JAVA

==========

What is Java?

Java is general purpose programming language that is class based, object-oriented, platform independent and platform to develop an application.

History of JAVA :-

1. Java was originally developed by James Gosling's at sun microsystem (which has been acquired by oracle in 2010) and released in 1995.
2. James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.
3. Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
4. it was called Oak and was developed as a part of the Green project.

Why was Java named as "Oak"?

5. Why Oak? Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.
6. In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.

Why is Java Programming named "Java"?

7. The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc.
8. Java was so unique, most of the team members preferred Java over other names.
9. Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.
10. In 1995, Time magazine called Java one of the Ten Best Products of 1995.

Java Version History

JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Each new version adds new features in Java.

1. It promises WORA = "***Write One Run Anywhere***"
2. Since Java SE 8 release, the Oracle corporation follows a pattern in which every even version is release in March month and an odd version released in September month.

Sr.No	Version	Release date
1	JDK Alpha and Beta	1995
2	JDK 1.0	23rd Jan 1996
3	JDK 1.1	19th Feb 1997
4	J2SE 1.2	8th Dec 1998
5	J2SE 1.3	8th May 2000
6	J2SE 1.4	6th Feb 2002
7	J2SE 5.0	30th Sep 2004
8	Java SE 6	11th Dec 2006
9	Java SE 7	28th July 2011
10	Java SE 8	18th Mar 2014
11	Java SE 9	21st Sep 2017
12	Java SE 10	20th Mar 2018
13	Java SE 11	September 2018
14	Java SE 12	March 2019
15	Java SE 13	September 2019
16	Java SE 14	Mar 2020
17	Java SE 15	September 2020
18	Java SE 16	Mar 2021
19	Java SE 17	September 2021
20	Java SE 18	to be released by March 2022

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1. Standalone Application

a. Also called as Desktop based or windows-based application. These are software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc.
b. AWT and Swing are used in Java for creating standalone applications.

2. Web Application

c. An application that runs on the server side and creates a dynamic page is called a web application. Examples Facebook, amazon etc.
d. Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3. Enterprise Application

e. An application that is distributed in nature, such as banking applications, etc. is called an enterprise application.
f. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4. Mobile Application

g. An application which is created for mobile devices is called a mobile application.
h. Currently, Android and Java ME are used for creating mobile applications.

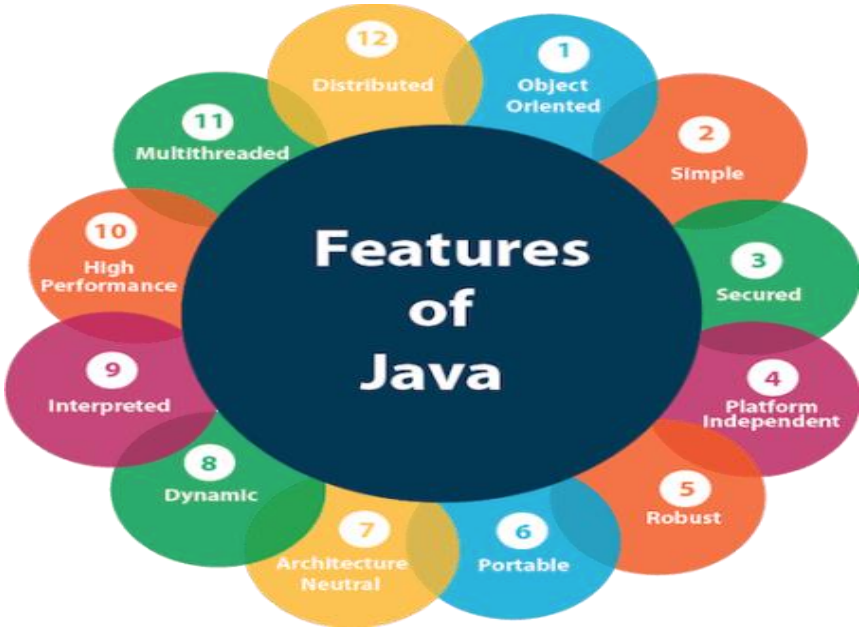
Java Platforms / Editions

1. **Java SE (Java Standard Edition)**
   a. It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc.
   b. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.
2. **Java EE (Java Enterprise Edition)**
   c. It is an enterprise platform that is mainly used to develop web and enterprise applications.
   d. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.
3. **Java ME (Java Micro Edition)**
   e. It is a micro platform that is dedicated to mobile applications.

Features of Java

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

1. Simple
2. Object-Oriented
3. Portable



4. Platform independent
5. Secured
6. Robust
7. Architecture neutral

8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

==========

Description about java program structure

1. **Project –**
    a. Project is a collection of multiple packages, multiple classes, user defined folder (for test data), supporting libraries, jar files and so on.
    b. We can create many packages and classes inside a project.
2. **Package –**
    c. Package is a collection of multiple classes
    d. There are two types of packages
        i. User defined package – If we create package
        ii. Default package – If we are created number of classes without generating package and shows under default package.
    e. Syntax – *package package_name ;*
    f. If package does not contain class then symbol should be colourless
    g. If class created inside package the symbol should be colourful
3. **Class**
    h. A class is collection of member functions(methods), variable, datatypes, printing statements, scanning statement, object, constructor etc.
    i. For all class name the first letter should be uppercase
    j. If several words are used to form a name of the class each inner words first letter should be in upper case
    k. Eg. *public class MyFirstJavaProgram*

==========

==========

Basis terms and Explanation

Case sensitivity

Java is a case sensitive which means identifier "Hello" and ""hello" would have different meaning in java

1. **ClassName**
   - for all classname start with uppercase letter
   - If have multi name class, then uppercase of each first letter of word

     Eg. public class MyFirstJavaProgram
2. **methodName**
   - all method starts with lower case letter
   - If have multi name method, then lowercase of first word and upper case of next words

     Eg. myMethodName()
3. **ProjectName**
   - for all project name start with uppercase letter
   - If have multi name project, then uppercase of each first letter of word

     Eg. MyProject
4. **PackageName**
   - all method starts with lower case letter
   - If have multi name package, then lowercase of first word and upper case of next words

     Eg. myPackage
5. **Comments(//)**
   - These lines are used for comments or writing some statement that will be not considered during execution of programs
6. **Signature Bracket ()**
   - also called argument bracket
   - This bracket we called as signature bracket or argument bracket
7. **+ Sign**
   - This + sign is ued for concatinating or we can say that join the string or statements
8. **JRE library**
   - In JRE library there are supporting jar files to write jave programs

9. { }

- It is called as body or curly brakes

  eg. class body, method body etc.

10.src

- src is a source folder where the project source file content programs

11.System.out.println()

- System - it is name of jave utility class
- out - it is object which belongs to system class to call predefined method
- println - it is predefined method or utility method which is used to send any string to console

==========

Method/Member Function in Java

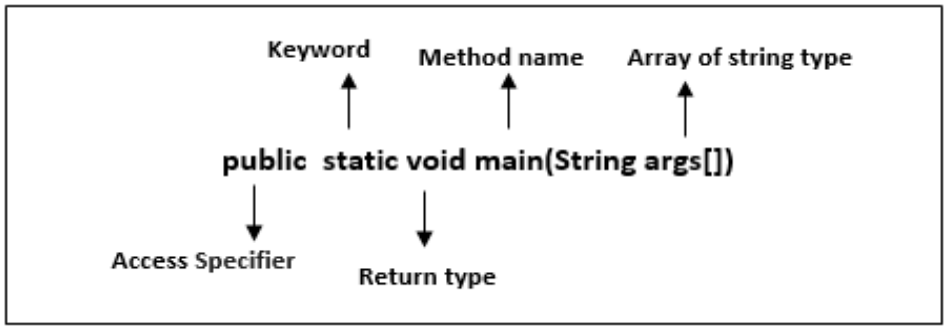
- Method in java is a collection of instructions that perform specific task. Without method we are not able to write program.
- It is also called member function
- It is memory location in JVM to store the data and information
- We can easily modify code using method
- Method is always declared inside class body.

There are two types of methods

1. Business method – main method
2. Regular method

Java main() method

- The main() is the starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program.
- The syntax of the main() method is:



- **public:** It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program.
- **static:** You can make a method static by using the keyword static. We should call the main() method without creating an object.
- **void:** In Java, every method has the return type. Void keyword in main() method does not return any value.
- **main():** It is a default method name which is predefined in the JVM.
- **String args[]:** The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array.

Regular method

Two types of methods

1. Static method
2. Non-Static method

1. Static Method

- The method which contains static keyword and used to perform particular task
- It is static in nature
- To call static method there is no need to create object.
- Main method is static method
- It can be represented as

```
public static void methodName()
{
        Body of the program for execution.
}
```

- It can call as ***ClassName.methodName();*** or only ***methodName();***

2. Non-Static Method

- The method which don't have static keyword
- It is dynamic in nature
- To call non-static method we need to create object

```
public void methodName()
{
        Body of the program for execution.
}
```

- It can call as object.methodName();

Object

- It is instance of class/example of class and used to call non-static method.
- An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical.

How to Create Object in Java using new keyword

ClassName object = new ClassName();

==========

==========

Data Types in Java

Datatypes are used to represent types of data or information that we going to used in the programming.

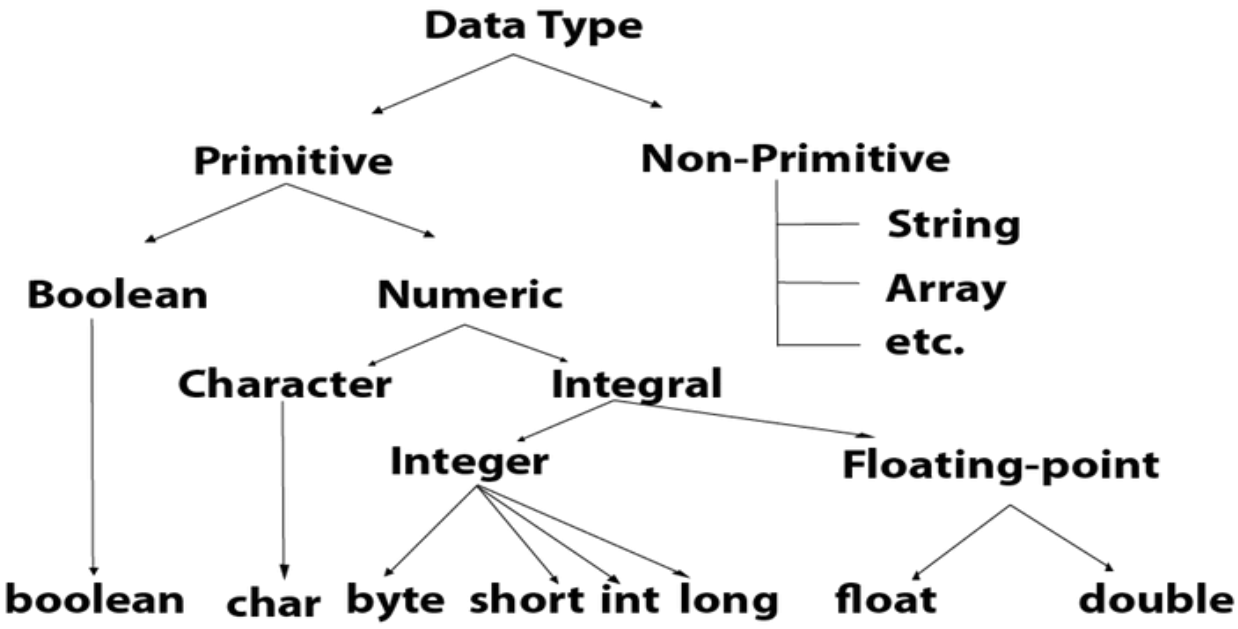
It is mandatory / compulsory to declare datatype before variable.

Types of data types in Java:

1. **Primitive data types:**
   - The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:**
   - The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.



There are 8 types of primitive data types:

1. boolean data type
2. byte data type
3. char data type
4. short data type

5. int data type
6. long data type
7. float data type
8. double data type

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

1. Boolean Data Type

- The Boolean data type is used to store only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.
- The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.
  Example: Boolean one = **false**

2. Byte Data Type

- The byte data type is an example of primitive data type.
- Its value-range lies between -128 to 127
- Its minimum value is -128 and maximum value is 127. Its default value is 0.

- The byte data type is used to save memory in large arrays where the memory savings is most required.
  Example: `byte a = 10, byte b = -20`

3. Short Data Type

- Its value-range lies between -32,768 to 32,767
- Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.
- The short data type can also be used to save memory just like byte data type.
  Example: `short s = 10000, short r = -5000`

4. Int Data Type

- Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1)
- Its minimum value is - 2,147,483,648 and maximum value is 2,147,483,647. Its default value is 0.
- The int data type is generally used as a default data type for integral values unless if there is no problem about memory.
  Example: `int a = 100000, int b = -200000`

5. Long Data Type

- Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1).
- Its minimum value is - 9,223,372,036,854,775,808 and maximum value is 9,223,372,036,854,775,807. Its default value is 0.
- The long data type is used when you need a range of values more than those provided by int.
  Example: `long a = 100000L, long b = -200000L`

6. Float Data Type

- The float data type is a single-precision 32-bit IEEE 754 floating point.
- Its value range is unlimited.
- It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers.
- The float data type should never be used for precise values, such as currency.
- Its default value is 0.0F.
  Example: `float f1 = 234.5f`

7. Double Data Type

- The double data type is a double-precision 64-bit IEEE 754 floating point.

- Its value range is unlimited.
- The double data type is generally used for decimal values just like float.
- The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.
  Example: `double d1 = 12.3`

8. Char Data Type

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

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

==========

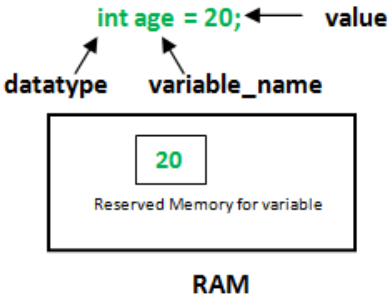
==========

Variables

- Variable is nothing but piece of memory used to store information.
- A variable is a name given to a memory location. It is the basic unit of storage in a program.

1. According to all programming language we cannot declare information directly, so variables are introduced.
2. To declare variable information to store information
3. In java programming variable can't be declare directly to store info we have declare datatype before it.
4. Reusable – It help us the info will use again and again

How to declare variables?



Datatype variable = information;

Types of Variable

1. Local Variable :-
   - The variable which are declared inside method body, method block or constructor called as local variables
   - The scope of variable remains within same method body
   - These variables are temporary variable
   - It cannot defined by static keyword
   - Utilize within method cannot be accessible outside of any method
   - It don't have any default values for variable
2. Static Variable
   - Static variables also known as class variable
   - Static variables are declared in class body but outside of method/block with static keyword before it.
   - We have only one copy of it.

- Initialization of static variable is not necessary
- Static variables are directly accessible to any method
- We have default values

int, byte, short, long	0
float double	0.0
String	null
boolean	false

-

3. Global Variable
    - Also called as Instance/non-static variable
    - Global variables are declared in class body but outside of method/block
    - Initialization of global variable is not necessary
    - We have default values as

int, byte, short, long	0
float double	0.0
String	null
boolean	false

==========

Constructor

==========

Constructor

- Constructor are used to initialize of data members(variables) of class and to load non-static member/methods into object
- Constructor is a block similar to method.
- Constructors are special member of class
- According to java each class must have a constructor, **if no constructor is present** then at time of compilation compiler will create a default constructor for the class and this default constructor will have a blank body.
- At the time of constructor declaration below are points need to followed
    - Constructor name should be same as class name
    - We should not declare any return type for Constructor
    - Any number of Constructor can be declared in class but name should same as class name but having different argument.

Ways to call a constructor

1. By creating object of a class.
    - Constructor executes automatically when we create an object.
2. By using 'new' keyword with constructor name with method signature followed by semicolon.
    - i.e ***new ConstructorName();***

There are 2 types of constructor

1. Default Constructor
2. User defined Constructor
    a. Zero Argument/ Non Argument Constructor
    b. Parameterized/ Argument Constructor

1. Default Constructor

- If Constructor is not declared in class then at the time compilation compiler will provide/consider the constructor.
- If programmer declared constructor then compiler will not provide any default constructor
- If constructor provided by compiler at the time of compilation is known as default constructors
- the best example of default constructor is main method

```java
package constructorExamples;

public class Example1 {

	public void method1()
	{
		String name ="Shivlila";
		System.out.println(name);
	}

	public static void main(String args[])
	{
		Example1 obj=new Example1();
		obj.method1();
	}

}
```

2. User defined constructor

- A programmer is declaring constructor in Java class then it is considered to be user defined constructor
- User defined constructor is divided into two types
  - Zero argument constructor and
  - parameterised constructor

1. Zero argument constructor

- We can initialise data member of a class
- constructor with no argument constructor is known as zero argument constructor or non argument constructor
- the signature is same as a default constructor

```java
	package constructorExamples;

	public class Example2 {

	//declaration
	static int a;
	static String name;

	//constructor
	Example2()
	{
		//initlition
		a=20;
```

```
        name="Ganesh";
        //usage
        System.out.println(a);
        System.out.println(name);
    }
    public static void main(String args[])
    {
        Example2 xyz=new Example2();
    }
    }
```

2. **Parametrize constructor**
   - Parametrize constructor a constructor is called as parameterised constructor
   - when it accepts a specific number of a parameter or argument in a signature bracket to initialise data members of a class with a distinct value
   - this constructor is used to allow multiple constructor by providing different types of arguments in simple words constructors with argument is known as parameters constructors
   - access scope of this constructor will be same as a class access code it means if class is a public then constructor is a public hip classes private then constructor is private or vice versa
   - constructors are going to invoke/use at the time of object creation at the time of object creation

```
    package constructorExamples;

    public class ArgumentCons {

    String city;
    ArgumentCons()
    {
        city="Pune";
        System.out.println(city);
    }

    ArgumentCons(int a)
    {
        System.out.println(a);
    }

    ArgumentCons(String name)
```

```
    {
        System.out.println(name);
    }

    public static void main(String args[])
    {
        ArgumentCons obj1=new ArgumentCons();
        ArgumentCons obj2=new ArgumentCons(852681256);
        ArgumentCons obj3=new ArgumentCons("Shivlila");
    }
    }
```

Operators in Java

==========

Operator in Java is a symbol that is used to perform operations.

Types :-

1. Unary Operator,
2. Arithmetic Operator,
3. Shift Operator,
4. Relational Operator,
5. Bitwise Operator,
6. Logical Operator,
7. Ternary Operator and
8. Assignment Operator

Java Operator Precedence

Operator	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr*
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >=
	equality	== !=
Bitwise	AND	&
	OR	|
	XOR	^
Logical	AND	&&
	OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>=

1. Java Unary Operator

- The Java unary operators require only one operand.
- Unary operators are used to perform various operations.
  - incrementing/decrementing a value by one
  - negating an expression

Operator	Meaning	Work
a++ a--	postfix	Print + operation
++a --a	prefix	Operation + print

2. Java Arithmetic Operators

- Java arithmetic operators are used to perform addition, subtraction, multiplication, and division.
- They act as basic mathematical operations.

Operator	Meaning	Work
+	Addition	To add two operands.
-	Subtraction	To subtract two operands.
*	Multiplication	To multiply two operands.
/	Division	To divide two operands.
%	Modulus	To get the area of the division of two operands.

3. Shift Operator

- It is used to shift all bits in value to left/right side of specified number in times

a. Left Shift Operator

- The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

b. Java Right Shift Operator

- The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

4. Logical Operators

- These operators are used to perform logical "AND", "OR" and "NOT" operation.
- They are used to combine two or more conditions.

Operator	Meaning	Working
&&	AND	True --- All conditions need true False --- any one condition will be false
||	OR	True --- any one condition is True False -- all condition false
!	Not	Invert conditions

5. Bitwise Operators

Operator	Meaning	Work
&	AND Operator	True - All condition true False - Any one condition false
|	OR Operator	True - any one condition is true False - all condition false
^	XOR Operator	False - Both condition same / T or F True - Have different values of conditions

6. Relational operators

- The Java Relational operators compare between operands and determine the relationship between them.
- The output of the relational operator is (true/false) boolean value

Operator	Meaning
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

7. Assignment Operators

- The Java Assignment Operators are used when you want to assign a value to the expression. The assignment operator denoted by the single equal sign =.
- In a Java assignment statement, any expression can be on the right side and the left side must be a variable name.

Operator	Example
=	C = A + B will assign value of A + B into C
+=	C += A is equivalent to C = C + A
-=	C -= A is equivalent to C = C – A
*=	C *= A is equivalent to C = C * A
/=	C /= A is equivalent to C = C / A
%=	C %= A is equivalent to C = C % A
<<=	C <<= 2 is same as C = C << 2
>>=	C >>= 2 is same as C = C >> 2
&=	C &= 2 is same as C = C & 2
^=	C ^= 2 is same as C = C ^ 2
|=	C |= 2 is same as C = C | 2

Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements.

Java provides three types of control flow statements.

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

Decision-Making statements:

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

==========

1) If Statement:

The "if" statement is used to evaluate a condition. The control of the program is depending upon the specific condition. The condition of If statement gives a Boolean value, either true or false.

There are four types of if-statements given below.

1. Simple if statement
2. if-else statement

3. if-else-if ladder
4. Nested if-statement

==========

1) Simple if statement:

It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax:-

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

Example.

```
public class Student{
    public static void main(String[] args) {
        int x = 10;
        int y = 12;
        if(x+y > 20)
        {
            System.out.println("x + y is greater than 20");
        }
  }
}
```

Output:

x + y is greater than 20

==========

2) if-else statement

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

Syntax:-

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

Example.

```
public class Student {
    public static void main(String[] args) {
        int x = 10;
        int y = 12;
        if(x+y < 10)
        {
                System.out.println("x + y is less than10");
        }
        else
        {
                System.out.println("x + y is greater than 20");
        }
    }
}
```

Output:

x + y is greater than 20

==========

3) if-else-if ladder:

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

Syntax:-

Syntax:

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

Example

```
public class Student {
    public static void main(String[] args) {
        String city = "Delhi";
        if(city == "Meerut")
        {
                System.out.println("city is meerut");
        }
        else if (city == "Noida")
        {
                System.out.println("city is noida");
```

```
            }
            else if(city == "Agra")
            {
                    System.out.println("city is agra");
            }
            else
           {
                   System.out.println(city);
             }
          }
    }
```

Output: Delhi

==========

4. Nested if-statement

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

Syntax:

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

Example

```
        public class Student {
              public static void main(String[] args) {
                    String address = "Delhi, India";
                if(address.endsWith("India"))
               {
```

```
                if(address.contains("Meerut"))
                {
                        System.out.println("Your city is Meerut");
                }
                else if(address.contains("Noida"))
                {
                        System.out.println("Your city is Noida");
                }
                else
                {
                        System.out.println(address.split(",")[0]);
                }
                }
                else
                {
                        System.out.println("You are not living in India");
                }
                }
                }
```

Output: Delhi

==========

Switch Statement:

In Java, Switch Statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.

The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:

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

- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

Syntax:

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

Example

```
public class Student implements Cloneable {
    public static void main(String[] args) {
        int num = 2;
        switch (num)
        {
            case 0:System.out.println("number is 0");
            break;
            case 1:System.out.println("number is 1");
            break;
            default:System.out.println(num);
        }
    }
}
```

Output: 2

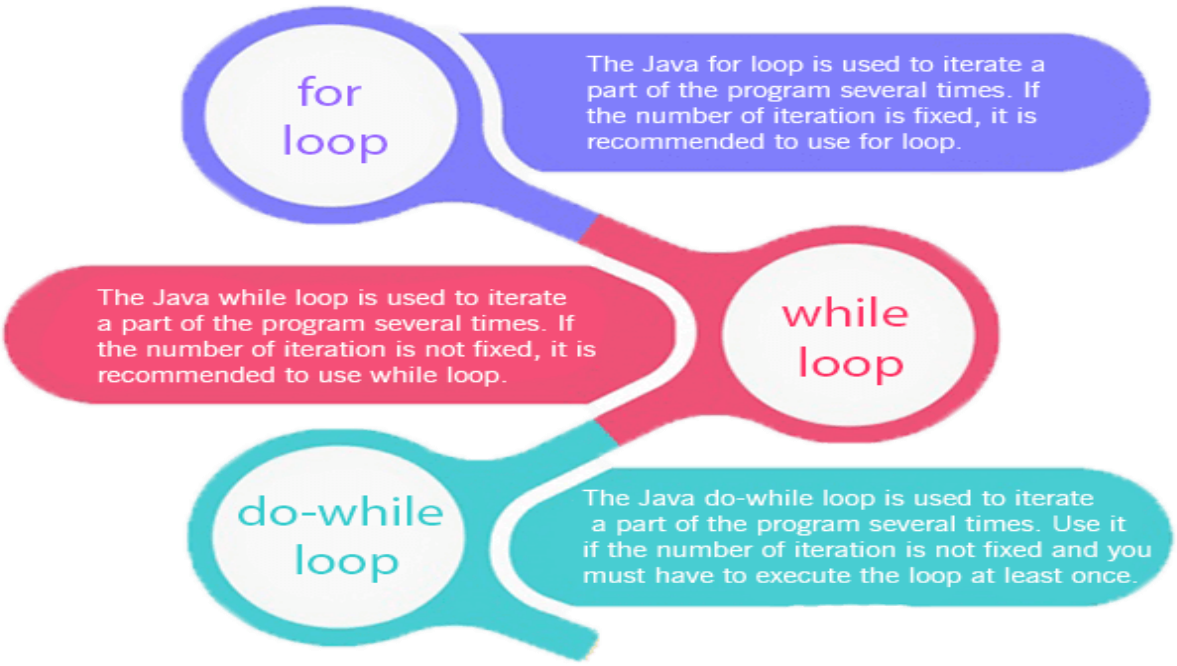
While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value. The switch permits only int, string, and Enum type variables to be used.

Loops in Java

Loop statements are used to execute the set of instructions in a repeated order. The set of instructions execute depends upon a particular condition and In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true.

In Java, we have three types of loops that execute similarly.

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



Java for loop

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

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

Syntax:

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

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false.
3. **Increment/Decrement**: It increments or decrements the variable value.
4. **Statements**: The statement of the loop is executed each time until the second condition is false.

Q. Print 1 to 10

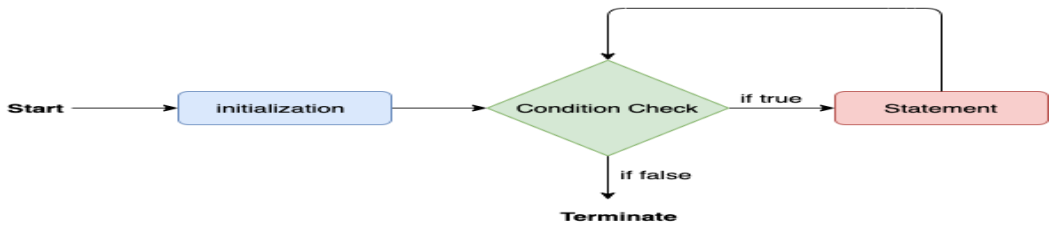
```
public class ForExample {
    public static void main(String[] args) {
        //Code of Java for loop
        for(int i=1;i<=10;i++)
        {
            System.out.println(i);
        }
    }
}
```

Output – 1

....

10

Flowchart:



Java Nested for Loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Syntax:

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

}
```

Example: NestedForExample.java

```
public class NestedForExample {
    public static void main(String[] args) {
        //loop of i
        for(int i=1;i<=3;i++)
        {
            //loop of j
            for(int j=1;j<=3;j++)
            {
                System.out.println(i+" "+j);
            }//end of i
        }//end of j
    }
}
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

Java while loop

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

1. The Java while loop is used to iterate a part of the programs repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.
2. The initialization and increment/decrement doesn't take place inside the loop statement in while loop.
3. It is also known as the entry-controlled loop since the condition is checked at the start of the loop.

Syntax:

```
1.          while(condition)
            {
                    statements;
                    Increment / decrement statement;
            }
```

Example:-

we print integer values from 1 to 10.

```
public class WhileExample {
    public static void main(String[] args) {
        int i=1;
        while(i<=10)
        {
            System.out.println(i);
            i++;
        }
    }
}
```

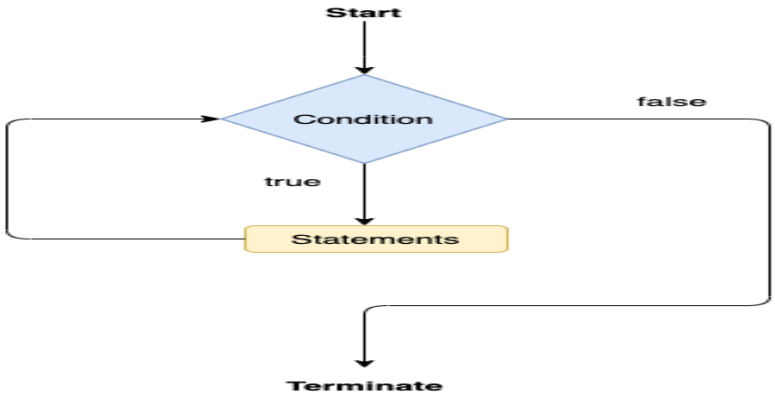
Output – 1

.

.

10

Flowchart



Java do-while loop

The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.

1. Java do-while loop is called an **exit control loop**.
2. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do
{
        //statements;
} while (condition);
```

Example:-

we print integer values from 1 to 10.

```
public class DoWhileExample {
        public static void main(String[] args) {
                int i=1;
                do
                {
                        System.out.println(i);
```

```
                i++;
            }while(i<=10);
        }
    }
```

Output – 1

.

.

10

Java for Loop vs while Loop vs do-while Loop

Comparison	for loop	while loop	do-while loop
Introduction	for loop is a control flow statement that iterates a part of the programs multiple times.	while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Syntax	for(init;condition;incr/decr){
	// code to be executed
	}
		while(condition){
		//code to be executed
		}
			do{
			//code to be executed
			}while(condition);

Syntax for infinitive loop	for(;;){
	//code to be executed
	}
		while(true){
		//code to be executed
		}
			do{
			//code to be executed
			}while(true);

Java User Input (Scanner)

==========

Scanner Class

- The Scanner class is mainly used to get the user input, and it belongs to the *java.util* package.
- In order to use the Scanner class, you can create an object of the class and use any of the Scanner class methods.

Input Types

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user
nextLine().chatAt(i)	Reads a char value from the user

Object creation

```
Scanner input = new Scanner(System.in);
```

Examples:-

```
import java.util.Scanner;  // Import the Scanner class
 public class Example1 {
            public static void main(String[] args) {
                    // Create a Scanner object
                    Scanner obj = new Scanner(System.in);
                    System.out.println("Enter username");
                    // Read user input
                    String userName =obj.nextLine();
                    // Output user input
                    System.out.println("Username is: " + userName);
                }
 }
```

Output :-

```
Enter username
My name is Khan
Username is: My name is Khan
```

```
import java.util.Scanner;
public class Example2 {
            public static void main(String[] args) {
                    Scanner myObj = new Scanner(System.in);
                    System.out.println("Enter name, age and salary:");
                    // String input
                    String name = myObj.nextLine();
                   // Numerical input
                   int age = myObj.nextInt();
                   double salary = myObj.nextDouble();
                   // Output input by user
                   System.out.println("Name: " + name);
                   System.out.println("Age: " + age);
                   System.out.println("Salary: " + salary);
                }
 }
```

Output :-

```
Enter name, age and salary:
Mayur
25
100000
Name: Mayur
Age: 25
Salary: 100000
```

Close Scanner

- Java Scanner class uses the "Close ()" method to close the Scanner.
- Input.close();

Do you need to close a Scanner class?

- It is better but not mandatory to close the Scanner class as if it is not closed, the underlying Readable interface of the Scanner class does the job for you. The compiler might flash some warning though if it is not closed.
- It is a good programming practice to explicitly close the Scanner using the Close () method once you are done using it.

==========

Java Math class

==========

Math class

Java Math class provides several methods to work on math calculations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs() etc.

Method	Description
Math.abs()	It will return the Absolute value of the given value.
Math.max()	It returns the Largest of two values.
Math.min()	It is used to return the Smallest of two values.
Math.round()	It is used to round of the decimal numbers to the nearest value.
Math.sqrt()	It is used to return the square root of a number.
Math.cbrt()	It is used to return the cube root of a number.
Math.pow()	It returns the value of first argument raised to the power to second argument.

Other than this the **java.lang.Math** class contains various such as the logarithm, cube root, and trigonometric functions etc.

```
public class JavaMathExample1
   {
      public static void main(String[] args)
      {
         double x = 28;
         double y = 4;

         // return the maximum of two numbers
         System.out.println("Maximum number of x and y is: " +Math.max(x, y));

         // return the square root of y
         System.out.println("Square root of y is: " + Math.sqrt(y));
```

```
        //returns 28 power of 4 i.e. 28*28*28*28
        System.out.println("Power of x and y is: " + Math.pow(x, y));

        // return the logarithm of given value
        System.out.println("Logarithm of x is: " + Math.log(x));
        System.out.println("Logarithm of y is: " + Math.log(y));

        // return the logarithm of given value when base is 10
        System.out.println("log10 of x is: " + Math.log10(x));
        System.out.println("log10 of y is: " + Math.log10(y));

        // return the log of x + 1
        System.out.println("log1p of x is: " +Math.log1p(x));

        // return a power of 2
        System.out.println("exp of a is: " +Math.exp(x));

        // return (a power of 2)-1
        System.out.println("expm1 of a is: " +Math.expm1(x));
    }
}
```

Output:-

```
Maximum number of x and y is: 28.0
Square root of y is: 2.0
Power of x and y is: 614656.0
Logarithm of x is: 3.332204510175204
Logarithm of y is: 1.3862943611198906
log10 of x is: 1.4471580313422192
log10 of y is: 0.6020599913279624
log1p of x is: 3.367295829986474
exp of a is: 1.446257064291475E12
expm1 of a is: 1.446257064290475E12
```

OOPs (Object-Oriented Programming System)

==========

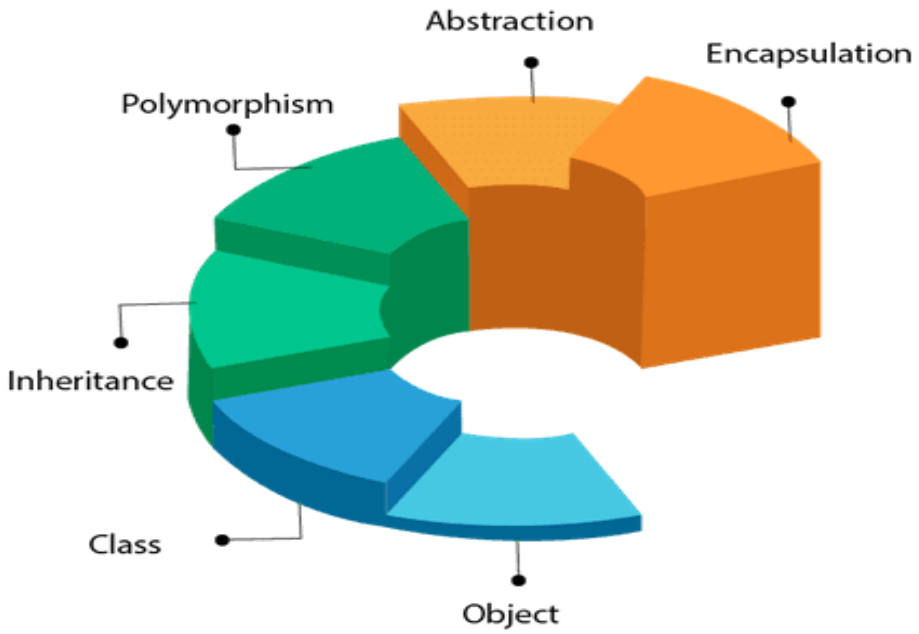
Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology to design a program using classes and objects.

Pillars/feature of OOPs:-

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

==========

OOPs (Object-Oriented Programming System)



==========

Object:-

Any entity that has state and behaviour is known as an object. For example a chair, pen, table, keyboard, bike, etc.

1. It can be physical or logical.
2. An Object can be defined as an instance of a class.
3. An object contains an address and takes up some space in memory.
4. Objects can communicate without knowing the details of each other's data or code.

Example: A table/bike is an object because it has states like color, name etc. as well as behaviors like running etc. 1. **State**: represents the data (value) of an object. 2. **Behaviour**: represents the behaviour (functionality) of an object such as deposit, withdrawal, etc.	

==========

Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

- **Inheritance in Java** is a mechanism in which one class acquires all the properties and behaviours of another class with the help of extends keyword.
- Inheritance is one of the most important Object-oriented programming language system
- Also, we can say that subclass can acquire the properties of superclass with extends keyword
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

1. For Code Reusability.
2. For code optimization

Terms used in Inheritance

- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

==========

Types of Inheritance

1. Single Level Inheritance
2. Multi-Level Inheritance
3. Multiple Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

==========

Single Level Inheritance

- For single level inheritance two classes are mandatory
- It is an operation where inheritance takes place between two classes to perform Single Level Inheritance
- In short we can say that when a class inherits another class it is known as single level Inheritance

```
package SingleLevelInh;

public class Bank {
```

```
        public void account()
        {
                System.out.println("This is the process for accout creation");
        }

        public void withdraw()
        {
                System.out.println("This is the process for withdraw");
        }
    }
```

➢

```
package SingleLevelInh;
public class HDFCClass extends Bank{

        public void bankName()
        {
                System.out.println("The name of bank is HDFC");
        }
}
```

```
package SingleLevelInh;
public class MasterClass {
        public static void main(String[] args) {
                HDFCClass obj=new HDFCClass();
                obj.bankName();
                obj.account();
                obj.withdraw();

                Bank obj1=new Bank();
                obj1.account();
                obj1.withdraw();
        }
}
```

Output:-

```
The name of bank is HDFC
This is the process for accout creation
This is the process for withdraw
This is the process for accout creation
This is the process for withdraw
```

==========

MultiLevel Inheritance

- Multilevel inheritance takes place between three or more classes.
- In multilevel inheritance one subclass can acquire the property of another superclass and this phenomenon continues so that we known as *"**Multi level Inheritance**"*
- In short when there is a chain of inheritance it is also known as multilevel inheritance.

```
package MUltiLevelIn;
public class ClassA {
    public void add()
    {
            int a=10;
            int b=20;
            System.out.println("I am from Class A");
            System.out.println("Addition result is:- "+(a+b));
    }
}
```

```
package MUltiLevelIn;
public class ClassB extends ClassA{
        //ClassB have properties ClassB+ClassA
        public void sub()
        {
                int a=10;
                int b=20;
                System.out.println("I am from Class B");
                System.out.println("sub result is:- "+(b-a));
        }
}
```

```
package MUltiLevelIn;
 public class ClassC extends ClassB{
        //preperties
        public void multi()
        {
                int a=10;
                int b=20;
                System.out.println("I am from Class C");
                System.out.println("multi result is:- "+(a*b));
        }
}
```

```
package MUltiLevelIn;
public class MasterClass {
	public static void main(String[] args) {

		ClassC c=new ClassC();
		c.add();
		c.sub();
		c.multi();

		ClassB b=new ClassB();
		b.add();
		b.sub();

		ClassA a=new ClassA();
		a.add();
	}
}
Output:-
I am from Class B
sub result is:- 10
I am from Class C
multi result is:- 200
I am from Class D
div result is:- 2
I am from Class B
sub result is:- 10
I am from Class C
multi result is:- 200
I am from Class B
sub result is:- 10
I am from Class A
Addition result is:- 30
```

==========

Multiple Inheritance

- In multiple inheritance one subclass acquiring properties of two superclasses at a same time so this known as multiple inheritance
- Java doesn't support multiple inheritance because it results diamond ambiguity problem

- It means the diamond like structure get created in jVM and object variable get confused for who need to inherited the property of superclass Ito subclass
- SuperClass of all the classes is object class so there diamond ambiguity forms
- To reduce the complexity and simplify the language multiple inheritance not supported in java

==========

Hierarchical Inheritance:-

- Hierarchical Inheritance takes place between one superclass and multiple subclass
- So the property of superclass can be acquired by multiple subclass this is kowm as Hierarchical Inheritance
- The two or more classes inherits a single class it is known as Hierarchical Inheritance
- This inheritance takes place between one superclass and multiple subclass

```
package Hierachicaln;
public class Bank{
    public void mainMethod()
    {
            System.out.println("This is main method from bank");
    }
}
```

```
package Hierachicaln;
public class HDFC extends Bank {
        public void hdfcMethod()
        {
                System.out.println("This is hdfc method from HDFC class");
        }
}
```

```
package Hierachicaln;
public class Axis extends Bank{
        public void axisMethod()
        {
                System.out.println("This is Axis method from axis class");
        }
}
```

```
package Hierachicaln;
public class SBI extends Bank{
```

```
        public void sbiMethod()
        {
                System.out.println("This is SBI method from SBI CLASS");
        }
}
```

```
package Hierachicaln;
public class MasterClass {
        public static void main(String[] args) {
                Bank b=new Bank();
                b.mainMethod();

                HDFC h=new HDFC();
                h.mainMethod();
                h.hdfcMethod();

                SBI s=new SBI();
                s.mainMethod();
                s.sbiMethod();

                Axis a=new Axis();
                a.mainMethod();
                a.axisMethod();
        }
}
```

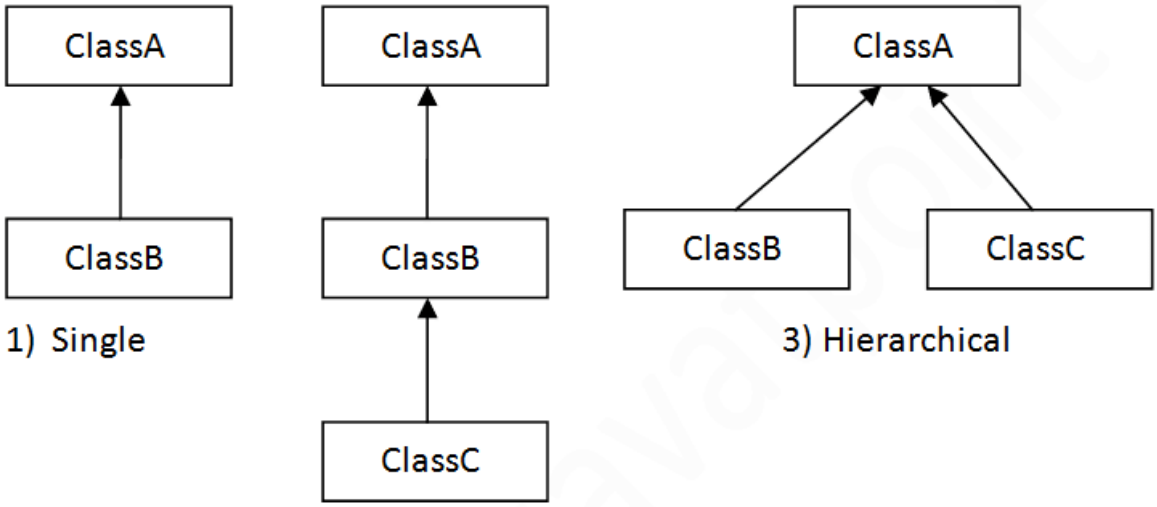
Output:-

```
This is main method from bank
This is main method from bank
This is hdfc method from HDFC class
This is main method from bank
This is SBI method from SBI CLASS
This is main method from bank
This is Axis method from axis class
```

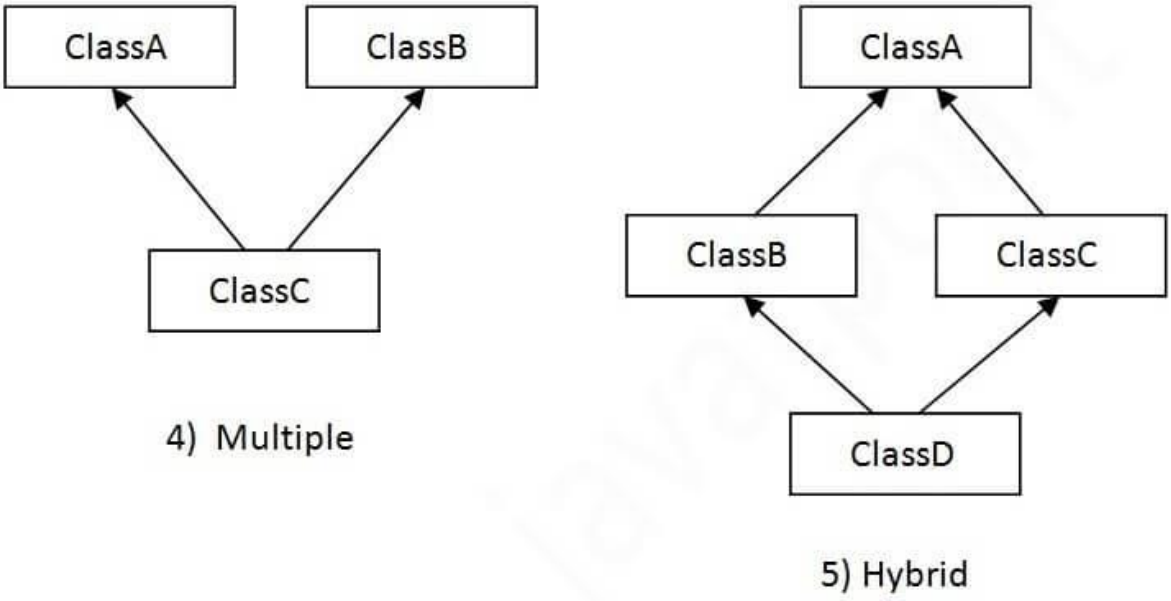
==========

Hybrid Inheritance:-

Hybrid Inheritance is a combination of Inheritances. Since in Java Multiple Inheritance is not supported directly we can achieve Hybrid inheritance also through Interfaces only.



Note: Multiple inheritance is not supported in Java through class.



What is the difference between an object-oriented programming language and object-based programming language?

Object-based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object-based programming languages.

Polymorphism

==========

Polymorphism in Java is a concept by which we can perform a single action in different way.

Polymorphism is derived form 2 Greek works: Ploy and morphs

- "Ploy" means many
- "morphs" mean forms

One object shows different behaviour at different stages of life cycle as called as "Polymorphism"

There are two types of polymorphism in JAVA

1. Compile-Time polymorphism
2. Run-Time polymorphism

==========

1. Compile Time Polymorphism: -

   In compile time polymorphism method declaration is going to get bonded to it's definition at compile time based on arguments so that we called it as *"Compile time polymorphism"*

   - At compile-time, Java knows which method to call by checking method signature so this is called compile time polymorphism.
   - Also called as *Static binding or Early binding*.
   - As binding takes place during compilation time so it is known as "Early binding"

   Method overloading is An example of compile time polymorphism.

```
package Ex1poly;

public class Example1MethodOverload {

	public void test() //1
	{
		System.out.println("This is test method with no arg");
	}
	public void test(String a) //2
	{
		System.out.println("This is test method with String arg");
	}
	public void test(String b, int c) //3
	{
		System.out.println("This is test method with no arg");
	}
```

```
        public void test(int d, String e) //4
        {
                System.out.println("This is test method with no arg");
        }

        public static void main(String args [])
        {
                Example1MethodOverload obj=new Example1MethodOverload();
                obj.test("hs", 10);
        }
}
```

==========

2. Run time polymorphism

In Run time polymorphism method declaration is going to get bonded to its definition during run time or execution time based on object creation so that we called it as *"runtime polymorphism"*.

- Runtime polymorphism also called as "Dynamic biding" or late binding
- Method overriding is an example of run-time polymorphism

```
package Ex1poly;
public class Test1 {
    public void test(int a)
    {
            System.out.println("Test method from test1");
    }
    public void display()
    {
            System.out.println("display method from test1");
    }
}
```

```
package Ex1poly;
public class Test2 extends Test1 {
public void test(int b)
{
System.out.println("Test method from test2");
}
public static void main(String args[])
{
Test2 t2=new Test2();
t2.test(10);
```

```
    t2.display();

    Test1 t1=new Test1();
    t1.test(20);
    }
    }
Test method from test2
display method from test1
Test method from test1
```

Difference between method overloading and method overriding in java

No.	***Method Overloading***	***Method Overriding***
1	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Casting / Conversions in Java

Converting one type of information into another type of information is called casting.

There are two types of casting

1. Primitive Casting
2. Non-primitive Casting

Primitive Casting

- Process that converts a data type into another data type in both ways manually and automatically called as type casting.
- Converting one datatype information into another datatype information is known as Primitive Casting.
- Primitive casting is also called as conversion casting or type casting

There are three subtypes of primitive casting

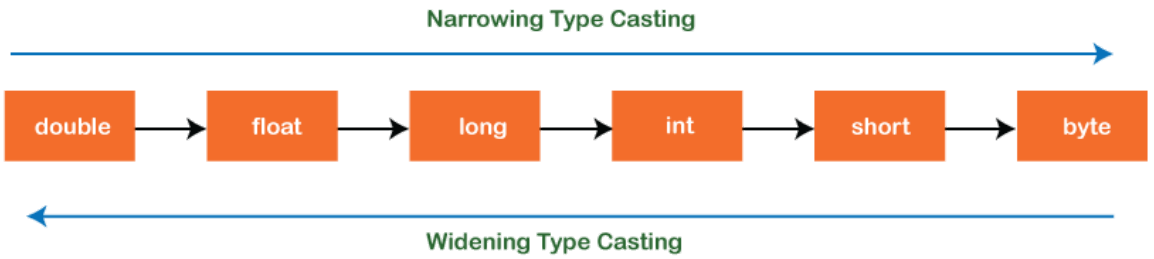
- Implicit Casting/Widening Conversion
- Explicit casting/ Narrowing Conversion
- Boolean Casting

1. implicit Casting/Widening Conversion

- Converting lower datatype information into higher datatype information is called as Implicit Casting.
- It is also called as widening casting or automatic casting/promotion
- It is done automatically. It is safe because there is no chance to lose data.
- It takes place when:
  - Both data types must be compatible with each other.
  - The target type must be larger than the source type.
- Implicit Casting format –
  - **byte** -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**
- That's why we called as widening casting where memory size goes on increasing

2. **Explicit casting/ manual casting / promotion**

- Converting a higher data type into a lower one is called Explicit casting.
- It is also called as narrowing casting/ manual casting
- In explicit casting data loss may be takes place
- It is done manually by the programmer.
- If we do not perform casting then the compiler reports a compile-time error.
- Here largest type specifies the desired type to convert the specified value to .
  - **double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**
- Note -
  - Narrowing casting must be done manually by placing the type in () in front of the value



Type Casting in Java

-
  - The automatic conversion is done by the compiler and manual conversion performed by the programmer.

3. **Boolean casting**

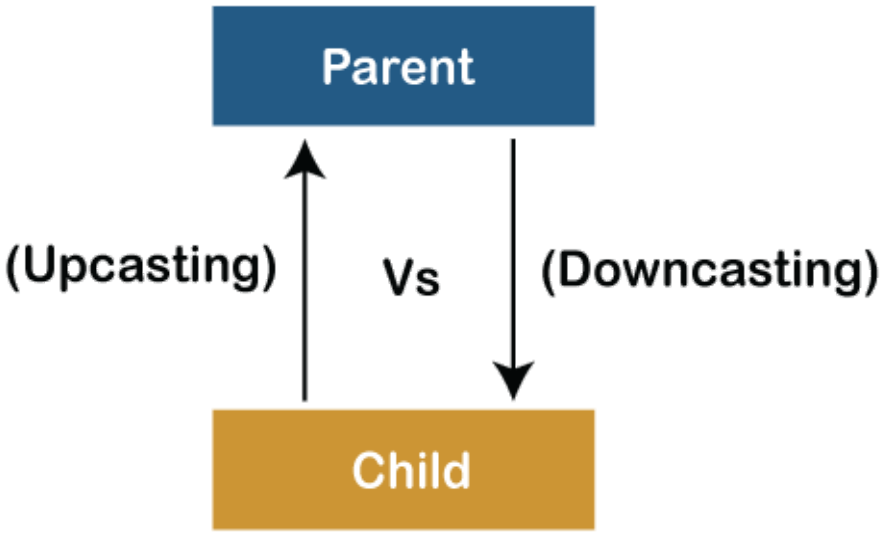
- It consider to be an incompatible casting type
- Boolean returns true or false so for converting true into false and false into true it is not possible
- Java doesn't support boolean casting so that's why it considers as incompatible type casting
- it is considered to be as incompatible casting means we cannot convert true into false it is not possible in java

==========

Non-Primitive Casting / Object Typecasting

Non-primitive casting means converting one type of class into another type of class is known as Non- primitive casting

- Up casting
- Down Casting



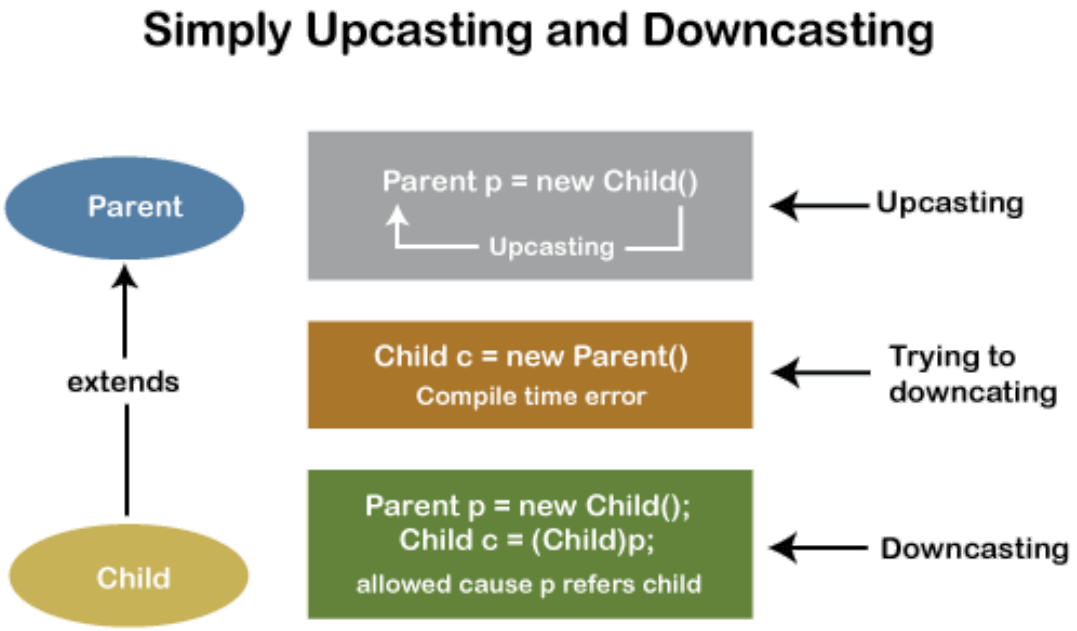
-

Up casting

- Assigning subclass property into superclass is known as Upcasting
- before performing upcasting operation inheritance operation take place in it.
- After performing inheritance properties which are present in super class that comes into subclass
- In subclass we have to declare new properties as well
- AT the time of upcasting the property which are inherited from super class are only eligible for operation
- The new property which where declare inside subclass are not eligible for upcasting operation
- Parent p = new child();

Down Casting

- Java doesn't support down casting/sampling
- before perform down casting perform upcasting operation first then perform down casting
- Down casting means typecasting of parent object to child object
- Down casting cannot be implicitly or automatic
- In this we can forcefully cast child to parent which is known as down casting Child c= (parent)p;
- Down casting has to be done externally and due to down casting a child object can acquire the properties of parent object



Access Modifiers in Java

==========

There are two types of modifiers in Java:

- **Access modifiers**
- **Non-Access modifiers**.

The Access Modifiers: -

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

==========

There are four types of Java access modifiers:

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

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

There are many non-access modifiers/keywords such as static, abstract, synchronized, native, volatile, transient, etc.

==========

1. Private

- The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

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

```
public 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
          }
 }
```

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

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

2. Default

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

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

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

==========

3. Protected

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
- It provides more accessibility than the default modifier.

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

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

==========

4. Public

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

```
//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();
          }
 }
```

==========

Java Access Modifiers with Method Overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

```
  class A{
        protected void msg()
       {
             System.out.println("Hello java");}
       }
public class Simple extends A{
       void msg()
       {
            System.out.println("Hello java");
       }//C.T.Error
      public static void main(String args[])
      {
            Simple obj=new Simple();
            obj.msg();
      }
}
```

The default modifier is more restrictive than protected. That is why, there is a compile-time error.

==========

Abstraction in Java

==========

- **Abstraction** is a process of hiding the implementation details/code/information and showing only functionality to the end user.
- Hiding internal functionality and showing external functionality to users.
- Another way, it shows only essential things to the user and hides the internal details

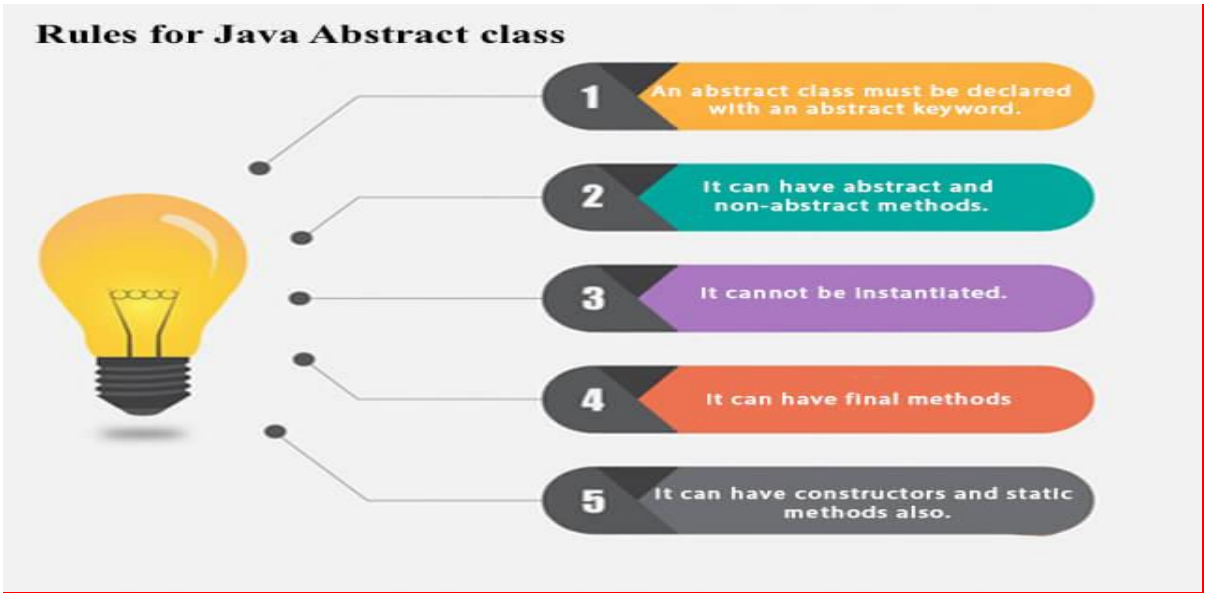
  Example > Sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

- A class which is declared with an abstract keyword is known as an **abstract class**.
- Abstract class has abstract and non-abstract methods (i.e. incomplete and complete methods).
- Abstract Method - Method declaration is present but don't have method body(implementation).
- It needs to be extended and its method implemented. It cannot be instantiated.



Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Abstract Method in Java

- A method which is declared with an abstract keyword and does not have implementation is known as an abstract method.
- A method does not have a method body.

Example of abstract method

```
abstract void printStatus(); //no method body and abstract
```

Concrete Class

- The class which is responsible to implement all the abstract methods from the abstract class.
- We can't create object of abstract class to create object of abstract class their is approach called as "concrete class".
- An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

Imp

- If there is an abstract method in a class, that class must be abstract.
- If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.

==========

Interface in Java

==========

- The interface in Java is *a mechanism to achieve abstraction*
- It is used to achieve abstraction and multiple inheritance in Java.
- An **interface in Java** is a blueprint of a class.
- There will be only abstract methods in the Java interface, not method body.
- It cannot be instantiated just like the abstract class.
- When an interface inherits another interface **extends** keyword is used whereas class use **implements** keyword to inherit an interface.

Why use the Java interface?

There are mainly three reasons to use interfaces. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling

Feature of Interface/Point to remember

- Data member declared Inside interface by default static and public
- methods declared inside interface are default public and abstract
- Constructor concept is not present inside interface
- Object of interface can not be created
- Interface supports multiple inheritance

- Since Java 8, we can have **default and static methods** in an interface.
- Since Java 9, we can have **private methods** in an interface.

How to declare an interface?

- An interface is declared by using the interface keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body and **public and abstract by default,** and all the fields/variables are **public, static and final by default.**

The Java compiler adds **public and abstract** keywords before the interface method. And it adds **public, static and final** keywords before data members.

Syntax:

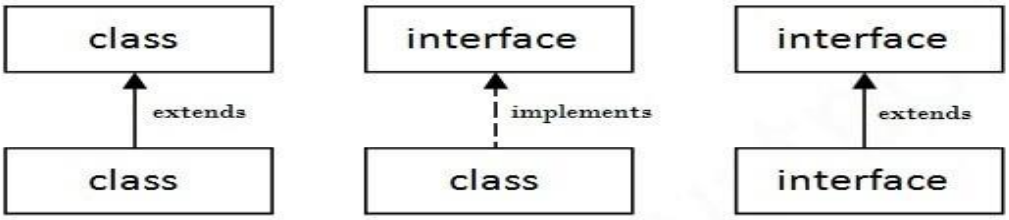
```
interface <interface_name>{
    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

Implementation Class :-

- A class which provides implementation for all the incomplete methods of interface with the help of implement keyword is known as Implementation class
- At the time of Implementation declared method with public access modifiers compulsory

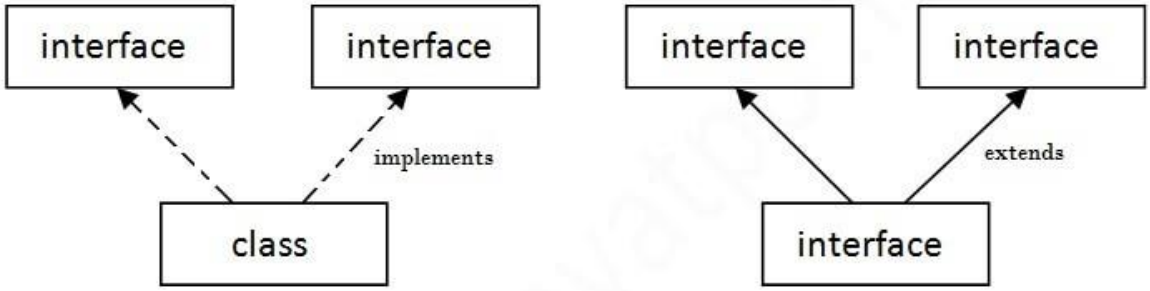
The relationship between classes and interfaces

A class extends another class, an interface extends another interface, but a **class implements an interface**.



Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

Q. Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

As we have explained in the inheritance, multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

Q. What is marker or tagged interface?

An interface which has no member (empty interface) is known as a marker or tagged interface,

==========

```java
package abstractionConcept;

public abstract class Vehicle {

	public abstract void wheels();	//33.3

	public abstract void run();

	public void testDrive()
	{
		System.out.println("Test Drive");
	}
}
```

```java
package abstractionConcept;

public class Bike extends Vehicle {

	public void wheels() {

		System.out.println("2 wheels");
	}

	@Override
	public void run()
	{
		System.out.println("Dur durrrrrrrrrrr");
	}
}
```

```java
package abstractionConcept;

public class MasterClass {

	public static void main(String[] args) {

		Bike b = new Bike();
		b.wheels();
		b.run();
		b.testDrive();

//		Vehicle v= new Vehicle();
	}
}
```

```java
package interfaceEx;

public interface Vehicle {

	int a=10;
	void test();

	void start();

	void stop();

	default void test1()
	{

	}
	public static void test2()
	{

	}
	private void test3()
	{

	}
}
```

```java
package interfaceEx;

public class Car implements Vehicle{

	@Override
	public void test() {
		System.out.println("Test method");
	}
	@Override
	public void start() {
		System.out.println("Start's with keys");
	}
	@Override
	public void stop() {
		System.out.println("Stop's with brake");
	}
}
```

```java
package interfaceEx;

public class Master {
```

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

                Car c= new Car();
                c.test();
                c.start();
                c.stop();
                c.test1();
                Vehicle.test2();
        }
}
```

Encapsulation

- The process of wrapping the data and corresponding methods into a single unit is called Encapsulation.
- Examples are Medicine capsule , School Bag.

- If any component follows data hiding and abstraction then it is called encapsulation.
  Encapsulation = Data Hiding + Abstraction

Steps to achieve encapsulation

1. We have to declare variables of class as private. (By declaring variable as private we have achieved data hiding means outside person cannot access this variable).
2. We have to use public getters and setters method to modify and view the variable values.

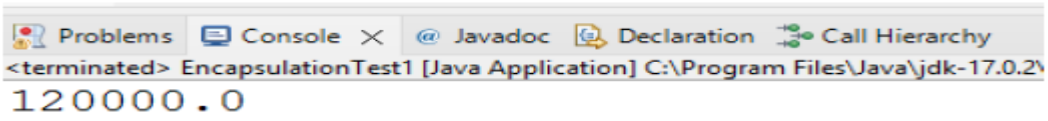
Few points about encapsulation

- User would never know of what is going on in the background, they would be only aware of update the field by set method and read a field by get method.
- Set and get words used in the method names are only for naming purpose so that programmer should understand which is set and get method.

- **Advantages are :-**
    1. Encapsulated class is easy to test so it is better for doing unit testing.
    2. It provides security.

```
package encapsulationPkg;

public class EncapsulationTest1 {
    private double balance; //GlobalorInstance

    //Setter
    public void setBalance(double balance) //Local
    {
        this.balance = balance;
    }

    //getter
    public double getBalance()
    {
        return balance;
    }

    public static void main(String[] args) {
        EncapsulationTest1 et = new EncapsulationTest1();
        et.setBalance(120000);
        et.getBalance();

        double z = et.getBalance();
        System.out.println(z);
    }
}
```

Output:



Super Keyword

"**super**" keyword is used to access global variable from super class or different class

Example:

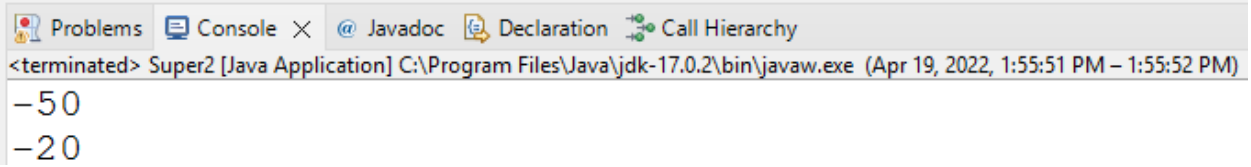
Super Class:

```
package superKeywordPckg;

public class Super1 {

    int a = 10;
    int b = 30;

    public void test()
    {
    }
}
```

Sub Class:

```
package superKeywordPckg;

public class Super2 extends Super1{
    int a = 50;
    int b = 100;

    public void test2()
    {
        System.out.println(a-b);
        System.out.println(super.a - super.b);
    }

    public static void main(String[] args) {
        Super2 s = new Super2();
        s.test2();
    }
}
```

Output:



Java Keywords

==========

Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

List of Java Keywords

1. **abstract:** Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. **boolean:** Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break:** Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. **byte:** Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. **case:** Java case keyword is used with the switch statements to mark blocks of text.
6. **catch:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. **char:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class:** Java class keyword is used to declare a class.
9. **continue:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. **default:** Java default keyword is used to specify the default block of code in a switch statement.
11. **do:** Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. **double:** Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. **else:** Java else keyword is used to indicate the alternative branches in an if statement.
14. **enum:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.

15. **extends:** Java extends keyword is used to indicate that a class is derived from another class or interface.
16. **final:** Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. **finally:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. **float:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. **for:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use a for loop.
20. **if:** Java if keyword tests the condition. It executes the if block if the condition is true.
21. **implements:** Java implements keyword is used to implement an interface.
22. **import:** Java import keyword makes classes and interfaces available and accessible to the current source code.
23. **instanceof:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. **int:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. **interface:** Java interface keyword is used to declare an interface. It can have only abstract methods.
26. **long:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. **new:** Java new keyword is used to create new objects.
29. **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. **package:** Java package keyword is used to declare a Java package that includes the classes.
31. **private:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.

32. **protected:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. **public:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. **return:** Java return keyword is used to return from a method when its execution is complete.
35. **short:** Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. **static:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. **strictfp:** Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. **super:** Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. **switch:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. **synchronized:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. **this:** Java this keyword can be used to refer the current object in a method or constructor.
42. **throw:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. **throws:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. **transient:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. **try:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. **void:** Java void keyword is used to specify that a method does not have a return value.
47. **volatile:** Java volatile keyword is used to indicate that a variable may change asynchronously.

48. **while:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

==========