Object Oriented Programming

Paper C8

P Veera Venkata Durga PraSad
DEPARTMENT OF COMPUTER SCIENCE (AWDC KKD)

UNIT -I:
Object Oriented Programming: Introduction to OOP, Objects and Classes, Characteristics of OOP, Difference between OOP and Procedure Oriented Programming. **Introduction to Java Programming:** Introduction, Features of Java, Comparing Java and other languages (C & C++), Java Development Kit, Structure of Java Program, Prerequisites for Compiling and Running Java Programs.
UNIT - II:
Java Language Fundamentals: Data types, variable declarations, Operators and Assignment, Control structures, Arrays, Strings, The String Buffer Class. **Java as an** OOP **Language:** Defining classes, Constructors, Overloading, Modifiers, Packages.
UNIT - III:
Inheritance, Interfaces, Exception Handling: Inheritance, Types of Inheritance, Interfaces, Interface Implementation, Exception Handling in Java, Throwing User-defined Exceptions, Advantages of Exception. **Multithreading:** An Overview of Threads, Creating Threads, Thread Life—cycle,Thread Priorities, Thread Synchronization, Daemon Threads, Communication of Threads.
UNIT - IV:
Files and I/O Streams: An Overview of FO streams, Java I/O, File Streams, FileInputStream and File Output Stream, Filter streams, Random Access File, Serialization. **Applets:** Introduction, Java applications versus Java Applets, Applet Life-cycle, Working with Applets, The HTML Applet Tag.
UNIT - V:
Database Handling Using JDBC: An Overview of DBMS, JDBC Architecture, Working with JDBC Servlets: Introduction, How to run servlets, The Life—cycle of the servlet, servlet API, Multitier Applications using JDBC from a servlet.

UNIT-I

❖ Introduction to OOPS / Features of OOPS:

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object:

- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc.
- It can be physical or logical.An Object can be defined as an instance of a class.
- An object contains an address and takes up some space in memory.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class:

- *Collection of objects* is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance:

- *When one object acquires all the properties and behaviors of a parent object,* it is known as inheritance.
- When we write a class, we inherit properties from other classes.
- It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism:

- If *one task is performed in different ways,* it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.
- In Java, we use method overloading and method overriding to achieve polymorphism.
- Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction:

- *Hiding internal details and showing functionality* is known as abstraction.
- For example phone call, we don't know the internal processing.
- In Java, we use abstract class and interface to achieve abstraction.

Encapsulation:

- *Binding (or wrapping) code and data together into a single unit are known as encapsulation*.
- For example, a capsule, it is wrapped with different medicines.
- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here**.**

Association:

- Association represents the relationship between the objects. Here, one object can be associated with one object or many objects.
- There can be four types of association between the objects:

  - One to One
  - One to Many
  - Many to One, and
  - Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be undirectional or bidirectional.

Aggregation:

- Aggregation is a way to achieve Association.
- Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects.
- It is also termed as a *has-a* relationship in Java. Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

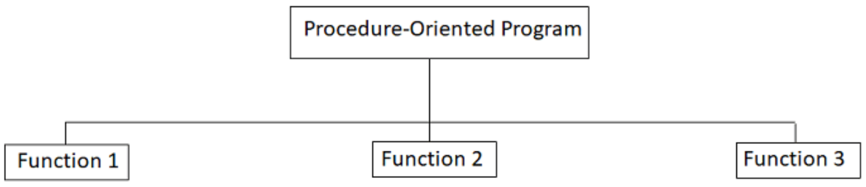
Composition:

- The composition is also a way to achieve Association.

- The composition represents the relationship where one object contains other objects as a part of its state.
- There is a strong relationship between the containing object and the dependent object.
- It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

❖ Procedure Oriented Approach:

- Procedure-Oriented Programming is the traditional way of programming, where an application problem is viewed as a sequence of steps (algorithms).
- As per the algorithm, the problem is broken down into many modules (functions) such as data entry, reporting, querying modules, etc. as shown in the figure.



Generalized Structure of a Procedure-Oriented Program:

- There are two types of data, which are associated with these modules- one is global and another is local data.
- Global data items are defined in the main program, whereas local data is define within associated functions.
- High-level languages like COBOL, Pascal, BASIC, Fortran, C, etc. are based on a procedure-oriented approach and hence are also called procedural languages.

❖ Differences between OOP and POP:

OOP

OOP, refers to Object Oriented Programming and its deals with objects and their properties. Major concepts of OOPs are –

- Class/objects
- Abstraction
- Encapsulation
- Polymorphism
- Inheritance

POP

POP, refers to Procedural Oriented Programming and its deals with programs and functions. Programs are divided into functions and data is global.

Following are the important differences between OOP and POP.

Sr. No.	Key	OOP	POP
1	Definition	OOP stands for Object Oriented Programing.	POP stands for Procedural Oriented Programming.
2	Approach	OOP follows bottom up approach.	POP follows top down approach.
3	Division	A program is divided to objects and their interactions.	A program is divided into funtions and they interacts.
4	Inheritance supported	Inheritance is supported.	Inheritance is not supported.
5	Access control	Access control is supported via access modifiers.	No access modifiers are supported.
6	Data Hiding	Encapsulation is used to hide data.	No data hiding present. Data is globally accessible.
7	Example	C++, Java	C, Pascal

❖ Classes and Objects:

✓ <u>Class:</u>

- ➢ Class is a set of object which shares common characteristics/ behavior and common properties/ attributes.
- ➢ Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
- ➢ Class does not occupy memory.
- ➢ Class is a group of variables of different data types and a group of methods.

Defining a Class in Java:

Java provides a reserved keyword **class** to define a class. The keyword must be followed by the class name. Inside the class, we declare methods and variables.

In general, class declaration includes the following in the order as it appears:

1. **Modifiers:** A class can be public or has default access.
2. **class keyword:** The class keyword is used to create a class.
3. **Class name:** The name must begin with an initial letter (capitalized by convention).
4. **Superclass (if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.

5. **Interfaces (if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. **Body:** The class body surrounded by braces, { }.

Syntax:

```
<access specifier> class class_name

{

 // member variables

 // class methods

}
```

✓ Object:

The **object** is a basic building block of an OOPs language. In **Java**, we cannot execute any program without creating an **object**. There is various way to **create an object in Java.**

Java provides five ways to create an object.

- Using **new** Keyword
- Using **clone()** method
- Using **newInstance()** method of the **Class** class
- Using **newInstance()** method of the **Constructor** class
- Using **Deserialization**

Using new Keyword:

- Using the new keyword is the most popular way to create an object or instance of the class.
- When we create an instance of the class by using the new keyword, it allocates memory (heap) for the newly created object and also returns the reference of that object to that memory.
- The new keyword is also used to create an array.

Syntax:

```
ClassName object = new ClassName();
```

❖ Introduction to Java:

Java is a high-level and purely **object oriented programming language**. It is platform independent, robust, secure, and multithreaded programming language which makes it popular among other OOP languages. It is widely used for software, web, and mobile application development, along with this it is also used in big data analytics and server-side technology. Before moving towards features of Java, let us see how Java originated.

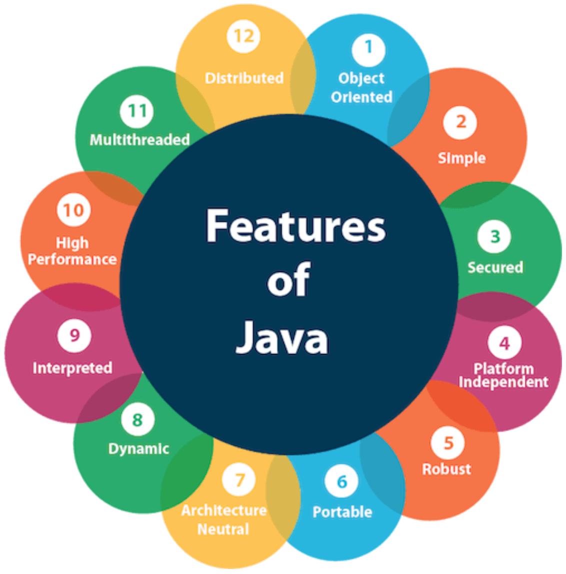
HISTORY:

- In 1990, Sun Microsystems Inc. started developing software for electronic devices. This project was called the Stealth Project (later known as Green Project).
- In 1991, Bill Joy, James Gosling, and Patrick Naughton started working on this project. Gosling decided to use C++ to develop this project, but the main problem he faced is that C++ is platform dependent language and could not work on different electronic device processors.
- As a solution to this problem, Gosling started developing a new language that can be worked over different platforms, and this gave birth to the most popular, platform-independent language known as Oak.
- Yes, you read that right, Oak, this was the first name of Java. But, later it was changed to Java due to copyright issues (some other companies already registered with this name).
- On 23 January 1996, Java's JDK 1.0 version was officially released by Sun Microsystems. This time the latest release of Java is JDK 20.0 (March 2023).
- Now Java is being used in Web applications, Windows applications, enterprise applications, mobile applications, etc. Every new version of Java comes with some new features.

❖ Features of java:

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

A list of the most important features of the Java language is given below.



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

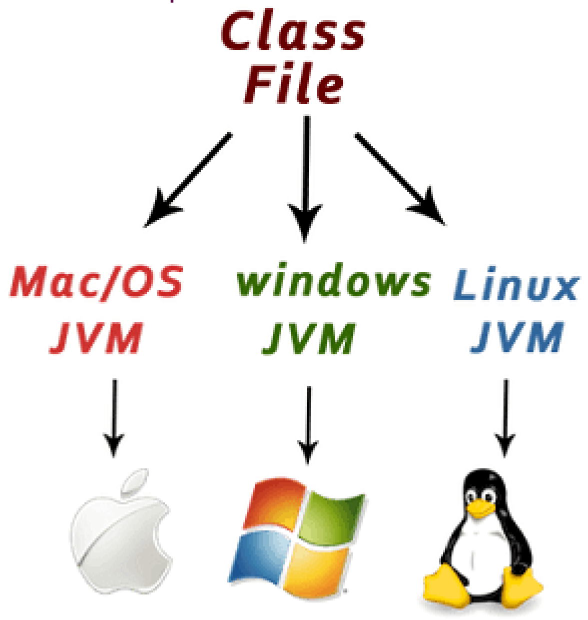
Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object

2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent



Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

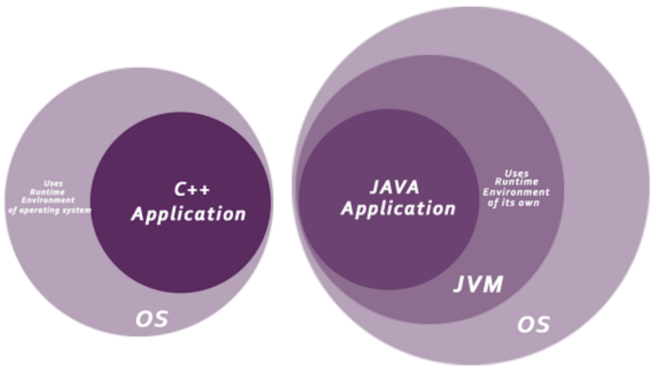
1. Runtime Environment
2. API(Application Programming Interface)

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**


Uses Runtime Environment of operating system
C++ Application
OS
JAVA Application
Uses Runtime Environment of its own
JVM
OS

- **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust

The English mining of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

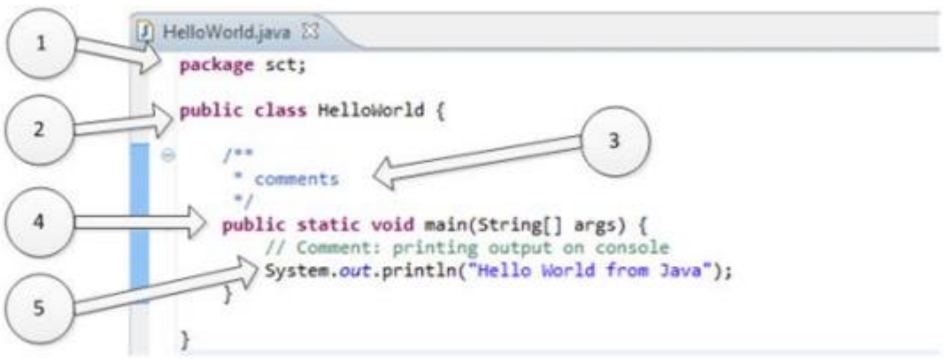
❖ Differences between C, C++ and Java

Metrics	C	C++	Java
Programming Paradigm	Procedural language	Object-Oriented Programming (OOP)	Pure Object Oriented Oriented
Origin	Based on assembly language	Based on C language	Based on C and C++
Developer	Dennis Ritchie in 1972	Bjarne Stroustrup in 1979	James Gosling in 1991
Translator	Compiler only	Compiler only	Interpreted language (Compiler + interpreter)
Platform Dependency	Platform Dependent	Platform Dependent	Platform Independent
Code execution	Direct	Direct	Executed by JVM (Java Virtual Machine)
Approach	Top-down approach	Bottom-up approach	Bottom-up approach
File generation	.exe files	.exe files	.class files
Pre-processor directives	Support header files (#include, #define)	Supported (#header, #define)	Use Packages (import)
keywords	Support 32 keywords	Supports 63 keywords	50 defined keywords
Datatypes (union, structure)	Supported	Supported	Not supported
Inheritance	No inheritance	Supported	Supported except Multiple inheritance
Overloading	No overloading	Support Function overloading (Polymorphism)	Operator overloading is not supported
Pointers	Supported	Supported	Not supported
Allocation	Use malloc, calloc	Use new, delete	Garbage collector
Exception Handling	Not supported	Supported	Supported
Templates	Not supported	Supported	Not supported
Destructors	No constructor neither destructor	Supported	Not supported
Multithreading/ Interfaces	Not supported	Not supported	Supported
Database connectivity	Not supported	Not supported	Supported
Storage Classes	Supported (auto, extern)	Supported (auto, extern)	Not supported

That’s all with the differences between C, C++, and Java. I hope you are clear with the basic concepts of these wonderful programming languages and helped you in adding value to your knowledge.

❖ JAVA PROGRAM STRUCTURE:

Let’s use example of HelloWorld Java program to understand structure and features of class. This program is written on few lines, and its only task is to print “Hello World from Java” on the screen. Refer the following picture.



```
package sct;
public class HelloWorld {
    /**
     * comments
     */
    public static void main(String[] args) {
        // Comment: printing output on console
        System.out.println("Hello World from Java");
    }
}
```

1. **“packagesct”:**

It is package declaration statement. The package statement defines a name space in which classes are stored. Package is used to organize the classes based on functionality. If you omit the package statement, the class names are put into the default package, which has no name. Package statement cannot appear anywhere in program. It must be first line of your program or you can omit it.

2. **“public class HelloWorld”:**

This line has various aspects of java programming.

a. public: This is access modifier keyword which tells compiler access to class. Various values of access modifiers can be public, protected,private or default (no value).

b. class: This keyword used to declare class. Name of class (HelloWorld) followed by this keyword.

3. **Comments section:** We can write comments in java in two ways.

a. Line comments: It start with two forward slashes (//) and continue to the end of the current line. Line comments do not require an ending symbol.

b. Block comments start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/).Block comments can also extend across as many lines as needed.

4. **“public static void main (String [ ]args)”:**

Its method (Function) named main with string array as argument.

a. public : Access Modifier

b. static: static is reserved keyword which means that a method is accessible and usable even though no objects of the class exist.

c. void: This keyword declares nothing would be returned from method. Method can return any primitive or object.

d. Method content inside curly braces. { }

5. **System.out.println("Hello World from Java") :**

a. System: It is name of Java utility class.

b. out: It is an object which belongs to System class.

c. println: It is utility method name which is used to send any String to console.

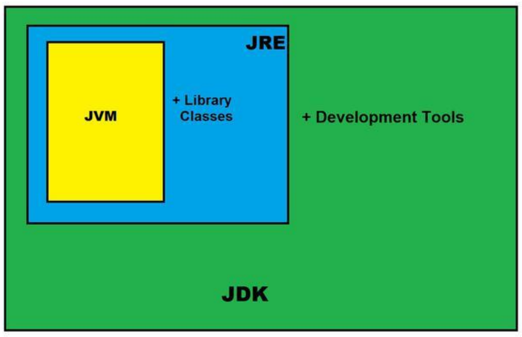
d. “Hello World from Java”: It is String literal set as argument to println method.

❖ Development Kit (JDK)

The Java Development Kit (JDK) is a cross-platformed software development environment that offers a collection of tools and libraries necessary for developing Java-based software applications and applets. It is a core package used in Java, along with the **JVM (Java Virtual Machine)** and the JRE (Java Runtime Environment).

Beginners often get confused with JRE and JDK, if you are only interested in running Java programs on your machine then you can easily do it using Java Runtime Environment. However, if you would like to develop a Java-based software application then along with JRE you may need some additional necessary tools, which is called JDK.

JDK=JRE+Development Tools



JAVA Development Kit (JDK)

The Java Development Kit is an implementation of one of the Java Platform:

- Standard Edition (Java SE),
- Java Enterprise Edition (Java EE),
- Micro Edition (Java ME),

Contents of JDK

The JDK has a private Java Virtual Machine (JVM) and a few other resources necessary for the development of a Java Application.

JDK contains:

- Java Runtime Environment (JRE),
- An interpreter/loader (Java),
- A compiler (javac),
- An archiver (jar) and many more.

The Java Runtime Environment in JDK is usually called Private Runtime because it is separated from the regular JRE and has extra content. The Private Runtime in JDK contains a JVM and all the class libraries present in the production environment, as well as additional libraries useful to developers, e.g, internationalization libraries and the IDL libraries.

Most Popular JDKs:

- **Oracle JDK:** the most popular JDK and the main distributor of Java11,
- **OpenJDK:** Ready for use: JDK 15, JDK 14, and JMC,
- **Azul Systems Zing:** efficient and low latency JDK for Linux os,
- **Azul Systems:** based Zulu brand for Linux, Windows, Mac OS X,
- **IBM J9 JDK:** for AIX, Linux, Windows, and many other OS,
- **Amazon Corretto:** the newest option with the no-cost build of OpenJDK and long-term support.

Compile and Run Java Code using JDK:

You can use the JDK compiler to convert your Java text file into an executable program. Your Java text segment is converted into **bytecode** after compilation which carries the **.class** extension.

First, create a Java text file and save it using a name. Here we are saving the file as Hello.java.

- Java

```
class Hello{

  public static void main (String[] args) {

    System.out.println("Hello Students!");

  }

}
```

After that just simply use the **javac** command, which is used for the compilation purpose in Java.

“C:\Program Files\Java\jdk-11.0.9\bin>javac Hello.java”

You can notice now that the *Hello.class* file is being created in the same directory as Hello.java. Now you can run your code by simply using the **java Hello** command, which will give you the desired result according to your code. Please remember that you don’t have to include the .class to run your code.

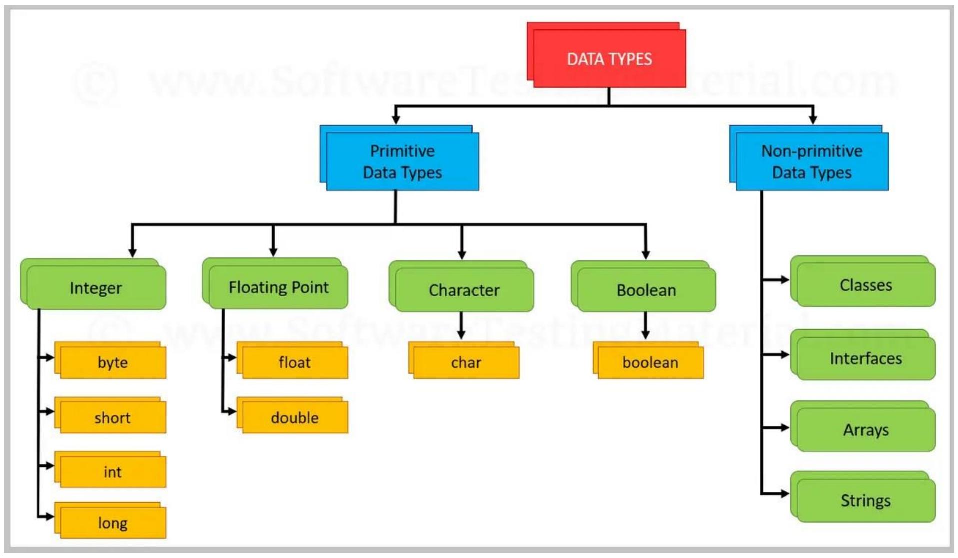
C:\ Program Files\Java\jdk-20\bin> java Hello

(Output:) Hello Students!

UNIT-II

❖ Datatypes:

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:



a) **Primitive Data Types:**In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

1. **Boolean:**

➢ The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

➢ The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

➢ Default value is false and default size is 1bit.

Example:

Boolean one = false

2. **Byte:**

➢ The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

➢ The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example:

byte a = 10, byte b = -20

3. **Short:**

- The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.
- The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example:

short s = 10000, short v = -5000

4. **Int:**

- The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.
- The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example:

int a = 100000, int b = -200000

5. **Long:**

- The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0.
- The long data type is used when you need a range of values more than those provided by int.

Example:

long a = 100000L, long b = -200000L

6. **Float:**

- The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited.
- It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers.
- The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example:

float f1 = 234.5f

7. **Double:**

- The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited.
- The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example:

double d1 = 12.3

8. Char:

- The char data type is a single 16-bit Unicode character.
- Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example:

char letterA = 'A'

b) **Non-primitive data types:**Unlike primitive data types, these are not predefined. These are user-defined data types created by programmers. These data types are used to store multiple values.

1.**Class:** A **class** in Java is a user defined data type i.e. it is created by the user. It acts a template to the data which consists of member variables and methods.

2.interface: An **interface** is similar to a class however the only difference is that its methods are abstract by default i.e. they do not have body. An interface has only the final variables and method declarations. It is also called a fully abstract class.

3.Array: An **array** is a data type which can store multiple homogenous variables i.e., variables of same type in a sequence. They are stored in an indexed manner starting with index 0. The variables can be either primitive or non-primitive data types.

4.String: A string represents a sequence of characters for example "Javanotes", Hello world", etc. String is the class of Java.

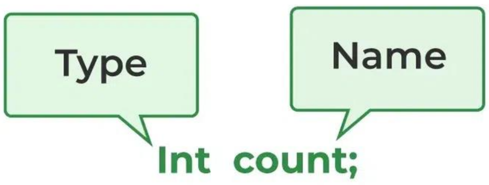
❖ Variables in Java

Java variable is a name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- Variables in Java are only a name given to a memory location. All the operations done on the variable affect that memory location.
- In Java, all variables must be declared before use.

Declare Variables in Java

We can declare variables in Java as pictorially depicted below as a visual aid.



From the image, it can be easily perceived that while declaring a variable, we need to take care of two things that are:

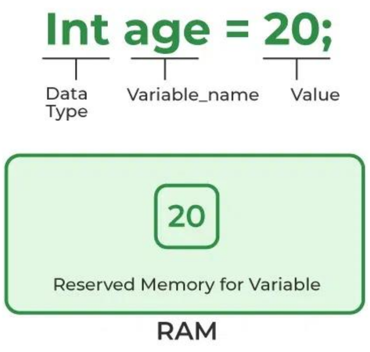
1. **datatype**: Type of data that can be stored in this variable.
2. **data_name:** Name was given to the variable.

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

Initialize Variables in Java

It can be perceived with the help of 3 components that are as follows:



- **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.

❖ Operators in java:

Operators are used to perform operations on variables and values.

There are many types of operators in Java which are given below:

- Unary Operator
- Arithmetic Operator
- Shift Operator
- Relational Operator
- Bitwise Operator

- Logical Operator
- Ternary Operator and
- Assignment Operator.

i. **Unary Operator:**

In Java, the **unary operator** is an operator that can be used only with an operand.

It is used to represent the **positive** or **negative** value, **increment/decrement** the value by 1, and **complement** a Boolean value.

erator name	**mbol**	**scription**	**ample**	**uivalent Expression**
ary Plus	**+**	is used to represent the **positive** value.	**+a**	**a**
ary Minus	**-**	is used to represent the **negative** value.	**-a**	**-**
rement Operator	**++**	**ncrements** the value of a variable by **1**.	**++a** **or** **a++**	**a=a+1**
crement operator	**--**	**lecrements** the value of a variable by **1**.	**--a** **or** **a--**	**a=a-1**
gical Complement Operator	**!**	**nverts** the value of a boolean variable.	**!true**	**-**

ii. **Arithmetic Operator:**

The way we calculate mathematical calculations, in the same way, Java provides **arithmetic operators** for mathematical operations. It provides operators for all necessary mathematical calculations.

erator	**me**	**scription**	**ample**
+	dition	ds two values together	**a+b**
-	btraction	btracts one value from another	**a-b**
*****	ıltiplication	ıltiplies two values	**a*b**

/	ision	ides one value by another	**a/b**
%	dulus	turns the division remainder	**a%b**

iii. **Shift Operator:**

In Java, **shift operators** are the special type of operators that work on the bits of the data. These operators are used to shift the bits of the numbers from left to right or right to left depending on the type of shift operator used. There are three types of shift operators in Java:

- **Signed Left Shift Operator (<<):**

The signed left shift operator is a special type of operator used to move the bits of the expression to the left according to the number specified after the operator.

- **Signed Right Shift Operator (>>):**

The signed right shift operator is a special type of operator used to move the bits of the expression to the right according to the number specified after the operator.

- **Unsigned Right Shift Operator (>>>):**

The unsigned right shift operator is a special type of right shift operator that does not use the signal bit to fill in the sequence. The unsigned sign shift operator on the right always fills the sequence by 0.

iv. Relational Operator:

Relational operators are used to check the relationship between two operands.

erator	**scription**	**ample**
==	qual To	= 5 returns **false**
!=	t Equal To	= 5 returns **true**
>	eater Than	5 returns **false**
<	ss Than	5 returns **true**
>=	eater Than or Equal To	= 5 returns **false**
<=	ss Than or Equal To	= 5 returns **true**

v. **Bitwise Operator:**

In Java, bitwise operators perform operations on integer data at the individual bit-level. Here, the integer data includes byte,int,short, and long types of data.

There are 7 operators to perform bit-level operations in Java.

erator	scription
|	wise OR
&	wise AND
^	wise XOR
~	wise Complement
<<	ft Shift
>>	ned Right Shift
>>>	signed Right Shift

vi. **Logical Operator:**

The **Java Logical Operators** work on the Boolean operand. It's also called Boolean logical operators. It operates on two Boolean values, which return Boolean values as a result.

erator	scription	rking
&&	gical AND	both operands are true then only "logical AND operator" evaluate true.
||	gical OR	e logical OR operator is only evaluated as true when one of its operands evaluates true. If either or both expressions evaluate to true, then the result is true.
!	gical Not	gical NOT is a Unary Operator, it operates on single operands. It reverses the value of operands, if the value is true, then it gives false, and if it is false, then it gives true.

vii. Ternary Operator:

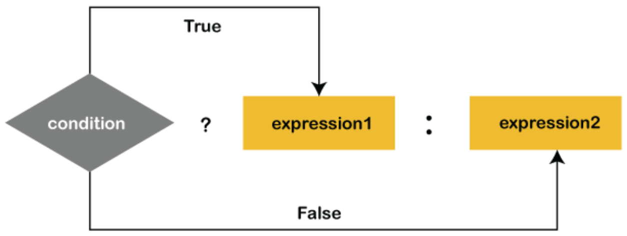
- The meaning of **ternary** is composed of three parts. The **ternary operator**

(? :) consists of three operands. It is used to evaluate Boolean expressions.

- The operator decides which value will be assigned to the variable. It is the only conditional operator that accepts three operands.
- It can be used instead of the if-else statement. It makes the code much more easy, readable, and shorter.

Syntax:

variable = (condition) ? expression1 : expression2



viii. Assignment Operations:

Assignment operators are used to assign values to variables.

erator	scription	ample
=	nple assignment operator. Assigns values from right side operands to left side operand.	= A + B will assign value of A + B into C
+=	d AND assignment operator. It adds right operand to the left operand and assigns the result to the left operand.	+= A is equivalent to C = C + A
-=	btract AND assignment operator. It subtracts right operand from the left operand and assigns the result to the left operand.	= A is equivalent to C = C – A
*=	ltiply AND assignment operator. It multiplies right operand with the left operand and assigns the result to the left operand.	= A is equivalent to C = C * A
/=	ide AND assignment operator. It divides left operand with the right operand and assigns the result to the left operand.	= A is equivalent to C = C / A
%=	dulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	%= A is equivalent to C = C % A
<<=	ft shift AND assignment operator.	<<= 2 is same as C = C << 2
>>=	ht shift AND assignment operator.	>>= 2 is same as C = C >> 2

&=	wise AND assignment operator.	&= 2 is same as C = C & 2
^=	wise exclusive OR and assignment operator.	^= 2 is same as C = C ^ 2
|=	wise inclusive OR and assignment operator.	= 2 is same as C = C | 2

❖ Control statements in java:

- Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear.
- However, Java provides statements that can be used to control the flow of Java code.
- Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
   - if statements
   - switch statement
2. Loop statements
   - do while loop
   - while loop
   - for loop
3. Jump statements
   - break statement
   - continue statement

▪ Decision Making statements:

- As the name suggests, decision-making statements decide which statement to execute and when.
- Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.
- There are two types of decision-making statements in Java, i.e., *If statement* and *switch statement*.

a) If Statement:

- The "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition.
- The condition of the If statement gives a Boolean value, either true or false.
- In Java, there are four types of if-statements given below.

i. <u>Simple if statement:</u>

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax:

```
if(condition)
{
 statement 1; //executes when condition is true
}
```

```
class Biggrst {
public static void main(String[] args) {
int x = 10;
int y = 5;
if(x<y) {
System.out.println(x+ " is greater value");
}
}
}
```

Output:

```
10 is greater value
```

ii. **<u>if-else statement:</u>**

The <u>if-else statement</u> is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
if(condition)
{
```

```
statement 1; //executes when condition is true

 }

 else

{

statement 2; //executes when condition is false

 }
```

```
class EvenOrOdd {
public static void main(String[] args) {
int x = 10;
if(x%2==0) {
System.out.println(x +  “is even");
}  else
System.out.println(x + “ is odd");
}
}
```

Output:

```
10 is even
```

iii. <u>if-else-if ladder:</u>

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax:

```
if(condition 1)

 {

statement 1; //executes when condition 1 is true
```

```
 }

**else if**(condition 2)

 {

 statement 2; //executes when condition 2 is true

 }

**else**

{

statement 2; //executes when all the conditions are false

}
```

```
import java.util.Scanner;
class Biggest{
public static void main(String []args){
int a,b,c;
Scanner s=new Scanner(System.in);
System.out.println("enter three numbers");
a=s.nextInt();
b=s.nextInt();
c=s.nextInt();
if((a>b)&&(a>c)){
System.out.print(a+"is biggest");
}
else if(b>c){
System.out.print(b+"is biggest");
}
else
System.out.print(c+"is biggest");
}
}
```

Output:

```
enter three numbers
30
40
20
20 is biggest
```

iv. **Nested if-statement:**

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax:

```
if(condition 1)

 {

 statement 1; //executes when condition 1 is true

if(condition 2)

{

statement 2; //executes when condition 2 is true

  }

 else

{

statement 2; //executes when condition 2 is false

 }

 }
```

b) Switch statement:

Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java.
- Cases cannot be duplicate.
- Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.

Syntax:

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

```
class Test {

  public static void main(String args[]) {

    char grade = 'C';

    switch(grade) {
      case 'A' :
        System.out.println("Excellent!");
        break;
      case 'B' :
      case 'C' :
        System.out.println("Well done");
        break;
      case 'D' :
        System.out.println("You passed");
      case 'F' :
        System.out.println("Better try again");
        break;
      default :
        System.out.println("Invalid grade");
    }
    System.out.println("Your grade is " + grade);
  }
}
```

Output

```
Well done
Your grade is C
```

- **Loop statements:**

- In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true.
- However, loop statements are used to execute the set of instructions in a repeated order.
- The execution of the set of instructions depends upon a particular condition.
- In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

a) for loop:

- In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.
- We use the for loop only when we exactly know the number of times, we want to execute the block of code.

Syntax:

```
for(initialization, condition, increment/decrement)

{

//block of statements

}
```

```
public class ForExample {
public static void main(String[] args) {
  for(int i=1;i<=10;i++){
    System.out.print(i+"\t");
  }
}
}
```

Output:

```
1          2          3          4          5          6          7          8          9          10
```

b) while loop:

- The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop.

- Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.
- It is also known as the entry-controlled loop since the condition is checked at the start of the loop.
- If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

Syntax:

```
while(condition)

{

 //looping statements

 }
```

```
public class WhileExample {
public static void main(String[] args) {
int i=10;
  while(i>0){
    System.out.print(i+”\t”);
    i--;
  }
}
}
```

Output:

```
10        9         8         7         6         5         4         3         2         1
```

c) <u>do-while loop:</u>

- The do-while loop checks the condition at the end of the loop after executing the loop statements.
- When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.
- It is also known as the exit-controlled loop since the condition is not checked in advance.

Syntax:

```
do

{

//statements

}

while (condition);
```

```
public class DoWhileExample {
public static void main(String[] args) {
int i=1;
  {
    System.out.print(i+"\t");
    i++;
  } while(i<10);
}
}
```

Output:

```
1          2          3          4          5          6          7          8          9          10
```

- ## Jump statements:

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

a) <u>Break statement:</u>

- The break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement.
- However, it breaks only the inner loop in the case of the nested loop.
- The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

Syntax:

break;

```
public class ForExample {
public static void main(String[] args) {
  for(int i=1;i<=10;i++){
    if(i==5)
    {
    break;
   }
    System.out.print(i+"\t");
  }
}
}
```

Output:

```
1          2          3          4          5
```

b) Continue statement:

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Syntax:

continue;

```
public class ForExample {
```

```
public static void main(String[] args) {
  for(int i=1;i<=10;i++){
    if((i>4)&& (i<7))
    {
    continue;
   }
    System.out.print(i+"\t");
  }
}
}
```

Output:

```
1          2          3          4          7          8          9          10
```

- **Return statement:**

In Java programming, the return statement is used for returning a value when the execution of the block is completed.

Returning a value from a method:

- In Java, every method is declared with a return type such as int, float, double, string, etc.
- These return types required a return statement at the end of the method. A return keyword is used for returning the resulted value.
- The void return type doesn't require any return statement. If we try to return a value from a void method, the compiler shows an error.

Following are the important points must remember while returning a value:

- The return type of the method and type of data returned at the end of the method should be of the same type. For example, if a method is declared with the float return type, the value returned should be of float type only.

Syntax:

```
return returnvalue;
```

❖ Java Arrays (One Dimensional array)

An array is a collection of similar types of data. The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

1. Declare an array in Java

In Java, here is how we can declare an array.

```
dataType[] arrayName;
```

- ❖ *dataType* - it can be primitive data types like `int`, `char`, `double`, `byte`, etc.
- *arrayName* - it is an identifier

For example,

```
double[] data;
```

Here, *data* is an array that can hold values of type `double`.

2. Memory allocation

- we have to allocate memory for the array in Java. *Memory representation*. When you use `new` to create an array, Java reserves space in memory for it (and initializes the values). This process is called *memory allocation*.

For example,

```
double[] data;
data = new double[10];
```

Here, the array can store **10** elements. We can also say that the **size or length** of the array is 10.

In Java, we can declare and allocate the memory of an array in one single statement. For example,

```
double[] data = new double[10];
```

3. Initialize Arrays in Java

In Java, we can initialize arrays during declaration. For example,

```
int[] age = {12, 4, 5, 2, 5};
```

Here, we have created an array named age and initialized it with the values inside the curly brackets.

Note that we have not provided the size of the array. In this case, the Java compiler automatically specifies the size by counting the number of elements in the array

In the Java array, each memory location is associated with a number. The number is known as an array index.

- Array indices always start from 0. That is, the first element of an array is at index 0.
- If the size of an array is *n*, then the last element of the array will be at index *n-1*.

4. Access Elements of an Array

We can use loops to access all the elements of the array at once.

Looping Through Array Elements

In Java, we can also loop through each element of the array. For example,

Example: Using For Loop

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

  int[] age = {12, 4, 5};
  System.out.println("Using for Loop:");

  for(int i = 0; i < age.length; i++) {
System.out.println(age[i]);
}
}
}
```

Output

```
Using for Loop:
12
4
5
```

In the above example, we are using the for Loop in Java to iterate through each element of the array. Notice the expression inside the loop,

```
age.length
```

Here, we are using the `length` property of the array to get the size of the array.

❖ Types of Array

In Java, there are two types of arrays:

1. Single-Dimensional Array
2. Multi-Dimensional Array

Single-Dimensional Array:

- ➢ An array that has **only one subscript** or one dimension is known as a single-dimensional array. It is just a list of the same data type variables.
- ➢ One dimensional array can be of either one row and multiple columns or multiple rows and one column.

- The declaration and initialization of an single-dimensional array is same as array's initialization and declaration.

Example:

int marks[] = {56, 98, 77, 89, 99};

Multi-Dimensional Array:

- A multi-dimensional array is just an array of arrays that represents multiple rows and columns.
- In multi-dimensional arrays, we have two categories:

  i. Two-Dimensional Arrays
  ii. Three-Dimensional Arrays

Two-Dimensional Arrays:

- An array involving two subscripts [] [] is known as a two-dimensional array. They are also known as the array of the array. Two-dimensional arrays are divided into rows and columns and are able to handle the data of the table.

Syntax:

DataTypeArrayName[row_size][column_size];

Example:

int arr[5][5];

Three-Dimensional Arrays:

- When we require to create two or more tables of the elements to declare the array elements, then in such a situation we use three-dimensional arrays.
- 3D array adds an extra dimension to the 2D array to increase the amount of space.
- Eventually, it is set of the 2D array. Each variable is identified with three indexes; the last two dimensions represent the number of rows and columns, and the first dimension is to select the block size.
- The first dimension depicts the number of tables or arrays that the 3D array contains.

Syntax: DataTypeArrayName[size1][size2][size3];

Example:

int a[5][5][5];

❖ Strings

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object. An array of characters works same as Java string.

For example:

```
char[] ch={'c','o','m','p','u','t','e','r'};
String s=new String(ch);
```

is same as:

```
String s="computer";
```

Creating Strings:

There are two ways to create String object:

1. By string literal
2. By new keyword

1) By string literal:

- Java String literal is created by using double quotes.
- String objects are stored in a special memory area known as the "string constant pool".
- Each time you create a string literal, the JVM checks the "string constant pool" first.
- If the string already exists in the pool, a reference to the pooled instance is returned.
- If the string doesn't exist in the pool, a new string instance is created and placed in the pool.
- The string literal concept is used to make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

Syntax:

<String_Type><string_variable> = "<sequence_of_string>";

Example:

String s="welcome";

2) By new keyword:

String s=new String("Welcome");//creates two objects and one reference variable

- In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool.
- The variable s will refer to the object in a heap (non-pool).

Program:

```
public class StringExample
{
public static void main(String args[])
{
String s1="java";
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);
String s3=new String("example");
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}
}
```

Output:

```
java
strings
example
```

String handling functions using Java

A **string** is a collection of characters. In Java, a string is an object that represents a collection of objects. A string is a predefined class used to create string objects. It is an **immutable object**, which means it can't be updated once created.

The string class has a set of built-in-methods, defined below.

- `charAt()`: It returns a character at a specified position.
- `equals()`: It compares the two given strings and returns a `Boolean`, that is, `True` or `False`.
- `concat()`: Appends one string to the end of another.
- `length()`: Returns the length of a specified string.
- `toLowerCase()`: Converts the string to lowercase letters.
- `toUpperCase()`: Converts the string to uppercase letters.
- `indexOf()`: Returns the first found position of a character.
- `substring()`: Extracts the substring based on index values, passed as an argument.

```
class StringFunction{

 public static void main(String []args)

 {

  String s1="ADITYA ";

  String s2="Degree College for Women";

  boolean x=s1.equals(s2);

  System.out.println("String1:"+s1);

  System.out.println("String2:"+s2);

  System.out.println("Compare s1 and s2:"+x);

  System.out.println("Character at 3rd position in "+s1+" is:"+s1.charAt(2));

  System.out.println("Cocadination of two strings is:"+s1.concat(s2));

  System.out.println("Lenth of"+s1+" is:"+s1.length());

  System.out.println("convert "+s1+" in Lower case:"+s1.toLowerCase());

  System.out.println("convert "+s2+" in Upper case:"+s2.toUpperCase());

  System.out.println("g index position in "+s2+" is:"+s2.indexOf('g'));

  System.out.println("substring upto 4 in "+s1+"is:"+s1.substring(0,4));

  System.out.println("substring of after 7th position in "+s2+" is :"+s2.substring(7));

 }

}
```

Output:

```
String1:ADITYA
```

```
String2:Degree College for Women

Compare s1 and s2:false

Character at 3rd position in ADITYA  is:I

Cocadination of two strings is:ADITYA Degree College for Women

Lenth ofADITYA  is:7

convert ADITYA  in Lower case:aditya

convert Degree College for Women in Upper case:DEGREE COLLEGE FOR WOMEN

g index position in Degree College for Women is:2

substring upto 4 in ADITYA is:ADIT

substring of after 7th position in Degree College for Women is :College for Women
```

❖ StringBuffer

StringBuffer is a class in Java that represents a mutable sequence of characters. It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time.

Here are some important features and methods of the StringBuffer class:

1. StringBuffer objects are mutable, meaning that you can change the contents of the buffer without creating a new object.
2. The initial capacity of a StringBuffer can be specified when it is created, or it can be set later with the ensureCapacity() method.
3. The append() method is used to add characters, strings, or other objects to the end of the buffer.
4. The insert() method is used to insert characters, strings, or other objects at a specified position in the buffer.
5. The delete() method is used to remove characters from the buffer.
6. The reverse() method is used to reverse the order of the characters in the buffer.

Mutable String

A String that can be modified or changed is known as mutable String. StringBuffer and StringBuilder classes are used for creating mutable strings.

1) StringBuffer Class append() Method

The append() method concatenates the given argument with this String.

StringBufferExample.java

```
class StringBufferExample{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");
System.out.println(sb);
}
}
```

Output:

```
Hello Java
```

2) StringBuffer insert() Method

The insert() method inserts the given String with this string at the given position.

StringBufferExample2.java

```
class StringBufferExample2{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");
System.out.println(sb);
}
}
```

Output:

```
HJavaello
```

3) StringBuffer replace() Method

The replace() method replaces the given String from the specified beginIndex and endIndex.

StringBufferExample3.java

```
class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);
}
}
```

Output:

```
HJavalo
```

4) StringBuffer delete() Method

The delete() method of the StringBuffer class deletes the String from the specified beginIndex to endIndex.

StringBufferExample4.java

```
class StringBufferExample4{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);
}
}
```

Output:

```
Hlo
```

5) StringBuffer reverse() Method

The reverse() method of the StringBuilder class reverses the current String.

StringBufferExample5.java

```
class StringBufferExample5{
```

```
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);
}
}
```

Output:

```
olleH
```

6) StringBuffer capacity() Method

The capacity() method of the StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

StringBufferExample6.java

```
class StringBufferExample6{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());
sb.append("Hello");
System.out.println(sb.capacity());
sb.append("java is my favourite language");
System.out.println(sb.capacity());
}
}
```

Output:

```
16
16
34
```

❖ Naming Conventions of the Different Identifiers

The following table shows the popular conventions used for the different identifiers.

Identifiers Type	Naming Rules	Examples

Class	It should start with the uppercase letter.
	It should be a noun such as Color, Button, System, Thread, etc.
	Use appropriate words, instead of acronyms.
		public class **Employee**
		{
		//code snippet
		}

Interface	It should start with the uppercase letter.
	It should be an adjective such as Runnable, Remote, ActionListener.
	Use appropriate words, instead of acronyms.
		interface **Printable**
		{
		//code snippet
		}

Method	It should start with lowercase letter.
	It should be a verb such as main(), print(), println().
	If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
		class Employee
		{
		// method
		void **draw()**
		{
		//code snippet
		}
		}

Variable	It should start with a lowercase letter such as id, name.
	It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore).
	If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
	Avoid using one-character variables such as x, y, z.
		class Employee
		{
		// variable
		int **id**;
		//code snippet
		}

Package	It should be a lowercase letter such as java, lang.
	If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.
		//package
		package **com.javatpoint;**
		class Employee
		{
		//code snippet
		}

Constant	It should be in uppercase letters such as RED, YELLOW.
	If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY.
	It may contain digits but not as the first letter.
		class Employee
		{
		//constant
		static final int **MIN_AGE** = 18;
		//code snippet
		}

❖ Objects and Classes in Java

Java Classes

A class in Java is a set of objects which shares common characteristics/ behavior and common properties/ attributes. It is a user-defined blueprint or prototype from which objects are created. For example, Student is a class while a particular student named Ravi is an object.

Properties of Java Classes

1. Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
2. Class does n
3. ot occupy memory.
3. Class is a group of variables of different data types and a group of methods.
4. A Class in Java can contain:
    - Data member
    - Method
    - Constructor
    - Nested Class
    - Interface

Class Declaration in Java

access_modifier **class** <*class_name*>
{
  data member;
  method;
  constructor;
  nested class;
  interface;
}

Components of Java Classes

In general, class declarations can include these components, in order:

1. ***Modifiers****: A class can be public or has default access (Refer this for details).*
2. ***Class keyword:*** *class keyword is used to create a class.*
3. ***Class name:*** *The name should begin with an initial letter (capitalized by convention).*
4. ***Superclass(if any):*** *The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.*
5. ***Interfaces(if any):*** *A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.*
6. ***Body:*** *The class body is surrounded by braces, { }.*

Constructors are used for initializing new objects. Fields are variables that provide the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

There are various types of classes that are used in real-time applications such as nested classes, anonymous classes, and lambda expressions.

Java Objects

An object in Java is a basic unit of Object-Oriented Programming and represents real-life entities. Objects are the instances of a class that are created to use the attributes and methods of a class. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. **State**: It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by the methods of an object. It also reflects the response of an object with other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog

Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, and “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Note: *When we create an object which is a non primitive data type, it’s always allocated on the heap memory.*

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be **instantiated**. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Example:

As we declare variables like (type name;). This notifies the compiler that we will use the name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variables , the type must be strictly a concrete class name. In general, we **can’t** create objects of an abstract class or an interface.

Dog tuffy;

If we declare a reference variable(tuffy) like this, its value will be undetermined(null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

Initializing a Java object

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

Example:

```
// Class Declaration

public class Dog {
  // Instance Variables
  String name;
  String breed;
  int age;
  String color;

  // Constructor Declaration of Class
  public Dog(String name, String breed, int age,
         String color)
  {
    this.name = name;
    this.breed = breed;
    this.age = age;
    this.color = color;
  }

  // method 1
  public String getName() { return name; }

  // method 2
  public String getBreed() { return breed; }

  // method 3
  public int getAge() { return age; }

  // method 4
  public String getColor() { return color; }

  @Override public String toString()
  {
    return ("Hi my name is " + this.getName()
        + ".\nMy breed,age and color are "
        + this.getBreed() + "," + this.getAge()
        + "," + this.getColor());
  }

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

```
    {
      Dog tuffy  = new Dog("tuffy", "papillon", 5, "white");
      System.out.println(tuffy.toString());
    }
  }
```

Output

Hi my name is tuffy.

My breed,age and color are papillon,5,white

Difference between Java Class and Objects

The differences between class and object in Java are as follows:

Class	Object
Class is the blueprint of an object. It is used to create objects.	An object is an instance of the class.
No memory is allocated when a class is declared.	Memory is allocated as soon as an object is created.
A class is a group of similar objects.	An object is a real-world entity such as a book, car, etc.
Class is a logical entity.	An object is a physical entity.
A class can only be declared once.	Objects can be created many times as per requirement.
An example of class can be a car.	Objects of the class car can be BMW, Mercedes, Ferrari, etc.

Let's see the example:

```
class Rectangle{
 int length;
 int width;
 void insert(int l,int w){
  length=l;
  width=w;
 }
 void calculateArea(){System.out.println(length*width);}
}
class TestRectangle2{
 public static void main(String args[]){
  Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
}
}
```

Output:

```
55
45
```

❖ Constructors

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Rules for creating Java constructor

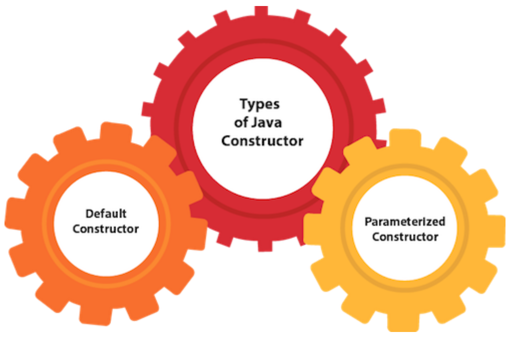
There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

1. <class_name>(){}

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation

```
class Data {
    //default constructor
        public Data() {
                System.out.println("Hello students");
        }
        public static void main(String[] args) {
                Data d = new Data();
        }
}
```

Output:

```
Hello students
```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Use the parameterized constructor

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student{
   int id;
   String name;

   Student(int i,String n){
   id = i;
   name = n;
   }

   void display(){System.out.println(id+" "+name);}

   public static void main(String args[]){

   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");

   s1.display();
   s2.display();
  }
}
```

Test it Now

Output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
class Student5{
   int id;
   String name;
   int age;
   //creating two arg constructor
   Student5(int i,String n){
   id = i;
   name = n;
   }
   //creating three arg constructor
   Student5(int i,String n,int a){
   id = i;
   name = n;
   age=a;
   }
   void display(){System.out.println(id+" "+name+" "+age);}

   public static void main(String args[]){
   Student5 s1 = new Student5(111,"Karan");
   Student5 s2 = new Student5(222,"Aryan",25);
   s1.display();
   s2.display();
  }
}
```

Output:

```
111 Karan 0
222 Aryan 25
```

❖ Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

❖ Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

1) Private

The private access modifier is accessible only within the class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
  A obj=new A();
  System.out.println(obj.data);//Compile Time Error
  obj.msg();//Compile Time Error
  }
}
```

2) Default

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A{
 void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
class B{
 public static void main(String args[]){
  A obj = new A();//Compile Time Error
  obj.msg();//Compile Time Error
 }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;

class B extends A{
 public static void main(String args[]){
  B obj = new B();
  obj.msg();
 }
}
```

```
Output:Hello
```

---

4) Public

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
//save by A.java

package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
//save by B.java

package mypack;
import pack.*;

class B{
 public static void main(String args[]){
  A obj = new A();
  obj.msg();
 }
}
```

```
Output:Hello
```

❖ Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
   System.out.println("Welcome to package");
  }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

1. javac -d directory javafilename

For **example**

1. javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java
To Run: java mypack.Simple

```
Output:Welcome to package
```

access package from another package

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

*1) Using packagename.**

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
 public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B{
 public static void main(String args[]){
  A obj = new A();
  obj.msg();
 }
}
```

```
Output:Hello
```

❖ Java Packages & API

A Java package is a group of related classes and interfaces that are stored together in a directory or subdirectory. Java API (Application Programming Interface) is a set of prewritten Java classes that are provided to developers as a means of performing common tasks, such as manipulating data or creating graphical user interfaces.

To use a Java package or API, you must first import it into your Java code using the "import" statement.

Built-in Packages

Java comes with several built-in packages that developers can use in their code. These packages are part of the Java API and provide a wide range of functionality, including data manipulation, networking, and user interface design.

Some examples of built-in packages in Java include:

java.lang: provides fundamental classes and interfaces, such as String, Integer, and Boolean.

java.util: provides utility classes and interfaces, such as ArrayList and Date.

java.io: provides classes for input and output operations, such as reading and writing files.

Import a Class

To import a class in Java, you need to use the import statement followed by the fully qualified name of the class.
For example:

```
 import java.util.ArrayList;
public class MyClass {
public static void main(String[] args) {
ArrayList myList = new ArrayList();
myList.add("Hello");
myList.add("World");
System.out.println(myList);
}
 }
```

Output:

```
Hello World
```

In the above example, we imported the **ArrayList** class from the **java.util** package using the import statement. We then created an instance of the **ArrayList** class and added two strings to it. Finally, we printed the contents of the **ArrayList** using the **System.out.println** statement.

Import a Package

To import an entire package in Java, you need to use the import statement followed by the name of the package.

For example:

```
import java.util.*;
public class MyClass {
public static void main(String[] args) {
ArrayList myList = new ArrayList();
myList.add("Hello");
myList.add("World");
System.out.println(myList);
}
}
```

Output:

```
Hello world
```

In the above example, we imported the entire **java.util** package using the import statement. We then used the **ArrayList** class from the **java.util** package in our code to create an instance of the **ArrayList** class and add two strings to it. Finally, we printed the contents of the **ArrayList** using the **System.out.println** statement.