Classes

```
class P {          // class
    P() {          // constructor
    }
}

class C extends P {
    C() {
        super();   // automatically calls constructor P and
                   memory assign
    }
}
```

⇒ Let's say we have some classes in multilevel inheritance manner
so how are we going to check for the presence of a variable & function

i.e.,

C_2 obj = new C_5();

(for variables)

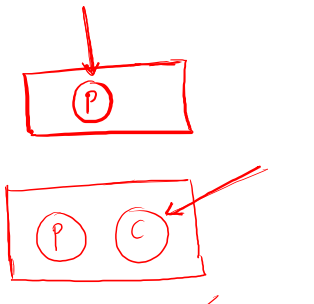
(for function)

Note:-
variable :- check from child class to parent class
function :- check from parent class to child class
(bcz function can be override but not variable)

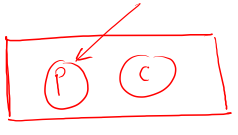
also

P o1 = new P(); // P's reference present inside P's class

C o2 = new C(); // C's reference present inside C's class but also includes P's reference bcz C extends to P



P o3 = new C(); // C's object containing references to both the classes but now we are only pointing to P's reference

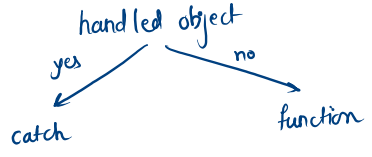


(X) C o4 = new P(); // Wrong (doesn't make any scence)
we cannot point c's reference in P's object bcz P's object can only contain P's reference.

Notes:-
1) Java can achieve 0 to 100% abstraction using abstract class but can achieve 100% abstraction using interfaces
2) Abstract class have 1 extra thing which is abstract method
3) Interface is blueprint of class and class is blueprint of object
Interface —blueprint→ class —blueprint→ object
4) Interface cannot have constructor because we cannot initialised any variable or method in interface.
5) Modifiers :- any keyword that changes property or accessibility or usage (but not value)
↳ access :- public, protected, default, private
↳ non-access :- static, final
6) Getter and setter :- used to get and set the properties

```
Public a {
    private int a = 3;
    public int get() {
        return a;
    }
    public int set(int a) {
        this.a = a;
    }
}
```

7) Parent class of exception class and error class is Throughable class



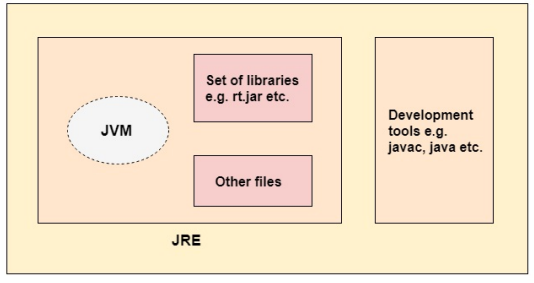
means we check if an object is handled or not, if not we will check for the same object in our parent classes functions

8) Throw is used to make a user defined exception.
Class is used to make a user defined data structure.

Difference b/w JDK, JRE and JVM (these 3 are platform dependent bcz configuration of each OS is different but Java is platform independent)

JVM (Java virtual machine)

1) JVM is an abstract machin and doesn't physically exist
2) It is a specification that provides a runtime environment in which java bytecode can be executed.
3) main notions of JVM : specification, implementation and instance.
4) mains tasks of JVM :-
   - loads code
   - verifies code
   - execute code
   - provide runtime environment



JRE (Java runtime environment)

1) JRE is set of tools used to develope java applications.
2) It is used to provide runtime environment
3) It is implimentation of JVM.
4) It physically exist and it contains some libraries and other files that JVM uses at runtime.

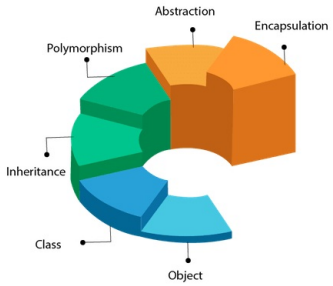
JDK (Java development Kit)

1) It is a software development environment which is used to develop Java applications & applet
2) It physically exist and contain JRE and some tools.

OOPs Concepts (Object Oriented Programming)

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 or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:



Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

Class

classes doesn't consume any space

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.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Abstraction

Hiding data

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding data

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

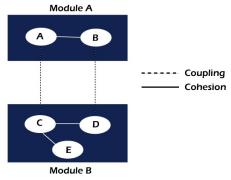
A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Polymorphism

we can use method overloading and method overriding to achieve polymorphism.

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

Some other terms in Java OOPs



↳ Coupling :- dependency of one class on another

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

Tight coupling : In general, Tight coupling means the two classes often change together. In other words, if A knows more than it should about the way in which B was implemented, then A and B are tightly coupled.

```
class Subject {
    Topic t = new Topic();
    public void startReading()
    {
        t.understand();
    }
}
class Topic {
    public void understand()
    {
        System.out.println("Tight coupling concept");
    }
}
```

Loose coupling : In simple words, loose coupling means they are mostly independent.

```
public interface Topic
{
    void understand();
}
class Topic1 implements Topic {
public void understand()
    {
        System.out.println("Got it");
    }
} class Topic2 implements Topic {
public void understand()
    {
        System.out.println("understand");
    }
} public class Subject {
public static void main(String[] args)
    {
        Topic t = new Topic1();
        t.understand();
    }
}
```

↳ Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

Note: *The more focused a class is, the more cohesiveness of that class is more.*

Fig: Low cohesion

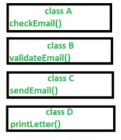


Fig: High cohesion

↳ Association

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

- ↳ one to one
- ↳ one to many
- ↳ many to one
- ↳ many to many

↳ Which is better?

Loose Coupling

Tight coupling:	Loose coupling:
1. More Interdependency	1. Less Interdependency
2. More coordination	2. Less coordination
3. More information flow	3. Less information flow

Example

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

Aggrigation :-

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

Composition :-

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

⇒ Advantages of OOPs

↳ makes development & maintainance much easier (easy to manage code)

↳ Provides data hiding

↳ Provides the ability to stimulate real world events much more effectively.

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.

Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

Note :- "new" keyword is used to allocate memory at runtime.
All objects get memory in heap area.

Note :- a java constructor cannot be static, final, abstract or synchronised.

Note :- There is NO copy constructor in Java.

Note :- Constructor also return some value which is reference of present class

Static

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

Java static block

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

Restriction on static methods

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

This Keyword

- this is a reference variable, that refers to the current object

Usage :- (This can be used to)

- refer current class instance variable
- Invoke current class method
- Invoke current class constructor — Call to this() must be the first statement in constructor.
- can be passed as an argument in method call
- can be passed as an argument in constructor call
- to return the current class instance from the method

⇒ Inheritance

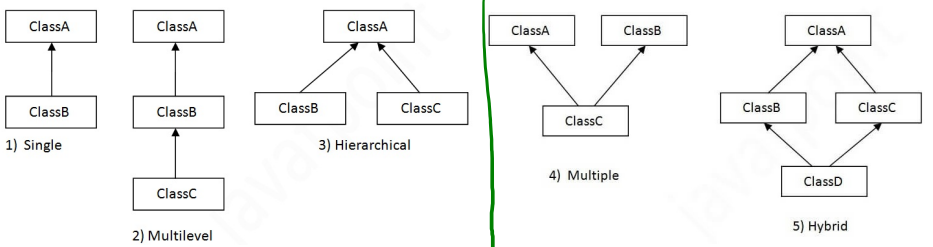
↳ It represents a IS-A relationship (parent-child relationship)

Why
- ↳ Code reusability
- ↳ for method overriding (so runtime polymorphism can be achived)

The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

Types * (multiple inheritance is not supported in java through classes)



not supported

Quel Why multiple inheritance is not allowed

↳ for simplicity

↳

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
  C obj=new C();
  obj.msg();//Now which msg() method would be invoked?
}
}
```

(?)

(we faced ambiguity)

⇒ Aggregation (It represents a HAS-A relationship)

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

⇒ Polymorphism

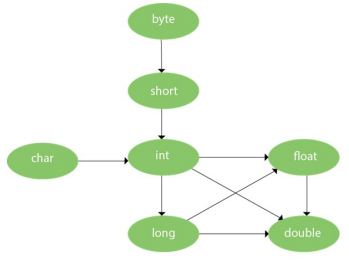
Method Overloading :- If a class have multiple methods having the same name but different parameters, then it is called method overloading

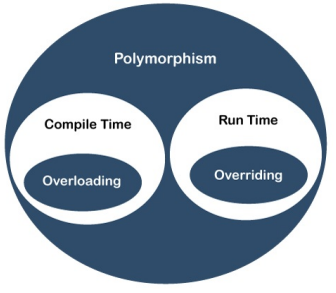
(Compile time Polymorphism)

Ways to Overload a method

- by changing no. of arguments
- by changing the data type

Method overloading and type promotion





Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading.
But JVM calls only that main which have array of string as parameter.

Method Overriding :- If a child class has a same method as declared in the parent class, then it is overriding

(Runtime Polymorphism)

Usage :-

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Note :- Static method is bound to class
instance method is bound to object

Note :- We cannot override a static method that's why we can't override main (bcz it's always static).

Why can we not override static method?

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

⇒ Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

⇒ Final Keyword in Java

It is used to restrict a user.

final can be a
- ↳ variable (cannot change its value)
- ↳ method (cannot override method)
- ↳ class (cannot extends a class)

final with variable

↳ as final can't be updated so we have to initialise it at the time of declaration

but what if we don't want to (now variable can be of 2 type)
- ↳ final int a; ⟶ (initialised in constructor only)
- ↳ final static int a; ⟶ (initialised in static block only)

Q) Can we declare a constructor final?

No, because constructor is never inherited.

⇒ Java Abstraction classes

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

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

Rule: If there is an abstract method in a class, that class must be abstract.

Rule: 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.

- class with abstract keyword
- can have abstract & non-abstract methods
- need to extended and method implemented
- cannot be instantiated
- Can have constructor and static also
- It can have final methods which will force the subclass not to change the body of method

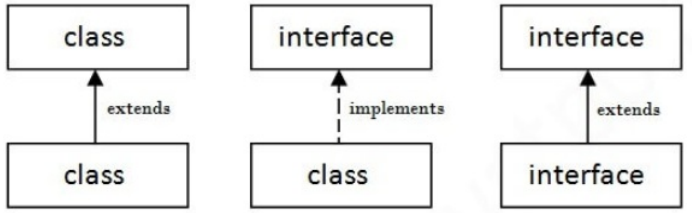
⇒ Interfaces

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also **represents the IS-A relationship**.

⇒ Relation b/w classes & Interface



↳ Usage

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

```
interface <interface_name>{

  // declare constant fields
  // declare methods that abstract
  // by default.
}
```

points :-

↳ declared using interface keyword

↳ all methods are declared in interface are with empty body (means provide total abstraction)

↳ all fields are public, static and final by default

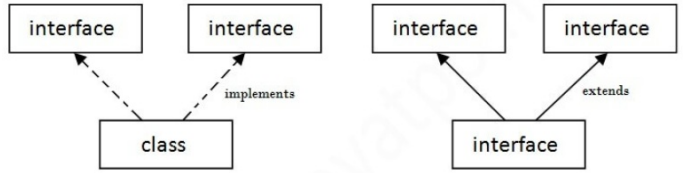
↳ Since Java 8, interface can have default and static methods which is discussed later.

↳ A class that implements an interface must implement all methods that are in the interface.

```
//Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

⇒ Multiple Inheritance in Java

It can be achieved when
↳ a class implements multiple interfaces
↳ a interface extends multiple interface



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

Ans) because there is no ambiguity in interfaces

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

(classes)

```
interface Printable{
void print();
}
interface Showable{
void print();
}

class TestInterface3 implements Printable, Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
 }
}
```

(interfaces)

Note:-
↳ maker or tagger interface is when an interface have no members (used to peform some essential info. to JVM)

```
public interface Serializable{
}
```

↳

```
interface printable{
 void print();
 interface MessagePrintable{
  void msg();
 }
}
```

nested interfaces

⇒ Difference b/w abstract class and Interfaces

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

⇒ Java Encapsulation

↳ **Packages** :- a java package is group of similar type of classes, interfaces and sub-classes (built in and user defined)

advantages:-
- ↳ easy management
- ↳ access protection
- ↳ prevent name collision

↳ **Access modifiers** :- modifies the accessibility and scope of a class, method or constructor.

(access modifiers and non-access modifiers)
↳ static, abstract

Access modifiers	within class	within Package	outside package but subclass only	outside Package
Private	✓	×	×	×
Default	✓	✓	×	×
Protected	✓	✓	✓	×
Public	✓	✓	✓	✓

Note: We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

Advantages:-
- ↳ Provide control over data
- ↳ It is a way to achieve data hiding
- ↳ an encapsulated class is easy to test
- ★ ↳ By providing only getter or setter method, we can make program read only or write only
- ↳ easy and fast to create such class

Some important Questions

1.) What do you understand by OOP? which revolve around object rather than fn & procedure

2.) What is the purpose of using OOPs concepts? It is to implement a real world entity like inheritance, hiding and polymorphism.

3.) Static member functions cannot be used for polymorphism, bcz they cannot overload or override.

4.) when there is a case of using fn overloading or abstract class, then the fn with highest priority is supposed to call first.

5.) Object based languages are those which support classes but not polymorphism.

6.) What is the extra feature in classes which was not in the structures? member functions.

7.) How many types of polymorphism in the C++ programming language? → runtime polymorphism & compile time polymorphism

8.) Why OOP is so popular? helps writing and maintaining code

9.) What are the advantages and disadvantages of OOP?

10.)

Advantages of OOP

- It follows a bottom-up approach.
- It models the real word well.
- It allows as the reusability of code.
- Avoids unnecessary data exposure to the user by using the abstraction.
- OOP forces the designers to have a long and extensive design phase that results in better design and fewer flaws.
- Decomposes a complex problem into smaller chunks.
- Programmer are able to reach their goals faster.
- Minimizes the complexity.
- Easy redesign and extension of code that does not affect the other functionality.

Disadvantages of OOP

- Proper planning is required.
- Program design is tricky.
- Programmer should be well-skilled.
- Classes tend to be overly generalized.

11.) What are the rules for creating a constructor?

It cannot have a return type.
It must have the same name as the Class name.
It cannot be marked as static.
It cannot be marked as abstract.
It cannot be overridden.
It cannot be final.

12.)

Basis of Comparison	Exception	Error
Recoverable/ Irrecoverable	Exception can be recovered by using the try-catch block.	An error cannot be recovered.
Type	It can be classified into two categories i.e. checked and unchecked.	All errors in Java are unchecked.
Occurrence	It occurs at compile time or run time.	It occurs at run time.
Package	It belongs to java.lang.Exception package.	It belongs to java.lang.Error package.
Known or unknown	Only checked exceptions are known to the compiler.	Errors will not be known to the compiler.
Causes	It is mainly caused by the application itself.	It is mostly caused by the environment in which the application is running.
Example	**Checked Exceptions:** SQLException, IOException **Unchecked Exceptions:** ArrayIndexOutOfBoundException, NullPointerException, ArithmaticException	Java.lang.StackOverFlow, Java.lang.OutOfMemoryError

13.)

19) What are the characteristics of an abstract class?

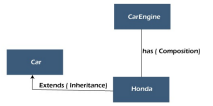
An abstract class is a class that is declared as abstract. It cannot be instantiated and is always used as a base class. The characteristics of an abstract class are as follows:

- Instantiation of an abstract class is not allowed. It must be inherited.
- An abstract class can have both abstract and non-abstract methods.
- An abstract class must have at least one abstract method.
- You must declare at least one abstract method in the abstract class.
- It is always public.
- It is declared using the **abstract**.

The purpose of an abstract class is to provide a common definition of the base class that multiple derived classes can share.

14.) Is it possible for a class to inherit the constructor of its base class?
No

15.) What is composition?
↳ It describes a class that reference one or more objects of the other classes in inheritance variables.



The main benefits of composition are:
- Reuse existing code
- Design clean APIs
- Change the implementation of a class used in a composition without adapting any external clients.

16.) What is the difference between new and override?

The new modifier instructs the compiler to use the new implementation instead of the base class function. Whereas, Override modifier helps to override the base class function.

17.) What are the types of variables in OOP?

↳ Instance variable:
1) It is an object level variable.
2) It should be declared inside a class but outside a method, block and constructor.
3) It is created when an object is created (with 'new' keyword)
4) It is accessed directly by calling the variable name inside the class.

↳ Static variable:-
1) It is a class level variable
2) defined in same area as of instance but with static keyword.
3) It stores in static memory and visibility same as instance variable
4) It can be accessed by calling objname.var_name.

↳ Local variable:-
1) It is a method level variable.
2) It can be declared inside a method, block and constructor.
3) Use of access modifiers are not allowed with these variables.

↳ Reference variable:-
1) It is variable that points to an object of class.

18.) Can we overload the main() method in Java also give an example?
Yes, but method signature must be different

19.) JVM, first execute the static blocks even before looking for the main method.

```
class demo{
    static {
        Syso("static block");
    }
    public static void main(String[] args){
        Syso("main");
    }
}
```