Advanced Java Programming Syllabus

UNIT – I

Introduction to Oops and Classes: OOPs concepts, Programming Paradigm, Basics of Java,Data Types, Variables, Operators, Control Statements, Loops and Arrays.

Classes and Objects: Class Fundamentals - Declaring Objects - Introducing Methods - Overloading methods – Constructors - Parameterized Constructors - this Keyword.Class Features:Garbage Collection - the finalize () Method - Introducing Access Control - Understanding static - Introducing nested and inner classes - String class - String Buffer Class - Command Line Arguments.

Practice:

1. Write a Java Program to generate Fibonacci sequence for n numbers
2. Write a Java Program that prompts the user for an integer and then prints out all the prime numbers up to that Integer
3. Write a Java Program to demonstrate String and StringBuffer class methods.
4. Implement the concept of constructor overloading

UNIT – II

Inheritance: Inheritance Basics - Multilevel Hierarchy- Using super - Method overriding - Dynamic Method Dispatch- Abstract keyword- Using final with inheritance - The Object Class.

Interfaces and Packages: Inheritance in java with Interfaces – Defining Interfaces - Implementing Interfaces - Extending Interfaces- Creating Packages - CLASSPATH variable - Access protection - Importing Packages - Interfaces in a Package.

Practice:

1. Implement the concept of inheritance, super, abstract and final keywords.
2. Write a java program for dynamic dispatch of methods.
3. Write a Java Program to create and demonstrate packages
4. Implement the concept of package and interface.

UNIT – III

Exception Handling: Exception Handling Fundamentals, Exception Types, Uncaught Exceptions, Using Try and Catch, Multiple Catch clauses, Nested try Statements, Java's Built-in Exceptions, Creating your own Exception subclasses,

Multithreading Java: Thread Model - Life cycle of a Thread - Java Thread Priorities - Runnable interface and Thread Class- Thread Synchronization – Inter Thread Communication.

Practice:

1. Write a Java program to illustrate exception handling mechanism using multiple catch clauses.
2. Implement the concept of multi threading.

UNIT – IV

JDBC: Introduction to JDBC- Connecting to the database- Basic JDBC Operations – Essential JDBC Classes – JDBC Drivers – JDBC-ODBC Bridge – Connecting to a database with driver manager – JDBC database URL, CRUD Operations.

Practice:

1. Write a JDBC program to perform CRUD operations

UNIT – V

Java Servlets: Servlets Basics – Life Cycle of a Servlet –A Simple Servlet - The Servlet API – Servlet Interfaces – Generic Servlet Class- HttpServletRequest Interface – HttpServeltResponse

JSP: The JSP development model – component of jsp page – Page directive – Action – scriptlet – JSP expression, JSP Syntax and semantics, JSP in XML.

Practice:

1. Write a program to receive two numbers from a HTML form and display their sum in HTTPServlet.
2. Write a program to Authenticate using JSP
3. Write a JSP program calculates factorial values for an integer number, while the input is taken from an HTML form.

Additional Practice:

1. Write a Java Program, using String Tokenizer class, which reads a line of integers and then displays each integer and the sum of all integers.
2. Write a program to Authenticate using Java Servlet
3. Write a program Program to store the user information into cookies, and to display the above stored information by retrieving from cookies.
4. Write a Java program to solve Producer-Consumer problem using synchronization

UNIT-I

1.1 Introduction to OOP

- OOP means Object-oriented programming
- The dictionary meaning of the object is an article or entity that exists in the real world. The meaning of oriented is interested in a particular kind of entity.
- The object-oriented programming is basically a computer programming design methodology that organizes/ models software design around data and objects rather than functions and logic.
- The major objective is to eliminate some of the flaws encountered in the procedural approach.
- OOP allows us to decompose a problem into a number of entities called Objects and then build data and functions (methods) around these entities.
- The combination of data and methods make up an object.
- The data of an object can be accessed only by the methods associated with that object.
- It is well suited for programs that are large, complex, and actively updated or maintained.
- It simplifies software development and maintenance by providing major concepts such as abstraction, inheritance, polymorphism, and encapsulation.
- Examples: Smalltalk, Objective C, C++, Java and Python.
- Some of the features of object-oriented paradigm are:
    1. Programs are divided into what are known as Objects.
    2. Data is hidden and cannot be accessed by external functions.
    3. Objects may communicate with each other through methods.
    4. New data and methods can be easily added whenever necessary.
    5. Follows bottom-up approach in program design.

1.2 Concepts of OOPs (Characteristics of OOP)

1. Class:

- Class is a group of similar entities. It is only a logical component and not the physical entity.
- A class is a collection of similar type of objects.
- It is commonly known as the 'blueprint of an object'.
- It is a template that describes the kinds of state and behavior that the object supports.
- Classes are user-defined data types and behave like the built-in types of a programming language.
- Examples of Class are: Student, Fruit, Car, Room…

2. Objects:

- Objects are real-world entities which have a certain state and behavior.
- Objects are commonly known as an 'instance of a class'.
- Objects are the basic runtime entities in an object oriented system.
- Objects interact with each other and share information. This technique is termed Message Passing.
- Examples of Objects are: S1, S2,… are objects of Student class. Mango, Apple, Guava,… are objects of Fruit class.

3. Data Abstraction:

- Abstraction refers to the act of representing essential features without including the background details or explanations.
- Hiding internal details and showing functionality is known as abstraction.
- Classes use the concept of abstraction and are defined as the list of abstract attributes such as size, weight and cost and methods that operate on these attributes.

4. Encapsulation:

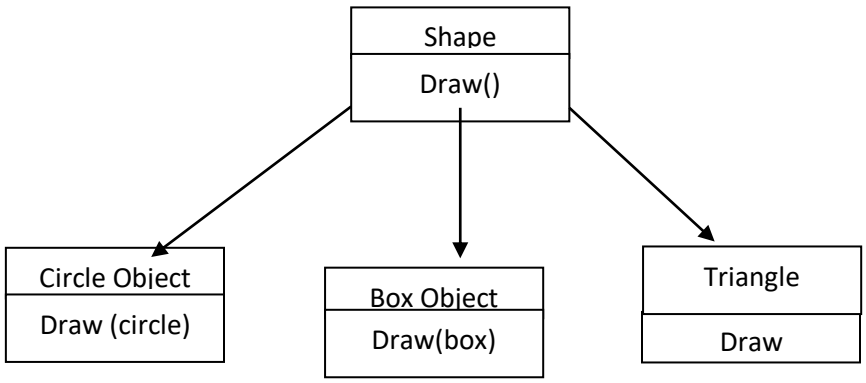
- The wrapping up of data and methods into a single unit (called class) is known as encapsulation.
- The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it.
- These methods provide the interface between the object's data and the program.

5. Inheritance:

- Inheritance is the process by which objects of one **class acquire the properties of objects of another class.**
- Inheritance supports the concept of hierarchical classification**.**
- In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes.

6. Polymorphism:

- Polymorphism means the ability to take more than one form.
- It is the ability to create a method, variable, or an object that has more than one form.
- For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.
- The following figure illustrates that a single function name can be used to handle different number and different types of arguments.



7. Dynamic Binding:

- Binding refers to the linking of a procedure **call to the code to be executed in response to the call**.
- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at runtime.
- It is associated with polymorphism and inheritance.

8. Message Communication:

- An object-oriented program consists of a set of objects that communicate with each other.

- The process of programming in an object-oriented language, involves the following basic steps:
  - Creating classes that define objects and their behaviour.
  - Creating objects from class definitions.
  - Establishing communication among objects.

1.3 Difference between OOP and Procedure Oriented Programming?

On the basis of	Procedural Programming	Object-oriented programming
Definition	It is a programming language that is derived from structure programming and based upon the concept of calling procedures.	It is a computer programming design methodology that organizes software design around data or objects rather than functions and logic.
Security	It is less secure than OOPs.	Data hiding is possible due to abstraction. So, it is more secure than procedural programming.
Approach	It follows a top-down approach.	It follows a bottom-up approach.
Data movement	In procedural programming, data moves freely within the system from one function to another.	In OOP, objects can move and communicate with each other via member functions.
Orientation	It is structure/procedure-oriented.	It is object-oriented.
Access modifiers	There are no access modifiers in procedural programming.	The access modifiers in OOP are named as private, public, and protected.
Inheritance	Procedural programming does not have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
Code reusability	There is no code reusability present in procedural programming.	It offers code reusability by using the feature of inheritance.
Overloading	Overloading is not possible in procedural programming.	In OOP, there is a concept of function overloading and operator overloading.
Importance	It gives importance to functions over data.	It gives importance to data over functions.
Complex problems	It is not appropriate for complex problems.	It is appropriate for complex problems.
Data hiding	There is not any proper way for data hiding.	There is a possibility of data hiding.
Program division	In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
Examples	Examples: C, Fortran, Pascal, and VB.	Examples: .NET, C#, Python, Java, VB.NET, and C++.

1.4 Introduction to Java

- Java is a general-purpose, object-oriented programming language developed by Sun Microsystems of USA in 1991.
- James Gosling, Mike Sheridan, and Patrick Naughton a team of Sun engineers known as the Green team initiated the Java language in 1991.
- James Gosling is known as the father of Java.

- Sun Microsystems released its first public implementation in 1996 as Java 1.0.
- Before Java, its name was Oak.
- It is mainly designed for the development of software for consumer electronic devices like TVs, VCRs, toaster and such other electronic machines.

1.5 Java Versions:

Java SE 1.0 (1996)	Java SE 12 (2019)
Java SE 1.1 (1997)	Java SE 13 (2019)
Java SE 1.2 (1998)	Java SE 14 (2020)
Java SE 1.3 (2000)	Java SE 15 (2020)
Java SE 1.4 (2002)	Java SE 16 (2021)
Java SE 5 (2004)	Java SE 17 (2021)
Java SE 6 (2006)	Java SE 18 (2022)
Java SE 7 (2011)	Java SE 19 (2022)
Java SE 8 (2014)	Java SE 20 (2023)
Java SE 9 (2017)	Java SE 21 (2023)
Java SE 10 (2018)	Java SE 22 (March 2024)
Java SE 11 (2018)	Java SE 23 (September 2024)

1.6 Features of Java

The inventors of Java wanted the language to be not only reliable, portable and distributed but also simple, compact and interactive. Sun Microsystems officially describes Java with the following features:

1. Compiled and Interpreted: Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making Java a two-stage system. First, Java compiler translates source code into bytecode instruction. Bytecodes are not machine instructions and therefore, in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program.

2. Simple, Small and Familiar: Java is a small and simple language. Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, goto statement and preprocessor header files etc. There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

3. Platform Independent: A platform is the hardware or software environment in which a program runs. 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).

4. Dynamic and Extensible: Java is a dynamic language. Java is capable of dynamically linking in new class libraries, methods and objects.

5. Robust: Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

6. Object-Oriented: Java is true object-oriented language. Almost everything in Java is an object. All program code and data reside within objects and classes. Java comes with an extensive set of classes,

arranged in packages, that we can use in our programs by inheritance. The object model in Java is simple and easy to extend.

7. Distributed: Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on Internet as easily as they can do in a local system.

8. Multithreaded: Multithreaded means handling multiple tasks simultaneously. We need not wait for the application to finish one task before beginning another. For example, listening to an audio clip while scrolling a page and downloading.

9. Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

10. High Performance: Java offers high performance using the JIT (Just In Time) compiler. The compiler only compiles that method which is being called. The JIT enhances the performance of interpreting byte code by caching interpretations.

1.7 How the Java is differ from other languages(C & C++)

Although Java was modeled after C and C++ languages, it differs from C and C++ in many ways. Java does not incorporate a number of features available in C and C++. A few major differences between C/C++ and Java languages include:

Java and C: Java is an object-oriented language and has mechanism to define classes and objects. In a effort to build a simple and safe language, the Java team did not include some of the C features in Java.

1. Java does not include the C unique statement keywords goto, sizeof and typedef.
2. Java does not contain the data types such as struct, union and enum.
3. Java does not define the type modifier keywords auto, extern, register, signed, and unsigned.
4. Java does not support an explicit pointer type.
5. Java does not have a preprocessor and therefore we cannot use #define, #include, and #ifdef statements.
6. Java does not support any mechanism for defining variable arguments to functions.
7. Java requires that the functions with no arguments must be declared with empty parenthesis and not with the void keyword as done in C.
8. Java adds new operators such as instanceof and >>>.
9. Java adds labeled break and continue statements.
10. Java adds many features required for object-oriented programming.

Java and C++: Java is a true object-oriented language while C++ is basically C with object-oriented extension. Listed below are some major C++ features that were differ from Java:

1. Java does not support operator overloading.
2. Java does not have template classes as in C++.
3. Java does not support multiple inheritance of classes. This is accomplished using a new feature called “interface”.
4. Java does not support global variables. Every variable and method I declared within a class and forms part of that class.
5. Java does not use pointers.
6. Java has replaced the destructor function with a finalize () function.
7. There are no header files in Java.

1.8 JDK (Java Development Kit)

The Java Development Kit (JDK) comes with a collecting of tools that are used for developing and running Java programs. They include:

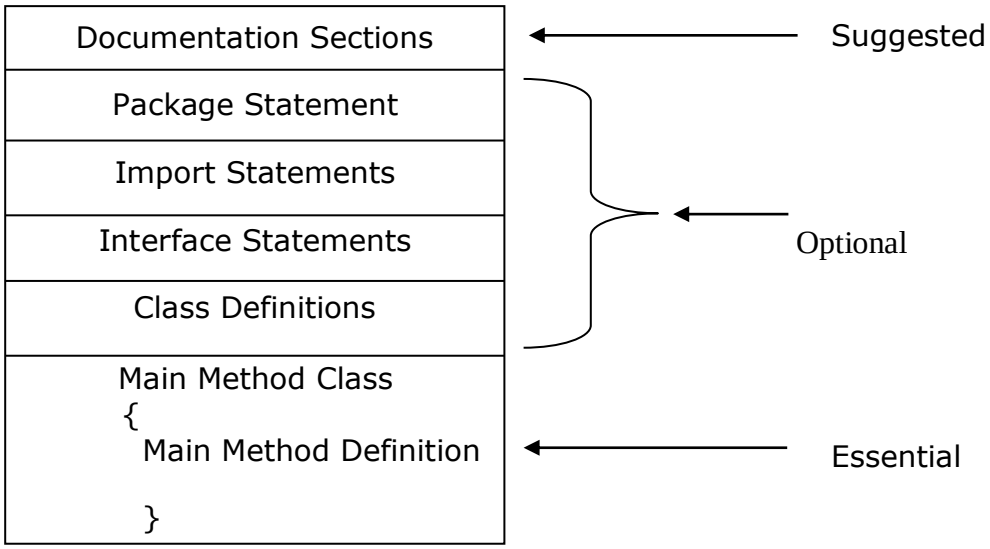
1. Appletviewer (for viewing Java applets) – Enables us to run Java applets.
2. Javac (Java Compiler) – Translates Java source code to byte code files that the interpreter can understand.
3. Java (Java Interpreter) – Runs applets and applications by reading and interpreting bytecode files.
4. Javap (Java disassembler) – Enables us to convert bytecode files into a program description.
5. Javah (for C header files) – Produces header files for use with native methods.
6. Javadoc (for creating HTML documents) – Creates HTML format documentation from Java source code files.
7. Jdb (Java debugger) – Helps us to find errors in our programs.

1.9 JVM

- Java Virtual Machine (JVM) is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language.
- In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.
- Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.
- JVM is a part of Java Runtime Environment (JRE).
- The JIT or Just-In-Time compiler is an essential part of the JRE (Java Runtime Environment)**.**
- In order to improve performance, JIT compilers interact with the Java Virtual Machine (JVM) at run time and compile suitable bytecode sequences into native machine code.

1.10 Structure of Java Program

A Java program may contain one or more sections as shown:



Documentation Section: It comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage. Comments must explain why and what of classes and how of algorithms. Java permits both the single-line (//….) comments and multi-line(/*….*/) comments. Further, Java also uses a third style of comment /**…*/ known as documentation comment, used for generating documentation automatically.

Package Statement: This statement declares a **package** name and informs the compiler that the classes defined here belong to this package.

Import Statements: This statement instructs the interpreter to load the classes available within the packages. This is similar to the **#include** statement in C/C++.

Interface Statements: An interface is like a class but includes a group of method declarations. These are used only when we wish to implement the multiple inheritance feature in the program.

Class Definitions: A Java program may contain multiple class definitions. Classes are primary and essential elements of a Java program. The number of classes used depends on the complexity of the problem.

Main Method Class: Since every Java stand-alone program requires a **main** method as its starting point, this class is the essential part of a Java program. The **main** method creates objects of various classes and establishes communications between them. On reaching the end of **main,** the program terminates and the control passes back to the operating system.

Simple java program:

```
class     First                    // Main class declaration
{                                  // main class definition start
    public static void main(String args[])     // Main method definition
    {
            System.out.println("Java is better than C++");    // output method
    }                          // End-of main method definition
}  // End-of Main class definition
```

The above program has only one class that contains the **main** method. A real-life application will generally require multiple classes. Let us discuss the program line by line and understand the unique features that constitute a Java program.

- **Class Declaration:** The first line class First;
  "class" is a keyword and declares that a new class definition follows. "First" is a Java identifier that specifies the name of the class to be defined.
- **Opening Brace:** Every class definition in Java begins with an opening brace "{" and ends with a matching closing brace "}", appearing in the last line in the example.
- **The Main Line:** The third line public static void main (String args [ ])
  Defines a method named **main.** Conceptually, this is similar to the **main()** function in C/C++. This is the starting point for the interpreter to begin the execution of the program. A Java application can have any number of classes but **only one** of them must include a **main** method to initiate the execution.
  This line contains a number of keywords, **public, static** and **void.**
  public: The keyword **public** is an access specifier that declares the **main** method is accessible to all other classes.

static: The keyword **static,** declares this method as one that belongs to the entire class and not a part of any objects of the class. The **main** must always be declared as **static** since the interpreter uses this method before any objects are created.

void: The type modifier **void** states that the **main** method does not return any value.

String args[]: declares a parameter named **args**, which contains an array of objects of the **String** class.

- **The Output Line:** The only executable statement in the above program is

  System.out.println ("Java is better than C++");

  This is similar to the **printf( )** statement of C. The **println( )** method is a member of the **out** object, which is a static data member of **System** class.
- Every Java statement must end with a semicolon (;).

1.11 Compile and Run the Standard Java program.

Here is a step by step process on how to run Java program:

Step 1: Open Notepad from Start menu by selecting Programs > Accessories > Notepad.

Step 2: Create a Source Code for your Java program.

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

Step 3: Save the file as **FirstProgram.java** in our working folder like **D:\javaprograms**

Step 4: Open the command prompt. Go to Directory **D:\javaprograms**. Compile the Java program using "javac" command, **D:\javaprograms>javac FirstProgram.java**

Step 5: Now the file named **FirstProgram.class** has been created.

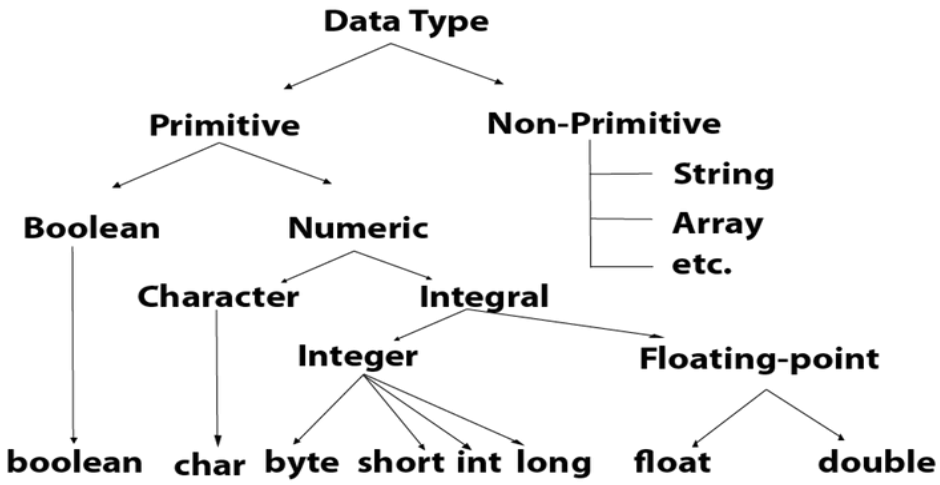
Step 6: To execute the code, enter the command "java" followed by the class name, as expected output **Hello World** is displayed now. **D:\javaprograms>java FirstProgram**

Note: Java is case sensitive Programming language. All code, commands, and file names should is used in consistent casing. **FirstProgram** is not same as **firstprogram.**

1.12 Data types in java

Data types: Data types specify the size and type of values that can be stored. Java language is rich in its data types. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Data types are mainly divided into two types. They are:

1. Primitive Data Types
2. Non Primitive Data Types / Reference / Object Data Types



1. Primitive Data Types:

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword.

byte:

- Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example : byte a = 100, byte b = -50

short:

- Short data type is a 16-bit signed two's complement integer
- Minimum value is -32,768 (-2^{15})
- Maximum value is 32,767 (inclusive) (2^{15} -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.
- Example: short s = 10000, short r = -20000

int:

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 (-2^{31})
- Maximum value is 2,147,483,647(inclusive) (2^{31} -1)
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: int a = 100000, int b = -200000

long:

- Long data type is a 64-bit signed two's complement integer
- Minimum value is -9,223,372,036,854,775,808(-2^{63})

- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^{63} -1)
- This type is used when a wider range than int is needed
- Default value is 0L
- Example: long a = 100000L, long b = -200000L

float:

- Float data type is a single-precision 32-bit IEEE 754 floating point
- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example : float f = 234.5f

double:

- double data type is a double-precision 64-bit IEEE 754 floating point
- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0d
- Example : double d = 123.4

boolean:

- boolean data type represents one bit of information
- There are only two possible values: true and false
- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example : boolean b = true

char:

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character
- Example − char ch = 'A'

2. Non Primitive Data Types (Reference variables):

Reference variables are created using defined constructors of the classes. They are used to access objects.

- Class objects and various type of array variables come under reference datatype.
- Default value of any reference variable is null.
- A reference variable can be used to refer any object of the declared type.
- Example: Animal a = new Animal("giraffe");

1.13 Variables in Java

- Variables are the data containers that save the data values during Java program execution.
- Every Variable in Java is assigned a data type that designates the type and quantity of value it can hold.
- A variable is a memory location name for the data.

- 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.
- In Java, all variables must be declared before use.
- A variable name can be chosen by the programmer in a meaningful way so as to reflect what it represents in the program.
- Variable names may consist of alphabets, digits, the underscore (_) and dollar characters, subject to the following conditions:
    1. They must not begin with a digit.
    2. Uppercase and lowercase are distinct, I.e. Total is not same as total or TOTAL.
    3. It should not be a keyword.
    4. White space is not allowed.
    5. Variable names can be of any length.

Declaration of variables: Declaration does three things:

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.
3. The place of declaration (in the program) decides the scope of the variable.

The general form of declaration of a variable is:

type variable1, variable2, ……..,variablen;

Variables are separated by commas. A declaration statement must end with a semicolon. Some valid declarations are:

int count;
float x,y;
char ch;

Giving values to variables: A variable must be given a value after it has been declared but before it is used in an expression. This can be achieved in two ways:

1. By using as assignment statement: A simple method of giving value to a variable is through the assignment statement as follows:

variablename = value;

For example,

count = 0;
x = y = 0;

Note: It is also possible to assign a value to a variable at the time of its declaration. This takes the form:

type variableName = value;

For example,

int count = 100;
char ch = 'x';

Note: The process of giving initial values to variables is known as the initialization.

2. Giving input at runtime: We may also give values to variables interactively through the keyboard using at runtime. There are two ways by which we can take Java input from the user or from a file.

1. BufferedReader Class 2. Scanner Class

Example program for Scanner class:

```
import java.util.*;
class Addition
```

```
{
    public static void main(String[] args)
    {
        int a,b,c;
        Scanner sc = new Scanner (System.in);
        System.out.println("Enter two numbers:");
        int a = sc.nextInt();
        int b= sc.nextInt();
        c=a+b;
        System.out.println("Sum=" + c);
    }
}
```

1.14 Different types of operators

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- Operators are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions.
- Java operators can be classified into a number of related categories as below;

1. Arithmetic operators
2. Relational Operators
3. Logical operators
4. Assignment operators
5. Increment & decrement operators
6. Bitwise operators
7. Conditional operators
8. Special operators

Arithmetic Operators Arithmetic operators are used to construct mathematical expressions as in algebra.

Java provides arithmetic operators. They are listed as +, -, *, and / . Arithmetic operators are used as shown below:

a-b a + b a*b a/b a % b

Example program:

```
class Test
{
    public static void main(String args[])
    {
        int a = 20, b =30;
        System.out.println("a+b = " + (a+b));
        System.out.println("a-b = " + (a-b));
        System.out.println("a*b = " + (a*b));
        System.out.println("a/b = " + (a/b));
    }
}
```

Relational Operators: We often compare two quantities, and depending on their relation, take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators like >, < , >= , <= ,==, != .

Program for Implementation of relational operators

```
class  Relational
{
    public static void main(String args[])
    {
        int a = 15, b = 20,  c = 15;
        System.out.println("a < b Is " + (a<b));
        System.out.println("a > b Is " + (a>b));
        System.out.println(" a == c Is " + (a==c));
        System.out.println("a <= c Is " + (a<=c));
        System.out.println("a >= b Is " + (a>=b));
        System.out.println(" b  != c Is " + (b!=c));
        System.out.println(" b == a+c is " + (b==a+c));
    }
}
```

Logical Operators: In addition to the relational operators, Java has three logical operators AND(&&), OR(||), NOT(!) . The logical operators && and | | are used when we want to form compound conditions by combining two or more relations. An example is: a > b && x == 10

Value of the expression

op1	op2	op1 && op2	op1 || op2
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Assignment Operators : Assignment operators are used to assign the value of an expression to a variable. We have seen the usual assignment operator, ' ='. In addition, Java has a set of 'shorthand' assignment operators which are used in the form

v op= exp;

where v is a variable, exp is an expression and op is a Java binary operator. The operator op = is known sn the shorthand assignment operator.

Increment and decrement Operators: Java has two very useful operators not generally found in many other languages. These are increment(++) and decrement(--) operators:

Operator ++ adds 1 to the operand while -- subtracts 1 from the operand. Both are unary operators.

Conditional Operator: The character pair "? :" is a ternary operator available in Java. This operator is used to construct conditional expressions of the form

expl ? exp2 : exp3 where expl, exp2, and exp3 are expressions.

In this expl is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of the conditional expression. If expl is false, exp3 is evaluated and its value becomes the value of the conditional expression.

a = 10; b = 15; x = (a > b) ? a : b;

Bitwise Operators : These operators are used for testing the bits, or shifting them to the right and left. Bitwise operators may not be applied to float or double.

Operator	Meaning
&	bitwise AND
!	bitwise OR
^	bitwise exclusive OR
~	one's complement
<<	shift left
>>	shift right
>>>	shift right with zero fill

Special Operators:

Java supports some special operators of interest such as **instanceof** operator Dot (.)

i. **instanceof Operator**: The instanceof is an object reference operator and returns true if the object on the left hand side is an instance of the class given on the right-hand side. This operator allows us to determine whether the object belongs to a particular class or not.

Example:

s1 instanceof Student; is true if the object person belongs to the class student; otherwise it is false.

ii. **Dot Operator:** The dot operator (.) is used to access the instance variables and methods of class objects.

Example:

s1. age --- Reference to the variable age

s1.salary( ) --- Reference to the method salary()

It is also used to access classes and sub-packages from a package.

1.15 Control statements in Java

Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. Java provides three types of control flow statements.

1. Decision-Making statements (Conditional statements):

Decision-making statements evaluate the condition and control the program flow depending upon the result of the condition provided.

i. **Simple if statement**: It evaluates a Boolean expression (condition), if the expression evaluates to true then program flow will enter into if block.

Syntax:

```
if(condition)
{
    statements;
}
```

Example program for simple if

ii. **if-else statement:** It evaluates a Boolean expression (condition), if the expression evaluates to true then Program flow will enter into if block otherwise flow will enter into else block.

Syntax:

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

```
else
{
      statements; //executes when condition is false
}
```

Example program for if-else

iii. if-else-if ladder: The if-else-if statement contains the if-statement followed by multiple else-if statements. We can also define an else statement at the end of the chain.

Syntax:

```
if(condition 1)
{
     statements; //executes when condition 1 is true
}
else if(condition 2)
{
    statements ; //executes when condition 2 is true
}
else
{
     statements; //executes when all the conditions are false
}
```

Example program for if else ladder.

iv. Nested if-else statement: In nested if- else statements, the if block or else block can contain a simple **if** or **if-else** statements.

Syntax:

```
if(condition 1)
{
   statements; //executes when condition 1 is true
   if(condition 2)
   {
        statements; //executes when condition 2 is true
   }
}
else
{
   statements; //executes when condition 1 is false
}
```

v. Switch Statement: The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.

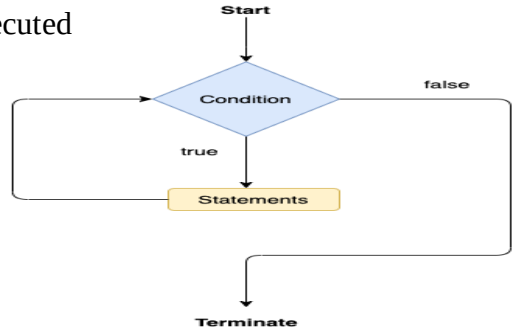
Syntax:

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

2. Loop Statements (Iterative statements):

Looping statements are used to execute the set of instructions repeatedly until given condition became false.

i. 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.

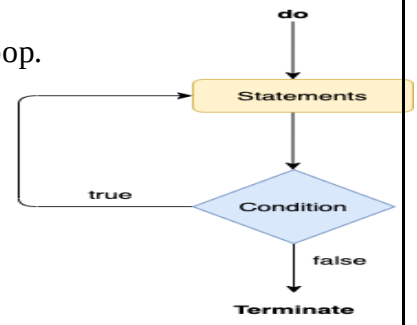


Syntax:

```
while(condition)
{
      //looping statements
}
```

Example program for while loop.

ii. do-while loop: The do-while loop checks the condition at the end of the loop. It is also known as the exit controlled loop since the condition is checked after the looping body executed .

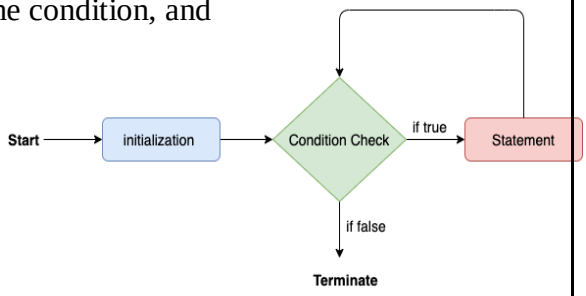


Syntax:

```
do
{
    Statements;
} while (condition);
```

Example program for do-while loop.

iii. for loop: It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.



Syntax:

```
for(initialization; condition; increment/decrement)
{
    block of statements
}
```

Example program for for-loop.

iv. for-each loop: Java provides an enhanced for loop to traverse the data structures like array. In the for-each loop, we don't need to update the loop variable.

Syntax:

```
for(data_type var : array_name)
{
        statements
}
```

Example program for for-each loop.

3. Jump Statements:

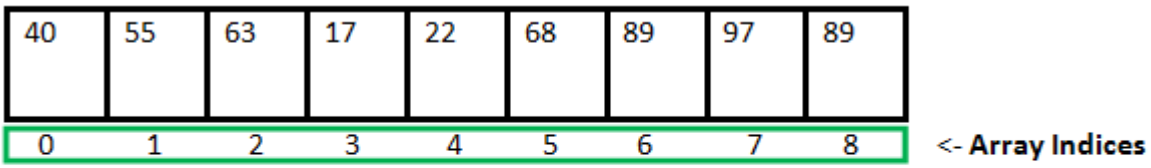
Jump statements are used to transfer the control of the program to the specific statements. There are two types of jump statements in Java, i.e., break and continue.

i. break: 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.

ii. continue: It skips the specific part of the loop and jumps to the next iteration of the loop immediately.

1.16 Arrays

- An Array is a group of **contiguous or related data items** that share a common name.
- In Java, all arrays are dynamically allocated.
- Arrays may be stored in contiguous memory.
- Since arrays are objects in Java, we can find their length using the object property "length".
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered, and each has an index beginning with 0.
- The **size** of an array must be specified by int or short value and not long.
- The direct superclass of an array type is Object.
- The size of the array cannot be altered(once initialized).
- Java supports **One Dimensional** Arrays and **Multi Dimensional** Arrays



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

<u>One Dimensional Array :</u> If an array variable have only one subscript then that array is called as One-Dimensional Array.

Declare the Array: The arrays in java may be declared in two forms

Datatype Arrayname [] or Datatype [] Arrayname

Ex : int number[] or int [] number

Creating memory for an Array: After declaring an array variable, we need to create memory. Java allows it by using "new" operator only.

Arrayname = new Datatype[size]

Ex : number = new int[5];

Initialization of array variable: we can initialize the array same as other variables

Type Arrayname []= { list of values }

Ex: int number [] = {34, 25, 67, 1 ,3};

<u>Two Dimensional Array:</u> If an array variable have two subscript then that array is called as Two Dimensional Array.

Datatype Arrayname [][] or Datatype[] [] Arrayname

Ex : int number[][] or int [][] number

Example program for Arrays.

1.17 String class

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.

<u>Creating a string object:</u> There are two ways to create String object:

1. By string literal
2. By new keyword

1. String Literal: Java String literal is created by using double quotes.

For Example: String s="welcome";

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. For example:

1. String s1="Welcome";
2. String s2="Welcome"; //It doesn't create a new instance

2. new keyword: In this case, JVM will create a new string object in normal (non-pool) heap memory, and the created literal will be placed in the string constant pool. The variable s will refer to the object in a heap.

```
String s=new String ("Welcome");        (OR)
char ch[]={'j', 'a', 'v', 'a'};
String s=new String(ch);
```

Example:

```
public class StringExample
{
    public static void main(String args[])
    {
        String s1="java";   //creating string by Java string literal
        char ch[]={'s','t','r','i','n','g','s'};
        String s2=new String(ch);    //converting char array to string
        String s3=new String("example");   //creating Java string by new keyword
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
 }
```

Output:

```
java
strings
example
```

String Arrays: We can also create and use arrays that contain strings. The statement

String s[ ] = new String[3];

will create s of size 3 to hold three string constants. We can assign the three strings to the s by using three different statements or using a for loop.

String Methods: **String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

Table of the most commonly used string methods, and their tasks.

Method Call	Task performed
s2= sl.toLowerCase;	Converts the string s1 to all lowercase
s2= sl.toUpperCase;	Converts the string s1 to all Uppercase
s2= sl.replace ('x', 'y')	Replace all appearances of x with y
s2 = sl.trim ( );	Remove white spaces at the beginning and end of the strings s1
s1.equals (s2)	Returns 'true' if sl equal to s2

s1.equalsIgnoreCase (s2)	Returns 'true' if sl =s2, ignoring the case of characters
s1.length ( )	Gives the length of sl
s1.chartAt (n)	Gives nth character of s1
s1.compareTo (s2)	Returns negative if s1 <s2, positive if sl>s2, and 0 if equal s2
s1.concat (s2)	Concatenates s1 and s2
s1.substring (n)	Gives substring starting from nth character
s1.substring (n, m)	Gives substring starting from nth character up to m^{th}
String.valueOf (p)	Creates a string object of the parameter p
p.toString ()	Creates a string representation of the object p
s1.indexOf ('x')	Gives the position of the first occurrence of 'x' in the strings s1
S1.indexOf ('x', n)	Gives the position of 'x' occurs after nth position in the string
String.valueOf (Variable)	Converts the parameter value to string representation

1.18 StringBuffer class

StringBuffer is a peer class of String. The **string** represents fixed-length, immutable character sequences while **StringBuffer** represents growable and writable character sequences means it is mutable in nature. In StringBuffer, we can insert characters and substrings in the middle of a siring, or append another string to the end.

```
StringBuffer s= new StringBuffer("Aditya");
```

Some methods of StringBuffer class are:

Method	Task
s1.setChartAt (n, 'x')	Modifies, the nth character to x
s1.append (s2)	Appends the strings s2 to sl at the end
s1.insert (n, s2)	Inserts the string2 at the position n of the string s1.
s1.setlength (n)	Sets the length of the string s1 to n.

1.19 Class, Object, Methods in Java

- class is a user-defined / Abstract data type.
- Classes provide a convenient method for packing related data items and functions that work together.
- In Java, the data items are called fields and the functions are called methods.
- By using 'class' we can create user defined classes.
- Once the class type has been defined, we can create "objects" / "variables" of that type.
- Each class declaration starts with the keyword **"class".**
- The basic form of a class definition is:

```
class  classname   // classname is any valid Java identifiers
{
    fields declaration;
    methods declaration;
}
```

Data Members / Variables / Fields Declaration: Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables or member variables

because they are created whenever an object of the class is instantiated. Declare the instance variables exactly the same way as declare local variables.

Example:
```
class Rectangle
{
    int length;
    int width;
}
```

The class Rectangle contains two integer type instance variables. Remember these variables are only declared and therefore no storage space has been created in the memory.

Methods Declaration: Methods are used to perform operations on data members. By using these methods only objects can handle the data from outside.

The general form of a method declaration is

```
type methodname (parameter-list)
{
    method-body;
}
```

Method declarations have four basic parts:

- The name or the method (method name)
- The type of the value the method returns (type)
- A List of parameters {parameter-list)
- The body of the method

Example:
```
class Rectangle
{
    int  length , width;
    void  getdata (int x, int y)  //  Method  declaration
    {
        length  =  x:
        width   =  y
    }
}
```

Creating Objects: Creating an object is also referred as instantiating an object. Objects in Java are created using the new operator. An object in Java is essentially a block of memory that contains space to store all the instance variables.

Here is an example of creating an object of type Rectangle.

```
Rectangle   r1;                  // declare the object no memory
r1    = new  Rectangle ( );      //  instantiate the object
```

the first statement declares a variable to hold the objects reference and the second one actually assigns the object reference to the variable. The variable r1 is now an object of the Rectangle class.

Both statements can be combined into one as shown below:

```
Rectangle rl = new Rectangle(  );
```

We can create any number of objects of Rectangle.

Accessing Class Members: By using the concerned object and the dot operator we can access the class members.

objectname. variablename = value;
objectname. methodname (perameter-list);

Example:

rl.length= 15:
rl.width =10;

Example program for class and object creation.

1.20 Method overloading

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- If we have to perform only one operation, having same name of the methods increases the readability of the program. Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as add2(int, int) for two parameters, and add3(int, int, int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading.
- There are two ways to overload the method in java
    1. By changing number of arguments
    2. By changing the data type

In Java, Method Overloading is not possible by changing the return type of the method only.

1. Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add() method performs addition of three numbers.

```
class Adder
{
    int add (int a, int b)
    {
        return a+b;
    }

    int add (int a, int b, int c)
    {
        return a+b+c;
    }
 }
class TestOverloading1
{
      public static void main(String[] args)
     {
           Adder obj = new Adder();
           int  r1= obj.add(10, 20);
           int  r2= obj.add(10, 20, 30);
           System.out.println("Sum=" +r1);
           System.out.println("Sum=" +r2);
      }
 }
```

Output:

Sum= 30
Sum= 60

2. Method Overloading: changing data type of arguments:

In this example, we have created two methods that differ in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder
{
    int add (int a, int b)
    {
        return a+b;
    }
    double add (double  a, double  b)
    {
       return a+b;
    }
}
class TestOverloading2
{
    public static void main(String[] args)
    {
        Adder obj = new Adder();
        int  r1= obj.add(10, 20);
        double  r2= obj.add(10.2,  20.3);
        System.out.println("Sum=" +r1);
        System.out.println("Sum=" +r2);
    }
}
```

Output:

Sum= 30

Sum= 30.5

1.21 Constructors

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created.

How Java Constructors are Different from Java Methods?

- Constructors must have the same name as the class within which it is defined but it is not necessary for the method in Java.
- Constructors do not return any type but methods have the return type or **void**(does not return any value).
- Constructors are called only once at the time of Object creation but methods can be called any number of times.

Types of Constructors in Java: In java there are mainly 3 types of constructers. That are

1. Default Constructor
2. Parameterized Constructor

1. Default Constructor: A constructor that has no parameters is known as default the constructor.

```
class Room
{
    double length, height, width;
    Room( )
    {
```

```
            length=12.3;
            height=34.2;
            width=16.9;
         }
        double volume()
        {
            return length * height * width;
         }
       }
  class TestRoom
  {
       public static void main(String args[])
       {
          double vol;
          Room r1=new Room( );
          vol=r1.volume();
          System.out.println("Volume of Room is:"+vol);
       }
  }
```

2. Parameterized Constructor: A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

```
   class Room
   {
     double length, height, width;
     Room(double l, double h, double w)
     {
        length=l;
        height=h;
      width=w;
     }
     double volume()
     {
              return length * height * width;
      }
   }
 class TestRoom
{
    public static void main(String args[])
    {
       double vol;
       Room r1=new Room(21.3, 34.1, 54.7);
       vol=r1.volume();
       System.out.println("Volume of Room is:"+vol);
    }
 }
```

1.22 this keyword

- this keyword refers to the current object in a method or constructor.
- The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name.
- this can also be used to:
    - Invoke current class constructor
    - Invoke current class method
    - Return the current class object
    - Pass an argument in the method call
    - Pass an argument in the constructor call
- If you omit the keyword in the below example, the output would be "0" instead of "111".

Example program:

```
class Student
{
    int rollno;
    Student(int rollno)
    {
        this.rollno=rollno;
    }
    void display()
    {
        System.out.println(rollno);
    }
}
class TestThis
{
   public static void main(String args[])
   {
        Student s1=new Student(111);
        s1.display();
   }
}
```

1.23 Garbage Collection

In C/C++, a programmer is responsible for both the creation and destruction of objects. Usually, programmer neglects the destruction of useless objects. Due to this negligence, at a certain point, sufficient memory may not be available to create new objects, and the entire program will terminate abnormally, causing OutOfMemoryErrors.

But in Java, the programmer need not care for all those objects which are no longer in use. Garbage collector destroys these objects. The main objective of Garbage Collector is to free heap memory by destroying unreachable objects. The garbage collector is the best example of the Daemon thread as it is always running in the background.

Java garbage collection is an automatic process. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.

1.24 finalize() method

The Java **finalize() method** of Object class is a method that the Garbage Collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection to perform clean-up activity. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection.Once the finalize() method completes immediately, Garbage Collector destroys that object.

Finalization: Just before destroying any object, the garbage collector always calls finalize() method to perform clean-up activities on that object. This process is known as Finalization in Java.

Note: The Garbage collector calls the finalize() method only once on any object.

Syntax:

protected void finalize throws Throwable{}

1.25 static keyword

- The static keyword in Java is used for memory management mainly.
- We can apply static keyword with variables, methods, blocks and nested classes.
- The static keyword belongs to the class than an instance of the class.

1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

```
class Student
{
    int rollno; //instance variable
    String name;
    static String college =”Aditya”; //static variable
    Student(int r, String n)
    {
       rollno = r;
       name = n;
    }
   void display ()
   {
        System.out.println(rollno+" "+name+" "+college);
   }
}
public class TestStatic
{
  public static void main(String args[])
  {
    Student s1 = new Student(111, ”Rani”);
    Student s2 = new Student(222, “Vani”);
    s1.display();
    s2.display();
  }
}
```

2) Java static method:

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

3) Java static block:

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

```
class A2
{
  static{ System.out.println("static block is invoked"); }
  public static void main(String args[])
  {
    System.out.println("Hello main");
  }
}
```

1.26 Nnested and Inner classes:

In Java, it is possible to define a class within another class, such classes are known as nested classes.

- The scope of a nested class is bounded by the scope of its enclosing class. Thus in the below example, the class NestedClass does not exist independently of the class OuterClass.
- A nested class has access to the members, including private members, of the class in which it is nested. But the enclosing class does not have access to the member of the nested class.
- A nested class is also a member of its enclosing class.
- Nested classes are divided into two categories:
    1. **static nested class:** Nested classes that are declared *static* are called static nested classes.
    2. **inner class:** An inner class is a non-static nested class.

Syntax:

```
class OuterClass
{
   ...
   class NestedClass
   {
      ...
   }
}
```

Static nested classes:

In the case of normal or regular inner classes, without an outer class object existing, there cannot be an inner class object. But in the case of static nested class, Without an outer class object existing, there may be a static nested class object. As with class methods and variables, a static nested class is associated with its outer class.

Syntax:

OuterClass.StaticNestedClass

For example, to create an object for the static nested class, use this syntax:

OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();

Inner classes:

To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

OuterClass.InnerClass innerObject = outerObject.new InnerClass();

1.27 Command line Arguments:

- Java command-line argument is an argument i.e. passed at the time of running the Java program.
- In Java, the command line arguments passed from the console can be received in the Java program and they can be used as input.
- We need to pass the arguments as space-separated values.
- We can pass both strings and primitive data types(int, double, float, char, etc) as command-line arguments.
- These arguments convert into a string array and are provided to the main() function as a string array argument.
- When command-line arguments are supplied to JVM, JVM wraps these and supplies them to args[].
- By using args.length method, we can find the length of args.
- JVM stores the first command-line argument at args[0], the second at args[1], the third at args[2], and so on.

```
class A
{
   public static void main(String args[])
   {
        for(int i=0;i<args.length;i++)
            System.out.println(args[i]);
    }
}
```

compile by > javac A.java

run by > java A 10 30 Aditya

Output:

10

30

Aditya