OBJECT ORIENTED PROGRAMMING THROUGH JAVA

UNIT-I

Object oriented thinking and Java Basics- Need for oop paradigm, summary of oop concepts, coping with complexity, abstraction mechanisms. A way of viewing world – Agents, responsibility, messages, methods, History of Java, Java buzzwords, data types, variables, scope and lifetime of variables, arrays, operators, expressions, control statements, type conversion and casting, simple java program, concepts of classes, objects, constructors, methods, access control, this keyword, garbage collection, overloading methods and constructors, method binding, inheritance, overriding and exceptions, parameter passing, recursion, nested and inner classes, exploring string class.

OBJECT ORIENTED THINKING

- When computers were first invented, programming was done manually by toggling in a binary machine instructions by use of front panel.
- As programs began to grow, high level languages were introduced that gives the programmer more tools to handle complexity.
- The first widespread high level language is FORTRAN. Which gave birth to structured programming in 1960's. The Main problem with the high level language was they have no specific structure and programs becomes larger, the problem of complexity also increases.
- So C became the popular structured oriented language to solve all the above problems.
- However in SOP, when project reaches certain size its complexity exceeds. So in 1980's a new way of programming was invented and it was OOP. OOP is a programming methodology that helps to organize complex programs through the use of inheritance, encapsulation & polymorphism.

NEED FOR OOP PARADIGM

- Traditionally, the structured programming techniques were used earlier.
- There were many problems because of the use of structured programming technique.
- The structured programming made use of a top-down approach.
- To overcome these problems the object oriented programming concept was created.
- The object oriented programming makes use of bottom-up approach.
- It also manages the increasing complexity.
- The description of an object-oriented program can be given as, a data that controls access to code.
- The object-oriented programming technique builds a program using the objects along with a set of well-defined interfaces to that object.

- The object-oriented programming technique is a paradigm, as it describes the way in which elements within a computer program must be organized.
- It also describes how those elements should interact with each other.
- In OOP, data and the functionality are combined into a single entity called an object.
- Classes as well as objects carry specific functionality in order to perform operations and to achieve the desired result.
- The data and procedures are loosely coupled in procedural paradigm.
- Whereas in OOP paradigm, the data and methods are tightly coupled to form objects.
- These objects helps to build structure models of the problem domain and enables to get effective solutions.
- OOP uses various principles (or) concepts such as abstraction, inheritance, encapsulation and polymorphism. With the help of abstraction, the implementation is hidden and the functionality is exposed.
- Use of inheritance can eliminate redundant code in a program. Encapsulation enables the data and methods to wrap into a single entity. Polymorphism enables the reuse of both code and design.

SUMMARY OF OOP CONCEPTS

- Everything is an object.
- Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending & receiving *messages*. A message is a request for an action bundled with whatever arguments may be necessary to complete the task.
- Each object has its own *memory*, which consists of other objects.
- Every Object is an *instance* of class. A class simply represents a grouping of similar objects, such as integers or lists.
- The class is the repository for *behavior* associated with an object. That is all objects that are instances of same class can perform the same actions.
- Classes are organized into a singly rooted tree structure, called *inheritance hierarchy.*

OOP CONCEPTS

OOP stands for Object-Oriented Programming. OOP is a programming paradigm in which every program is follows the concept of object. In other words, OOP is a way of writing programs based on the object concept.

The object-oriented programming paradigm has the following core concepts.

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

Class

Class is a blue print which is containing only list of variables and methods and no memory is allocated for them. A class is a group of objects that has common properties.

Object

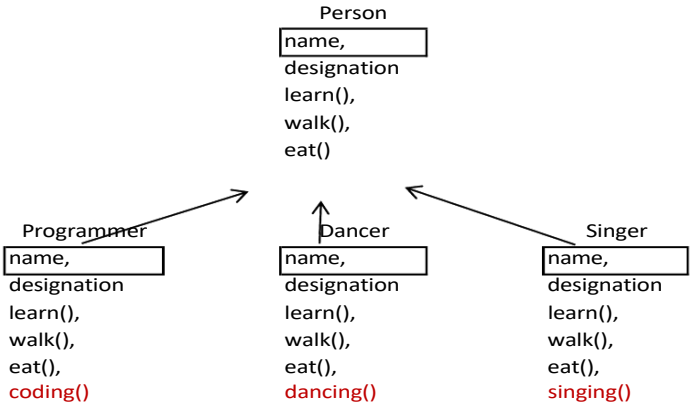
- Any entity that has state and behavior is known as an object.
- For example a chair, pen, table, keyboard, bike, etc. It can be physical or logical. An Object can be defined as an instance of a class.
- **Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Encapsulation

- Encapsulation is the process of combining data and code into a single unit.
- In OOP, every object is associated with its data and code.
- In programming, data is defined as variables and code is defined as methods.
- The java programming language uses the class concept to implement encapsulation.

Inheritance

- Inheritance is the process of acquiring properties and behaviors from one object to another object or one class to another class.
- In inheritance, we derive a new class from the existing class. Here, the new class acquires the properties and behaviors from the existing class.
- In the inheritance concept, the class which provides properties is called as parent class and the class which recieves the properties is called as child class.



Polymorphism

- Polymorphism is the process of defining same method with different implementation. That means creating multiple methods with different behaviors.
- The java uses method overloading and method overriding to implement polymorphism.
- Method overloading - multiple methods with same name but different parameters.
- Method overriding - multiple methods with same name and same parameters.

Abstraction

- Abstraction is hiding the internal details and showing only essential functionality.
- In the abstraction concept, we do not show the actual implementation to the end user, instead we provide only essential things.
- For example, if we want to drive a car, we does not need to know about the internal functionality like how wheel system works? how brake system works? how music system works? etc.

COPING WITH COMPLEXITY

- Coping with complexity in Java, or any programming language, is an essential skill for software developers.
- As your Java projects grow in size and complexity, maintaining, debugging, and extending your code can become challenging.

Here are some strategies to help you cope with complexity in Java:

1. **Modularization**: Breaking down the code into smaller, self-contained modules, classes, or packages to manage complexity. Each module should have a specific responsibility and interact with others through well-defined interfaces.
2. **Design Patterns: Using** established design patterns to solve common architectural and design problems. Design patterns provide proven solutions to recurring challenges in software development.
3. **Encapsulation**: Restricting access to class members using access modifiers (public, private, protected) to hide implementation details and provide a clear API for interacting with the class.
4. **Abstraction**: Creating abstract classes and interfaces to define contracts that classes must adhere to, making it easier to work with different implementations.
5. **Documentation**: Writing comprehensive documentation, including code comments and Javadoc, to explain how the code works, its purpose, and how to use it.
6. **Testing**: Implementing unit tests to ensure that individual components of the code function correctly, which helps identify and prevent bugs.
7. **Code Reviews**: Collaborating with team members to review and provide feedback on code to catch issues and ensure code quality.
8. **Version Control**: Using version control systems like Git to manage changes, track history, and collaborate with others effectively.
9. **Refactoring**: Regularly improving and simplifying the codebase by removing redundancy and improving its structure.

ABSTRACTION MECHANISM

- In Java, abstraction is a fundamental concept in object-oriented programming that allows you to hide complex implementation details while exposing a simplified and well-defined interface.
- Abstraction mechanisms in Java include the use of abstract classes and interfaces.

A WAY OF VIEWING WORLD

- A way of viewing the world is an idea to illustrate the object-oriented programming concept with an example of a real-world situation.
- Let us consider a situation, I am at my office and I wish to get food to my family members who are at my home from a hotel. Because of the distance from my office to home, there is no possibility of getting food from a hotel myself. So, how do we solve the issue?
- To solve the problem, let me call zomato (an agent in food delevery community), tell them the variety and quantity of food and the hotel name from which I wish to delever the food to my family members.

AGENTS AND COMMUNITIES

Let us consider a situation, I am at my office and I wish to get food to my family members who are at my home from a hotel. Because of the distance from my office to home, there is no possibility of getting food from a hotel myself. So, how do we solve the issue?

To solve the problem, let me call zomato (an **agent** in food delevery community), tell them the variety and quantity of food and the hotel name from which I wish to delever the food to my family members. **An object-oriented program is structured as a community of interacting agents, called objects. Where each object provides a service (data and methods) that is used by other members of the community.**

In our example, the online food delivery system is a community in which the agents are zomato and set of hotels. Each hotel provides a variety of services that can be used by other members like zomato, myself, and my family in the community.

RESPONSIBILITIES

In object-oriented programming, behaviors of an object described in terms of responsibilities.

In our example, my request for action indicates only the desired outcome (food delivered to my family). The agent (zomato) free to use any technique that solves my problem. By discussing a problem in terms of responsibilities increases the level of abstraction. This enables more independence between the objects in solving complex problems.

MESSAGES & METHODS

To solve my problem, I started with a request to the agent zomato, which led to still more requestes among the members of the community until my request has done. Here, the members of a community interact with one another by making requests until the problem has satisfied.

In object-oriented programming, every action is initiated by passing a message to an agent (object), which is responsible for the action. The receiver is the object to whom the message was sent. In response to the message, the receiver performs some method to carry out the request. Every message may include any additional information as arguments.

In our example, I send a request to zomato with a message that contains food items, the quantity of food, and the hotel details. The receiver uses a method to food get delivered to my home.

HISTORY OF JAVA

- Java is a object oriented programming language.
- Java was created based on C and C++.
- Java uses C syntax and many of the object-oriented features are taken from C++.
- Before Java was invented there were other languages like COBOL, FORTRAN, C, C++, Small Talk, etc.
- These languages had few disadvantages which were corrected in Java.
- Java also innovated many new features to solve the fundamental problems which the previous languages could not solve.
- Java was developed by James Gosling, Patrick Naughton, Chris warth, Ed Frank and Mike Sheridon at Sun Microsystems in the year 1991.
- This language was initially called as “OAK” but was renamed as “Java” in 1995.
- The primary motivation behind developing java was the need for creating a platform independent Language (Architecture Neutral), that can be used to create a software which can be embedded in various electronic devices such as remote controls, micro ovens etc.
- The problem with C, C++ and most other languages is that, they are designed to compile on specific targeted CPU (i.e. they are platform dependent), but java is platform Independent which can run on a variety of CPU’s under different environments.
- The secondary factor that motivated the development of java is to develop the applications that can run on Internet. Using java we can develop the applications which can run on internet i.e. Applet. So java is a platform Independent Language used for developing programs which are platform Independent and can run on internet.

JAVA BUZZWORDS(JAVA FEATURES)

- Java is the most popular object-oriented programming language.
- Java has many advanced features, a list of key features is known as Java Buzz Words.

The Following list of Buzz Words

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Architecture-neutral (or) Platform Independent
- Multi-threaded
- Interpreted
- High performance
- Distributed
- Dynamic

Simple

- Java programming language is very simple and easy to learn, understand, and code.
- Most of the syntaxes in java follow basic programming language C and object-oriented programming concepts are similar to C++.
- In a java programming language, many complicated features like pointers, operator overloading, structures, unions, etc. have been removed.
- One of the most useful features is the garbage collector it makes java more simple.

Secure

- Java is said to be more secure programming language because it does not have pointers concept.
- java provides a feature "applet" which can be embedded into a web application.
- The applet in java does not allow access to other parts of the computer, which keeps away from harmful programs like viruses and unauthorized access.

Portable

- Portability is one of the core features of java .
- If a program yields the same result on every machine, then that program is called portable.
- Java programs are portable
- This is the result of java System independence nature.

Object-oriented

- Java is an object oriented programming language.
- This means java programs use objects and classes.

Robust

- Robust means strong.
- Java programs are strong and they don't crash easily like a C or C++ programs

There are two reasons

- Java has got excellent inbuilt exception handling features. An exception is an error that occurs at runtime. If an exception occurs, the program terminates suddenly giving rise to problems like loss of data. Overcoming such problem is called exception handling.
- Most of the C and C++ programs crash in the middle because of not allocating sufficient memory or forgetting the memory to be freed in a program. Such problems will not occur in java because the user need not allocate or deallocate the memory in java. Everything will be taken care of by JVM only.

Architecture-neutral (or) Platform Independent

- Java has invented to archive "write once; run anywhere, anytime, forever".
- The java provides JVM (Java Virtual Machine) to archive architectural-neutral or platform-independent.
- The JVM allows the java program created using one operating system can be executed on any other operating system.

Multi-threaded

- Java supports multi-threading programming.
- Which allows us to write programs that do multiple operations simultaneously.

Interpreted

- Java programs are compiled to generate byte code.
- This byte code can be downloaded and interpreted by the interpreter in JVM.
- If we take any other language, only an interpreter or a compiler is used to execute the program.
- But in java, we use both compiler and interpreter for the execution.

High performance

- The problem with interpreter inside the JVM is that it is slow.
- Because of Java programs used to run slow.
- To overcome this problem along with the interpreter.
- Java soft people have introduced JIT (Just in Time) compiler, to enhance the speed of execution.
- So now in JVM, both interpreter and JIT compiler work together to run the program.

Distributed

- Information is distributed on various computers on a network.
- Using Java, we can write programs, which capture information and distribute it to the client.
- This is possible because Java can handle the protocols like TCP/IP and UDP.

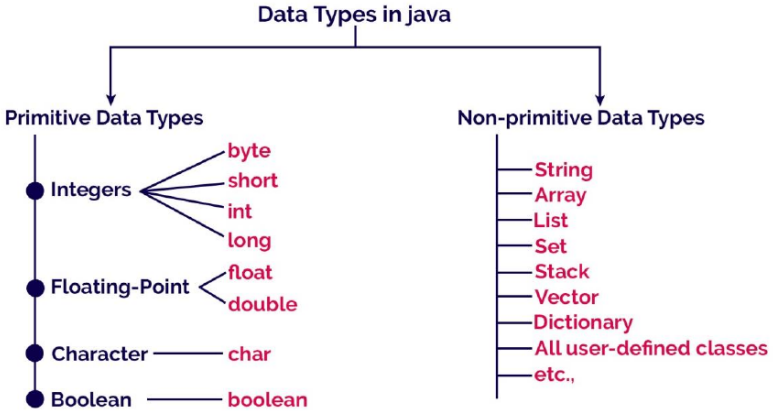
Dynamic

Java is said to be dynamic because the java byte code may be dynamically updated on a running system and it has a dynamic memory allocation and deallocation (objects and garbage collector).

DATA TYPES IN JAVA

Java programming language has a rich set of data types. The data type is a category of data stored in variables. In java, data types are classified into two types and they are as follows.

- Primitive Data Types
- Non-primitive Data Types



Primitive Data Types

The primitive data types are built-in data types and they specify the type of value stored in a variable and the memory size.

Integer Data Types

Integer Data Types represent integer numbers, i.e numbers without any fractional parts or decimal points.

Data Type	Memory Size	Minimum and Maximum values	Default Value
byte	1 byte	-128 to +128	0
short	2 bytes	-32768 to +32767	0
int	4 bytes	-2147483648 to +2147483647	0
long	8 bytes	-9223372036854775808 to +9223372036854775807	0L

Float Data Types

Float data types are represent numbers with decimal point.

Data Type	Memory Size	Minimum and Maximum values	Default Value
float	4 byte	-3.4e38 to -1.4e-45 and 1.4e-45 to 3.4e38	0.0f
double	8 bytes	-1.8e308 to -4.9e-324 and 4.9e-324 to 1.8e308	0.0d

Note: Float data type can represent up to 7 digits accurately after decimal point.

Double data type can represent up to 15 digits accurately after decimal point.

Character Data Type

Character data type are represents a single character like a, P, &, *,..etc.

Data Type	Memory Size	Minimum and Maximum values	Default Value
char	2 bytes	0 to 65538	\u0000

Boolean Data Types

Boolean data types represent any of the two values, true or false. JVM uses 1 bit to represent a Boolean value internally.

Data Type	Memory Size	Minimum and Maximum values	Default Value
boolean	1 byte	0 or 1	0 (false)

VARIABLES

Variable is a name given to a memory location where we can store different values of the same data type during the program execution.

The following are the rules to specify a variable name...

- A variable name may contain letters, digits and underscore symbol
- Variable name should not start with digit.
- Keywords should not be used as variable names.
- Variable name should not contain any special symbols except underscore(_).
- Variable name can be of any length but compiler considers only the first 31 characters of the variable name.

Declaration of Variable

Declaration of a variable tells to the compiler to allocate required amount of memory with specified variable name and allows only specified datatype values into that memory location.

Syntax: datatype variablename;

Example : int a;

Syntax : data_type variable_name_1, variable_name_2,...;

Example : int a, b;

Initialization of a variable:

Syntax: datatype variablename = value;

Example : int a = 10;

Syntax : data_type variable_name_1=value, variable_name_2 = value;

Example : int a = 10, b = 20;

SCOPE AND LIFETIME OF A VARIABLE

- In programming, a variable can be declared and defined inside a class, method, or block.
- It defines the scope of the variable i.e. the visibility or accessibility of a variable.
- Variable declared inside a block or method are not visible to outside.
- If we try to do so, we will get a compilation error. Note that the scope of a variable can be nested.
- **Lifetime** of a variable indicates how long the variable stays alive in the memory.

TYPES OF VARIABLES IT'S SCOPE

There are three types of variables in Java:

1. local variable
2. instance variable
3. static variable

Variable Type	Scope	Lifetime
Instance variable	Troughout the class except in static methods	Until the object is available in the memory
Class variable	Troughout the class	Until the end of the program
Local variable	Within the block in which it is declared	Until the control leaves the block in which it is declared

Local Variables

- Variables declared inside the methods or constructors or blocks are called as local variables.
- The scope of local variables is within that particular method or constructor or block in which they have been declared.
- Local variables are allocated memory when the method or constructor or block in which they are declared is invoked and memory is released after that particular method or constructor or block is executed.
- Access modifiers cannot be assigned to local variables.
- It can't be defined by a static keyword.
- Local variables can be accessed directly with their name.

Program

```
class LocalVariables
{
	public void show()
	{
		int a = 10;
		System.out.println("Inside show method, a = " + a);
	}
	public void display()
	{
		int b = 20;
		System.out.println("Inside display method, b = " + b);
		//System.out.println("Inside display method, a = " + a); // error
	}
	public static void main(String args[])
	{
		LocalVariables obj = new LocalVariables();
		obj.show();
		obj.display();
	}
}
```

Instance Variables:

- Variables declared outside the methods or constructors or blocks but inside the class are called as instance variables.
- The scope of instance variables is inside the class and therefore all methods, constructors and blocks can access them.
- Instance variables are allocated memory during object creation and memory is released during object destruction. If no object is created, then no memory is allocated.
- For each object, a separate copy of instance variable is created.
- Heap memory is allocated for storing instance variables.
- Access modifiers can be assigned to instance variables.
- It is the responsibility of the JVM to assign default value to the instance variables as per the type of
- Variable.
- Instance variables can be called directly inside the instance area.
- Instance variables cannot be called directly inside the static area and necessarily requires an object reference for calling them.

Program

```
class InstanceVariable
{
    int x = 100;
    public void show()
    {
        System.out.println("Inside show method, x = " + x);
        x = x + 100;
    }
    public void display()
    {
        System.out.println("Inside display method, x = " + x);
    }
    public static void main(String args[])
    {
        ClassVariables obj = new ClassVariables();
        obj.show();
        obj.display();
    }
}
```

Static variables

- Static variables are also known as class variable.
- Static variables are declared with the keyword 'static ' .
- A static variable is a variable whose single copy in memory is shared by all the objects, any modification to it will also effect other objects.
- Static keyword in java is used for memory management, i.e it saves memory.
- Static variables gets memory only once in the class area at the time of class loading.
- Static variables can be invoked without the need for creating an instance of a class.
- Static variables contain values by default. For integers, the default value is 0. For Booleans, it is false. And for object references, it is null.

Syntax: static datatype variable name;

Example: static int x=100;

Syntax: classname.variablename;

Example

```
class Employee
{
   static int empid=500;
   static void emp1()
        {
                empid++;
                System.out.println("Employee  id:"+empid);
        }
}
class Sample
{
        public static void main(String args[])
        {
                Employee.emp1();
                Employee.emp1();
                Employee.emp1();
                Employee.emp1();
                Employee.emp1();
                Employee.emp1();

        }
}
```

ARRAYS

- An array is a collection of similar data values with a single name.
- An array can also be defined as, a special type of variable that holds multiple values of the same data type at a time.
- In java, arrays are objects and they are created dynamically using new operator.
- Every array in java is organized using index values.
- The index value of an array starts with '0' and ends with 'zise-1'.
- We use the index value to access individual elements of an array.

In java, there are two types of arrays and they are as follows.

- One Dimensional Array
- Multi Dimensional Array

One Dimensional Array

In the java programming language, an array must be created using new operator and with a specific size. The size must be an integer value but not a byte, short, or long. We use the following syntax to create an array.

Syntax

```
data_type array_name[ ] = new data_type[size];
        (or)
data_type[ ] array_name = new data_type[size];
```

Example

```
class Onedarray
{
    public static void main(String args[])
    {
        int a[]=new int[5];
        a[0]=10;
        a[1]=20;
        a[2]=70;
        a[3]=40;
        a[4]=50;
        for(int i=0;i<5;i++)
        System.out.println(a[i]);
    }
}
```

- In java, an array can also be initialized at the time of its declaration.
- When an array is initialized at the time of its declaration, it need not specify the size of the array and use of the new operator.
- Here, the size is automatically decided based on the number of values that are initialized.

Example

```
int list[ ] = {10, 20, 30, 40, 50};
```

Multidimensional Array

- In java, we can create an array with multiple dimensions. We can create 2-dimensional, 3-dimensional, or any dimensional array.
- In Java, multidimensional arrays are arrays of arrays.
- To create a multidimensional array variable, specify each additional index using another set of square brackets.

Syntax

```
data_type array_name[ ][ ] = new data_type[rows][columns];
                (or)
data_type[ ][ ] array_name = new data_type[rows][columns];
```

- When an array is initialized at the time of declaration, it need not specify the size of the array and use of the new operator.
- Here, the size is automatically decided based on the number of values that are initialized.

Example

```
class Twodarray
{
    public static void main(String args[])
    {
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                System.out.print(arr[i][j]+"  ");
            }
             System.out.println();
        }
    }
}
```

OPERATORS

An operator is a symbol that performs an operation. An operator acts on some variables called operands to get the desired result.

Example: a + b

Here a, b are operands and + is operator.

Types of Operators

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment or Decrement operators
6. Conditional operator
7. Bit wise operators

1. Arithmetic Operators: Arithmetic Operators are used for mathematical calculations.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modular

Program: Java Program to implement Arithmetic Operators

```
class ArithmeticOperators
{
    public static void main(String[] args)
        {
            int a = 12, b = 5;
            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));
            System.out.println("a % b = " + (a % b));
        }
}
```

2. Relational Operators: Relational operators are used to compare two values and return a true or false result based upon that comparison. Relational operators are of 6 types

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

Program: Java Program to implement Relational Operators

```
class  RelationalOperator
{
        public static void main(String[] args)
        {
                int a = 10;
                int b = 3;
                int c = 5;
                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));
                System.out.println("a == c: " + (a == c));
                System.out.println("a != c: " + (a != c));
        }
}
```

3. Logical Operator: The Logical operators are used to combine two or more conditions .Logical operators are of three types

1. Logical AND (&&),
2. Logical OR (||)
3. Logician NOT (!)

1. Logical AND (&&) : Logical AND is denoted by double ampersand characters (&&).it is used to check the combinations of more than one conditions. if any one condition false the complete condition becomes false.

Truth table of Logical AND

Condition1	Condition2	Condition1 && Condition2
True	True	True
True	False	False
False	True	False
False	False	False

2. Logical OR (||) : Logical OR is denoted by double pipe characters (||). it is used to check the combinations of more than one conditions. if any one condition true the complete condition becomes true.

Truth table of Logical OR

Condition1	Condition2	Condition1 && Condition2
True	True	True
True	False	True
False	True	True
False	False	False

3. Logician NOT (!): Logical NOT is denoted by exclamatory characters (!), it is used to check the opposite result of any given test condition. i.e, it makes a true condition false and false condition true.

Truth table of Logical NOT

Condition1	!Condition2
True	False
False	True

Example of Logical Operators

```
class  LogicalOp
{
        public static void main(String[] args)
        {
                int x=10;
                System.out.println(x==10 && x>=5));
                System.out.println(x==10 || x>=5));
                System.out.println ( ! ( x==10 ));
        }
}
```

4. Assignment Operator: Assignment operators are used to assign a value (or) an expression (or) a value of a variable to another variable.

Syntax : variable name=expression (or) value

Example : x=10;

y=20;

The following list of Assignment operators are.

Operator	Description	Example	Meaning
+=	Addition Assignment	x + = y	x= x + y
-=	Addition Assignment	x - = y	x= x - y
*=	Addition Assignment	x * = y	x= x * y
/=	Addition Assignment	x / = y	x= x / y
%=	Addition Assignment	x % = y	x= x % y

Example of Assignment Operators

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

5: Increment And Decrement Operators : The increment and decrement operators are very useful. ++ and == are called increment and decrement operators used to add or subtract. Both are unary operators.

The syntax of the operators is given below.

These operators in two forms : prefix (++x) and postfix(x++).

++<variable name> --<variable name>

<variable name>++ <variable name>--

Operator	Meaning
++x	Pre Increment
--x	Pre Decrement
x++	Post Increment
x--	Post Decrement

Where

1 : ++x : Pre increment, first increment and then do the operation.

2 : - -x : Pre decrement, first decrements and then do the operation.

3 : x++ : Post increment, first do the operation and then increment.

4 : x- - : Post decrement, first do the operation and then decrement.

Example

```
class Increment
{
    public static void main(String[] args)
    {
        int var=5;
        System.out.println (var++);
        System.out.println (++var);
        System.out.println (var--);
        System.out.println (--var);
    }
}
```

6 : Conditional Operator: A conditional operator checks the condition and executes the statement depending on the condition. Conditional operator consists of two symbols.

1 : question mark (?).

2 : colon (:).

Syntax: condition ? exp1 : exp2;

It first evaluate the condition, if it is true (non-zero) then the "exp1" is evaluated, if the condition is false (zero) then the "exp2" is evaluated.

Example :

```
class  ConditionalOperator
{
        public static void main(String[] args)
        {
            int februaryDays = 29;
            String result;
            result = (februaryDays == 28) ? "Not a leap year" : "Leap year";
            System.out.println(result);
        }
}
```

7. Bitwise Operators:

- Bitwise operators are used for manipulating a data at the bit level, also called as bit level programming. Bit-level programming mainly consists of 0 and 1.
- They are used in numerical Computations to make the calculation process faster.
- The bitwise logical operators work on the data bit by bit.
- Starting from the least significant bit, i.e. LSB bit which is the rightmost bit, working towards the MSB (Most Significant Bit) which is the leftmost bit.

A list of Bitwise operators as follows...

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise Complement
<<	Left Shift
>>	Right Shift

1. Bitwise AND (&):

- Bitwise AND operator is represented by a single ampersand sign (&).
- Two integer expressions are written on each side of the (&) operator.
- if any one condition false (0) the complete condition becomes false (0).

Truth table of Bitwise AND

Condition1	Condition2	Condition1 & Condition2
0	0	0
0	1	0
1	0	0
1	1	1

Example : int x = 10;

int y = 20;

x & y = ?

x = 0000 1010

y = 0000 1011

x & y = 0000 1010 = 10

2. Bitwise OR:

- Bitwise OR operator is represented by a single vertical bar sign (|).
- Two integer expressions are written on each side of the (|) operator.
- if any one condition true (1) the complete condition becomes true (1).

Truth table of Bitwise OR

Condition1	Condition2	Condition1 | Condition2
0	0	0
0	1	1
1	0	1
1	1	1

Example : int x = 10;

int y = 20;

x | y = ?

x = 0000 1010

y = 0000 1011

x | y = 0000 1011 = 11

3. Bitwise Exclusive OR :

- The XOR operator is denoted by a carrot (^) symbol.
- It takes two values and returns true if they are different; otherwise returns false.
- In binary, the true is represented by 1 and false is represented by 0.

Truth table of Bitwise XOR

Condition1	Condition2	Condition1 ^ Condition2
0	0	0
0	1	1
1	0	1
1	1	0

Example : int x = 10;

int y = 20;

x ^ y = ?

x = 0000 1010

y = 0000 1011

x ^ y = 0000 0001 = 1

4. Bitwise Complement (~):

- The bitwise complement operator is a unary operator.
- It is denoted by ~, which is pronounced as tilde.
- It changes binary digits 1 to 0 and 0 to 1.
- bitwise complement of any integer N is equal to - (N + 1).
- Consider an integer 35. As per the rule, the bitwise complement of 35 should be -(35 + 1) = -36.

Example : int x = 10; find the ~x value.

x = 0000 1010

~x= 1111 0101

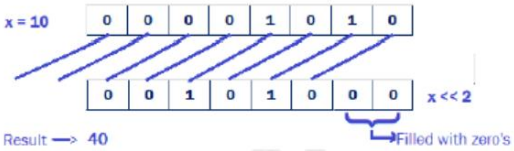
5. Bitwise Left Shift Operator (<<) :

- This Bitwise Left shift operator (<<) is a binary operator.
- It shifts the bits of a number towards left a specified no.of times.

Example:

int x = 10;

x << 2 = ?



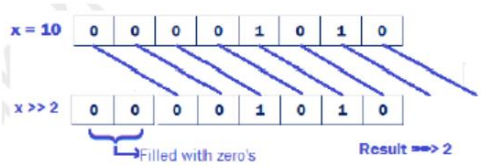
6. Bitwise Right Shift Operator (>>) :

- This Bitwise Right shift operator (>>) is a binary operator.
- It shifts the bits of a number towards right a specified no.of times.

Example:

int x = 10;

x >> 2 = ?



EXPRESSIONS

- In any programming language, if we want to perform any calculation or to frame any condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression. **In the java programming language, an expression is defined as follows..**
- An expression is a collection of operators and operands that represents a specific value.
- In the above definition, an operator is a symbol that performs tasks like arithmetic operations, logical operations, and conditional operations, etc.

Expression Types

In the java programming language, expressions are divided into THREE types. They are as follows.

- **Infix Expression**
- **Postfix Expression**
- **Prefix Expression**

The above classification is based on the operator position in the expression.

Infix Expression

The expression in which the operator is used between operands is called infix expression. The infix expression has the following general structure.

Example

a+b

Postfix Expression

The expression in which the operator is used after operands is called postfix expression. The postfix expression has the following general structure.

Example

ab+

Prefix Expression

The expression in which the operator is used before operands is called a prefix expression. The prefix expression has the following general structure.

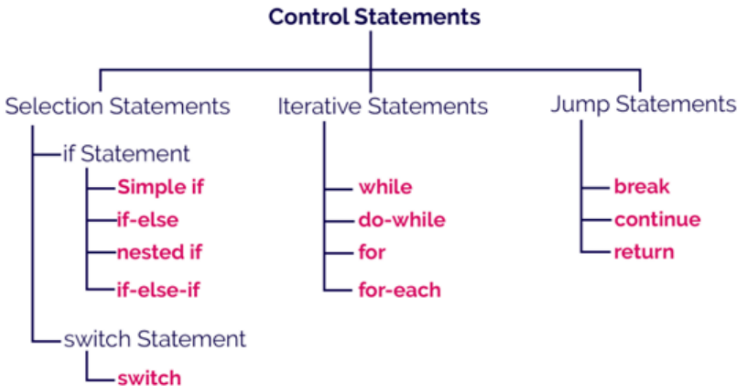
Example

+ab

CONTROL STATEMENTS

- In java, the default execution flow of a program is a sequential order.
- But the sequential order of execution flow may not be suitable for all situations.
- Sometimes, we may want to jump from line to another line, we may want to skip a part of the program, or sometimes we may want to execute a part of the program again and again.
- To solve this problem, java provides control statements.

Types of Control Statements



1. Selection Control Statements

In java, the selection statements are also known as decision making statements or branching statements. The selection statements are used to select a part of the program to be executed based on a condition.

Java provides the following selection statements.

- if statement
- if-else statement
- if-elif statement
- nested if statement
- switch statement

if statement in java

- In java, we use the if statement to test a condition and decide the execution of a block of statements based on that condition result.
- The if statement checks, the given condition then decides the execution of a block of statements. If the condition is True, then the block of statements is executed and if it is False, then the block of statements is ignored.

Syntax

```
if(condtion)
{
        if-block of statements;
}
statement after if-block;
```

Example

```
public class IfStatementTest
{
        public static void main(String[] args)
       {
                int x=10;
                if(x>0)
                x++;
                System.out.println("x value is:"+x);
        }
}
```

In the above execution, the number 12 is not divisible by 5. So, the condition becomes False and the condition is evaluated to False. Then the if statement ignores the execution of its block of statements.

if-else statement in java

- In java, we use the if-else statement to test a condition and pick the execution of a block of statements out of two blocks based on that condition result.
- The if-else statement checks the given condition then decides which block of statements to be executed based on the condition result.
- If the condition is True, then the true block of statements is executed and if it is False, then the false block of statements is executed.

Syntax

```
if(condtion)
{
        true-block of statements;
}
else
{
        false-block of statements;
}
statement after if-block;
```

Example

```
public class IfElseStatementTest
{
        public static void main(String[] args)
       {
                int a=29;
                if(a % 2==0)
                        System.out.println("Even Number is :"+a);
                 else
                        System.out.println("Odd Number is :"+a);
        }
}
```

Nested if statement in java

Writing an if statement inside another if-statement is called nested if statement.

Syntax

```
if(condition_1)
{
   if(condition_2)
   {
      inner if-block of statements;
      ...
   }
   ...
}
```

Example

```
public class NestedIfStatementTest
{
        public static void main(String[] args)
       {
                int num=1;
                if(num<10)
                {
                        if(num==1)
                        {
                                System.out.print("The value is equal to 1);
                        }
                        else
                        {
                                System.out.print("The value is greater than 1");
                        }
                }
                else
                {
                        System.out.print("The value is greater than 10");
                }
                System.out.print("Nested if - else statement ");
        }
}
```

if-else if statement in java

Writing an if-statement inside else of an if statement is called if-else-if statement.

Syntax

```
if(condition_1)
{
    condition_1 true-block;
                ...
}
else if(condition_2)
{
                condition_2 true-block;
                condition_1 false-block too;
                ...
}
```

Example

```
public class IfElseIfStatementTest
{
        public static void main(String[] args)
        {
                int x = 30;
                if( x == 10 )
                {
                        System.out.print("Value of X is 10");
                }
                else if( x == 20 )
                {
                        System.out.print("Value of X is 20");
                }
                else if( x == 30 )
                {
                        System.out.print("Value of X is 30");
                }
                else
                {
                        System.out.print("This is else statement");
                }
        }
}
```

Switch

- Using the switch statement, one can select only one option from more number of options very easily.
- In the switch statement, we provide a value that is to be compared with a value associated with each option. Whenever the given value matches the value associated with an option, the execution starts from that option.
- In the switch statement, every option is defined as a **case**.

Syntax:

```
switch (expression)
{
    case value1: // statement sequence
        break;
    case value2: // statement sequence
        break;
     ....
     case valueN:
}
```

Example

```
class SampleSwitch
{
    public static void main(String args[])
    {
        char color ='g';
        switch(color )
        {
            case 'r':
                System.out.println("RED") ; break ;
            case 'g':
                System.out.println("GREEN") ; break ;
            case 'b':
                System.out.println("BLUE") ; break ;
            case 'w':
                System.out.println("WHITE") ; break ;
            default:
                System.out.println("No color") ;
        }
    }
}
```

2. Iteration Statements

- The java programming language provides a set of iterative statements that are used to execute a statement or a block of statements repeatedly as long as the given condition is true.
- The iterative statements are also known as looping statements or repetitive statements. Java provides the following iterative statements.
    1. while statement
    2. do-while statement
    3. for statement
    4. for-each statement

while statement in java

The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE. The while statement is also known as Entry control looping statement.

Syntax

```
while(condition)
{
    // body of loop
}
```

Example

```
public class WhileTest
{
    public static void main(String[] args)
    {
        int num = 1;
        while(num <= 10)
        {
            System.out.println(num);
            num++;
        }
        System.out.println("Statement after while!");
    }
}
```

do-while statement in java

- The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE.
- The do-while statement is also known as the Exit control looping statement.

Syntax

```
do
{
    // body of loop
} while (condition);
```

Example

```
public class DoWhileTest
{
        public static void main(String[] args)
        {
                int num = 1;
                do
              {
                        System.out.println(num);
                        num++;
                }while(num <= 10);
                System.out.println("Statement after do-while!");
         }
}
```

for statement in java

The for statement is used to execute a single statement or a block of statements repeatedly as long as the given condition is TRUE.

Syntax

```
for(initialization; condition; inc/dec)
{
    // body
}
```

If only one statement is being repeated, there is no need for the curly braces.

In for-statement, the execution begins with the **initialization** statement. After the initialization statement, it executes **Condition**. If the condition is evaluated to true, then the block of statements executed otherwise it terminates the for-statement. After the block of statements execution, the **modification** statement gets executed, followed by condition again.

Example

```
public class ForTest
{
        public static void main(String[] args)
       {
                for(int i = 0; i < 10; i++)
              {
                        System.out.println("i = " + i);
                }
                System.out.println("Statement after for!");
        }
}
```

3. Jump Statements

The java programming language supports jump statements that used to transfer execution control from one line to another line.

The java programming language provides the following jump statements.

1. break statement
2. continue statement

break

When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

Example

```
class BreakStatement
{
        public static void main(String args[] )
        {
                int i;
                i=1;
                while(true)
                {
                        if(i >10)
                        break;
                        System.out.print(i+" ");
                        i++;
                }
        }
}
```

Continue

This command skips the whole body of the loop and executes the loop with the next iteration. On finding continue command, control leaves the rest of the statements in the loop and goes back to the top of the loop to execute it with the next iteration (value).

Example

```
/* Print Number from 1 to 10 Except 5 */
class NumberExcept
{
        public static void main(String args[] )
        {
                int i;
                for(i=1;i<=10;i++)
                {
                        if(i==5)
                           continue;
                        System.out.print(i +" ");
                }
        }
}
```

TYPE CONVERSION AND CASTING

Type Casting

- When a data type is converted into another data type by a programmer using the casting operator while writing a program code, the mechanism is known as **type casting**.
- In typing casting, the destination data type may be smaller than the source data type when converting the data type to another data type, that's why it is also called **narrowing conversion**.

Syntax

```
destination_datatype = (target_datatype)variable;
```

(): is a casting operator.

target_datatype : is a data type in which we want to convert the source data type.

Example

```
float x;
byte y;
y=(byte)x;
```

Program

```
public class NarrowingTypeCastingExample
{
    public static void main(String args[])
    {
        double d = 166.66;
        int i = (int)d;
        System.out.println("Before conversion: "+d);
        System.out.println("After conversion into int type: "+i);
    }
}
```

Output

```
Before conversion: 166.66
After conversion into int type: 166
```

Type Conversion

- If a data type is automatically converted into another data type at compile time is known as type conversion.
- The conversion is performed by the compiler if both data types are compatible with each other.
- Remember that the destination data type should not be smaller than the source type.
- It is also known as **widening** conversion of the data type.

Example

```
int a = 20;
Float b;
b = a;     // Now the value of variable b is 20.000
```

Program

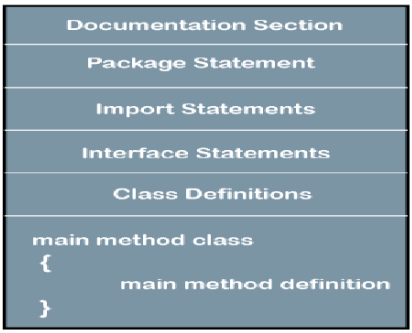
```
public class WideningTypeCastingExample
{
    public static void main(String[] args)
    {
        int x = 7;
        float y = x;
        System.out.println("After conversion, float value "+y);
}
}
```

Output :

After conversion, the float value is: 7.0

STRUCTURE OF JAVA PROGRAM

Structure of a Java program contains the following elements:



Structure of Java Program

Documentation Section

The documentation section is an important section but optional for a Java program.

It includes **basic information** about a Java program. The information includes the **author's name, date of creation, version, program name, company name,** and **description** of the program. It improves the readability of the program. Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program. To write the statements in the documentation section, we use **comments**.

Comments there are three types

1. Single-line Comment: It starts with a pair of forwarding slash **(//)**.

Example : //First Java Program

2. Multi-line Comment: It starts with a **/*** and ends with ***/.** We write between these two symbols.

Example : /* It is an example of
multiline comment */

3. Documentation Comment: It starts with the delimiter **(/**)** and ends with ***/**.

SAMPLE JAVA PROGRAM

```
/* This is First Java Program */
        Class sample
        {
                public static void main(String args[])
                {
                        System.out.println("Hello Java Programming");
                }
        }
```

Parameters used in First Java Program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

class keyword is used to declare a class in java.

public keyword is an access modifier which represents visibility. It means it is visible to all.

static is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.

void is the return type of the method. It means it doesn't return any value.

main represents the starting point of the program execution

String[] args is used for command line argument.

System.out.println() is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class.

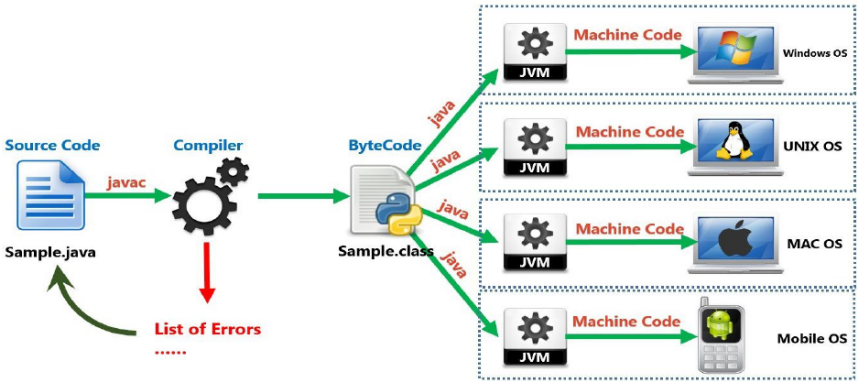
How to Compile and Run the Java Program

To Compile : javac Sample.java [program name]

To Run : java Sample

Output : Hello Java Programming

EXECUTION PROCESS OF JAVA PROGRAM



WHAT IS JVM

Java Virtual Machine is the heart of entire java program execution process. It is responsible for taking the .class file and converting each byte code instruction into the machine language instruction that can be executed by the microprocessor.

CLASSES AND OBJECTS IN JAVA

CLASSES

- In Java, classes and objects are basic concepts of Object Oriented Programming (OOPs) that are used to represent real-world concepts and entities.
- classes usually consist of two things: instance variables and methods.
- The class represents a group of objects having similar properties and behavior.
- For example, the animal type **Dog** is a class while a particular dog named **Tommy** is an object of the **Dog** class.
- 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.
- The java class is a template of an object.
- Every class in java forms a new data type.
- Once a class got created, we can generate as many objects as we want.

Class Characteristics

Identity - It is the name given to the class.

State - Represents data values that are associated with an object.

Behavior - Represents actions can be performed by 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 not 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

Creating a Class

In java, we use the keyword class to create a class. A class in java contains properties as variables and behaviors as methods.

Syntax

```
class className
{
        data members declaration;
        methods definition;
}
```

- The ClassName must begin with an alphabet, and the Upper-case letter is preferred.
- The ClassName must follow all naming rules.

Example

Here is a class called Box that defines three instance variables: width, height, and depth.

```
class Box
{
    double  width;
    double  height;
    double  depth;
    void volume()
    {
        ……..……..
    }
}
```

OBJECT

- In java, an object is an instance of a class.
- Objects are the instances of a class that are created to use the attributes and methods of a class.
- All the objects that are created using a single class have the same properties and methods. But the value of properties is different for every object.

Syntax

```
ClassName objectName = new ClassName( );
```

- The objectName must begin with an alphabet, and a Lower-case letter is preferred.
- The objectName must follow all naming rules.

Example

```
Box  mybox = new Box();
```

The new operator dynamically allocates memory for an object.

Example

```
class Box
{
      double  width;
      double  height;
      double depth;
}
class BoxDemo
{
    public static void main(String args[])
    {
          Box mybox = new Box();
          double vol;
          mybox.width = 10;
          mybox.height = 20;
          mybox.depth = 15;
          vol = mybox.width * mybox.height * mybox.depth;
          System.out.println("Volume is " + vol);
    }
}
```

METHODS

- A method is a block of statements under a name that gets executes only when it is called.
- Every method is used to perform a specific task. The major advantage of methods is code re-usability (define the code once, and use it many times).
- In a java programming language, a method defined as a behavior of an object. That means, every method in java must belong to a class.
- Every method in java must be declared inside a class.

Every method declaration has the following characteristics.

- **returnType** - Specifies the data type of a return value.
- **name** - Specifies a unique name to identify it.
- **parameters** - The data values it may accept or recieve.
- **{ }** - Defienes the block belongs to the method.

Creating a method

A method is created inside the class

Syntax

```
class ClassName
{
    returnType methodName( parameters )
    {
        // body of method
    }
}
```

Calling a method

- In java, a method call precedes with the object name of the class to which it belongs and a dot operator.
- It may call directly if the method defined with the static modifier.
- Every method call must be made, as to the method name with parentheses (), and it must terminate with a semicolon.

Syntax

```
objectName.methodName(actualArguments );
```

Example

```
//Adding a Method to the Box Class
```

```
Class Box
{
        double width, height, depth;
        void volume()
        {
                System.out.print("Volume is ");
                System.out.println(width * height * depth);
        }
}
class BoxDemo3
{
        public static void main(String args[])
        {
                Box mybox1 = new Box();
                Box mybox2 = new Box();
                mybox1.width = 10;
                mybox1.height = 20;
                mybox1.depth = 15;
                mybox2.width = 3;
                mybox2.height = 6;
                mybox2.depth = 9;
                mybox1.volume();
                mybox2.volume();
        }
  }
```

CONSTRUCTORS

- Constructor in Java is a special member method which will be called automatically by the JVM whenever an object is created for placing user defined values in place of default values.
- In a single word constructor is a special member method which will be called automatically whenever object is created.
- The purpose of constructor is to initialize an object called object initialization. Initialization is a process of assigning user defined values at the time of allocation of memory space.

Syntax

```
ClassName()
{
.....
.....
}
```

Types Of Constructors

Based on creating objects in Java constructor are classified in two types. They are

1. Default or no argument Constructor
2. Parameterized constructor

1. Default Constructor

- A constructor is said to be default constructor if and only if it never take any parameters.
- If any class does not contain at least one user defined constructor then the system will create a default constructor at the time of compilation it is known as system defined default constructor.

Note: System defined default constructor is created by java compiler and does not have any statement in the body part. This constructor will be executed every time whenever an object is created if that class does not contain any user defined constructor.

Example

```
class Test
{
        int a, b;
        Test()
        {
                a=10;
                b=20;
```

```
                System.out.println("Value of a: "+a);
                System.out.println("Value of b: "+b);
            }
    }
    class TestDemo
    {
        public static void main(String args[])
        {
            Test t1=new Test();
        }
    }
```

2. Parameterized Constructor

If any constructor contain list of variables in its signature is known as paremetrized constructor. A parameterized constructor is one which takes some parameters.

Example

```
class Test
{
    int a, b;
    Test(int n1, int n2)
    {
        a=n1;
        b=n2;
        System.out.println("Value of a = "+a);
        System.out.println("Value of b = "+b);
    }
}
class TestDemo
{
    public static void main(String args[])
    {
        Test t1=new Test(10, 20);
    }
}
```

ACCESS CONTROL(MEMBER ACCESS)

In Java, Access modifiers help to restrict the scope of a class, constructor, variable, method, or data member. It provides security, accessibility, etc to the user depending upon the access modifier used with the element.

Types of Access Modifiers in Java

There are four types of access modifiers available in Java:

1. Default – No keyword required
2. Private
3. Protected
4. Public

1. Default Access Modifier

- When no access modifier is specified for a class, method, or data member – It is said to be having the **default** access modifier 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

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.

2. Private

- The private access modifier is accessible only within the class.
- The private access modifier is specified using the keyword **private**.
- The methods or data members declared as private are accessible only **within the class** in which they are declared.
- Any other **class of** the **same package will not be able to access** these members.
- Top-level classes or interfaces can not be declared as private because private means "only visible within the enclosing class".

Example

- 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
        }
    }
```

3. Protected

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

Example

- 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();
    }
}
```

4. Public

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

Example

//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();
        }
}
```

Table: Class Member Access

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	YES	NO	NO	NO
Default	YES	YES	NO	NO
Protected	YES	YES	YES	NO
Public	YES	YES	YES	YES

'this' KEYWORD IN JAVA

this is a reference variable that refers to the current object. It is a keyword in java language represents current class object

Why use this keyword in java ?

- The main purpose of **using this keyword** is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then JVM get ambiguity (no clarity between formal parameter and member of the class).
- To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

Syntax: this.data member of current class.

Example without using this keyword

```
class Employee
{
    int id;
    String name;
    Employee(int id,String name)
    {
        id = id;
        name = name;
    }
    void show()
    {
        System.out.println(id+"  "+name);
    }
}
class ThisDemo1
{
    public static void main(String args[])
    {
        Employee e1 = new Employee(111,"Harry");
        e1.show();
    }
}
```

Output: 0 null

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Example of this keyword in java

```
class Employee
{
        int id;
        String name;
        Employee(int id,String name)
        {
                this.id = id;
                this.name = name;
        }
void show()
{
        System.out.println(id+" "+name);
}
class ThisDemo2
{
        public static void main(String args[])
        {
                Employee e1 = new Employee(111,"Harry");
                e1.show();
        }
}
```

Output: 111 Harry

GARBAGE COLLECTION IN JAVA

Garbage collection in Java is the process by which Java programs perform automatic memory management.

How Does Garbage Collection in Java works?

- 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.
- An unused or unreferenced object is no longer referenced by any part of your program.
- So the memory used by an unreferenced object can be reclaimed.
- The programmer does not need to mark objects to be deleted explicitly.
- The garbage collection implementation lives in the JVM.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How Can an Object be Unreferenced?

There are many ways:

1. By nulling the reference
2. By assigning a reference to another
3. By anonymous object etc.

1. By nulling a reference:

```
1. Employee e=new Employee();
2. e=null;
```

2. By assigning a reference to another:

```
1. Employee e1=new Employee();
2. Employee e2=new Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection
```

3. By anonymous object:

```
1. new Employee();
```

OVERLOADING METHODS AND CONSTRUCTORS

OVERLOADING METHODS

- Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as method overloading.
- Method overloading in Java is also known as Compile-time Polymorphism, Static Polymorphism, or Early binding.

Example

```
class Addition
{
        void sum(int a, int b)
        {
        System.out.println(a+b);
        }
        void sum(int a, int b, int c)
        {
        System.out.println(a+b+c);
        }
        void sum(float a, float b)
        {
        System.out.println(a+b);
        }
}
class Methodload
{
    public static void main(String args[])
    {
            Addition obj=new Addition();
            obj.sum(10, 20);
            obj.sum(10, 20, 30);
            obj.sum(10.05f, 15.20f);
    }
}
```

OVERLOADING CONSTRUCTORS

Constructor overloading is a concept of having more than one constructor with different parameters list, so that each constructor performs a different task.

Example

```
public class Person
{
        Person()
        {
                System.out.println("Introduction:");
        }
        Person(String name)
        {
                System.out.println("Name: " +name);
        }
        Person(String scname, int rollNo)
        {
                System.out.println("School name: "+scname+ ", "+"Roll no:"+rollNo);
        }
        public static void main(String[] args)
        {
                Person p1 = new Person();
                Person p2 = new Person("John");
                Person p3 = new Person("ABC", 12);
        }
}
```

METHOD BINDING

Connecting a method call to the method body is known as binding.

There are two types of binding

1. Static Binding (also known as Early Binding).
2. Dynamic Binding (also known as Late Binding).

Static Binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

Example

```
class Dog
{
    private void eat(){System.out.println("dog is eating...");}
    public static void main(String args[])
    {
            Dog d1=new Dog();
            d1.eat();
    }
}
```

Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

Example

```
class Animal
{
    void eat()
    {
            System.out.println("animal is eating...");
    }
}
class Dog extends Animal
{
    void eat()
    {
            System.out.println("dog is eating...");
    }
    public static void main(String args[])
    {
            Animal a=new Dog();
            a.eat();
     }
}
```

In the above example object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal. So compiler doesn't know its type, only its base type.

PARAMETER PASSING METHODS

Parameter passing in Java refers to the mechanism of transferring data between methods or functions. Java supports two types of parameters passing techniques

1. Call-by-value
2. Call-by-reference.

1. Call-by-Value

In Call-by-value the copy of the value of the actual parameter is passed to the formal parameter of the method. Any of the modifications made to the formal parameter within the method do not affect the actual parameter.

Example

```
public class CallByValueExample
{
    public static void main(String[] args)
    {
            int num = 10;
            System.out.println("Before calling method:"+num);
            modifyValue(num);
            System.out.println("After calling method:"+num);
    }
    public static void modifyValue(int value)
    {
             value=20;
             System.out.println("Inside  method:"+value);
    }
}
```

Output:

```
Before calling method: 10
Inside method: 20
After calling method: 10
```

Call-by-Reference

call by reference" is a method of passing arguments to functions or methods where the memory address (or reference) of the variable is passed rather than the value itself. This means that changes made to the formal parameter within the function affect the actual parameter in the calling environment.

In "call by reference," when a reference to a variable is passed, any modifications made to the parameter inside the function are transmitted back to the caller. This is because the formal parameter receives a reference (or pointer) to the actual data.

Example

```
class CallByReference
{
        int a,b;
        CallByReference(int x,int y)
        {
                a=x;
                b=y;
        }
         void changeValue(CallByReference obj)
        {
                obj.a+=10;
                obj.b+=20;
        }
}
public class CallByReferenceExample
{
     public static void main(String[] args)
     {
                CallByReference object=new CallByReference(10, 20);
                System.out.println("Value of a: "+object.a +" & b: " +object.b);
                object.changeValue(object);
                System.out.println("Value of a:"+object.a+ " & b: "+object.b);
    }
}
```

Output:

```
Value of a: 10 & b: 20
Value of a: 20 & b: 40
```

RECURSION IN JAVA

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method. It makes the code compact but complex to understand.

Syntax:

```
returntype methodname()
{
    methodname();
}
```

Example

```
public class RecursionExample3
{
    static int factorial(int n)
    {
        if (n == 1)
            return 1;
        else
            return(n * factorial(n-1));
    }
    public static void main(String[] args)
    {
        System.out.println("Factorial of 5 is: "+factorial(5));
    }
}
```

INNER CLASSES

- Inner class means one class which is a member of another class.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Syntax of Inner class

```
class Outer_class
{
    //code
    class Inner_class
    {
        //code
    }
}
```

Types of Inner classes

There are four types of inner classes.

1. Member Inner class
2. Local inner classes
3. Anonymous inner classes
4. Static nested classes

1. MEMBER INNER CLASS

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

```
class Outer
{
    //code
    class Inner
    {
        //code
    }
}
```

Example

```
class TestMemberOuter
{
        private int data=30;
        class Inner
        {
         void msg()
         {
              System.out.println("data is "+data);
        }
}
public static void main(String args[])
{
        TestMemberOuter obj=new TestMemberOuter();
        TestMemberOuter.Inner in=obj.new Inner();
        in.msg();
}
}
```

2. ANONYMOUS INNER CLASS

- In Java, a class can contain another class known as nested class. It's possible to create a nested class without giving any name.
- A nested class that doesn't have any name is known as an anonymous class.
- An anonymous class must be defined inside another class. Hence, it is also known as an anonymous inner class.

Example

```
abstract class Person
{
     abstract void eat();
}
class TestAnonymousInner
{
          public static void main(String args[])
```

```
        {
            Person p=new Person()
            {
                void eat()
                {
                    System.out.println("nice  fruits");
                }
            };
            p.eat();
        }
    }
```

1. A class is created, but its name is decided by the compiler, which extends the Person class and provides the implementation of the eat() method.
2. An object of the Anonymous class is created that is referred to by 'p,' a reference variable of Person type.

3. LOCAL INNER CLASS

- A class i.e. created inside a method is called local inner class in java.
- If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

Example

```
public class localInner
{
    private int data=30;
    void display()
    {
        class Local
        {
            void msg()
            {
                System.out.println(data);
            }
        }
        Local l=new Local();
        l.msg();
    }
    public static void main(String args[])
    {
        localInner obj=new localInner();
        obj.display();
    }
}
```

4. STATIC NESTED CLASS

- A static class i.e. created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.
- It can access static data members of outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

Example

```
class TestOuter
{
        static int data=30;
        static class Inner
        {
              void msg()
              {
               System.out.println("data is "+data);
               }
        }
        public static void main(String args[])
        {
            TestOuter.Inner obj=new TestOuter.Inner();
            obj.msg();
        }
 }
```

In this example, you need to create the instance of static nested class because it has instance method msg(). But you don't need to create the object of Outer class because nested class is static and static properties, methods or classes can be accessed without object.

EXPLORING STRING CLASS

- A string is a sequence of characters surrounded by double quotations. In a java programming language, a string is the object of a built-in class **String**.
- The string created using the **String** class can be extended. It allows us to add more characters after its definition, and also it can be modified.

Example

```
String siteName = "javaprogramming";
siteName = "javaprogramminglanguage";
```

String handling methods

In java programming language, the String class contains various methods that can be used to handle string data values.

The following table depicts all built-in methods of String class in java.

S.No	Method	Description
1	charAt(int)	Finds the character at given index
2	length()	Finds the length of given string
3	compareTo(String)	Compares two strings
4	compareToIgnoreCase(String)	Compares two strings, ignoring case
5	concat(String)	Concatenates the object string with argument string.
6	contains(String)	Checks whether a string contains sub-string
7	contentEquals(String)	Checks whether two strings are same
8	equals(String)	Checks whether two strings are same
9	equalsIgnoreCase(String)	Checks whether two strings are same, ignoring case
10	startsWith(String)	Checks whether a string starts with the specified string
11	isEmpty()	Checks whether a string is empty or not
12	replace(String, String)	Replaces the first string with second string
13	replaceAll(String, String)	Replaces the first string with second string at all occurrences.
14	substring(int, int)	Extracts a sub-string from specified start and end index values
15	toLowerCase()	Converts a string to lower case letters
16	toUpperCase()	Converts a string to upper case letters
17	trim()	Removes whitespace from both ends
18	toString(int)	Converts the value to a String object

Example

```
public class JavaStringExample
{
        public static void main(String[] args)
        {
                String title = "Java Programming";
                String siteName = "String Handling Methods";
                System.out.println("Length of title: " + title.length());
                System.out.println("Char at index 3: " + title.charAt(3));
                System.out.println("Index of 'T': " + title.indexOf('T'));
                System.out.println("Empty: " + title.isEmpty());
                System.out.println("Equals: " + siteName.equals(title));
                System.out.println("Sub-string: " + siteName.substring(9, 14));
                System.out.println("Upper case: " + siteName.toUpperCase());
        }
}
```