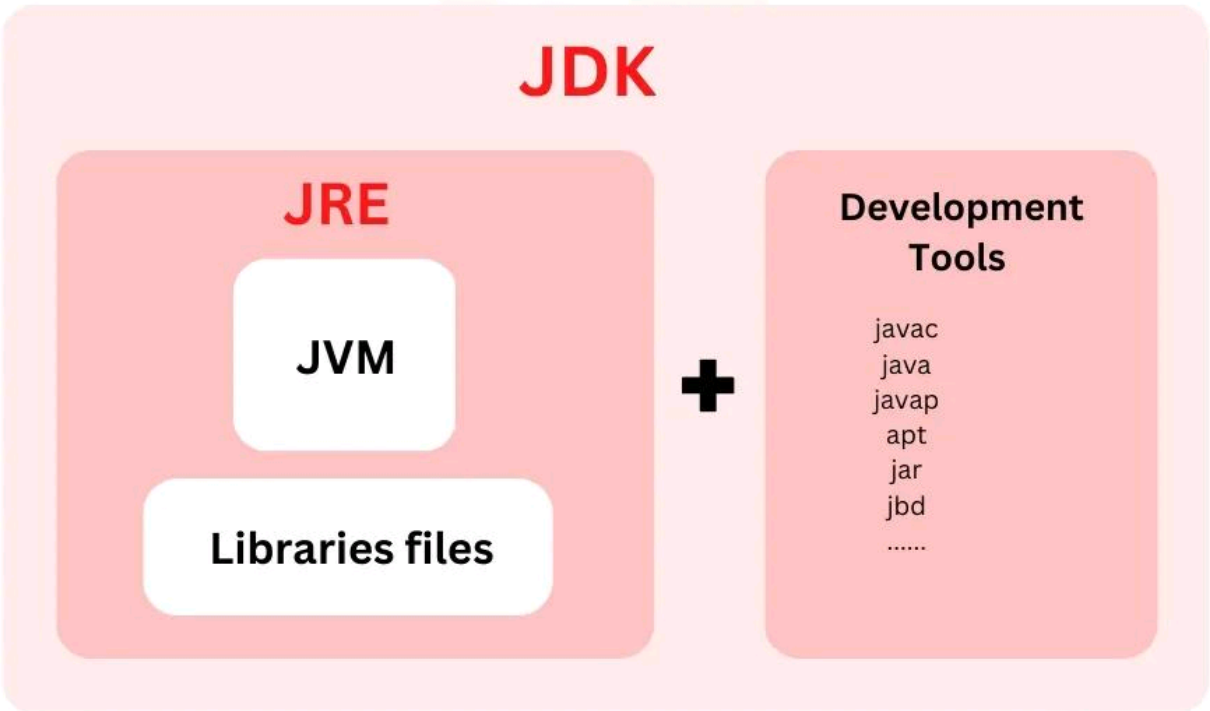
Java is an object-oriented, class-based, secured, platform-independent, and general-purpose programming language. Java was originally developed by **James Gosling** at **Sun Microsystems** and released in **1995** as a core component of Sun Microsystem's Java platform.

Why Java programming language is named JAVA ?

The language was initially called Oak after an oak tree that stood outside Gosling's office. Later the project went by the name Green and was finally renamed Java, from Java coffee, a type of coffee from Indonesia.

Java Terminology

JDK, **JRE**, and **JVM** are the most important parts of the Java programming language. Without these, you can not develop and run java programs on your machine.



JDK: JDK stands for Java Development Kit. JDK provides an environment to develop and execute the java program. JDK is a kit that includes two things - Development Tools to provide an environment to develop your java programs and JRE to execute your Java programs.

JRE: JRE stands for Java Runtime Environment. JRE provides an environment to only run (not develop) the java programs onto your machine. JRE is only used by the end-users of the system. JRE consists of libraries and other files that JVM uses at runtime.

JVM: JVM stands for Java Virtual Machine, which is a very important part of both JDK and JRE because it is inbuilt in both. Whatever java program you run using JDK and JRE goes into the JVM and JVM is responsible for executing the java program line by line.

Features and Uses of Java

- Simple
- Secure
- Robust
- Portable
- Distributed
- Multithreaded
- Object-Oriented
- High Performance
- Dynamic flexibility
- Platform independent

JAVA

HELLO WORLD

Java is one of the most popular and widely used programming languages and platforms. 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.

The process of Java programming can be simplified in three steps:

- Create the program by typing it into a text editor and saving it to a file – **HelloWorld.java**.
- Compile it by typing **“javac HelloWorld.java”** in the terminal window.
- Execute (or run) it by typing **“java HelloWorld”** in the terminal window.

Hello World

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

Let's understand of the code.

- **Class definition**

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

- **public:** So that JVM can execute the method from anywhere.

- **static:** The main method is to be called without an 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. The main method must be inside the class definition. The compiler executes the codes starting always from the main function.

- **String[]:** The main method accepts a single argument, i.e., an array of elements of type String.

- **System.out.println :** This is used to print anything on the console like printf in C language.

JAVA
VARIABLES

A **Variable** is a name given to a memory location. It is used to store a value that may vary. Java is a statically typed language, and hence, all the variables are declared before use.

Variable Declaration

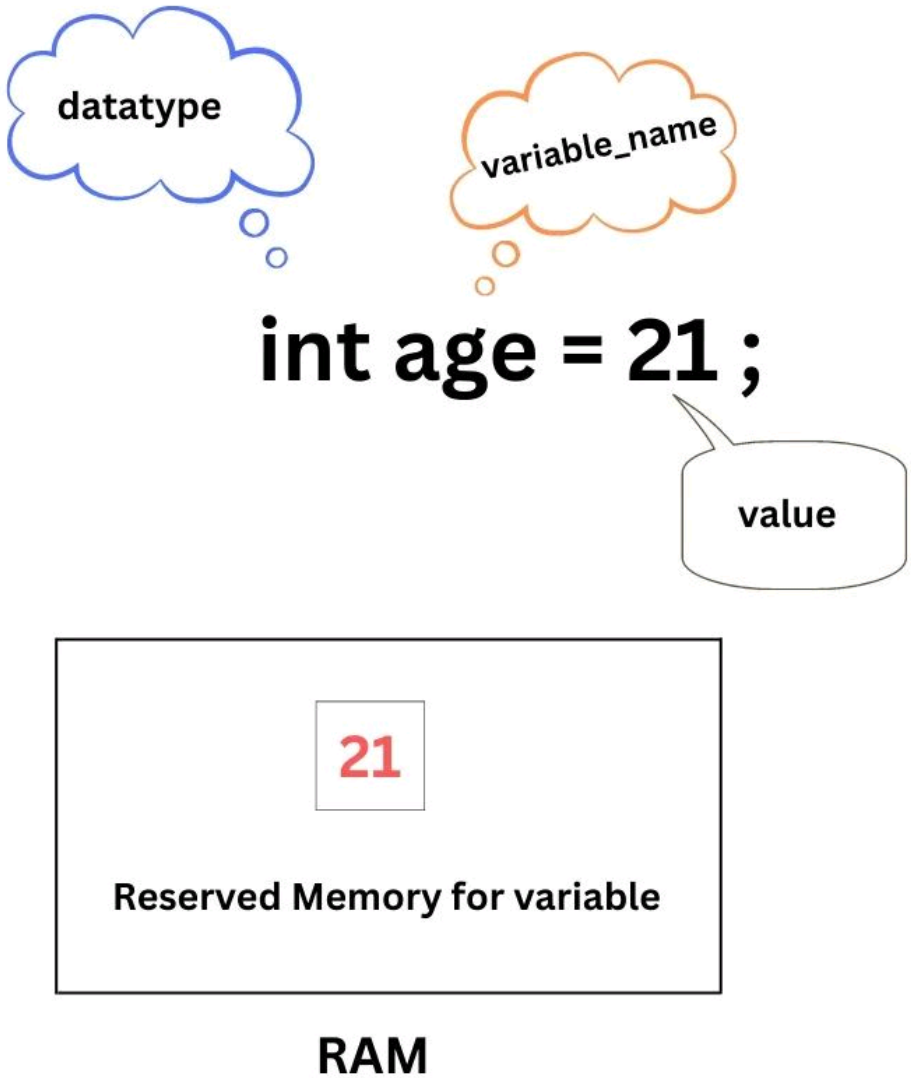
1. datatype: Type of data that can be stored in this variable.
2. data_name: Name given to the variable.

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

- Variable Initialization
- Assigning value by taking input

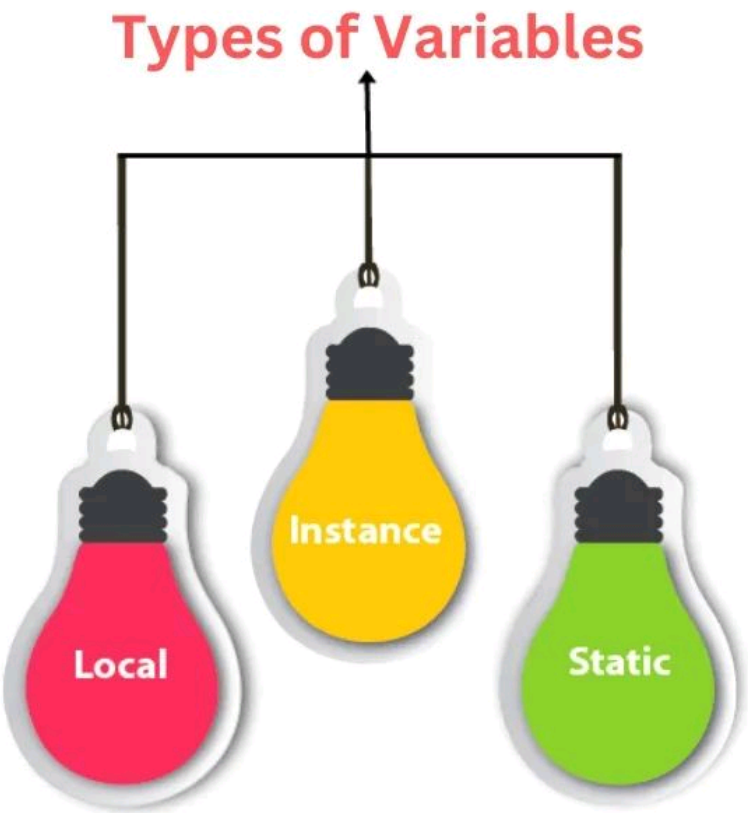
Variable Initialization

- **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.



Variables naming in Java

- A variable name should be short and meaningful.
- It should begin with a lowercase letter.
- It can begin with special characters such as _ (underscore) and **$** (dollar) sign.
- If the variable name contains multiple words, then use the camel case, i.e. variable name should start with a lowercase letter followed by an uppercase letter. For eg: iamRupnath, instaGram.
- Always try to avoid single character variable names such as i, j, and k except for the temporary variables.
- A variable name can not contain whitespaces.
- We can’t use keywords(pre-defined literals) as the variable names.



1. Local Variables

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

```java
import java.io.*;

class rupnath {
    public static void main(String[] args)
    {
        int var = 102; // Local Variable

        System.out.println("Local Variable: " + var);
    }
}
```

2. Instance Variables

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.
It is called an instance variable because its value is instance-specific and is not shared among instances.

```java
class Student {

    String name; // instance variables
    int rollno;
}

public class StudentRecords {
    public static void main(String args[]) {

            Student obj = new Student(); // Creating Student class object

            obj.name = "Rupnath";  // Assigning values in the variables
            obj.rollno = 102;

            System.out.println(obj.name);
            System.out.println(obj.rollno);
    }
}
```

3. Static Variables

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

```java
class Student {


    // static variables
    public static int rollno;
    public static String name = "Ram";
}


public class StudentDemo {
    public static void main(String args[])
    {

        // accessing static variable without creating object
        Student.rollno = 10;
        System.out.println(Student.name + " 's rollno is :" + Student.rollno);
    }
}
```

JAVA

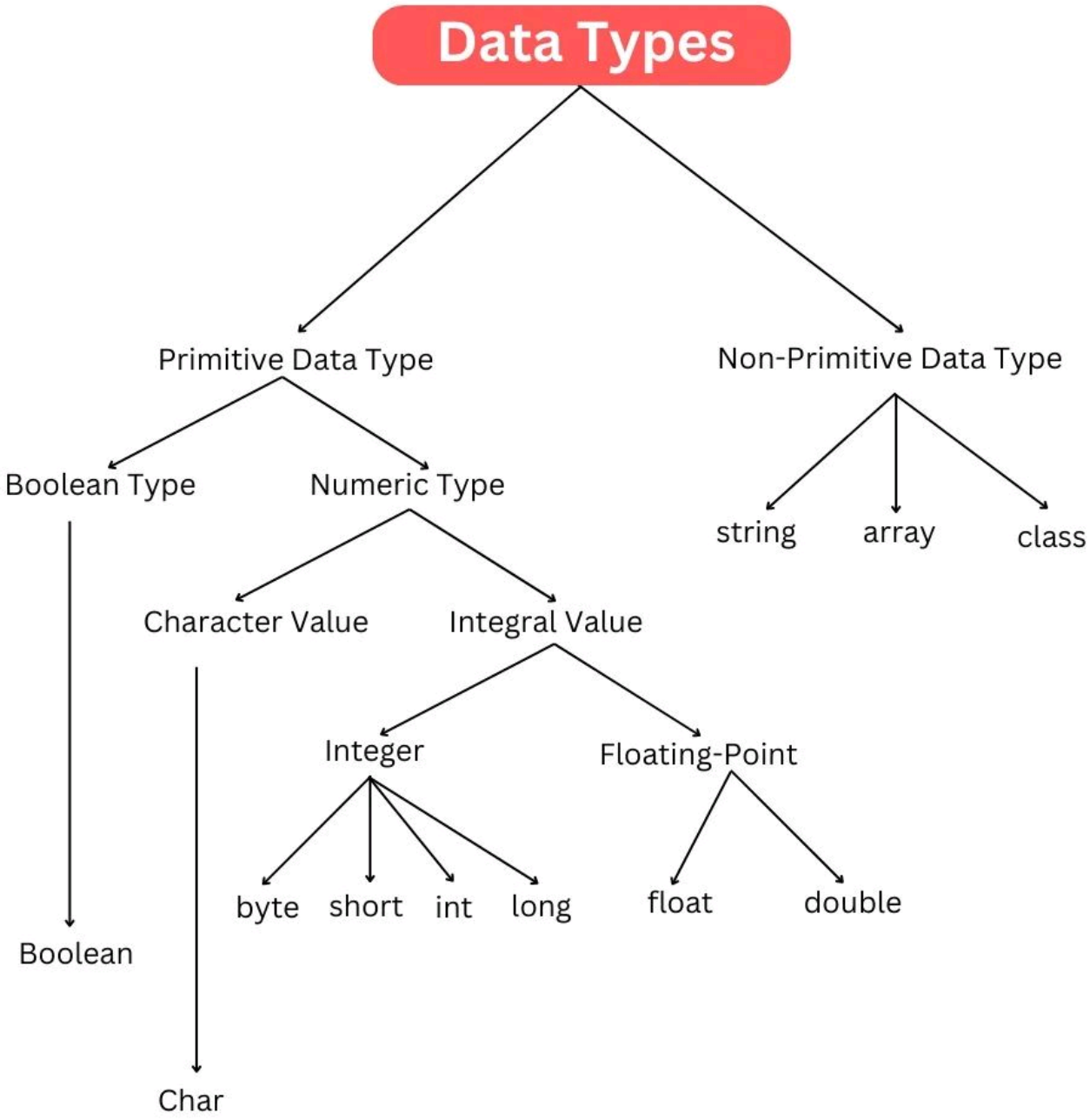
DATA TYPES

Data types specify the different sizes and values that can be stored in the variable.

There are two types of data types -

Primitive Data Type

Non-Primitive Data Type


Data Types
Primitive Data Type
Non-Primitive Data Type
Boolean Type
Numeric Type
string
array
class
Character Value
Integral Value
Integer
Floating-Point
byte
short
int
long
float
double
Boolean
Char

Data Types

Type	Default Value	Default size	RANGE OF VALUES
boolean	false	1 bit	true, flase
char	'\u0000'	2 byte	character representation of ASCII values 0 to 255
byte	0	1 byte	-128 to 127
short	0	2 byte	-32,768 to 32,767
int	0	4 byte	-2,147,483,648 to 2,147,483,647
long	0L	8 byte	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	0.0f	4 byte	upto 7 decimal digits
double	0.0d	8 byte	upto 16 decimal digits

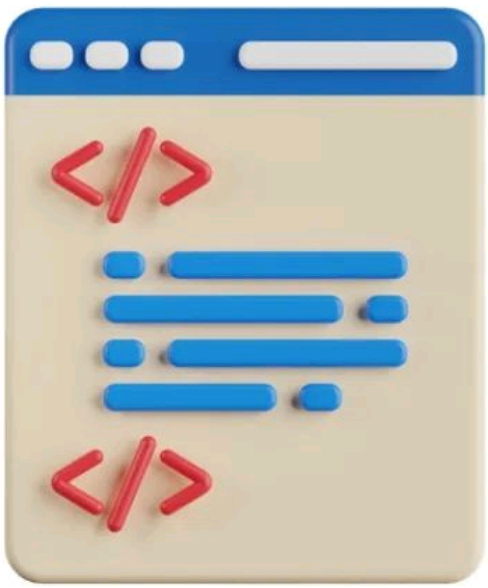
Example:

```java
class rupnath {

    public static void main(String args[])
    {

        char a = 'G';
        int i = 89;
        byte b = 4;
        short s = 56;
        double d = 4.355453532;
        float f = 4.7333434f;
        long l = 12121;

        System.out.println("char: " + a);
        System.out.println("integer: " + i);
        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
        System.out.println("long: " + l);
    }
}
```

JAVA
OPERATORS

Operators are symbols that perform operations on variables and values.

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



Java Operator Precedence

Operator Type	Category	Precedence	Associativity
Unary	**postfix**	**a++, a--**	**Right to left**
	prefix	**++a, --a, +a, -a, ~, !**	**Right to left**
Arithmetic	**Multiplication**	***, /, %**	**Left to Right**
	Addition	**+, -**	**Left to Right**
Shift	**Shift**	**<<, >>, >>>**	**Left to Right**
Relational	**Comparison**	**<, >, <=, >=, instanceOf**	**Left to Right**
	equality	**==, !=**	**Left to Right**
Bitwise	**Bitwise AND**	**&**	**Left to Right**
	Bitwise exclusive OR	**^**	**Left to Right**
	Bitwise inclusive OR	**|**	**Left to Right**
Logical	**Logical AND**	**&&**	**Left to Right**
	Logical OR	**||**	**Left to Right**
Ternary	**Ternary**	**? :**	**Right to Left**
Assignment	**assignment**	**=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=**	**Right to Left**

Java Unary Operator

Example: ++ and --

```java
public class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
  }
}
```

Example: ~ and !

```java
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);//-11 (minus of total positive value which starts from 0)
System.out.println(~b);//9 (positive of total minus, positive starts from 0)
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true
    }
}
```

Java Arithmetic Operators

```java
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
    }
}
```

Java Shift Operator

```java
public class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>>2);
    }
}
```

Java Logical Operators

```
public class OperatorExample{
public static void main(String args[]){
int a, b, c;
a=10; b=5; c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a>b||a<c);//true || true = true
    }
}
```

Java Bitwise Operator

```
public class OperatorExample{
public static void main(String args[]){
int a, b, c;
a=10; b=5; c=20;
System.out.println(a<b&a<c);//false & true = false
System.out.println(a>b|a<c);//true | true = true
System.out.println(a>b^a<c);
    }
}
```

Java Ternary Operators

```java
public class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
    }
}
```

Java Assignment Operator

```java
class Main {
  public static void main(String[] args) {
    int a = 4;
    int var;
    var = a; // assign value using =
    System.out.println("var using =: " + var);
    var += a; // assign value using =+
    System.out.println("var using +=: " + var);
    var *= a; // assign value using =*
    System.out.println("var using *=: " + var);
  }
}
```

Find the output 3

```java
public class Solution {
    public static void main(String args[]) {
        int a = 10, b = 1;
        System.out.println(!(a < b));
    }
}
```

A) true

B) false

C) Compilation error

D) Runtime error

Find the output 4

```java
public class Solution {

    public static void main(String args[]) {
        int num = 18;
        num %= 2;
        System.out.println(num);
    }
}
```

A) 0

B) 18

C) 9

D) Compile time error

JAVA

IF-ELSE

A programming language uses control statements to control the flow of execution of a 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.

- **if**
- **if-else**
- **nested-if**
- **if-else-if**

if statements

The Java if statement tests the condition. It executes the if block if condition is true.

Syntax

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

Example -

```java
class IfStatement {
  public static void main(String[] args) {

    int number = 10;
    // checks if number is less than 0
    if (number < 0) {
      System.out.println("The number is negative.");
    }
    System.out.println("Statement outside if block");
  }
}
```

if -else statements

The if-else statement executes a block of code if a specified condition is true. If the condition is false, another block of code can be executed.

Syntax

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

Example -

```java
class Main {
  public static void main(String[] args) {

 int i = 10;

        if (i < 15)
            System.out.println("i is smaller than 15");
        else
            System.out.println("i is greater than 15");
    }
}
```

nested-if statements

The nested if statement represents the if block within another if block. Here, the inner if block condition executes only when outer if block condition is true.

Syntax

```
if (condition 1) {
    //code to be executed
        if (condition 2) {
          //code to be executed
  }
}
```

Example -

```java
class Main {
  public static void main(String[] args) {
        int i = 10;
        if (i == 10 || i<15) {
            if (i < 15)
                System.out.println("i is smaller than 15 ");
            if (i < 12)
                System.out.println(
                    "i is smaller than 12 too");
        } else{
                System.out.println("i is greater than 15");
        };
    }
}
```

if-else-if statements

The if-else-if ladder statement executes one condition from multiple statements.

Syntax

```
if (condition 1)
    statement;
else if (condition 2)
    statement;
. . .
else
    statement;
```

Example -

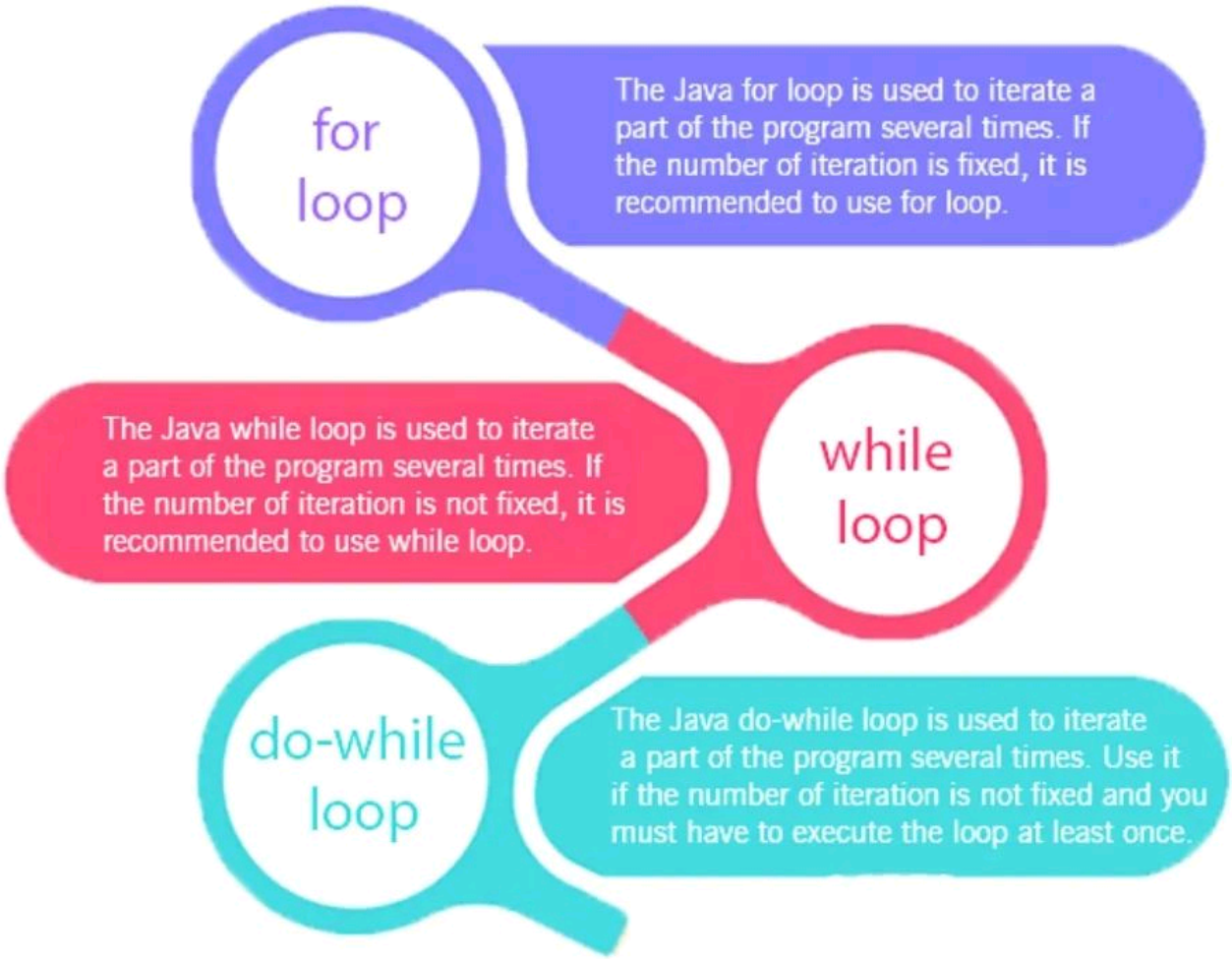
```java
class Main {
  public static void main(String[] args) {
          int i = 20;

        if (i == 10)
            System.out.println("i is 10");
        else if (i == 15)
            System.out.println("i is 15");
        else if (i == 20)
            System.out.println("i is 20");
        else
            System.out.println("i is not present");
    }
}
```

JAVA

LOOPS

The Java for **loop** is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are **three types** of for loops in Java.



For Loop

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.

Syntax

```
for(initialization; condition; increment/decrement)
{
//statement or code to be executed
}
```

Initialization: The initialization initializes and/or declares variables and executes only once.

Condition: The condition is evaluated. If the condition is true, the body of the for loop is executed.

Increment/Decrement: It increments or decrements the variable value. It is an optional condition.

Statement: The statement of the loop is executed each time until the second condition is false.

Example:

```java
public class ForExample {
public static void main(String[] args) {
    //Code of Java for loop
    for(int i=1;i<=10;i++){
        System.out.println(i);
         }
    }
}
```

While Loop

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax

```
while (test_expression)
{
//code to be executed
 update_expression;
}
```

Test Expression: In this expression, we have to test the condition. If the condition evaluates to true then we will execute the body of the loop and go to update expression. Otherwise, we will exit from the while loop.

Example: `i <= 10`

Update Expression: After executing the loop body, this expression increments/decrements the loop variable by some value.

Example: `i++ ;`

Example:

```
class Main {
  public static void main(String[] args) {

    // declare variables
    int i = 1, n = 5;

    // while loop from 1 to 5
    while(i <= n) {
      System.out.println(i);
      i++;
    }
  }
}
```

do-while Loop

do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.

Syntax

```
do {
  //code to be executed / loop body
  //update statement
} while (condition)
```

Condition: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. As soon as the condition becomes false, loop breaks automatically.

Example: `i <= 100`

Update Expression: Every time the loop body is executed, the this expression increments or decrements loop variable.

Example: `i++ ;`

Example:

```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;

    do{
        System.out.println(i);
    i++;
    }while(i<=10);

     }
}
```

Infinite Loop

One of the most common mistakes while implementing any sort of looping is that it may not ever exit, that is the loop runs for infinite time. This happens when the condition fails for some reason.

Infinite for loop

```java
public class ForExample {
public static void main(String[] args) {
    //Using no condition in for loop
    for(;;){
        System.out.println("infinitive loop");
      }
    }
}  // Stop ⟶ ctrl+c
```

Infinite while loop

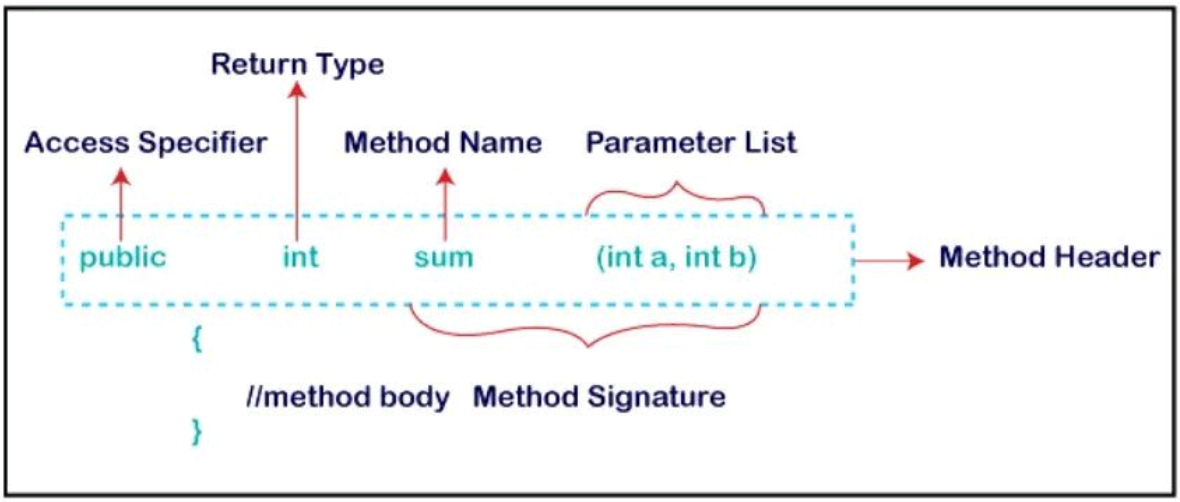
```java
public class WhileExample {
public static void main(String[] args) {
 // setting the infinite while loop by passing true to
the condition
    while(true){
        System.out.println("infinitive while loop");
      }
    }
}
```

JAVA

FUNCTIONS / METHOD

A **method** is a block of code that performs a specific task.

Method Declaration



1. Modifier: It defines the access type of the method i.e. from where it can be accessed in your application. In Java, there 4 types of access specifiers.

- **public:** It is accessible in all classes in your application.
- **protected:** It is accessible within the class in which it is defined and in its subclass.
- **private:** It is accessible only within the class in which it is defined.

- **default:** It is declared/defined without using any modifier. It is accessible within the same class and package within which its class is defined.

2. The return type: The data type of the value returned by the method or void if does not return a value.

3. Method Name: the rules for field names apply to method names as well, but the convention is a little different.

4. Parameter list: Comma-separated list of the input parameters is defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ( ).

5. Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the method name and parameter list.

6. Method body: it is enclosed between braces. The code you need to be executed to perform your intended operations.

Types of Methods in Java

- **User-defined Methods:** We can create our own method based on our requirements.
- **Predefined Method:** These are built-in methods in Java that are available to use.

Method Calling

we have declared a method named **addNumbers( ).** Now, to use the method, we need to call it.

```
class Main {
  // create a method
  public int addNumbers(int a, int b) {
    int sum = a + b;
    return sum;  // return value
  }
  public static void main(String[] args) {
    int num1 = 25;
    int num2 = 15;
    Main obj = new Main();  // create an object of Main
    int result = obj.addNumbers(num1, num2);  // calling method
    System.out.println("Sum is: " + result);
  }
}
```

Method Return Type

A Java method may or may not return a value to the function call. We use the **return statement** to return any value.

```java
class Main {
// create a method
  public static int square(int num) {
    // return statement
    return num * num;
  }
  public static void main(String[] args) {
    int result;
    // call the method
    // store returned value to result
    result = square(10);
    System.out.println("Squared value of 10 is: " + result);
  }
}
```

Method Parameters

```java
// method with two parameters
int addNumbers(int a, int b) {
  // code
}

// method with no parameter
int addNumbers(){
  // code
}
```

```java
class Main {
  // method with no parameter
  public void display1() {
    System.out.println("Method without parameter");
  }
  // method with single parameter
  public void display2(int a) {
    System.out.println("Method with a single parameter: " + a);
  }
  public static void main(String[] args) {
    Main obj = new Main();  // create an object of Main
    obj.display1();  // calling method with no parameter
    obj.display2(24);  // calling method with the single parameter
  }
}
```

Predefined Method

Predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc.

```java
public class Main {
  public static void main(String[] args) {
    // using the sqrt() method
    System.out.print("Square root of 4 is: " + Math.sqrt(4));
  }
}
```

What are the advantages of using methods?

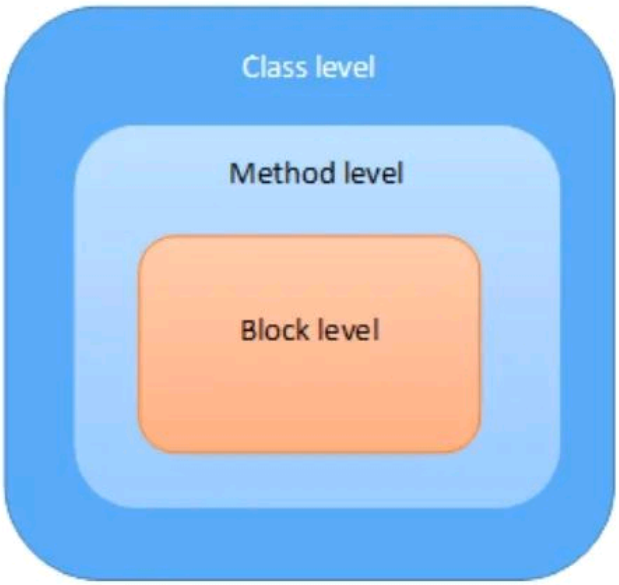
The main advantage is **code reusability.** We can write a method once, and use it multiple times. We do not have to rewrite the entire code each time. Think of it as, "write once, reuse multiple times".

```java
public class Main {
  // method defined
  private static int getSquare(int x){
    return x * x;
  }
  public static void main(String[] args) {
    for (int i = 1; i <= 5; i++) {
      int result = getSquare(i);  // method call
      System.out.println("Square of " + i + " is: " + result);
    }
  }
}
```

JAVA

METHOD & SCOPE

The **scope** tells the compiler about the segment within a program where the variable is accessible or used.

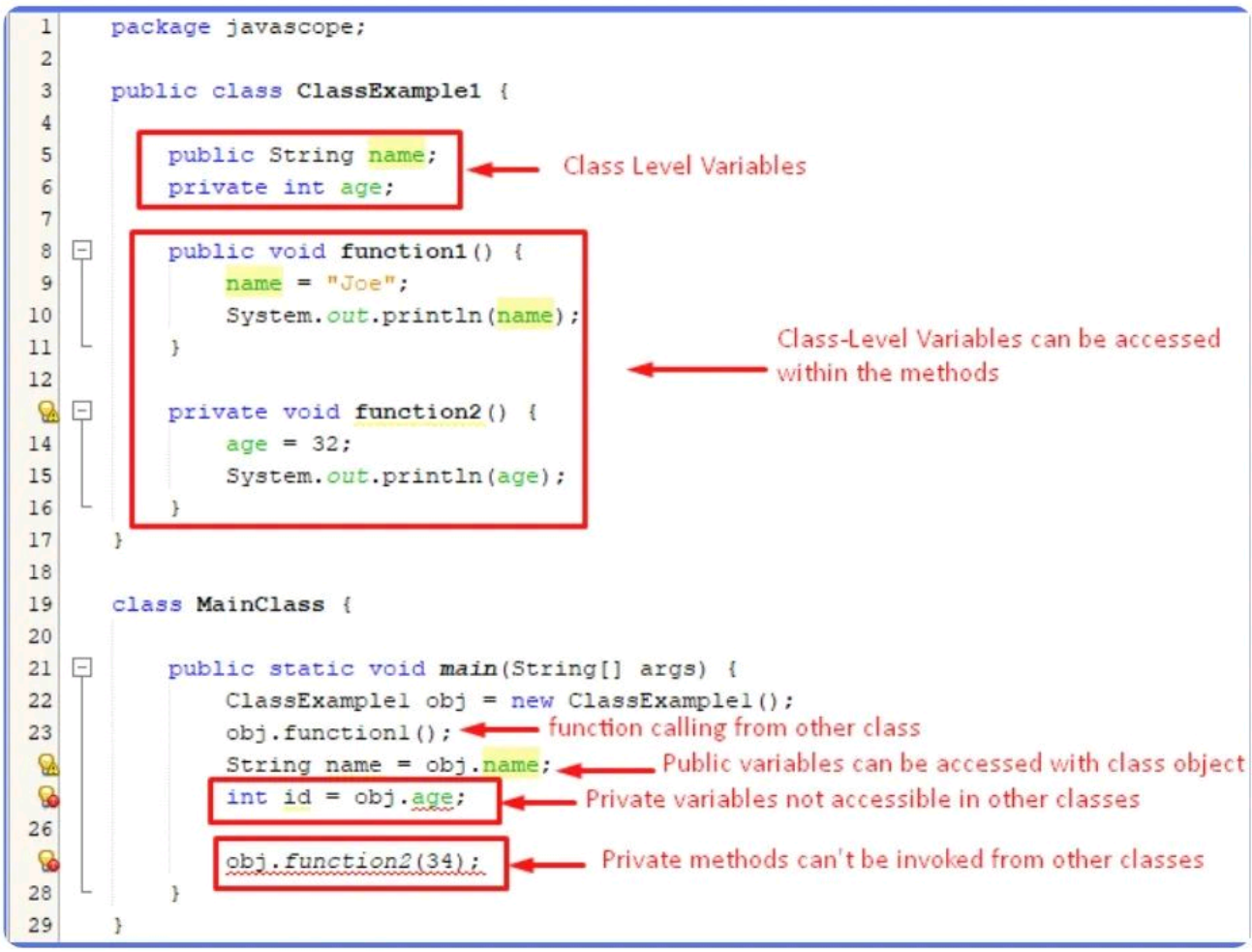


Class-level Scope

The variables declared inside a class can be accessed by all the functions in that class depending on its access modifier/specifier i.e. public, private, etc.

	Within Same Class	Within same package	Outside the package-(Subclass)	Outside the package-(Global)
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes (only to derrived class)	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

Example -->



Method-level Scope

These are the variables that are declared inside the method and cannot be accessed outside the method.

Example -->

```
3   public class MethodScope {
4
5       public void funAge(){
6           int age = 24;
7       }
8       public void print(){
9           System.out.println(age);
10
11          funAge();
12      }
13      public static void main(String[] args) {
14
15
16      }
17  }
```

Variable created within "funAge()" method

Error

calling funAge() method

Block-level Scope

These are the variables that are declared inside the pair of brackets ' { ' and ' } ' .

```
4
5       public static void main(String[] args) {
6
7           String name = "John";
8           {
9               int id = 0;
10
11              for (int i = 0; i <= 5; i++) {
12
13                  id++;
14                  if (id == 4) {
15
16                      System.out.println("id: " + id);
17                      System.out.println("name: " + name);
18                  }
19
20              }
21
22          }
23      }
24  }
```

Variable Declared at Method Level

Block Level Starts Here

Accessing the Method level Variable in Block level

Block Level ends Here

Call by Value and Call by Reference in Java

There is only **call by value in java, not call by reference.** If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example -->

In case of call by value original value is not changed.

```
class Operation{
 int data=50;

 void change(int data){
 data=data+100;//changes will be in the local variable only
 }
 public static void main(String args[]){
   Operation op=new Operation();
   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);
 }
}


//  Output:before change 50
//  after change 50
```

Example -->

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value.

```
class Operation2{
 int data=50;

 void change(Operation2 op){
 op.data=op.data+100; //changes will be in the instance variable
 }

 public static void main(String args[]){
   Operation2 op=new Operation2();

   System.out.println("before change "+op.data);
   op.change(op);//passing object
   System.out.println("after change "+op.data);

 }
}


//  Output:before change 50
//  after change 150
```

JAVA

ARRAYS

An **Array** is a data structure use to store a collection of data. The elements of an array are stored in a **contiguous memory location.**

Why do we need Arrays?

Arrays are used **to store multiple values in a single variable,** instead of declaring separate variables for each value.

Declaration of Arrays

Syntax

```
dataType[ ] arrayName;
```

Example :

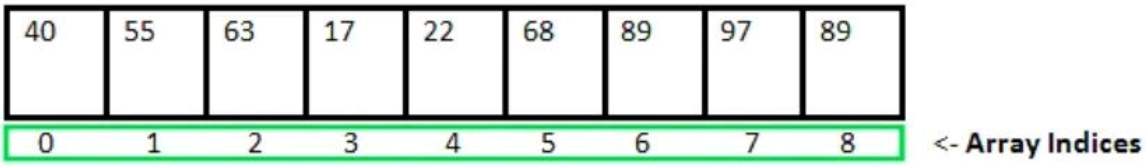
```
int[ ] arr;
OR
int arr[ ];
```

Creating An Array

```
int [ ] arr = new int [ 10 ];
```

- `int` — type of each element
- `arr` — name of array
- `10` — subscript (integer or constant expresssion for number of elements.)

Initializing An Array



Array Length = 9
First Index = 0
Last Index = 8

Access Array Elements

```java
class Main {
 public static void main(String[] args) {

   // create an array
   int[] age = {40,55,63,17,22};

   // access each array elements
   System.out.println("Accessing Elements of Array:");
   System.out.println("First Element: " + age[0]);
   System.out.println("Second Element: " + age[1]);
   System.out.println("Third Element: " + age[2]);
   System.out.println("Fourth Element: " + age[3]);
   System.out.println("Fifth Element: " + age[4]);
 }
}
```

Compute Sum and Average of Array Elements

```
class Main {
    public static void main(String[] args) {

    int[ ] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
    int sum = 0;
    Double average;

  // access all elements using for each loop
  // add each element in sum
       for (int number: numbers) {
       sum += number;
  }

  // get the total number of elements
       int arrayLength = numbers.length;

  // calculate the average
  // convert the average from int to double
       average =  ((double)sum / (double)arrayLength);

       System.out.println("Sum = " + sum);
       System.out.println("Average = " + average);
     }
}
```

Passing Array to a Method

```
class Testarray{
//creating a method which receives an array as a
//parameter
      static void min(int arr[ ]){
      int min=arr[0];
      for(int i=1;i<arr.length;i++)
      if(min>arr[i])
      min=arr[i];

      System.out.println(min);
 }

     public static void main(String args[]){

         //declaring and initializing an array
     int a[ ]={33,3,4,5};

        //passing array to method
    min(a);

      }
}
```

ArrayIndexOutOfBoundsException

```
public class TestArrayException{

public static void main(String args[]){

int arr[]={50,60,70,80};
for(int i=0;i<=arr.length;i++){

System.out.println(arr[i]);
            }
    }
}
```

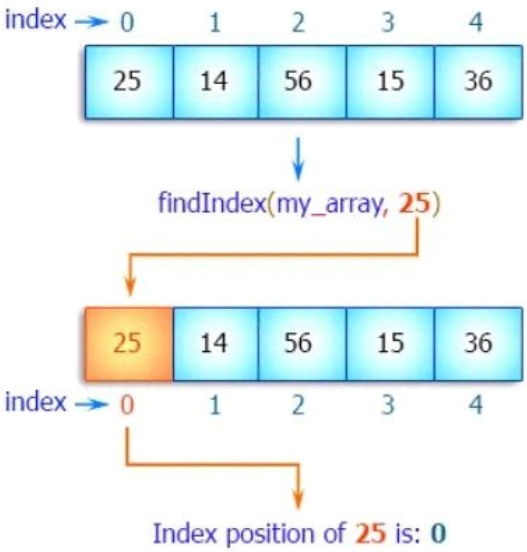
Output:

```
Exception in thread "main"
    java.lang.ArrayIndexOutOfBoundsException: 4
    at TestArrayException.main(TestArrayException.java:5)
50
60
70
80
```

JAVA

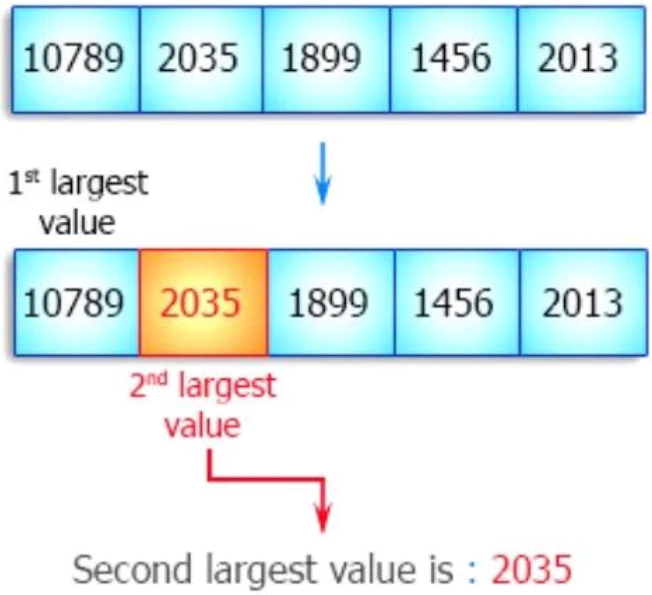
ARRAY PROBLEMS

Write a Java program to find the index of an array element.



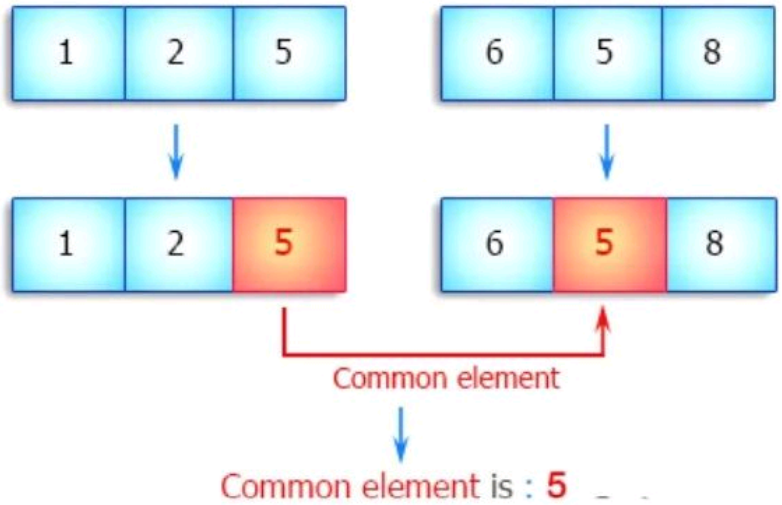

```
public class Main {
      public static int  findIndex (int[] my_array, int t) {
       if (my_array == null) return -1;
       int len = my_array.length;
       int i = 0;
       while (i < len) {
          if (my_array[i] == t) return i;
          else i=i+1;
       }
       return -1;
    }
    public static void main(String[] args) {
      int[] my_array = {25, 14, 56, 15, 36, 56, 77, 18, 29, 49};
      System.out.println("Index position of 25 is: " +
findIndex(my_array, 25));
      System.out.println("Index position of 77 is: " +
findIndex(my_array, 77));
      }
}
```

Write a Java program to find the second largest element in an array.



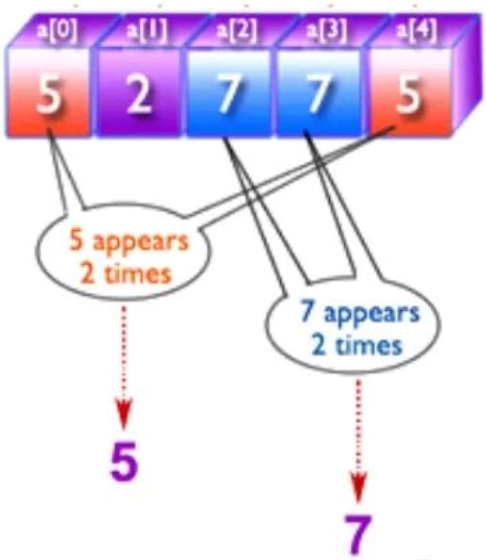

```
import java.util.Arrays;
public class Main {
public static void main(String[] args) {
int[] my_array = {
        10789, 2035, 1899, 1456, 2013,
        1458, 2458, 1254, 1472, 2365,
        1456, 2165, 1457, 2456};
System.out.println("Original numeric array :
"+Arrays.toString(my_array));
Arrays.sort(my_array);
int index = my_array.length-1;
while(my_array[index]==my_array[my_array.length-1]){
index--;
}
System.out.println("Second largest value: " +
my_array[index]);
}
}
```

Write a Java program to find the common elements between two arrays of integers.



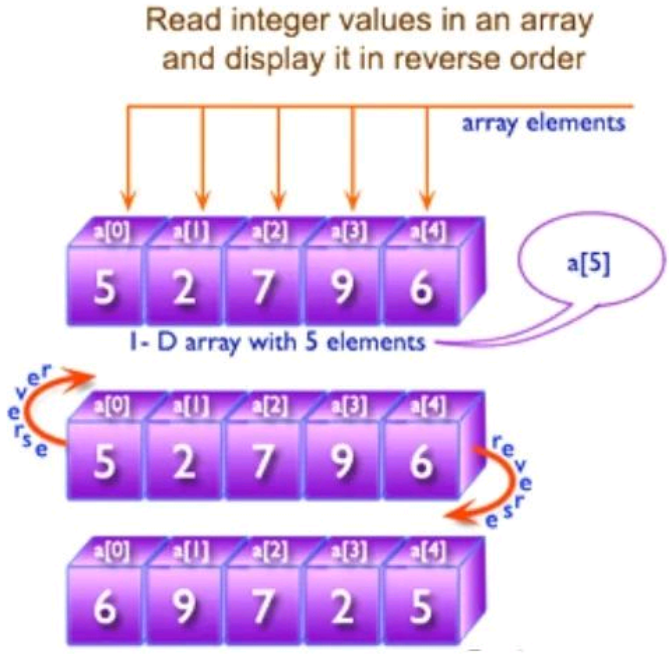

```
import java.util.Arrays;
public class Main {
public static void main(String[] args) {
    int[ ] array1 = {1, 2, 5, 5, 8, 9, 7, 10 };
    int[ ] array2 = {1, 0, 6, 15, 6, 4, 7, 0 };
     System.out.println("Array1 : "+Arrays.toString(array1));
     System.out.println("Array2 : "+Arrays.toString(array2));
      for (int i = 0; i < array1.length; i++) {
         for (int j = 0; j < array2.length; j++) {
            if(array1[i] == (array2[j] ) ) {
             System.out.println("Common element is : "+(array1[i]));
             }
         }
      }
   }
}
```

Write a Java program to find the duplicate values of an array of integer values.




```
import java.util.Arrays;
public class Main {
 public static void main(String[] args)
  {
     int[] my_array = {1, 2, 5, 5, 6, 6, 7, 2};

     for (int i = 0; i < my_array.length-1; i++)
     {
        for (int j = i+1; j < my_array.length; j++)
        {
           if ((my_array[i] == my_array[j]) && (i != j))
           {
              System.out.println("Duplicate Element :
"+my_array[j]);
           }
        }
     }
  }
}
```

Write a Java program to reverse an array of integer values.




```
import java.util.Arrays;
public class Main {
public static void main(String[] args){
   int[] my_array1 = {
         1789, 2035, 1899, 1456, 2013,
         1458, 2458, 1254, 1472, 2365,
         1456, 2165, 1457, 2456};
 System.out.println("Original array :
"+Arrays.toString(my_array1));
  for(int i = 0; i < my_array1.length / 2; i++)
 {
   int temp = my_array1[i];
   my_array1[i] = my_array1[my_array1.length - i - 1];
   my_array1[my_array1.length - i - 1] = temp;
 }
   System.out.println("Reverse array :
"+Arrays.toString(my_array1));
 }
}
```

JAVA

2D-ARRAYS

Multidimensional Arrays can be defined in simple words as array of arrays. Data in multidimensional arrays are stored in tabular form (in row major order).

Declaration of Arrays

Syntax { data_type[][] array_name = new data_type[x][y];

Example : int[ ][ ] a = new int[3][4];

we have created a multidimensional array named **[a]**. It is a 2-dimensional array, that can hold a maximum of 12 elements

	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Initialize a 2d array in Java

```
int[ ][ ] a = {
    {1, 2, 3},
    {4, 5, 6, 9},
    {7},
} ;
```

Initialization of 2-dimensional Array

	Column 1	Column 2	Column 3	Column 4
Row 1	1 a[0][0]	2 a[0][1]	3 a[0][2]	
Row 2	4 a[1][0]	5 a[1][1]	6 a[1][2]	9 a[1][3]
Row 3	7 a[2][0]			

Example: 2d-Array

```
import java.io.*;
import java.util.*;
public class solution { public static void main(String[] args) {
                        int[ ][ ] arr = {{2, 5},
                                        {4, 0},
                                        {9, 1}};

                for (int i = 0; i < 3; i++) {
                      for (int j = 0; j < 2; j++) {

            System.out.println("arr[" + i + "][" + j + "] = " + arr[i][j] + " ");
        }
      }

  }
}
```

Output:

arr[0][0] = 2

arr[0][1] = 5

arr[1][0] = 4

arr[1][1] = 0

arr[2][0] = 9

arr[2][1] = 1

Advantages of Array in Java

- The elements in arrays can be accessed at random using the index number.
- Since it generates a single array of several elements, it requires fewer lines of code.
- All of the array's elements are easily accessible.
- Using a single loop, traversing the array becomes easy.
- Sorting becomes simple because it can be done with fewer lines of code.
- Matrices are represented using two-dimensional arrays.

Disadvantages of Array in Java

- Arrays are Strongly Typed.
- Arrays does not have add or remove methods.
- We need to mention the size of the array. Fixed length.
- So there is a chance of memory wastage.
- To delete an element in an array we need to traverse through out the array so this will reduce performance.

JAVA

ARRAYLIST

Java **ArrayList** class uses a dynamic array for storing the elements. It is like an array, but there is no size limit. **We can add or remove elements anytime.** So, it is much more flexible than the traditional array. It is found in the **java.util package**. It is like the Vector in C++.

Important points about ArrayList

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non-synchronized.
- Java ArrayList allows random access because the array works on an index basis.
- We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. **Example -**

```
ArrayList<int> al = ArrayList<int>( ); // does not work
ArrayList<Integer> al = new ArrayList<Integer>( ); // works fine
```

Create ArrayList

```java
import java.util.ArrayList;

class Main {
  public static void main(String[] args){

    // create ArrayList
    ArrayList<String> languages = new ArrayList<>();

    // Add elements to ArrayList
    languages.add("Java");
    languages.add("Python");
    languages.add("Swift");
    System.out.println("ArrayList: " + languages);
  }
}
```

Basic Operations on ArrayList

The **ArrayList** class provides various methods to perform different operations on arraylists.

- **Add** elements
- **Access** elements
- **Change** elements
- **Remove** elements

1. Add Elements

To add a single element to the arraylist, we use the **add( ) method** of the ArrayList class.

```java
import java.util.ArrayList;

class Main {
  public static void main(String[] args){
    // create ArrayList
    ArrayList<String> languages = new ArrayList<>();

    // add() method without the index parameter
    languages.add("Java");
    languages.add("C");
    languages.add("Python");
    System.out.println("ArrayList: " + languages);
  }
}
```

2. Access Elements

To access an element from the arraylist, we use the **get( ) method** of the ArrayList class.

```java
import java.util.ArrayList;

class Main {
  public static void main(String[] args) {
    ArrayList<String> animals = new ArrayList<>();

    // add elements in the arraylist
    animals.add("Cat");
    animals.add("Dog");
    animals.add("Cow");
    System.out.println("ArrayList: " + animals);

    // get the element from the arraylist
    String str = animals.get(1);
    System.out.print("Element at index 1: " + str);
  }
}
```

3. Change Elements

To change elements of the arraylist, we use the **set( ) method** of the ArrayList class.

```java
import java.util.ArrayList;

class Main {
  public static void main(String[] args) {
    ArrayList<String> languages = new ArrayList<>();

    // add elements in the array list
    languages.add("Java");
    languages.add("Kotlin");
    languages.add("C++");
    System.out.println("ArrayList: " + languages);

    // change the element of the array list
    languages.set(2, "JavaScript");
    System.out.println("Modified ArrayList: " + languages);
  }
}
```

4. Remove Elements

To remove an element from the arraylist, we can use the **remove( ) method** of the ArrayList class.

```java
import java.util.ArrayList;
class Main {
  public static void main(String[] args) {
    ArrayList<String> animals = new ArrayList<>();

    // add elements in the array list
    animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
    System.out.println("ArrayList: " + animals);

    // remove element from index 2
    String str = animals.remove(2);
    System.out.println("Updated ArrayList: " + animals);
    System.out.println("Removed Element: " + str);
  }
}
```