SREENIDHI
UNIVERSITY
KINDLE THE LIGHT WITHIN

Yamnampet, Ghatkesar, Hyderabad – 501301, Telangana

Course Information:

24CSEJ303 – Object Oriented Programming through Java

24CSEJ303	Object Oriented Programming through Java	**L**	**T**	**P**	**S**	**I**	**C**
		2	0	**4**	0	0	**4**
Pre-requisite	**24CSE102 Problem Solving and Computer Programming through C++**						
Co-requisite	**NIL**						

Course Outcomes

Upon successful completion of the course students will be able to:

1. Understand **Java syntax, variables, operators, control structures** for programming. [L2- Understand]
2. Utilize **classes, constructors, method overloading and rec**ursion. [L3-Apply]
3. Apply **1D and multidimensional arrays, and string functions** for data manipulation. [L3-Apply]
4. Apply **inheritance, use abstract classes, interfaces, and packages** to structure applications efficiently. [L3-Apply]
5. Examine **exceptions, multithreading for concurrent processing, and understand thread synchronization**. [L4-Analyze]
6. Develop programs for real-world applications through Java concepts, including OOP, arrays, inheritance, exception handling, and multithreading. [L6-Create]

Course Contents

Unit	Title	Key Topics
I	**Fundamentals of Java Language**	**History and Evolution**, **Overview of Java**, **Data Types**, **Operators**, **Control Statements**
II	**Introduction to Classes and Methods**	**Class Structure**, **Objects**, **Constructors**, **Method Overloading**, **this Keyword**, **Garbage Collection**, Scanner, BufferedReader
		Autoboxing/Unboxing, **Wrapper Classes**, **static**, **final**, **Command Line Args**, **Varargs**, **Recursion**
III	**Arrays and Strings**	**1D & Multidimensional Arrays**, Arrays Class, Vector Class, **String Handling**, **String Comparison**, **StringBuffer Methods**

Course Contents

Unit	Title	Key Topics
IV	Inheritance, Packages & Interfaces	**Inheritance Basics, super, Method Overriding, Dynamic Dispatch, Abstract Classes, Object Class, Interfaces, Packages**
		Default & Static Methods, Access Protection, Importing Packages
V	Exception Handling & Multithreading	**try-catch-finally, throw/throws, Custom Exceptions, Thread Creation, isAlive(), join(), Thread Priorities, Sync**

Course Evaluation Policy

				Continuous Internal Evlaluation (Components)									CIE Marks
Course Type	Evaluation Type	Course Code	Courses	Term Test	MOOCS(Self Learning Module) / Assignments	SIA Tests - 5 Objective	SIA Tests - 10 Live Tets	Practical Internal Test	Regular Lab Performance Assessment	Project Work Assessment	Regular Skill Performance Evaluation	Skill Certification	
Theory with Practical - II	ETP2	24CSEJ303	Object Oriented Programming through Java	Sum of 2 Term Tests (20 marks)	---	7.5	2.5	15	5	---	---	---	50

SEE Components				SEE Marks	CIE Pass Mark out of 50	SEE Pass Mark out of 50	Minimum mark for passing the course (CIE+SEE) out of 100	Total
Theory	Practical	Project	Skill					
30	20	---	---	50	---	10.5 for Theory (10.26 and above is pass) and 7 for Practical (6.5 and above is pass)	50	100

Neet of Programming

- Everyone uses **computers, mobile phones** and many other gadgets in **day-to-day l**ife to **access various applications.**
- Different business domains like **Education, Banking, E-Commerce, Healthcare, Insurance** etc. use computer applications.
- These **applications are built using programming.**
- The knowledge of programming is essential to bring innovation

Understanding Programming

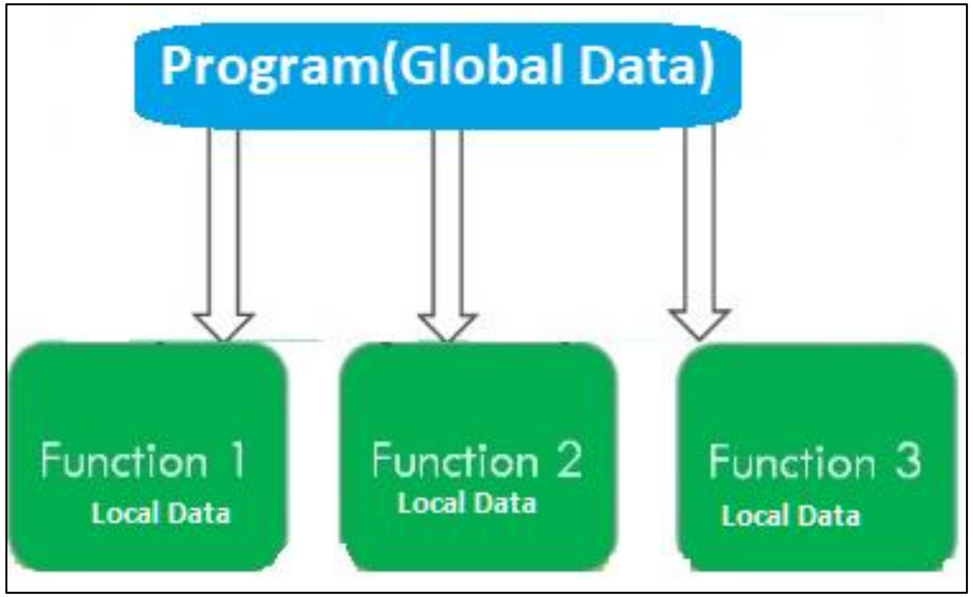
- A customer wants to ORDER FOOD using app.
- The customer can
  1. Open menu
  2. select the food items and
  3. place an order.
- If the order is placed successfully, the food gets delivered to the customer.
- we give instructions to the computing devices
- A **set of such instructions** is known as a PROGRAM and the act of creating a programs is known as PROGRAMMING.

Programs - solve the problems, automate the processes and reduce repetitive and/or manual work.

- To write these programs, you need a PROGRAMMING LANGUAGE

POP - Procedural Oriented Programming

- In POP, a program is dividend into **a set of procedures/functions** to accomplish a task.
- A **procedure** is a **collection of instructions** executed in sequential order.
- Ex. read(),add(),display()..etc



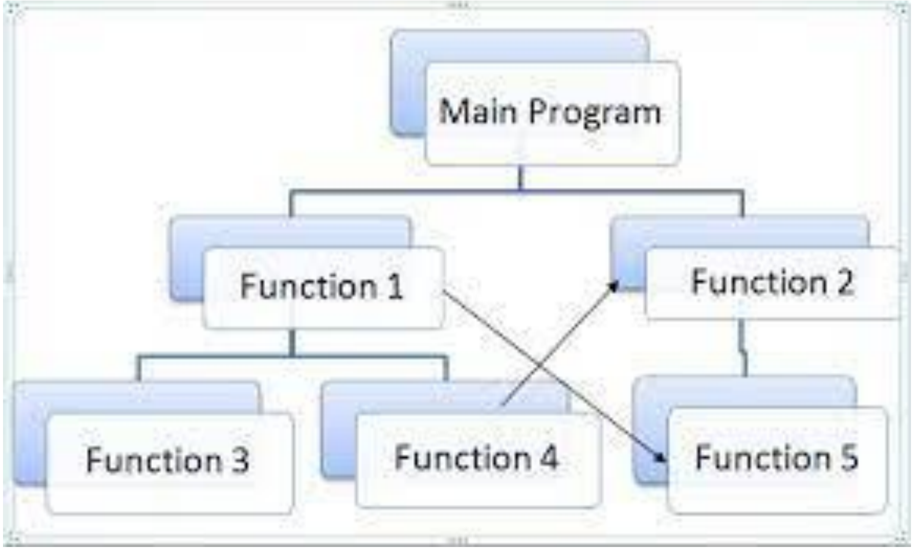
Features of POP:

- **Top-Down** Approach:
  Begin with the main task, then break it into smaller sub-tasks (procedures/functions).
- **Data Flow:**
  Data is passed explicitly between functions as arguments and return values.
- **Scope Limitation:**
  Global data can be accessed from multiple functions, so data visibility is less controlled

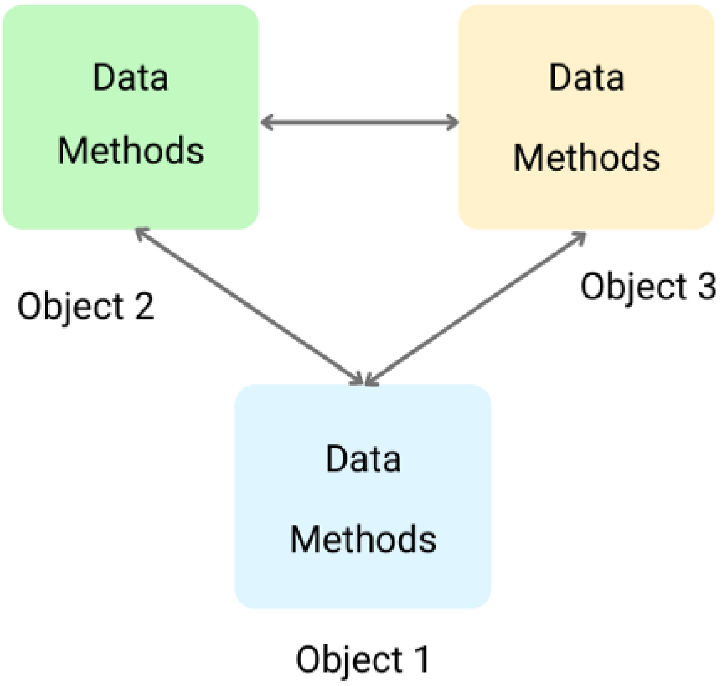
POP - Procedural Oriented Programming

Problems with POP:

- **Emphasis on Procedures, Not Data**: Focuses **primarily on functions or procedures**, often leading to data being treated as secondary
- **Lack of Data Security & Encapsulation**: **Data move publicly;** therefore **data security cannot be maintained. Data hiding is not possible** as there are no access specifiers.
- **Limited Code Reusability** : The **Procedural code is often not reusable**, recreation is need in another application.
- **Difficulty Modeling Real-World:** Struggles to represent complex real-world objects and their behaviors.
- **Scalability Complexity Issues:** As programs become more complex, it's **difficult to manage the increasing number of functions** and dependencies.
- **Code Maintenance:** As programs grow larger, it **becomes harder to maintain due to the lack of modular structure**. Changes in one part of the program can affect other parts.



OOP -Object Oriented Programming



Object: A real-world entity (e.g., pen, chair, computer) with **data** and **behavior**..

- **Modularity and Reusability**: Build systems using **reusable components** (objects).
- **OOP Paradigm**: Programs are designed using **classes and objects**.
- Program is divided into **set of objects.**
- **Data-Centric**: Emphasizes **data over procedures**.
- In general, an object is a **real-world entity** which contains **data** and **behavior**.
- An object in programming can be created by using a **Class.**
- i.e.. An object is an **instance** of a class.
- A class is **Template or blueprint** to create objects.

Key OOP Principles:

- Encapsulation – Data hiding
- Inheritance – Reuse of code
- Polymorphism – Same function, different behavior
- Abstraction – Hiding complexity

Java – History and Evolution

- Java is an **Object-Oriented Programming** language.

Where It All Began

- **James Gosling** and his team at "**Sun Micro Systems**" in initiated in1991 called **Green Project**
- Purpose: To create software for embedded systems in electronic appliances like **set-top boxes, digital cameras**, etc.
- In 1992 called **Oak** – named after an oak tree outside Gosling's office.

The Big Idea Behind Java

- **Platform independence:** "Write Once, Run Anywhere" (WORA)
- Need for **secure, portable, and robust** programs.
- C and C++ had limitations for **distributed and network-based environments.**

Java – History and Evolution

Oak to Java – Name Change

- **Oak** is a symbol of strength.
- In **1995**, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.
- Team brainstormed names over coffee ☕
- Java is an island where the first coffee was produced (called Java coffee).

Java – History and Evolution

Java’s Timeline – Key Milestones

Year	Milestone
1996	Java 1.0 released – Applets & AWT
1998	Java 2 – Swing, Collections
2000–2004	Rise of Enterprise Java (J2EE) – Servlets, JSPs, EJB
2004-2010	Java 5 – Era of Generics, Enhanced for-loop
2011	Oracle acquires; Java 7 released Performance Improvement.
2014	Java 8 – Lambdas, Streams
2017+	The Modern Java: **Java 9** and Beyond support LTS(Long Term Support)
March 2025	Java 24 released

Java – History and Evolution

Major Java Versions *and Features*

Java 5: Foundation of Modern Java

Generics	Enables code reusability and **type safety** by allowing classes and methods to operate on typed parameters (e.g., **List<String>).**
For-each loop	Simplifies **iteration** through arrays or collections **(for (int x : arr)).**
Enums	Defines a **fixed set of constants** (e.g., enum Day { MON, TUE, ... }).
Annotations	Used to provide metadata for code (e.g., @Override, @Deprecated).

Java 8: Functional Programming Revolution

Lambda Expressions	Allow you to write **inline, concise code** for functional interfaces **((a, b) -> a + b).**
Streams API	**Processes sequences of data** (like lists) using a fluent, functional style **(list.stream().filter(...)).**
Default Methods	Adds **default method** implementation in **interfaces**.
Functional Interfaces	Interfaces with a **single abstract method**, used with lambda expressions (Runnable, Comparator).

Java 11: Long-term support (LTS), new HTTP client

Java 17: Latest LTS with sealed classes, pattern matching

Overview of Java

Features / buzzwords of Java

- Simple
- Object-Oriented
- Portable
- Platform independent
- Secured
- Robust
- Architecture neutral
- Interpreted
- High Performance
- Multithreaded
- Distributed
- Dynamic

Features of Java

Simple:

Its syntax is simple, syntax is based on C++

Object Oriented : Everything in Java is treated as an object.

Portable:

Bytecode can run on any system with out recompilaiton

Platform independent :

Java code can be executed on multiple platforms(any OS) using JVM

Secured:

No explicit pointers, Programs run inside a virtual machine sandbox

Robust:

Automatic garbage collection, exception handling

Architecture neutral:

Byte-code is not dependent on any machine architecture

Interpreted:

Java byte code is translated on the fly to native machine instructions

High Performance :

Just-In-Time compilers enables high performance.

Multithreaded:

Supports concurrent tasks execution

Distributed:

Designed for use in networked/distributed environments

Dynamic:

classes are loaded on demand.

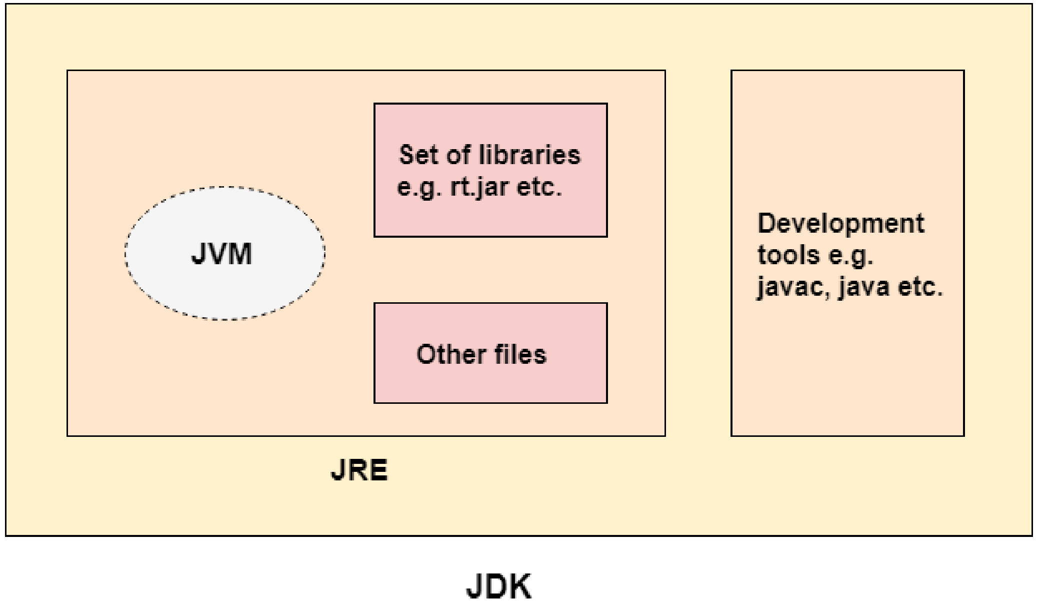
Comparison Table: Java vs. C++

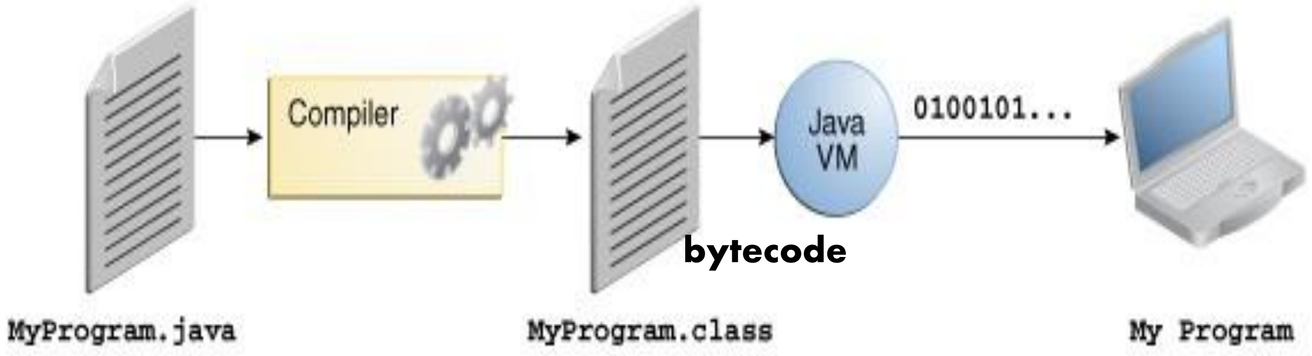
Feature	C++	Java
Platform Dependency	Platform dependent	Platform independent (JVM)
Memory Management	Manual (delete, free)	Automatic (Garbage Collector)
Multiple Inheritance	Supported	Not directly (uses interfaces)
Pointers	Supports pointers	No pointers (security)
Compilation	Compiler only	Compiler + Interpreter
GUI Support	Requires libraries	Built-in (JavaFX/Swing)
Multithreading	Requires libraries	Built-in

Java Architecture

Components:

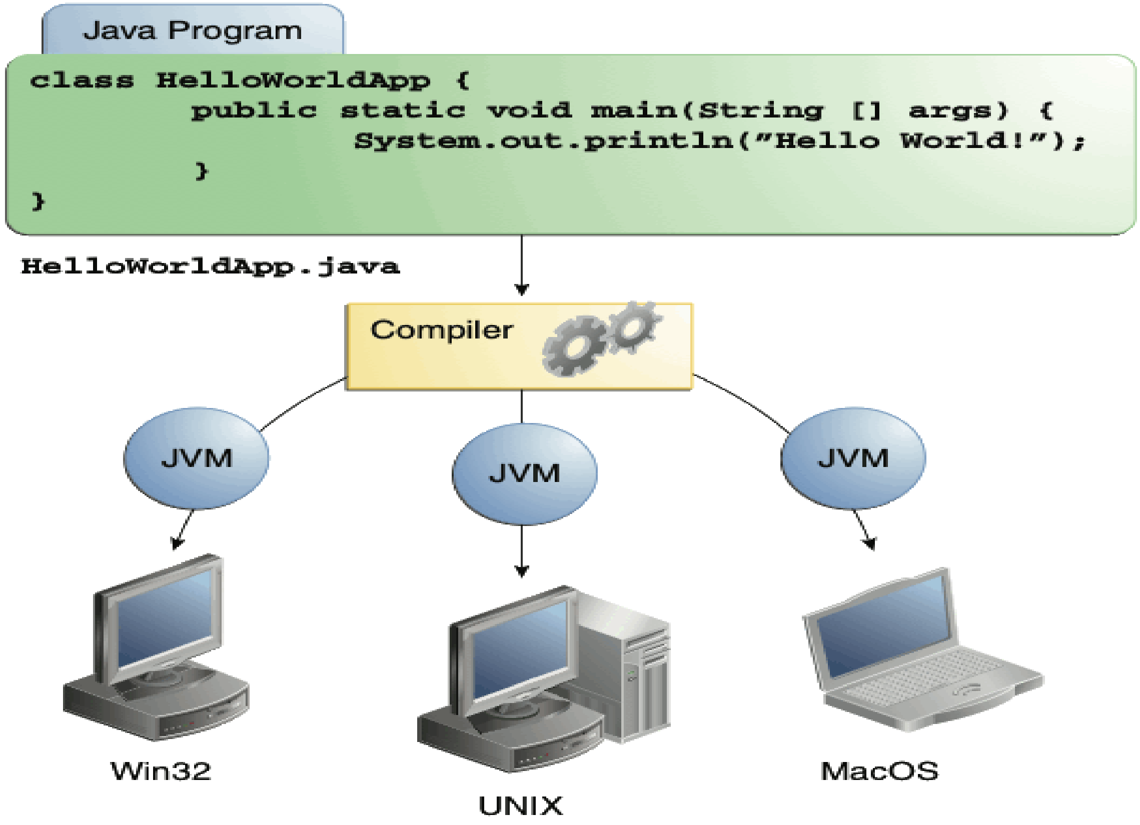
- **JDK (Java Development Kit)**: Includes development tools like compiler, debugger, etc + **JRE**
  - **Java Compiler**: Converts .java files to .class files (bytecode)
- **JVM (Java Virtual Machine):** Executes the compiled .class **bytecode** on any platform.
- **JRE (Java Runtime Environment)**: Provides the environment **(JVM + libraries)** to run Java programs


JVM
Set of libraries e.g. rt.jar etc.
Other files
Development tools e.g. javac, java etc.
JRE
JDK

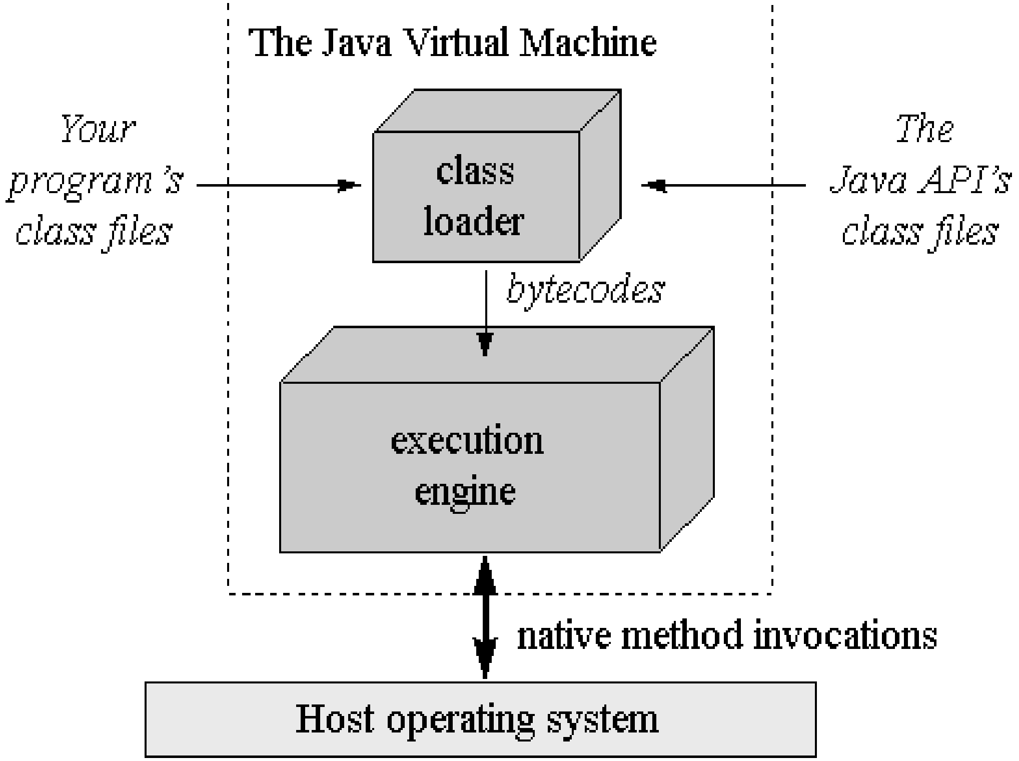


JVM (Java Virtual Machine)

- The **Java Virtual Machine (JVM)** is a **core part of the Java** platform, enabling **Java programs to run on different systems**.
- It acts as an **intermediary** between the compiled Java code (bytecode) and the underlying **operating system and hardware**, enabling Java applications to be **platform-independent.**



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



Crating a First Java Program

```
//documentation section
Import java.io.*;          //import statments
class Sample               //class definition
 {
    public static void main(String args[ ])
     {
       System.out.println(“Welcome to  Java");
      }
}
```

- class keyword is used to declare a class in Java.
- public access modifier that represents visibility..
- static - no need to create an object to invoke the static method.
- void : it doesn't return any value.
- main represents the starting point of the program.
- String[] args or String args[] is used for command line arguments.
- System.out.println() is used to print statement.

```
compile:    javac Sample.java
execute:    java Sample
```

Java Identifies

Identifiers:

Name given to entities such as variables, functions, structures etc. An identifier is a **long sequence of letters**(a-z & A-Z) and numbers(0-9).

Ex: int sum; float marks; void swap(int a, int b); //sum, marks, swap - **Identifiers**

int, float – Keywords

Naming Conventions:

- Can contain letters **(A–Z, a–z),** digits **(0–9),** and underscore (_)
- **Cannot start with a digit**
- **No special characters** allowed except underscore **(_)**
- **Cannot use Java keywords** (like int, float, class, etc.)
- Java is **case-sensitive** (Total and total are different identifiers)
- Should be meaningful and follow **naming conventions** (e.g., studentName)

Input and output in java

In Java, handling console input and output involves using the **Scanner** class for input and **System.out** for output

1. Console Output

- To print output to the console, use System.out.println() or **System.out.print().**

- System.out.println: Prints text followed by a new line.
- System.out.print: Prints text without a new line.

2. Console Input

- To read input from the console, use the Scanner class from the **java.util package.**
- You create a Scanner object and use its methods to read different types of data.

```
 // Create a Scanner object to read input
      Scanner scanner = new Scanner(System.in);
```

Methods:

- **nextInt() – reads integer**
- **nextLine() – reads string**
- **nextDouble() – reads float or double**

```
import java.util.Scanner;

public class InputOutputExample {
    public static void main(String[] args) {
        // Create a Scanner object to read input
        Scanner sc = new Scanner(System.in);

        // Prompt the user for their name
        System.out.print("Enter your name: ");
        String name = sc.nextLine();

        // Prompt the user for their age
        System.out.print("Enter your age: ");
        int age = sc.nextInt();

        // Display the collected information
        System.out.println("Hello, " + name + "!");
        System.out.println("You are " + age + " years old.");

}
}
```

Variables in java

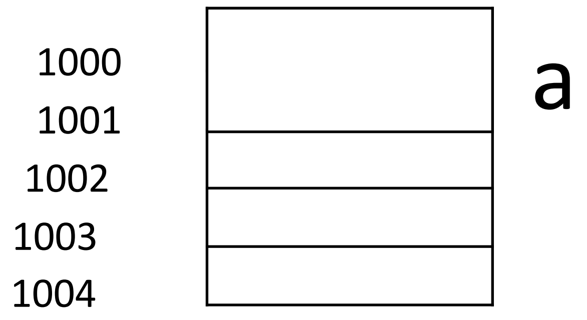
A Variable is a **name** given to the **memory location**.

Ex.

Variable = labeled jar in kitchen (stores specific type of ingredient)

It is used to **store data**

Its value can be changed, and it can be reused many times.



RAM Visualization

Syntax: type variable_name;

Eg : int **a**;

float b;

char c;

Variable Declaration:

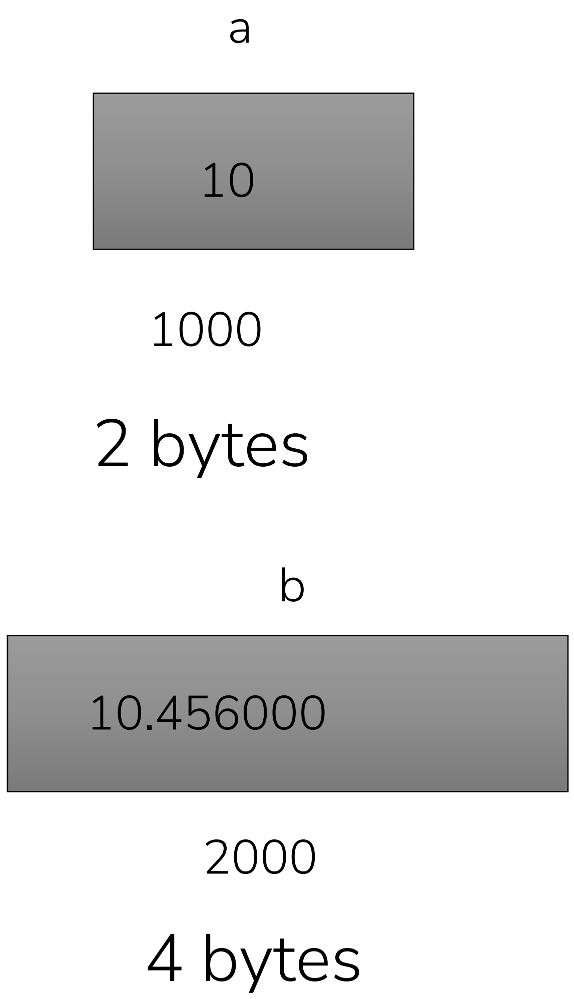
```
    int  a;

  float  b;
```

Variable Initialization:

```
     a = 10;

    b = 10.456;
```



a

10

1000

2 bytes

b

10.456000

2000

4 bytes

Problem: Area & Perimeter of a Rectangle

Write a Java program to calculate the **area** and **perimeter** of a rectangle.

Consider a rectangle with the following integer variables :

- Length = **11**
- Breadth = **13**

Formulas:

- **Area** = length × breadth
- **Perimeter** = 2 × (length + breadth)

📃 **Output:**

Your program should print:

1. The area of the rectangle.
2. The perimeter of the rectangle.

Sample Output:

```
143
48
```

```
class RectangleCalc {
    public static void main(String[] args) {
        int length = 11;
        int breadth = 13;

        int area = length * breadth;
        int perimeter = 2 * (length + breadth);

        System.out.println(area);      // Output: 143
        System.out.println(perimeter); // Output: 48
    }
}
```

Types of Java Variables

THREE types of variables in java:

1).Local 2).Instance and 3) Static.

1) Local Variable

- Declared **inside a method.**
- Accessible **only within that method**.
- **Not visible** to other methods.

2) Instance Variable

- Declared **inside the class but outside methods**..
- Each object gets its **own copy** (instance-specific).

3) Static variable

- A single copy of the static variable can be shared among all the instances of the class.
- Memory allocation for static variables happens only once when the class is loaded in the memory.

```
public class A
{
   static int m=100;//static variable
    int data=50;//instance variable
   void method()
   {
      int n=90;//local variable
   }
   public static void main(String args[])
   {
      ..
   }
}//end of class
```

Data Types in Java

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

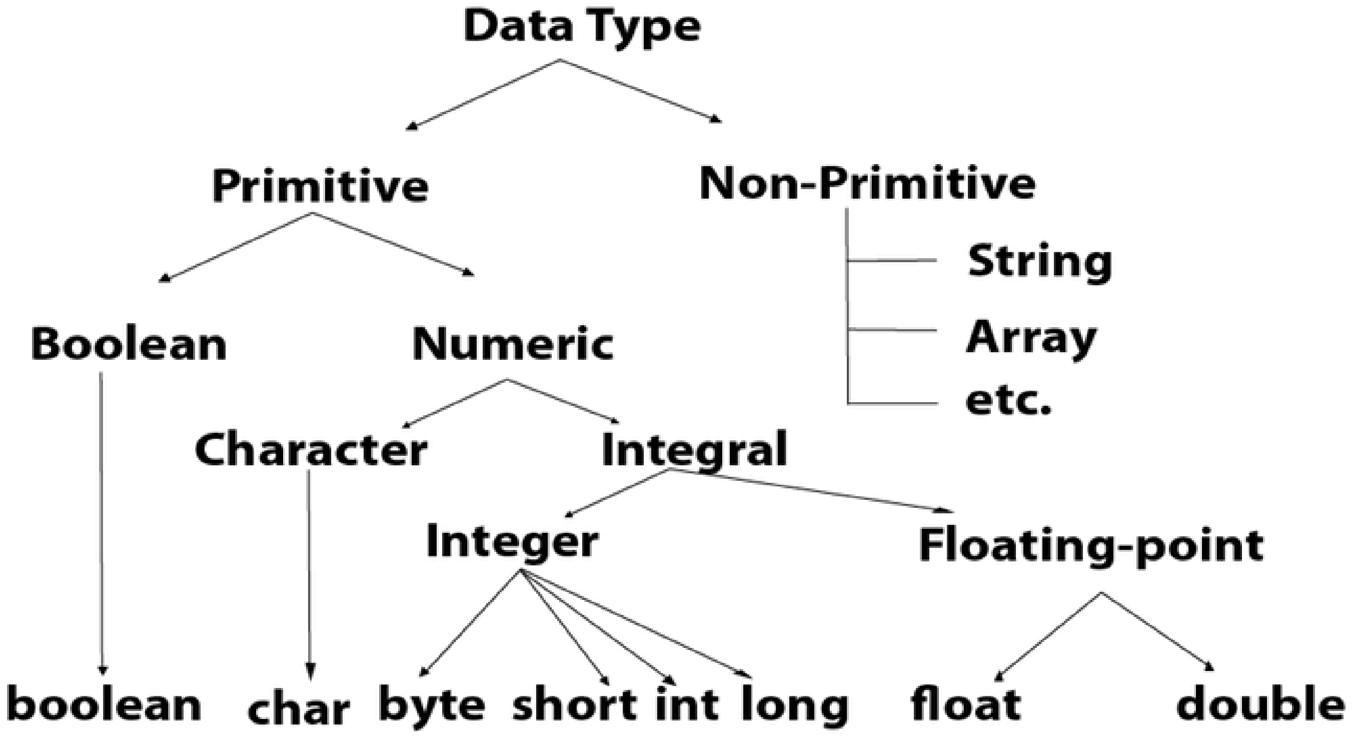
- There are two types of data types in Java:

  1. Primitive Data Types:

  - int, float, double, char, boolean, byte, short, long

  2.Non-Primitive:

  - Arrays, Strings, Classes, Interfaces



Data Types in Java

Type	Data Type	Size	Range	Example	Usage
Integer Types	**byte**	**1 byte**	-128 to 127	**byte age = 25;**	**Small data like age, scores**
	short	**2 bytes**	-32,768 to 32,767	**short year = 2024**;	**For medium numbers**
	int	**4 bytes**	-2B to 2B	int rollNo = 1023;	**Default for whole numbers**
	long	**8 bytes**	(about) -10E18 to 10E18	**long distance = 123456789L;**	**Big numbers like distance, population**
Floating-point	float	4 bytes	~±3.4E38	**float price = 12.99f;**	Decimal numbers (less precision)
	double	8 bytes	~±1.7E308	double pi = 3.14159;	More accurate decimals
Character	char	2 bytes	**0 to 65,535 (Unicode)**	**char grade = 'A';**	Any single character
Boolean	boolean	1 bit (virtually 1 byte)	**true or false**	**boolean isPass = true;**	Logic-based conditions

Operators in Java

- An **operator is symbol** that **specify operation** to be perform certain operation.

ARITHMETIC

+, -, *, /, %

RELATIONAL:

==, !=, >, <

LOGICAL:

&&, ||, !

ASSIGNMENT:

=, +=, -=, ETC.

BITWISE:

&, |, ^, <<, >>

INCREMENT AND DECREMENT OPERATORS ++, --

CONDITIONAL OPERATORS ?:

Real-time Examples:

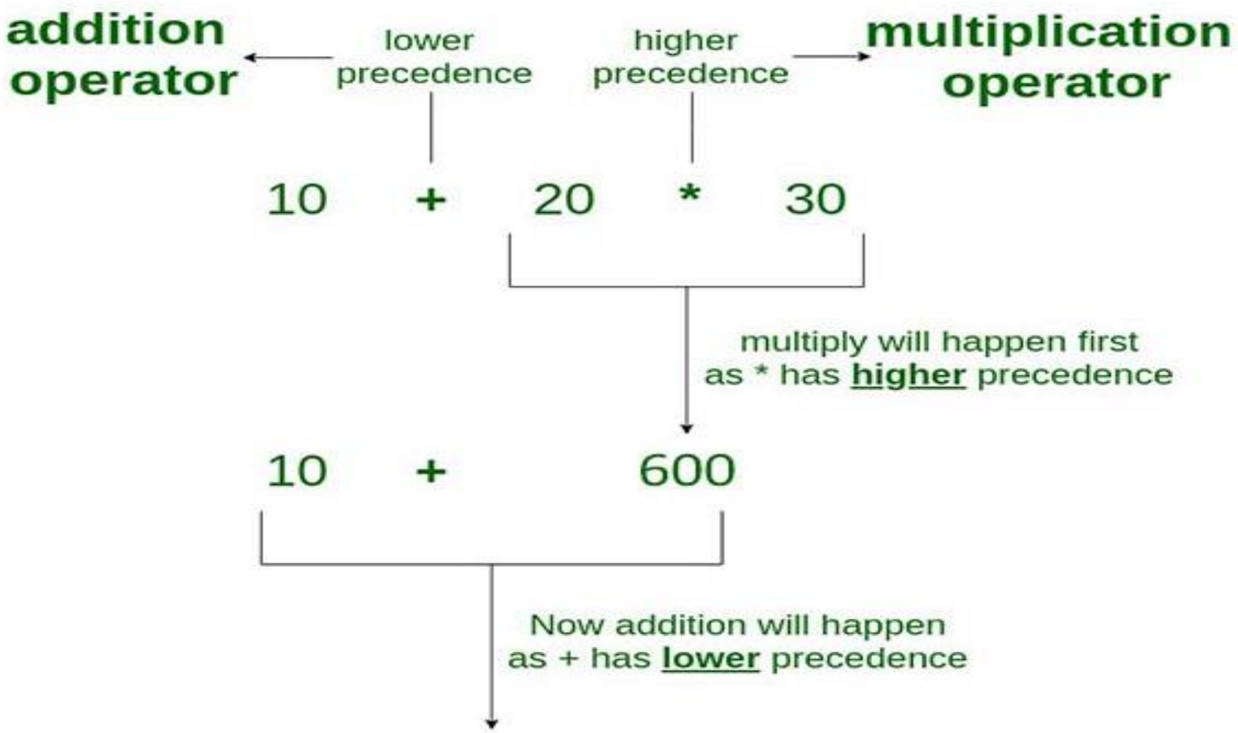
- Arithmetic: **Calculator app**
- Relational: **Comparing prices on Amazon**
- Logical: **Login systems using AND/OR**

Precedence and associativity of Operators

- The **precedence** of operator **specifies that which operator will be evaluated first and next.**

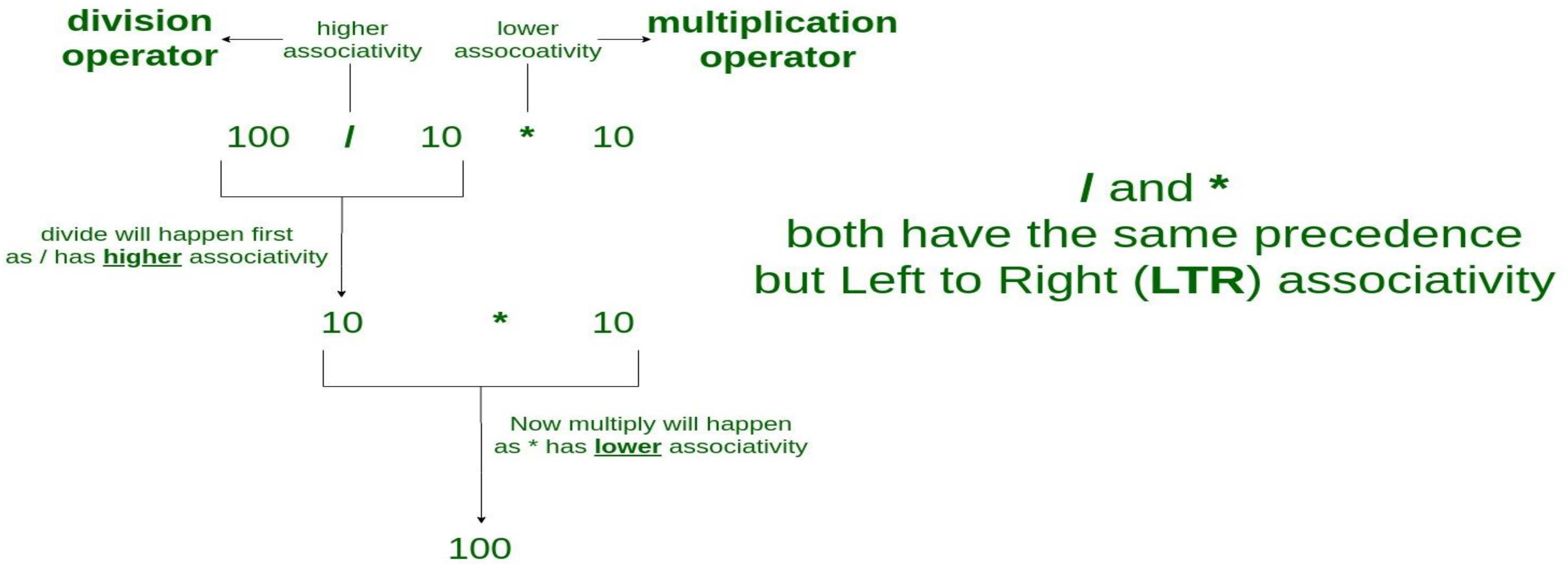
  int value=10+20*10;

- The value variable will contain 210 because * (multiplicative operator) is evaluated before + (additive operator).


addition operator
lower precedence
higher precedence
multiplication operator
10 + 20 * 30
multiply will happen first as * has higher precedence
10 + 600
Now addition will happen as + has lower precedence

Associativity of Operators

- The **associativity** specifies the **operator direction to be evaluated**; it may be **left to right** or **right to left**.
- ‘*’ and ‘/’ have same precedence and their associativity is Left to Right.
- E.g. 100 / 10 * 10



divide will happen first as / has **higher** associativity

Now multiply will happen as * has **lower** associativity

/ and * both have the same precedence but Left to Right (**LTR**) associativity

Operator Hierarchy

Precedence	Category	Operator	Associativity
1	**Method,Aarray access & post fix**	**() [] . ++ - -** (post incr/ decr)	**Left to right**
2	**Unary**	+ - ! ~ (unary) **++ - -** (pre incr)	**Right to left**
3	**Multiplicative**	*** / %**	**Left to right**
4	**Additive**	**+ -**	**Left to right**
5	**Shift**	**<< >> >>>**	**Left to right**
6	**Relational**	**< <= > >= instamceof**	**Left to right**
7	**Equality**	**== !=**	**Left to right**
8	**Bitwise AND**	**&**	**Left to right**
9	**Bitwise XOR**	**^**	**Left to right**
10	**Bitwise OR**	**|**	**Left to right**
11	**Logical AND**	**&&**	**Left to right**
12	**Logical OR**	**||**	**Left to right**
13	**Conditional**	**?:**	**Right to left**
14	**Assignment**	**= += -= *= /= %=>>= <<= &= ^= |=**	**Right to left**
15	**Comma**	**,**	**Left to right**

Control statements

I. Decision-making Statements:

if

if-else

switch

II. Looping Statements:

for

while

do-while

III. Jump Statements:

- break
- continue
- return

Real-Life Examples:

- if: Check if ATM has enough balance
- for: Send OTP to 100 users
- while: Continue until user exits app

Control statements

In java program, control structure is can divide in three parts:

I. **Selection Statement/ Conditional Statement**
II. **Loops/Iteration Statement**
III. **Unconditional / Jumps In Statement**

I . SELECTION statements:

- Selection statements used for **Decision making.**
- Decision making is about **deciding the order of execution of statements** based on **certain conditions**

Types:

1. **if statement (conditional)**
2. **switch statement**

1. if statement (conditional)

i. If (simple if)

ii. if – else

iii. Cascaded(if else - if)

iv. Nested if

Control Structures/control statements:

i) simple if

Syntax :

```
if(condition)
{
        // Statements inside.

}
//remaining statements
//statement outside
```

```
// Check if person 1 is taller
if (h1 > h2)
{
  System.out.println("Person 1 is taller");
}
```

ii) if – else statement

Syntax :

```
if(condition)
{
   //block1: code to be executed if condition is true
}
else
{
   //block2: code to be executed if condition is false
 }
```

```
// Check if person 1 is taller
 if (h1 > h2) {
 System.out.println("Person 1 is taller");
} else
 {
 System.out.println("Person 2 is taller");
}
```

Control Structures/control statements:

iii) If else if ladder

Syntax :

```
if(condition1)
{
            //block1: code to be executed if condition1 is true
}
else if(condition2)
{
            //block2:code to be executed if condition2 is true
}
else if(condition3){
            //block3:code to be executed if condition3 is true
}
...
else{
  //:code to be executed if all the conditions are false

}
```

Ex Decision Making in PUBG

- PUBG players are not going to get a Chicken Dinner anytime soon without the ability to aim at targets and take them down with relative ease.
- So to aim the target they must use **scope**. It can take hundreds of rounds before you become more comfortable with all the weapons on offer and start landing your shots, but we're here to help speed that process up.

Conditions:

- If you have 8x scope, **Use sniper gun.**
- If you have 6X scope, **Use AUG A3, GROZA, QBZ, M16A4, M416 .**
- If you have 4x Scope, Use UMP9, AKM, SCAR-L, Cross Bow .
- If you have 2x Scope, almost all guns.
- If you don't have scope, find one.

Now Let's help them by writing a program which helps them to select the gun based on the scope .

Control Structures/control statements:

iv) Nested if

Syntax:

```
if (condition1)
{
    // code to be executed if condition1 is true
    if (condition2)
    {
        // code to be executed if condition2 is true
    }
}
```

```
If(age >= 12)
{
    If(weight >= 40)
    {
      If(weight <= 110)
             {
             print("He can Jump");
              }
             else
            {
             print("Extra ropes will be added");
             }
      }
       else
      {
      print("He can't Jump");
      }
}
else
{
   Print("He can't Jump");
}
```

Control Structures/control statements:

Switch Case (Multiple Branching Statement)

Syntax:

```
            switch(expression){
                        case 1:
                                    //code to be executed;
                                    break;
                         case 2:
                                    //code to be executed;
                                    break;
                                    ......
                        default:
                                    code to be executed if all
cases are not matched;
            }
```

- The switch statement allows us to **execute one code block among many alternatives.**
- **A**llows us to **execute multiple operations** for the **different possible values** of a **single variable** called switch variable.
- We can define **various statements** in the **multiple cases** for the different values of a **single variable.**

```
switch(number)
{
  case 1:
           Print(“Welcome to Erangel Map. You are Inside a Forest”);
           break;
  case 2:
           Print(“Welcome to Miramar Map. You are Inside a
Desert”);
           break;
  case 3:
           Print(“Welcome to Sanhok Map. You are Inside a Rain
Forest”);
           break;
  case 4:
           Print(“Welcome to Vikendi Map. You are Inside a Snow
Forest”);
           break;
  default:
          Print(“Invalid Input”);
}
```

Loops / Iterations

- The process of repeatedly executing a statements and is called as looping.

Types:

- In Iteration statement, there are three types of operation:

1. for loop
2. while loop
3. do-while loop

1. Java for loop:

- ***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.

Syntax:

```
    for(initialization;condition; incr/decr)
    {
    Statement block;
    }
 for(int i=1;i<=10;i++){
    System.out.println(i);
 }
```

for-each loop

- Used to traverse the array or collection elements

- Syntax:

```
 for(data_type variable : array | collection){
 //body of for-each loop
 }


    for(int i=1;i<=10;i++){
 System.out.println(i);
}
```

Ex.,

```
 int arr[]={12,13,14,44};
 //traversing the array with for-each loop
 for(int i:arr)
  {
   System.out.println(i);
  }
```

Loops / Iterations

2 while loop

- The while loop loops through a block of code as long as a **specified condition is true:**

Syntax:

```
while (condition)
 {
   // code block to be executed
 }
```

Ex.

```
int i = 0;
while (i < 5)
 {
  System.out.println(i);
  i++;
}
```

3.The Do/While Loop

- This loop will **execute the code block once**, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

```
do
{
  // code block to be executed
}
while (condition);
```

Ex.

```
 int i=1;
    do{
        System.out.println(i);
          i++;
    }while(i<=10);
```

Unconditional / Jumps In Statement

break :

- The **break** statement immediately terminates the loop

```
for (int i = 0; i < 10; i++)
 {
    if (i == 5)
     {
        break; // Exits the loop when i equals 5
    }
    System.out.println(i);
}


System.out.println("Loop ended.");
```

continue:

- The **continue** statement in Java is used within loops to skip the current iteration and proceed to the next iteration of the loop.

```
for (int i = 0; i < 10; i++)
{       if (i % 2 == 0)
         {
           continue; // Skips the even numbers
         }
  System.out.println(i);
}
```