Inheritance
==========
It is one of the pillar of Object Orientation.
It always speaks about reusability.
In java inheritance is achieved through "extends" keyword.

```
eg:: class Parent{
         public void methodOne(){System.out.println("methodOne from
parent");}
     }
     class Child extends Parent{
         public void methodTwo(){System.out.println("methodTwo from
child");}
     }
     public class TestApp{
        public static void main(String... args){
                Parent p=new Parent();
                     p.methodOne();

                Child c=new Child();
                     c.methodOne();
                c.methodTwo();

                Parent p1=new Child();
                p1.methodOne();
                p1.methodTwo();//CE: can't find the symbol methodTwo in
Parent
          }
     }
```

Note:: Whatever is their in the Parent class by default would be available to Child class.
       Whatever is their in the Child class by default won't be available to Parent class.
       On Child reference we can call both Parent class and Child class methods where as
       using Parent reference we can call only Parent class methods.
       Child class Object reference can be holded by Parent class reference whereas Parent class
       Object reference can't be holded by the Child class reference.

eg:: Loan activities
The common methods which are required for HousingLoan,EducationLoan,VehicleLoan is defined in seperate class Loan class and it can be reused so that production cost can be increased effectively and time consumption for building project is less.

```
class Loan{
     //common methods for every type of  loan is written here
}
class VehicleLoan extends Loan{
    //specific methods for VehicleLoan is written here
}
class EducationLoan extends Loan{
   //specific methods for EducationLoan is written here
}
class HomeLoan extends Loan{
   //specific methods for EducationLoan is written here
}
```

Note:: All the common methods which every java class needs is a part of Object class, because Object class is parent class for all the inbuilt class in java.
For all Exception and Error commonly required methods is a parent of "Throwable",hence
Throwable is parent class for all "Exception and Error".

Different types of inheritance supported in java
==========
1. Multiple Inheritance
Having more than one Parent class at same level is called "Mulitple Inheritance".

eg::
```
class Parent1{

}
class Parent2{

}
class Child extends Parent1,Parent2{

}
```
Any java class can extends only one class at a time,it can't extends more than one class at a time
so we say java doesnot support "MultipleInheritance".
In java MulitpleInheritance is supported through "Interface".

Java doesnot support "Multiple Inheritance", becoz it would bring in ambiguity problem
eg#1.
```
class Parent1{
    public void methodOne(){}
}
class Parent2{
    public void methodOne(){}
}
class Child extends Parent1,Parent2{//CE

}
Child c=new Child();
c.methodOne();//Ambiguity problem
```

Note:: If a java class doesnot contain any parent by default compiler will make Object class as
Parent through which inheritance is promoted to every class by default.

2. MultiLevel Inheritance
If our class extends from any other class by default for the extended class Object class acts as parent through which it supports "MultiLevel Inheritance".
eg#1.
```
  class A{}
```

eg#2.
```
  class A{}
  class B extends A{}
```

3. Cyclic inheritance
       It is not allowed in java

eg#1.
```
class A extends B{

}
class B extends A{//CE:cyclic inheritance involving A

}
```

eg#2.
```
 class A extends A{}//CE: cyclic inheritance involving A.
```

Usage of super keyword in java
==========
If parent class properties and child class properties are same, then with in the child class if
we access the properties then compiler/jvm will always give preference for child class properties
only, to make it available from parent class to child class we use "super" keyword.

eg#1.
```
class Parent{
      public int i=10;

}
class Child extends Parent{
      public int i=20;
      public void display(){
            System.out.println("Parent class i is:: "+super.i);
            System.out.println("child class i is :: "+i);
      }
}
class Test {
      public static void main(String[] args) {
            Child c = new Child();
            c.display();
      }
}
```

eg#2.
```
class Parent{
      private int i=10;

}
class Child extends Parent{
      private int i=20;
      public void display(){
            System.out.println("Parent class i is:: "+super.i);//CE:: i is private
            System.out.println("child class i is :: "+i);
      }
}
class Test {
      public static void main(String[] args) {
            Child c = new Child();
            c.display();
```

```
      }
}

eg#3
class Parent{
      private int i=10;
        public void display(){
            System.out.println("Parent class i is ::"+i);
      }
}
class Child extends Parent{
      private int i=20;
      public void display(){
            super.display();
            System.out.println("child class i is :: "+i);
      }
}
class Test {
      public static void main(String[] args) {
            Child c = new Child();
            c.display();
      }
}

Usage of super() in java
========================
 It is basically used to make a call to parent class constructor.
 Internally jvm uses super() to promote constructor chaining at
inheritance level.

eg#1::class Parent{

       }
       class Child extends Parent{

       }

   ==========Compiler generated code ===========
    class Parent extends Object{
        Parent(){super();}
    }
    class Child extends Parent{
          Child(){super();}
    }

eg#2.
  class Parent{
      int i;
      Parent(int i){this.i=i;}
  }
  class Child extends Parent{

  }
    ==========Compiler generated code=========
  class Parent extends Object{
      int i;
        Parent(int i){super();this.i=i;}
  }
  class Child extends Parent{
      Child(){super();}
```

```
  }
Output:: CompileTime Error, Be careful when parent class has
Parameterized constructor in
         child class also we need to have Parameterized constructor.

instance control flow in parent to child relationship
=====================================================
eg#1.
class Parent{
      int x=10;
      {
            methodOne();
            System.out.println("Parent first instance block");
      }
      Parent(){
            System.out.println("Parent class constructor");
      }

      public static void main(String... args){
            Parent p=new Parent();
            System.out.println("Parent class main()");
      }
      public void methodOne(){
            System.out.println(y);
      }
      int y=20;
}
class Child extends Parent{
      int i=100;
      {
            methodTwo();
            System.out.println("Child first instance block");
      }
      Child(){
            System.out.println("Child class constructor");
      }

      public static void main(String... args){
            Child c=new Child();
            System.out.println("Child class main()");
      }
      public void methodTwo(){
            System.out.println(j);
      }
      int j=200;
}
public class Test {
      public static void main(String[] args) {

      }
}

Rule:: Whenever we create child class Object,the follwing things take
place
 a. Identification of instance variables,instance block from parent to
child.
 b. Execution of instance variables assignement,instance block only of
parent class.
 c. Execution of parent class constructor.
```

d. Execution of instance variables assignement,instance block only of child class.
e. Execution of child class constructor.

Output:: java Parent
0
Parent first instance block
Parent class constructor
Parent class main()

Output:: java Child
0
Parent first instance block
Parent class constructor
0
Child first instance block
Child class constructor
Child class main()

Note::
As noticed above we can say Object creation is most costly operation in java, so it is not recomended to create an object if their is no specific requirement.

Static control flow in parent to child class
==========

```
class Parent{
	static int i=10;
	static{
		methodOne();
		System.out.println("Parent class static block");
	}
	public static void methodOne(){
		System.out.println(j);
	}
	public static void main(String... args){
		methodOne();
		System.out.println("Parent class main()");
	}
	static int j=20;
}
class Child extends Parent{
	static int x=100;
	static{
		methodTwo();
		System.out.println("Child class static block");
	}
	public static void methodTwo(){
		System.out.println(y);
	}
	public static void main(String... args){
		methodTwo();
		System.out.println("Child class main()");
	}
	static int y=20;
}
```

Rule:: Whenever we are executing Child class the following sequence of events will be performed
a. Identification of static varaibles and static blocks from top to bottom(parent to child)

b. Execution of static variables and static block from top to bottom(parent to child)
c. Execution of child class main()

Output::
java Parent
0
Parent class static block
20
Parent class main()

Output::
java Child
0
Parent class static block
0
Child class static block
200
Child class main()

Note:: Whenever we are loading child class, automatically parent class will be loaded but when ever we are loading parent class only parent class will loaded not the child class.

Note::
new => creates an object
constructor => initialize the object.
it gets called automatically at the time of creating a object.

Whenever we create an object of child class, will the parent class constructor be called?
Ans. Yes

Whenever we create an object of child class will the object of parent class be called?
Ans. No,constructor of parent class is called just to initialize the properties/fields/members
and to bring those initalized properties to child class.

eg#1.
```
class Parent{
      Parent(){
            System.out.println(this.hashCode());
      }
}
class Child extends Parent{
      Child(){
            System.out.println(this.hashCode());
      }
}
public class TestApp{
      public static void main(String... args){
            Child c= new Child();
            System.out.println(c.hashCode());
      }
}
```

HAS-A Relationship

==========
It is also known as "Composition" or "Aggregation".
There is no specific keyword to implement HAS-A relatioship.
The main advantage of HAS-A relationship is "reusability".

```
eg::
 class Engine{
      //Engine specific functionality
 }
 class Car{
      Engine e=new Engine();
      .......
      .......
      .......
 }
```

In HAS-A relationship, We have 2 things
  a. Dependant Object
  b. Target Object

The main disadvntage of HAS-A relationship is there is always a dependancy b/w components and it
always creates "maintainence problem".

Composition
==========
Without existing Container object,there is no possiblity of existing Contained Object then such
type of relationship is called "Composition".

ex:: University => Target Object
     Department => Dependent Object

Without existing University Object there is no chance of existing Department Object hence these
two objects are tightly related which we say it as "Composition".

Aggregation
==========
Without existing Container object, if there is a possibility of existing Contained Object then
we say relationship is "Aggregation".

ex::Department => Target Object
    Professor => Dependant Object

If we close the department, still professor will be there who would work for other departments, such type of association is called "weak association" or it is also called as "Aggregation".

Note:: In Composition, Container object holds the Contained Object directly,so we say association
       b/w Container object and Contained Object is "strong".
       In Aggregation, Container object holds only the reference of Contained Object, so we say
       association b/w Container object and Container Object is "weak".

Association
==========

1. One to One Association
   eg:: One Student has One Account.
2. One to Many Association
   eg:: One Department has Many Employees.
3. Many to One Association
   eg:: Mulitple Students have joined with Single Branch.
4. Many to Many Association
   eg:: Mulitple Students have taken Mulitple courses.

Relationships in JAVA:
As part of Java application development,we have to use entities as per the application requirements.
In Java application development,if we want to provide optimizations over memory utilization,code Reusability, Execution Time,Sharability then we have to define relationships between entities.

There are three types of relationships between entities.
1.Has-A relationship (extensively used in projects)
2.IS-A relationship
3.USE-A relationship(not popular)

Q)What is the difference between HAS-A Relationship and IS-A relationship?
Ans:
Has-A relationship will define associations between entities in Java applications,here associations between entities will improve communication between entities and data navigation between entities.

IS-A Relationship is able to define inheritance between entity classes, it will improve "Code Reusability" in java applications.

Associations in JAVA
==========
There are four types of associations between entities
1.One-To-One Association
2.One-To-Many Association
3.Many-To-One Association
4.Many-To-Many Association

To achieve associations between entities,we have to declare either single reference or array of reference variables of an entity class in another entity class.

```
class Address{
      ----
}
class Account{
      ----
}
class Employee{
      ----
      Address[] addr;//It will establish One-To-Many Association
      Account account; // One-To-One Association
}
```

1.One-To-One Association:
It is a relation between entities, where one instance of an entity should be mapped with exactly oneinstance of another entity.

EX: Every employee should have exactly one Account.

```
class Account{
      String accNo;
      String accName;
      String accType;

      Account(String accNo,String accName,String accType){
                  this.accNo=accNo;
                  this.accName=accName;
```

```
                this.accType=accType;
        }
}

class Employee{
        String eid;
        String ename;
        String eaddr;

        Account acc;

        Employee(String eid,String ename,String eaddr,Account acc){
                this.eid=eid;
                this.ename=ename;
                this.eaddr=eaddr;
                this.acc=acc;
        }
        public void getEmployee(){
                        System.out.println("Employee Details");
                        System.out.println("-----------------");
                        System.out.println("Employee Id :"+eid);
                        System.out.println("Employee Name :"+ename);
                        System.out.println("Employee Address :"+eaddr);

                        System.out.println();
                        System.out.println("Account Details");
                        System.out.println("----------------");
                        System.out.println("Account Number :"+acc.accNo);
                        System.out.println("Account Name :"+acc.accName);
                        System.out.println("Account Type :"+acc.accType);
        }
}
class OneToOneEx{
                public static void main(String args[]){
                                Account acc=new
Account("abc123","Sachin","Savings");
                                Employee emp=new Employee("E-
111","Sachin","Hyd",acc);
                                emp.getEmployee();
                }
}

OP:
Employee Details
------------------------
Employee Id :E-111
Employee Name:Sachin
Employee Address:MI

Account Details
----------------------
Account Number :abc123
Account Name :Sachin
Account Type :Savings
```

2.One-To-Many Association:
It is a relationship between entity classes, where one instance of an entity should be mapped with multiple instances of another entity.
Example:Single department has multiple employees.

```
class Employee{
	String eid;
	String ename;
	String eaddr;

	Employee(String eid,String ename,String eaddr){
		this.eid=eid;
		this.ename=ename;
		this.eaddr=eaddr;
	}
}
class Department{
	String did;
	String dname;

	Employee[] emps;

	Department(String did,String dname,Employee[] emps){
		this.did=did;
		this.dname=dname;
		this.emps=emps;
	}
	public void getDepartmentDetails(){
		System.out.println("Department Details");
		System.out.println("--------------------");
		System.out.println("Department Id :"+did);
		System.out.println("Department Name:"+dname);
		System.out.println();
		System.out.println("EID ENAME EADDR");
		System.out.println("-----------------");
		for(int i=0;i<emps.length;i++){
			Employee e=emps[i];
			System.out.println(e.eid+" "+e.ename+" "+e.eaddr);
		}
	}
}
class OneToManyEx{
		public static void main(String args[]){
			Employee e1=new Employee("E-111","AAA","Hyd");
			Employee e2=new Employee("E-222","BBB","Hyd");
			Employee e3=new Employee("E-333","CCC","Hyd");
			Employee[] emps=new Employee[3];
				emps[0]=e1;
				emps[1]=e2;
				emps[2]=e3;
			Department dept=new Department("D-111","Admin",emps);
			dept.getDepartmentDetails();
		}
}
```

```
OP:
Department Details
----------------------------
Department Id :D-111
Department Name:Admin

EID ENAME EADDR
---------------------------------
E-111 AAA Hyd
E-222 BBB Hyd
```

E-333 CCC Hyd

Many-To-One Association:
==========
It is a relationship between entities,where multiple instances of an entity should be mapped with exactly one instance of another entity.

EX:Multiple Student have joined with a single branch.

```
class Branch{
	String bid;
	String bname;
	Branch(String bid,String bname){
		this.bid=bid;
		this.bname=bname;
	}
}

class Student{
	String sid;
	String sname;
	String saddr;

	Branch branch;

	Student(String sid,String sname,String saddr,Branch branch){
			this.sid=sid;
			this.sname=sname;
			this.saddr=saddr;
			this.branch=branch;
	}
	public void getStudentDetails(){
			System.out.println("Student Details");
			System.out.println("----------------");
			System.out.println("Student Id :"+sid);
			System.out.println("Student name :"+sname);
			System.out.println("Student Address:"+saddr);
			System.out.println("Branch Id :"+branch.bid);
			System.out.println("Branch Name :"+branch.bname);
			System.out.println();
	}
}
class ManyToOneEx{
	public static void main(String args[]){
			Branch branch=new Branch("B-111","CS");

			Student std1=new Student("S-111","AAA","Hyd",branch);
			Student std2=new Student("S-222","BBB","Hyd",branch);
			Student std3=new Student("S-333","CCC","Hyd",branch);

			std1.getStudentDetails();
			std2.getStudentDetails();
			std3.getStudentDetails();
	}
}
```

OP:

```
Student Details
----------------------
Student Id :S-111
Student name:AAA
```

```
Student Address:Hyd
Branch Id :B-111
Branch Name:CS
Student Details
-----------------------
Student Id :S-222
Student name:BBB
Student Address:Hyd
Branch Id :B-111
Branch Name:CS
Student Details
----------------------
Student Id :S-333
Student name:CCC
Student Address:Hyd
Branch Id :B-111
Branch Name:CS
```

Many-To-Many Associations
==========
It is a relationship between entities,Where multiple instances of an entity should be mapped with multiple instances of another entity.
EX:Multiple Students Have Joined with Multiple Courses

```
EX: class Course{
            String cid;
            String cname;
            int ccost;

            Course(String cid,String cname,int ccost){
                  this.cid=cid;
                  this.cname=cname;
                  this.ccost=ccost;
            }
}
class Student{
      String sid;
      String sname;
      String saddr;

      Course[] crs;

      Student(String sid,String sname,String saddr,Course[] crs){
                  this.sid=sid;
                  this.sname=sname;
                  this.saddr=saddr;
                  this.crs=crs;
      }
      public void getStudentDetails(){
                  System.out.println("Student Details");
                  System.out.println("----------------");
                  System.out.println("Student Id :"+sid);
                  System.out.println("Student name :"+sname);
                  System.out.println("Student Address:"+saddr);
                  System.out.println("CID CNAME CCOST");
                  System.out.println("----------------------");
                  for(int i=0;i<crs.length;i++){
                        Course c=crs[i];
                        System.out.println(c.cid+" "+c.cname+" "+c.ccost);
                  }
```

```
                System.out.println();
            }
}
class ManyToManyEx{
      public static void main(String[] args){
                Course c1=new Course("C-111","C",500);
                Course c2=new Course("C-222","C++",1000);
                Course c3=new Course("C-333","JAVA",5000);
                Course[] crs=new Course[3];
                crs[0]=c1;
                crs[1]=c2;
                crs[2]=c3;
                Student std1=new Student("S-111","AAA","Hyd",crs);
                Student std2=new Student("S-222","BBB","Hyd",crs);
                Student std3=new Student("S-333","CCC","Hyd",crs);
                std1.getStudentDetails();
                std2.getStudentDetails();
                std3.getStudentDetails();
      }
}
OP:
Student Details
----------------------
Student Id :S-111
Student name :AAA
Student Address:Hyd
CID CNAME CCOST
-----------------------------------
C-111 C 500
C-222 C++ 1000
C-333 JAVA 5000
Student Details
------------------------
Student Id :S-222
Student name:BBB
Student Address:Hyd
CID CNAME CCOST
-----------------------------------
C-111 C 500
C-222 C++ 1000
C-333 JAVA 5000
Student Details
------------------------
Student Id :S-333
Student name:CCC
Student Address:Hyd
CID CNAME CCOST
-----------------------------------
C-111 C 500
C-222 C++ 1000
C-333 JAVA 5000
```

In java applications, Associations are existed in the following two form

1. Aggregation
2. Composition

Q)What is the difference between Aggregation and Composition?

Ans:

1.Where Aggregation is representing weak association that is less dependency, but composition is representing strong association that is

more dependency.
2.In case of Aggregation, if contained object is existed even without container object.
In case of Composition ,if contained object is not existed without container object.
3.In the case of Aggregation,the life of the contained Object is independent of the container Object life.
In the case of composition,the life of contained objects is depending on the container object life that is same as container object life.

EX:
If we take an association between
1. "Library" and "Students"
2. "Library" and "Books".

"Students" can exist without "Library" , so that,the association between "Library" and "Student" is Aggregation.
"Books" can not exist without "Library", so that,the association between "Library" and "Books" is Composition.

Dependancy Injection
==========
The process of injecting dependant object into target object is called as "Dependancy injection".

We can achieve dependancy injection in 2 ways
a. Constructor dependancy injection
b. Setter dependancy injection

a. Constructor dependancy Injection
Injecting dependant object into target object through a constructor is called as "Constructor Dependancy Injection".

b. Setter dependancy Injection
Injecting dependant object into target object through a setter is called as "Setter Dependancy Injection".

Note:
Circular Dependancy Injection
==========
Injecting dependant objects in circular fashion is called "Circular Dependancy Injection".
eg: A is depending on B and B is depending on A.

In java applications, we are unable to achieve circular dependancy injection through consturctor dependancy injection,we are able to achieve circulr dependancy injection through setter dependancy injection.

eg:Constructor Dependancy injection
```
class A{
	B b;
	A(B b){

	}
}
class B{
	A a;
	B(A a){

	}
```

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

      }
}

Eg#2.Setter Dependancy Injection
class A{
      B b;
      public void setB(B b){

      }
}
class B{
      A a;
      public void setA(A a){

      }
}
class Test{
      public static void main(String[] args){
            A a =new A();
            B b =new B();
            a.setB(b);
            b.setA(a);
      }
}
```

Method Overriding
==========
Whatever the parent has by default available to child through inheritance,if the child is not satisifed with parent class method implementation then child is allow to redefine that parent class method in child class in its own way this process is called as "Overriding".

The parent class method which is overriden is called "Overridden method".
The child class method which is overriding is called "Overriding method".

```
eg1::
class Parent{
      public void property(){System.out.pritnln("Gold+cash+land");}
        public void marry(){System.out.pritnln("Relative girl only u
should marry!!!");}
}
class Child extends Parent{
       public void property(){System.out.pritnln("Gold+cash+land");}

       @Override
       public void marry(){System.out.println("I will marry
Deepika!!!");}

}
class Test{
      pubilc static void main(String[] args){
            Parent p=new Parent();
            p.marry();//Relative girl only u should marry!!!

            Child c=new Child();
            c.marry();//I will marry Deepika!!!

            Parent p =new Child();
            p.marry();//I will marry Deepika!!!
      }
}
```

In method overriding, method resolution is always based on the runtime object created by JVM, so we say method overriding as "RunTime Polymorphism/Dynamic dispatch/Late Binding".
Here reference type is just like a dummy template.

Rules of Overriding
==========
To apply the rules of Overriding, the method signature should be same only then the methods is participating in Overriding.

Note::
1. In overriding method names and arguments must be same. That is method signature must be same.
2. Until 1.4 version the return types must be same but from 1.5 version onwards covariant return types are allowed.
3. According to this Child class method return type need not be same as Parent class method return type its Child types also allowed.

```
eg#1.
class Parent{
      public Object methodOne(){ return null;}
}
```

```
class Child extends Parent{
      public String methodOne(){return null;}
}
```
It compiles succesfully.

```
case studies::
    Object(P) => String(C)  //valid
    Number(P) => Integer(P) //valid
    String(P) => Object(C) //invalid
   Integer(P) => int(c)    //invalid
```

Rule::Co-Variant type is applicable only on objects not on primitive types.

Scenario2::
Private methods are not visible in the Child classes hence overriding concept is not applicable for private methods. Based on own requirement we can declare the same Parent class private method
in child class also. It is valid but not overriding.

eg#1.
```
class Parent{ private void methodOne(){}}
class Child extends Parent{private void methodOne(){}}
```

2. If the method is declared as final in parent class then those methods we cannot override in
   child class,it would result in compile time error.

```
    eg1:: class Parent{public final void methodOne(){}}
          class Child extends Parent{public void methodOne(){}}//Compile
```
time error

3. Parent class non final methods can be made as final in child class
```
    eg1:: class Parent{public  void methodOne(){}}
          class Child extends Parent{public final void methodOne(){}}
```

4. In the parent class, if the method is abstract then in the child class we should compulsory override to give the implementation
```
       eg1:: abstract class Parent{public abstract void methodOne();}
             class Child extends Parent{public void methodOne(){}}
```

5. We can override NonAbstract method as abstract, this approach is helpful to stop the availablity of parent method implementation to next level child classes.
```
      eg1:: class Parent{public  void methodOne(){}}
            abstract class Child extends Parent{public abstract void
```
methodOne();}

```
Note:: final => Nonfinal (invalid)
       Nonfinal => final (valid)
       abstract => nonabstract(valid)
       native   => nonnative(valid)
       synchronized => nonsychronized(valid)
       strictfp     => nonstrictfp(valid)
```

Rules with respect to scope
==========
While Overriding we cannot reduce the scope of accessmodifier

```
   eg1:: class Parent{public  void methodOne(){}}
         class Child extends Parent{protected void
methodOne(){}}//Compile time error
Note
public => public
protected => protected/public
default   => default/protected/public
private   => Overriding concept is not applicable

Overrding w.r.t static methods
==============================
1. we can't override static method as instance method, it would result in
compile time error.

eg#1.
class Parent{
      public static void methodOne(){ }
}
class Child extends Parent{
      public void methodOne(){}
}

2. we can't override instance method as static,it would result in compile
time error.

eg#2.
class Parent{
      public void methodOne(){ }
}
class Child extends Parent{
      public static void methodOne(){}
}

3.
class Parent{
      public static void methodOne(){ }
}
class Child extends Parent{
      public static void methodOne(){}
}
It seems to be overriding,but it is not overriding it is method hiding.
In case of static method, compiler will take the resolution of method
call.

Difference b/w method Overriding and method hiding?
Overriding =>In both parent and child class method should be instance
           Method resolution is based on runtime object.
           JVM will take the call so it is called as
runtimepolymorphsim/dynamic dispatch.

Hiding     =>In both parent and child class method should be static.
           Method resolution is based on reference type.
           Compiler will take the call so it is called as Compile time
polymorphism.

eg#1.
class Parent{
      public static void methodOne(){System.out.println("I am from
parent"); }
}
```

```
class Child extends Parent{
      public static void methodOne(){System.out.println("I am from
child");}
}
public class Test {
      public static void main(String[] args) {
                  Parent p= new Parent();
                  p.methodOne();//I am from parent

                  Child c=new Child();
                  c.methodOne();//I am from child

                  Parent p1=new Child();
                  p1.methodOne();//I am from parent
      }
}
```

Overriding w.r.t var arg method
==========
var arg method should be overriden with var arg only,if we try to override w.r.t normal method
then it would become Overloading nor overriding.

```
eg#1.
class Parent{
      public  void methodOne(int... i){System.out.println("I am from
parent"); }
}
class Child extends Parent{
      public  void methodOne(int i){System.out.println("I am from
child");}
}
public class Test {
      public static void main(String[] args) {
                  Parent p= new Parent();
                  p.methodOne(10);// I am from Parent

                  Child c=new Child();
                  c.methodOne(10);.// I am from Child

                  Parent p1=new Child();
                  p1.methodOne(10);// I am from Parent
      }
}
```

If we replace child class method with var arg method then it will become Overriding.

```
eg#2.
class Parent{
      public  void methodOne(int... i){System.out.println("I am from
parent"); }
}
class Child extends Parent{
      public  void methodOne(int... i){System.out.println("I am from
child");}
}
public class Test {
      public static void main(String[] args) {
                  Parent p= new Parent();
```

```
                p.methodOne(10);//I am from parent

                Child c=new Child();
                c.methodOne(10);//I am from child

                Parent p1=new Child();
                p1.methodOne(10);//I am from child
      }
}
```

Overridng w.r.t variables
==========
1. Overriding concept is not applicable for variables.
2. In case of Overriding compiler will resolve the call based on the reference type.

eg#1.

```
class Parent{int x= 888;}
class Child extends Parent{int x=999;}
public class Test {
      public static void main(String[] args) {
            Parent p= new Parent();
            System.out.println("X value is ::"+p.x);//X value is :: 888

            Child c=new Child();
            System.out.println("X value is :: "+c.x);//X value is :: 999

            Parent p1=new Child();
            System.out.println("X value is :: "+p1.x);//X value is :: 888
      }
}
```

In the above pgm,if the variable is static then also there is no change in the output,becoz compiler will resolve the call based on the reference type.

Difference b/w Overloading vs Overriding?

```
     Overloading
MethodName     : same
ArgumentType   : must be different
MethodSignature: same
  return type  : no restrictions
 access modifer: no restrictions
private,final  : no restrictions
   static
  throws       : no restrictions
 binding       : compiler
alternativename: CompileTimebinding/early binding/eager binding/static
binding.

     Overriding
MethodName     : same
ArgumentType   : must be same
MethodSignature: same
  return type  : same or covariant type
 access modifer: same or increase the
scope(private,default,protected,public)
private,final  : can't be Overriden
   static
```

throws : if child class throws some exception then parent should throw the same or its
Parent type(rule applicable only for CheckedException not for UnCheckedException)
binding : JVM(based on run time object)
alternativename: RunTimePolymorphism/lazy binding/dynamic dispatch.

Note:
1. In overloading we have to check only method names (must be same) and arguments
(must be different) the remaining things like return type extra not required to check.
2. But In overriding we should compulsory check everything like method names, arguments, return types, throws keyword, modifiers etc.

Consider the method in parent class
Parent: public void methodOne(int i)throws IOException

In the child class which of the following methods we can take..
1. public void methodOne(int i) //valid(overriding)
2. private void methodOne()throws Exception//valid(overloading)
3. public native void methodOne(int i);//valid(overriding)
4. public static void methodOne(double d)//valid(overloading)
5. public static void methodOne(int i)
Compile time error :methodOne(int) in Child cannot override methodOne(int) in Parent; overriding method is static
6. public static abstract void methodOne(float f) Compile time error :
1. illegal combination of modifiers: abstract and static
2. Child is not abstract and does not override abstract method methodOne(float) in Child