

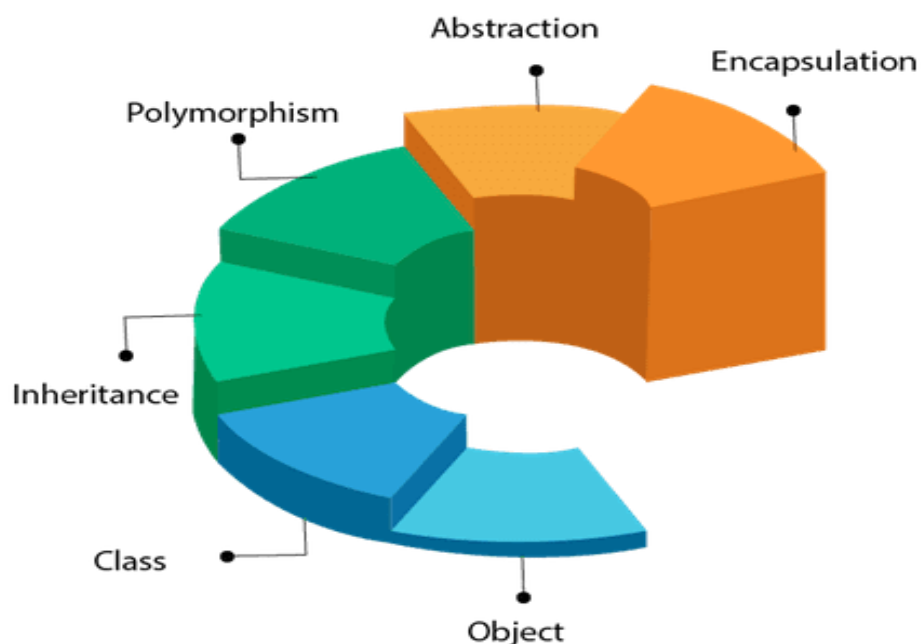
OOPs (Object-Oriented Programming System)

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)



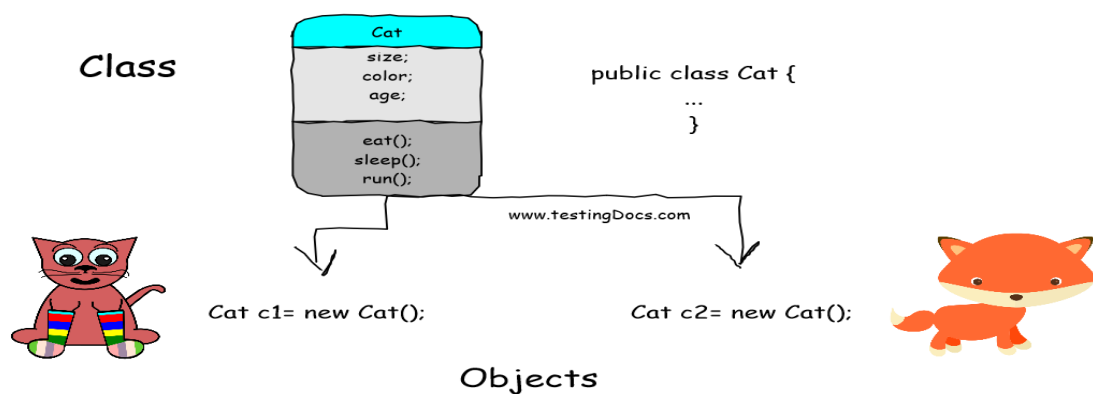
Object: -

Any entity which have state and behaviour is known as object. For example a car, book, clock, laptop, bike, etc.



- ❖ **State:** represents the data (value) of an object.
- ❖ **Behaviour:** represents the behaviour (functionality) of an object such as deposit, withdrawal, etc.
- An Object can be defined as an instance of a class, and there can be multiple instances of a class in a program.
- It can be physical or logical entity.
- An object contains an address and takes up some space in memory.

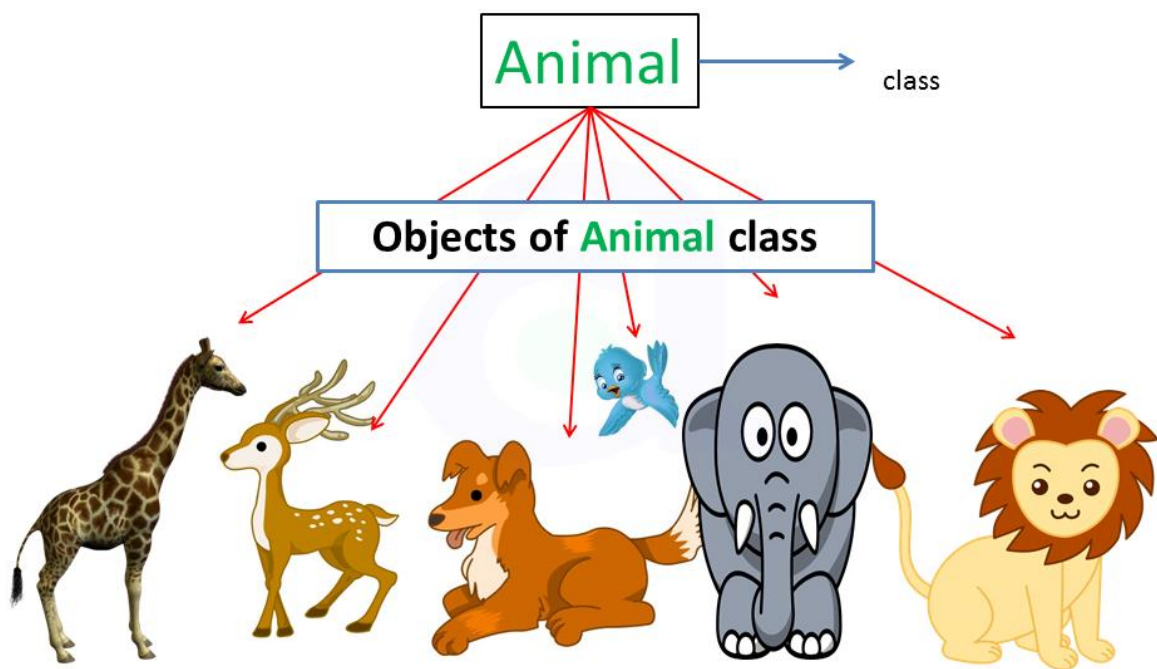
Example: A cat is an object as it has states like color, name etc. as well as behaviors like running, eating etc.



Class: -

- Class is collection of objects, methods, constructors, variables, statements, datatypes etc.
- Class is a blueprint or a set of instructions to build a specific type of object.
- Class in Java determines how an object will behave and what the object will contain.
- It is a logical entity, which doesn't consume any space.
- A class can also be defined as a blueprint or protocol from which objects are created.

Class and Object Relation -



=====

Inheritance

1. 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.
2. Inheritance is one of the most important Object-oriented programming language system.
3. Also, we can say that subclass can acquire the properties of superclass with the help of extends keyword
4. 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

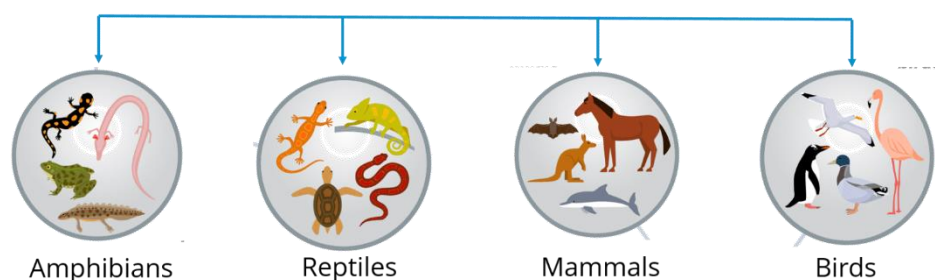
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a *derived class, extended class, or child class*.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a *base class or a parent class*.

Super Class



Animals

Child Class



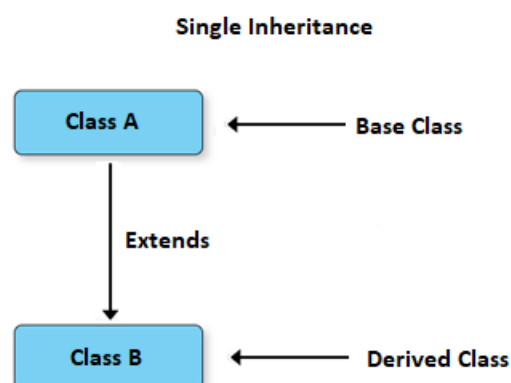
Types of Inheritance

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

=====

Single Level Inheritance

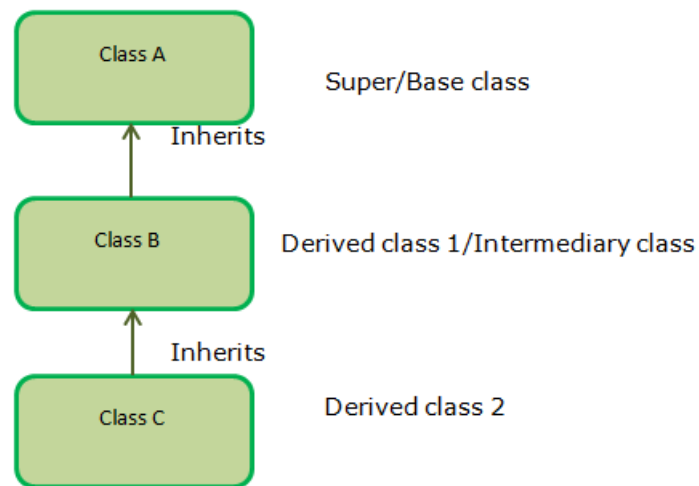
- The process of inheritance in which a class acquires/inherits the properties of another class by using extends keyword is called as Single Level Inheritance.
- It is an operation where inheritance takes place between two classes only to perform Single Level Inheritance



Eg. ClassB inherits classA.

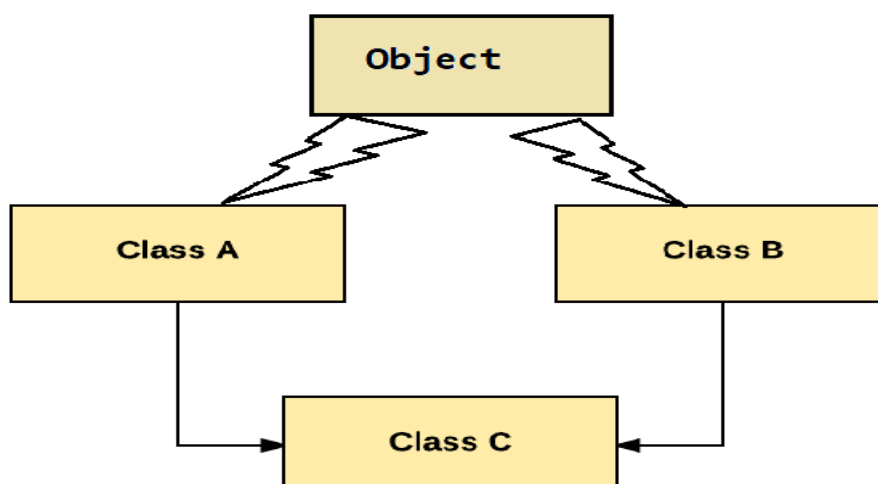
Multi-Level Inheritance

- Multilevel inheritance takes place between three or more classes.
- The process of inheritance in which one subclass acquires the properties of another superclass with the help of extends keyword and this phenomenon continues so that we called as ***"Multi level Inheritance"***
- In short when there is a chain of inheritance it is also known as multilevel inheritance.



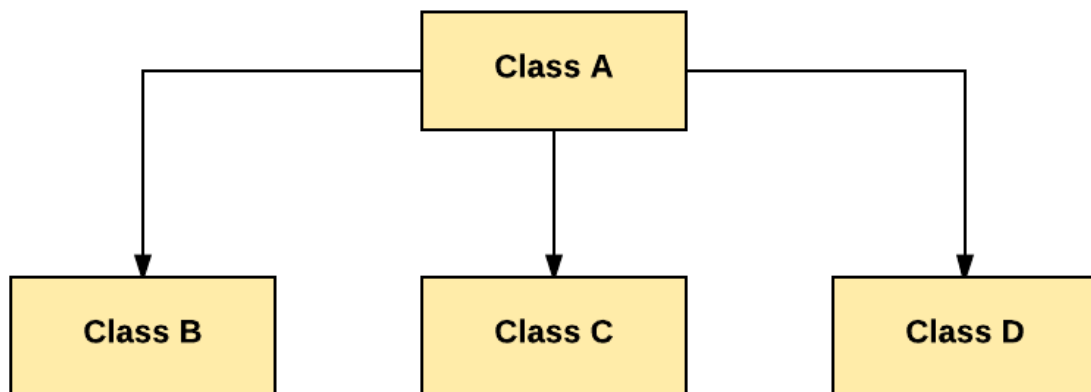
Multiple Inheritance

- The Process of inheritance in which one subclass acquires the properties of two or more super-classes at a same time with the help of extends keyword is called as Multiple Inheritance.
- Java doesn't support multiple inheritance because it results diamond ambiguity problem.
- It means when we try to inherits properties from more than two classes at the same time, the diamond like structure gets created in JVM and object variable gets confused for who need to inherited the property of superclass into subclass.
- Super Class of all the classes is object class so their diamond ambiguity forms
- To reduce the complexity of language multiple inheritance not supported by java, but we can achieve with the help of interface.



Hierarchical Inheritance

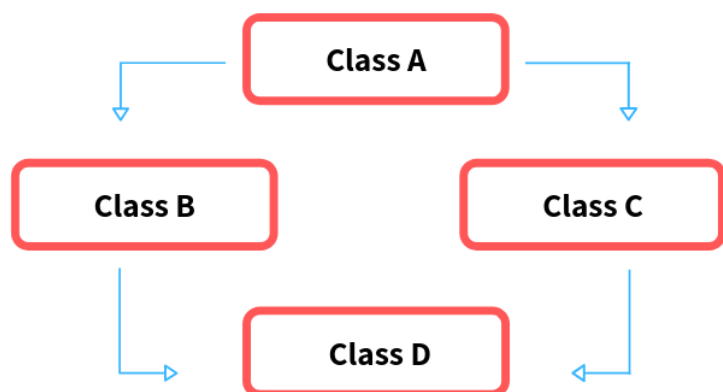
- The process of inheritance in which multiple subclasses acquires the properties one superclass with help of extends keyword is called as Hierarchical Inheritance.
- Hierarchical Inheritance takes place between one superclass and multiple subclasses.
- The two or more classes inherits a single class it is known as Hierarchical Inheritance



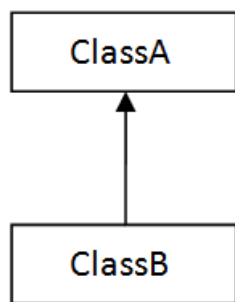
Hybrid Inheritance

- Hybrid Inheritance is a combination of Inheritances. Since in Java Multiple Inheritance is not supported directly so hybrid inheritance is also not support but we can achieve Hybrid inheritance through Interfaces.

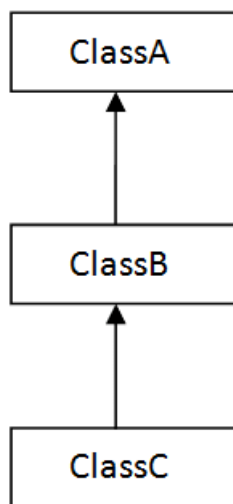
**COMBINATION OF
SINGLE
AND
MULTIPLE
INHERITANCE.**



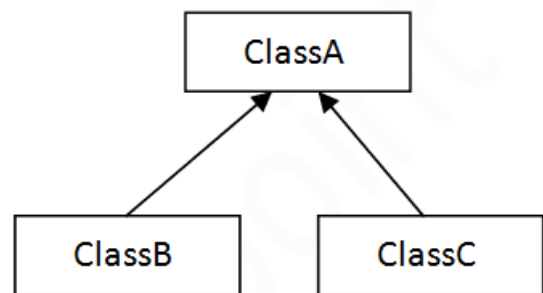
=====



1) Single

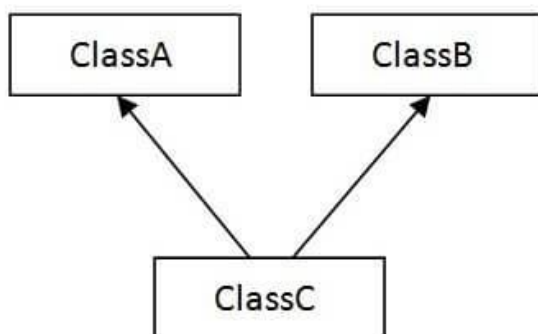


2) Multilevel

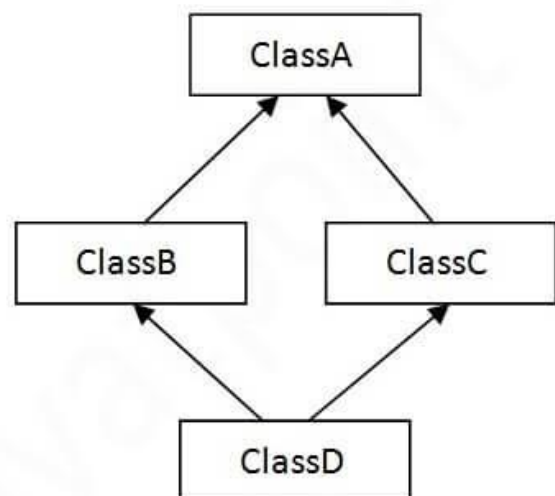


3) Hierarchical

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



4) Multiple



5) Hybrid

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.

Who is the superclass of all the classes?

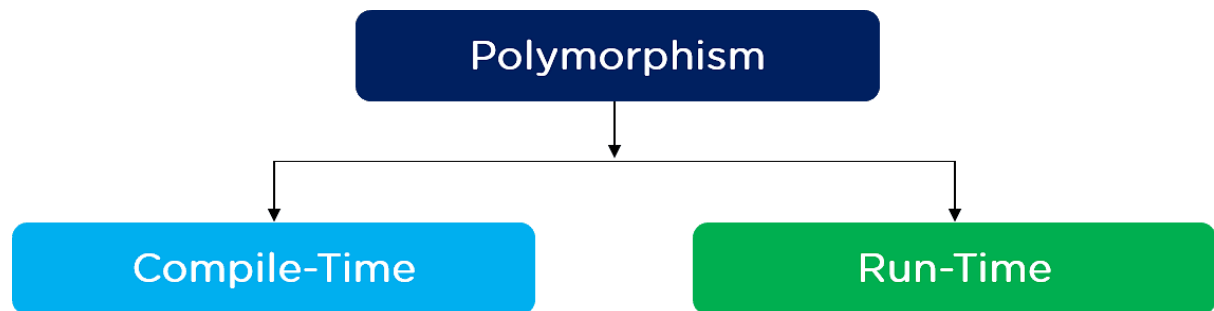
=====

Polymorphism

- Polymorphism means “Many Forms”, is a concept by which we can perform a single action in different way.
- It occurs when we have many classes that are related to each other by inheritance.
- Polymorphism is derived from 2 Greek words: Ploy and morphs which means
 - “Ploy” means many
 - “morphs” mean forms
- One object shows different behaviour at different stages of life cycle as called as “Polymorphism”
- Eg. A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So, the same person possesses different behaviour in different situations. This is called polymorphism.



There are two types of polymorphism in JAVA



Compile Time Polymorphism

- The process of polymorphism in which method declaration is going to get bonded with its own definition during compile time based on arguments is called as *"Compile time polymorphism"*
- At compile-time, Java knows which method to call by checking method signature.
- 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.

Run time polymorphism

- The process of polymorphism in which method declaration is going to get bonded with its own definition during run time or execution time based on object creation is called as *"runtime polymorphism"*.
- Also called as *Dynamic binding or Late binding*.
- As binding takes place during execution time so it is known as *"Late binding"*
- Method overriding is an example of run-time polymorphism.

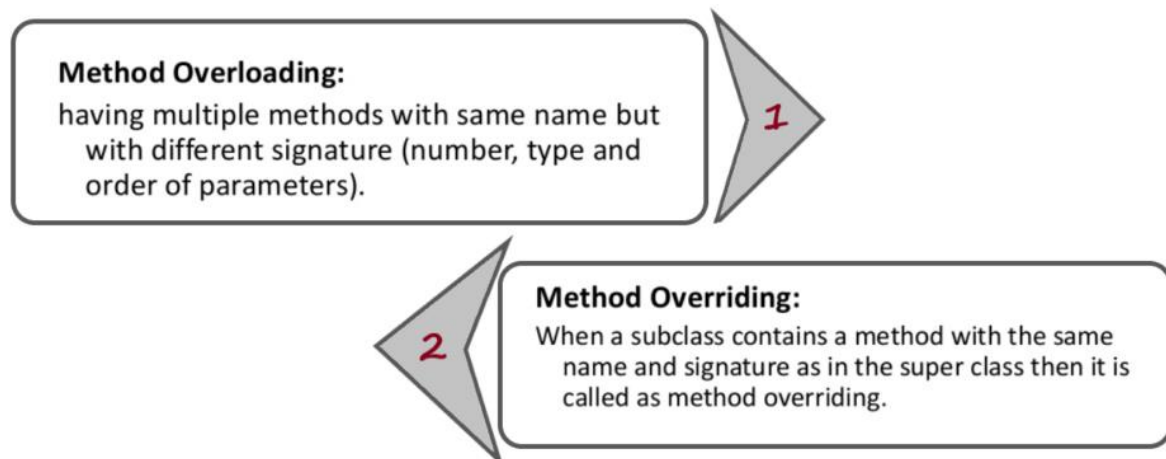
=====

Method Overloading

- In a class we have multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.
- There are two ways to overload the method in java
 - By changing number of arguments
 - By changing the data type

Method Overriding

- If subclass (child class) have the same method as declared in the parent class, it is known as **method overriding**.
- If we have different classes, same method name, same argument having inheritance relationship between those classes.



- Eg. Suppose we have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as `a(int,int)` for two parameters, and `b(int,int,int)` for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs.

Difference between method overloading and method overriding

No.	Method Overloading	Method Overriding
1	It is used to increase the readability of the program.	It is used to provide the specific implementation of the method that is already provided by its super class.
2	It occurred within class.	It occurred in two classes that have IS-A (inheritance) relationship.
3	Parameter must be different.	Parameter must be same.
4	It is the example of compile time polymorphism.	It 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.
6	Private and final methods can be overloaded.	Private and final methods can't be overridden.

Method Overloading

- Compiler
- Same name
- Same Class
- Different arguments
 - Number of Arguments
 - Sequence of Arguments
 - Types of Arguments

Method Overriding

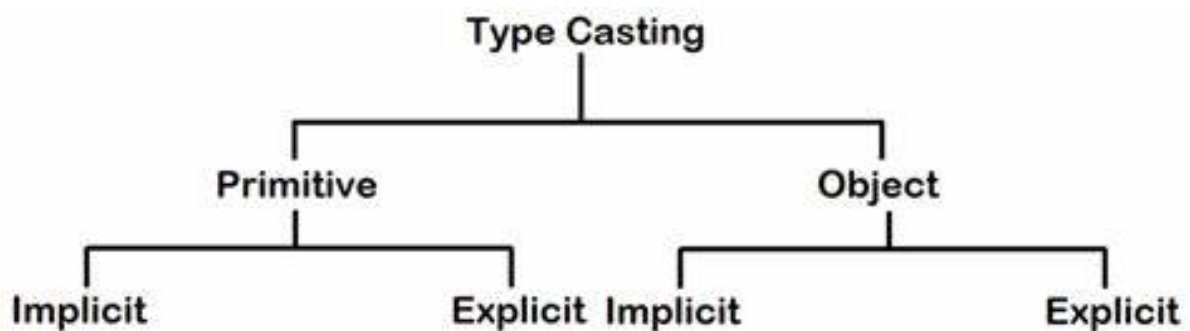
- JVM
- Inheritance (IS-A Relationship)
- Same name
- Different Class
- Same arguments
 - Number of arguments
 - Sequence of Arguments
 - Types of Arguments

=====

Casting / Conversions

- Converting one type of information into another type of information is called as casting.
- It is also called as promotion, conversion or type casting.

There are two types of casting



- ❖ Primitive Casting
- ❖ Object (non-primitive) Casting

Primitive Casting

- The process of converting one datatype information into another datatype information is known as Primitive Casting.

There are three subtypes of primitive casting

1. Implicit Casting/Widening Conversion
2. Explicit casting/ Narrowing Conversion
3. Boolean Casting

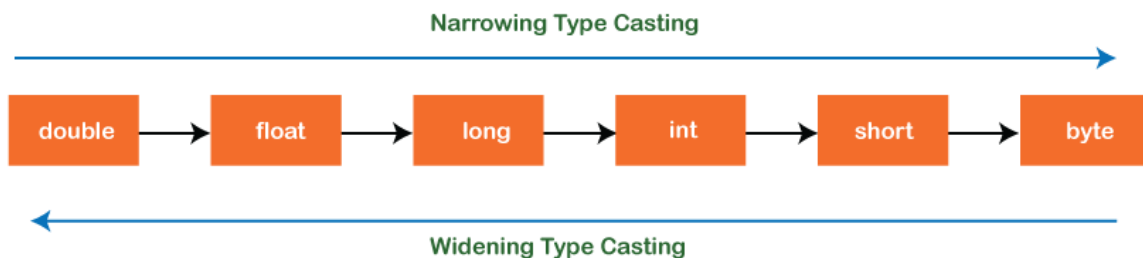
1. implicit Casting/Widening Conversion

- Converting lower datatype information into higher datatype information is known as Implicit Casting.
- It is also called as widening casting or automatic casting.
- 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 higher data type information into a lower datatype information is known Explicit casting.
- It is also called as narrowing casting or manual casting.
- It is done manually by the programmer, and data loss may be takes place
- 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

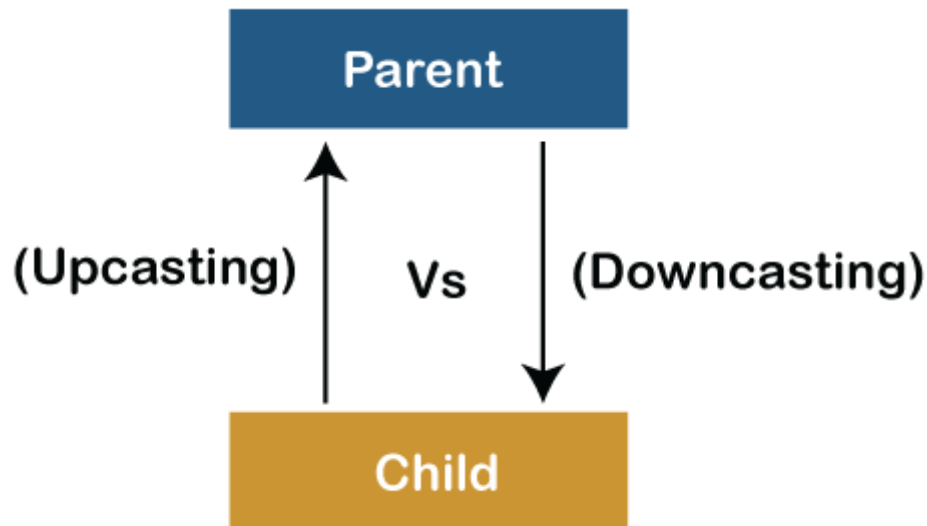
- It considers to be an incompatible casting type
- Boolean returns true or false so for converting true into false and false into true it is not possible so Java doesn't support boolean casting.
- it is considered to be as incompatible casting means we cannot convert true into false it is not possible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

Non-Primitive Casting / Object Typecasting

- The process of converting one type of class into another type of class is known as non-primitive casting.

There are two subtypes of non-primitive casting

1. Up casting
2. Down Casting



1. Up casting

- The process of assigning subclass property into superclass is known as Upcasting
- Upcasting is the typecasting **of a child object to a parent object**.
- Before performing upcasting it is mandatory to have inheritance relationship between classes. After performing inheritance properties which are present in super class that comes into subclass
- Upcasting can be done implicitly.
- Upcasting gives us the flexibility to access the parent class members but it is not possible to access all the child class members using this feature.

Parent p = new child();

2. Down Casting

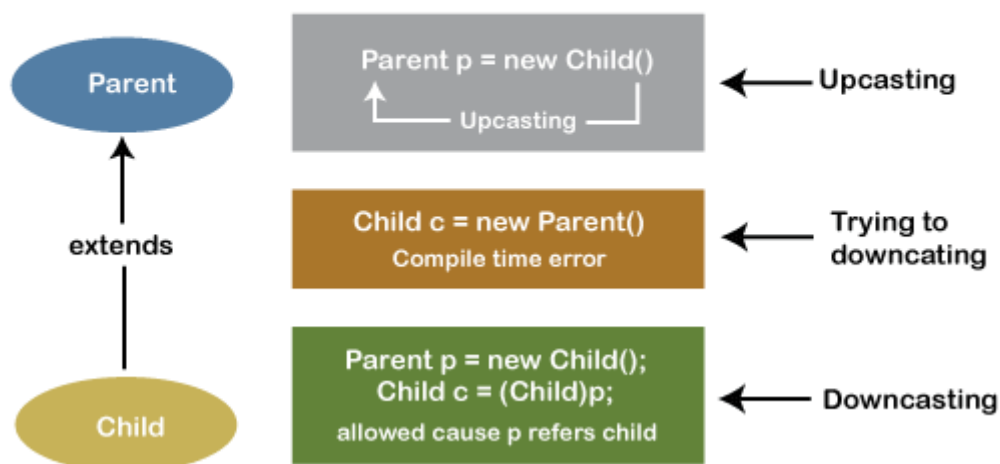
- Downcasting is the typecasting of a **parent object to a child object**.
- Downcasting cannot be implicit.
- before perform down casting perform upcasting operation first then perform down casting
- So, we have to cast child to parent forcefully which is known as down casting

Child c= (parent)p;

- So, Java doesn't support down casting/sampling
- Down casting has to be done externally and due to down casting a child object can acquire the properties of parent object

=====

Simply Upcasting and Downcasting



Example

Let us there is a parent class. There can be many children of a parent. Let's take one of the children into consideration. The child inherits the properties of the parent. Therefore, there is an "is-a" relationship between the child and parent.

Therefore, the child can be implicitly **upcasted** to the parent. However, a parent may or may not inherits the child's properties. However, we can forcefully cast a parent to a child which is known as **downcasting**.

After we define this type of casting explicitly, the compiler checks in the background if this type of casting is possible or not. If it's not possible, the compiler throws a `ClassCastException`.

=====

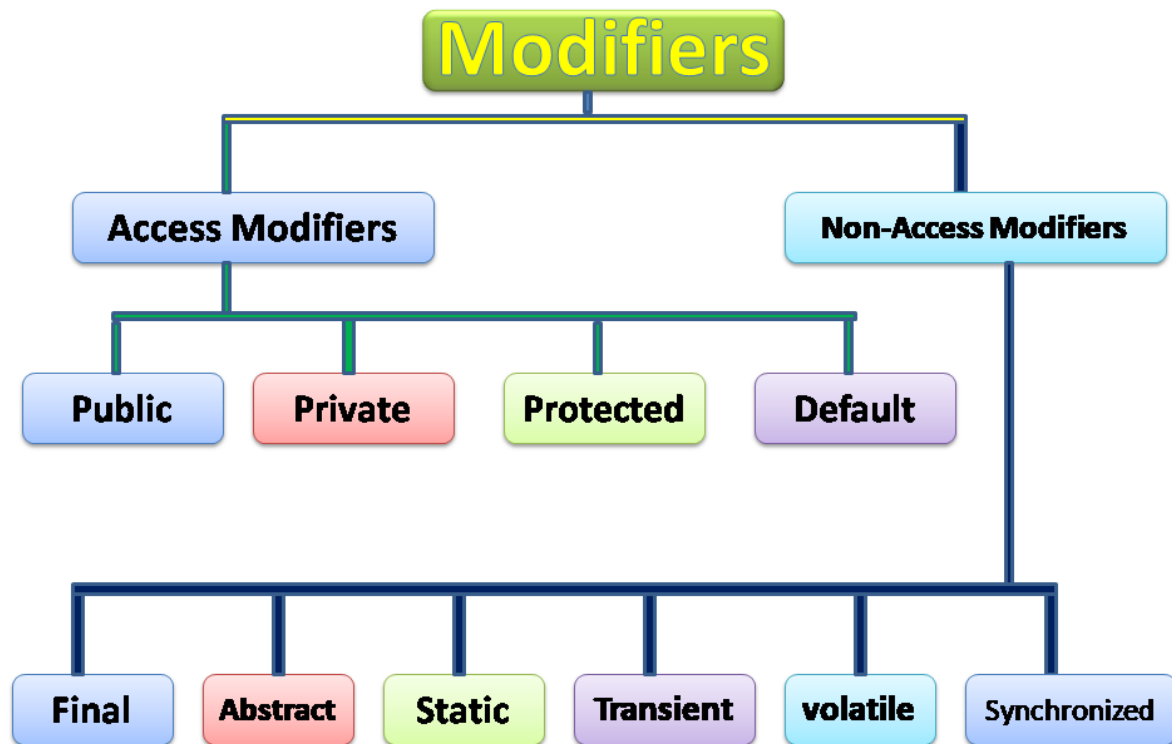
Access Modifiers in Java

Modifiers

- It is used to set the access scope/level for classes, variables, methods and constructors.

There are two types of modifiers in Java:

- Access Modifiers** - controls the access level
- Non-Access Modifiers** - do not control access level, but provides other functionality.



Access Modifiers

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

MODIFIER	ACCESS LEVELS			
	Class	Package	Subclass	Everywhere
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	N	N
private	Y	N	N	N

1. Private

- The access level of a private modifier is only within the class.
- It cannot be accessed from outside the class.
- Private Constructor - If you make any class constructor private, you cannot create the instance of that class from outside the class

2. Default

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

3. Protected

- The access level of a protected modifier is within the package.
- But if we use inheritance and make child class outside the package, it can be accessed from outside.
- 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.

4. Public

- The access level of a public modifier is everywhere.

- It has the widest scope among all other modifiers as it can be accessed from within the class, outside the class, within the package and outside the package.

<i>Declaration Of Access modifiers with each field</i>				
<i>Access Modifiers</i>	<i>default</i>	<i>public</i>	<i>protected</i>	<i>private</i>
<i>Class</i>	<i>Yes</i>	<i>Yes</i>	<i>No</i>	<i>No</i>
<i>Method</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>
<i>Variable</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>
<i>Constructor</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>

=====

Abstraction in Java

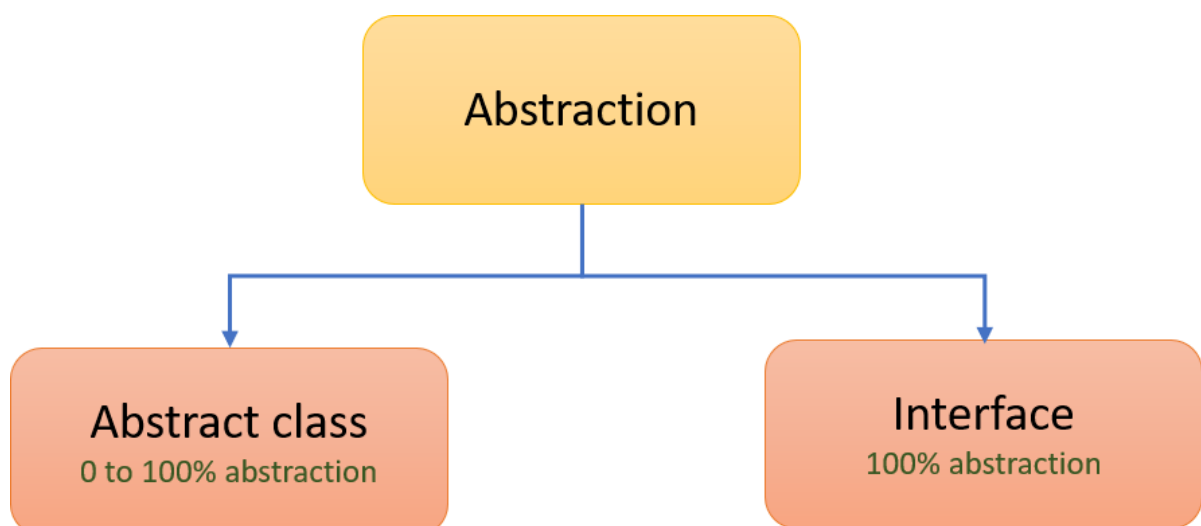
- ❖ **Abstraction** is a process of hiding the implementation details/code/information and showing only functionality to the end user.
- ❖ Hiding internal functionality and showing external functionality to users.
- ❖ Another way, it shows only essential things to the user and hides the internal details

Examples >

- Sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.
- When we derive a car, we do not know **how is the car moving** or **how internal components are working?** But we know **how to derive a car**. It means it is not necessary to know how the car is working, but it is important how to derive a car. The same is an abstraction.



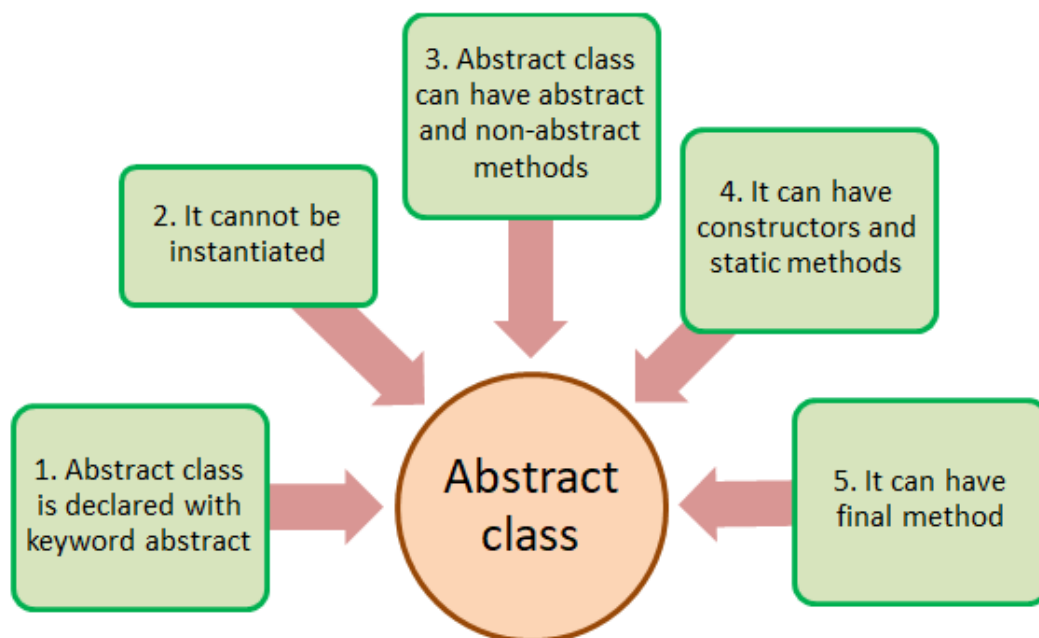
There are two ways to achieve abstraction in java



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.

Important

- 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.

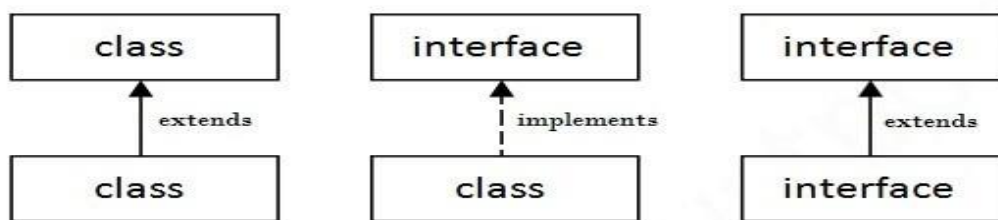
=====

Interface in Java

- Interface in Java is a blueprint of a class, which is used to achieve abstraction and multiple inheritance in Java.
- It consists of only abstract methods in the interface, that is method without 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.

The relationship between classes and interfaces

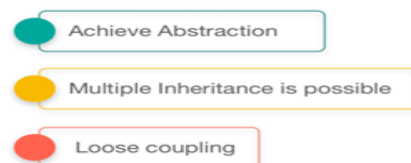
A class extends another class, an interface extends another interface, but a **class implements an interface**.



Why use the Java interface?

There are mainly three reasons to use interfaces:

- It is used to achieve abstraction.
- It is used to achieve multiple inheritance.
- It can be used to achieve loose coupling.



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**.

```
public interface InterfaceName
{
    //only public abstract methods
    //only public static final variables
}
```

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 cannot 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.

Note

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

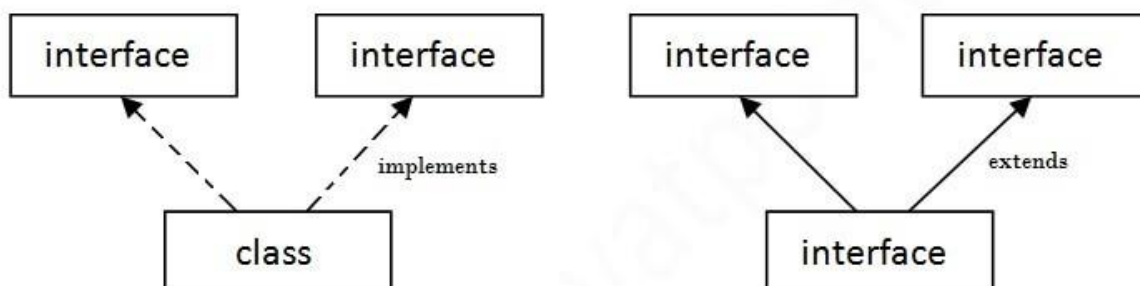
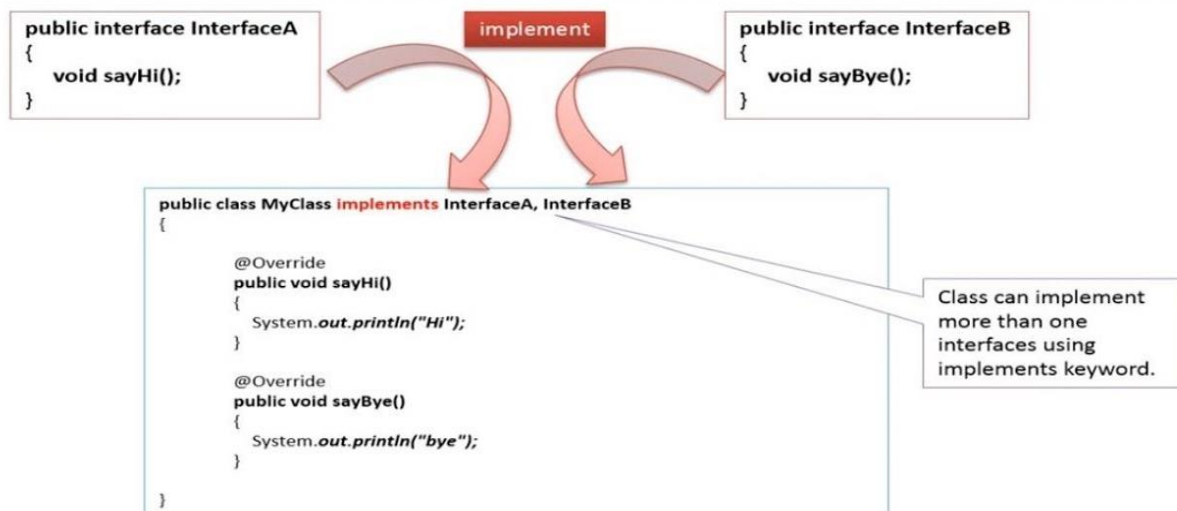
Implementation Class

- A class which provides implementation for all the incomplete methods of interface with the help of implement keyword is known as Implementation class
- To access the interface methods, the interface must be "implemented" (like inherited) by another class with the implements keyword (instead of extends).
- At the time of Implementation declared method with public access modifiers compulsory.

=====

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

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.
- because the class can implement multiple interfaces. Note: To implement multiple interfaces, separate them with a comma.
- 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.

What is marker or tagged interface?

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

What is Functional Interface?

- Functional Interface is an interface that has only pure one abstract method.
- It can have any number of static and default methods and also even public methods of java.lang.Object classes

=====

Interface

I only know method names that I will require for my job to be done. You have to provide body for those methods.

Interface

Implementer

contract



Abstract class

abstract class

Some methods I know. Some methods I don't know and I will depend upon you to provide.

Implementer 2

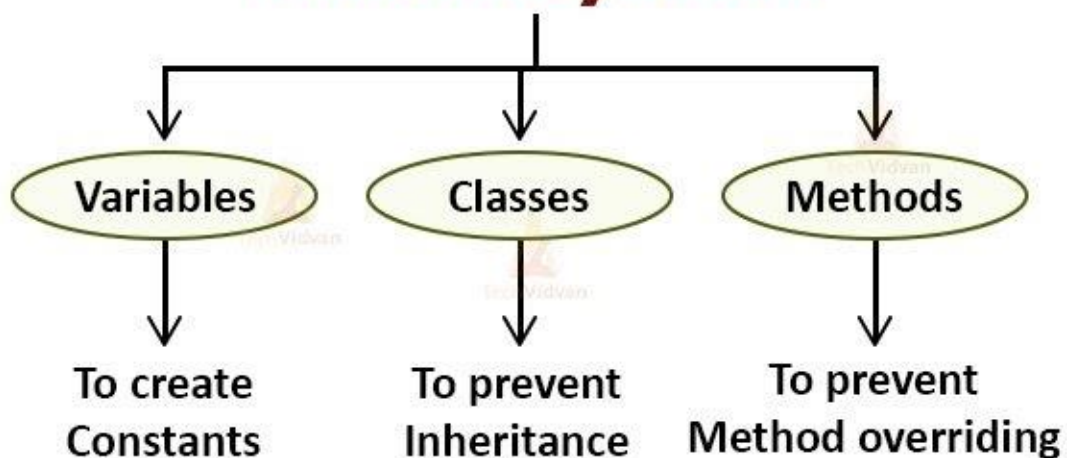


Some methods I know. Some methods I don't know and I will depend upon you to provide.

Implementer 1



Final Keyword



=====

In Real Life...



Abstraction



Login Validation
Process Hidden

Encapsulation



Inheritance



Polymorphism

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.

=====

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

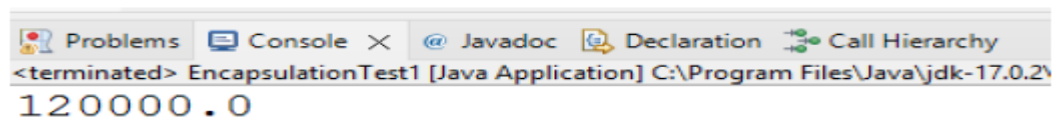
    //Setter
    public void setBalance(double balance) //Local
    {
        this.balance = balance;
    }

    //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:



The screenshot shows an IDE window with tabs for Problems, Console, Javadoc, Declaration, and Call Hierarchy. The Console tab is active, displaying the output of the Java application: 120000.0. The text in the console is: <terminated> EncapsulationTest1 [Java Application] C:\Program Files\Java\jdk-17.0.2\ 120000.0

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:

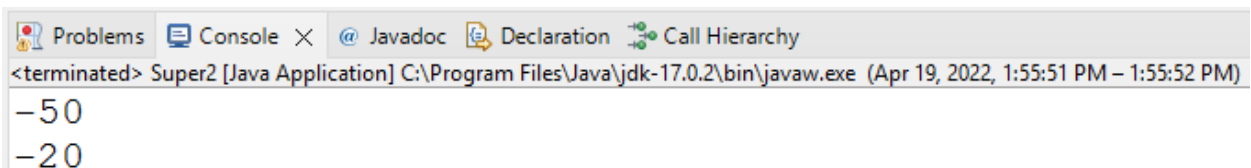
```
package superKeywordPckg;

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:



The screenshot shows an IDE interface with tabs for Problems, Console, Javadoc, Declaration, and Call Hierarchy. The Console tab is active, displaying the output of the Java application. The output consists of two lines: -50 and -20. The console title bar indicates the application is 'Super2 [Java Application]' and the command prompt is 'C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe'.

```
<terminated> Super2 [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe (Apr 19, 2022, 1:55:51 PM – 1:55:52 PM)
-50
-20
```