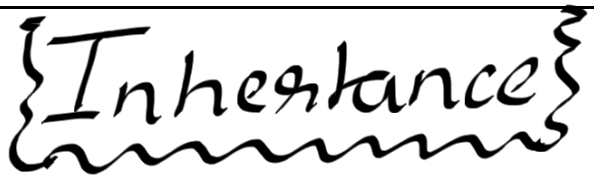
List of Concepts Involved:



- Inheritance introduction
- extends keyword
- Types of Inheritance
- Important key points (5 key points)
- Types of methods Inherited, overridden, specialised
- Rules to override method
- Constructor execution in case of Inheritance

Inheritance Introduction

- It is one of the pillars of Object Orientation.
- It always speaks about reusability.
- Using inheritance productivity of the code can be improved.
- If we use inheritance, lines of code can be reduced in the application.
- In java inheritance is achieved through the "extends" keyword.

Extends keyword

If we use extends keyword, then we can take the properties and behaviours from parent class to child class.

Example

```
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();

        }
 }
```

Output

methodOne from parent
methodOne from parent
methodOne from child

Q) Does private member of a class participate in inheritance?

Soln → No, pvt. member of a class do not participate in inheritance

Q) Does Constructor of Parent class participate in inheritance?

Soln No. Constructor of Parent class do not participate in inheritance
∵ obj of child class calls for child class constructor ∴
it can not inherit Parent class constructor

Example :- Class Human1
{
private String name;
int age;

Human1()
{
Sop("Human class Constructor");
}
Void sleep()
{
age = 18;
Sop("Human needs good sleep");
Sop(age);
}
}

Class Student extends Human1
{
void disp()
{
Sop("The age is: " + age);
Sop("The Name is: " + name);
}

(Error) : cuz name is private and private property does not participate in inheritance.

public class Inheritance2
{
p.s.v.m (String[] args)
{
Student s = new Student1();

s.Sleep();

s.disp();
}
}

O/p :- Human class Constructor
Human needs good sleep

// How does Human class constructor is executing even though it does not participate in inheritance

Q) How does Human class constructor is executing even though constructor does not participate in inheritance

Sol^{n} → In above case, we know when a Student class object is created then a default constructor is created inside the student class with default super() method in 1st line.

```
Class Student
{
  Public Student()
    {
      super(); // this super refers to the
    }          Parent Class
}
```

So, Human() class constructor is executing cuz when Student class object is created the default constructor is created inside student which contains super method that refers to the parent class constructor.
and not buccause Parent constructor participates in inheritance, as it does not

Note:

- Whatever is there in the Parent class by default would be available to the Child class.
- Whatever is there in the Child class by default won't be available to the Parent class.
- On Child reference we can call both Parent class and Child class methods whereas using Parent reference we can call only Parent class methods.

Loan activities

The common methods which are required for HousingLoan,EducationLoan,VehicleLoan are defined in separate class Loan class and it can be reused so that production cost can be increased effectively and time consumption for building projects can be reduced.

Example

```
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
}
```

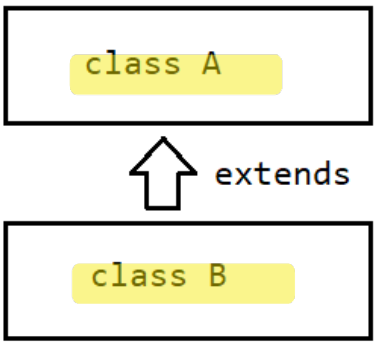
Types of inheritance in java

- Single-level inheritance
- Multi-level Inheritance
- Hierarchical Inheritance
- Multiple Inheritance — (not allowed in Java)
- Hybrid Inheritance

Single-Level Inheritance

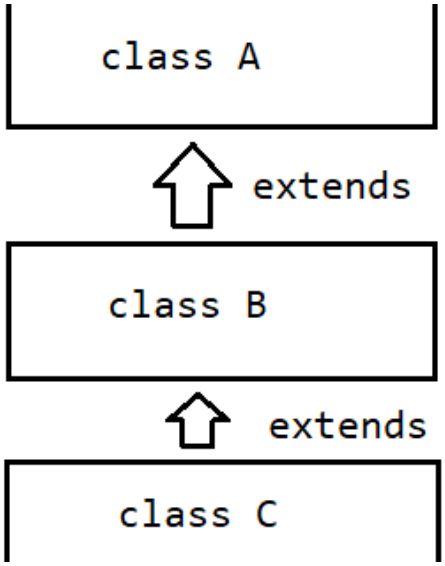
As the name suggests, this type of inheritance occurs for only a single class. Only one class is derived from the parent class. In this type of inheritance, the properties are derived from a single parent class and not more than that. As the properties are derived from only a single base class the reusability of a code is facilitated along with the addition of new features.

The flow diagram of a single inheritance is shown below:



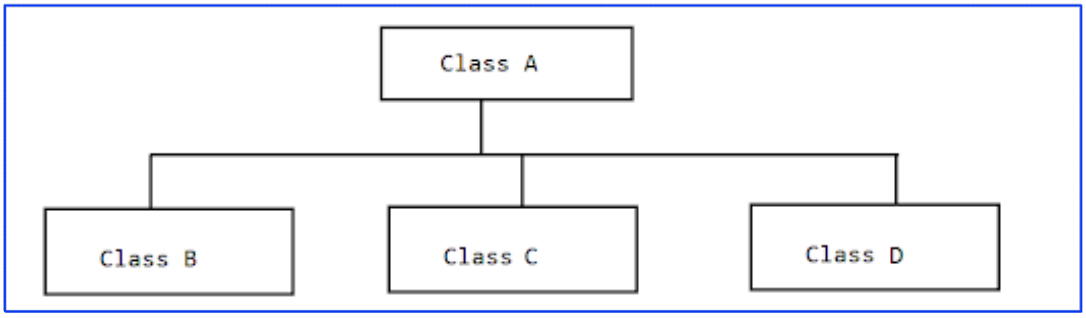
MultiLevel Inheritance

- The multi-level inheritance includes the involvement of at least two or more than two classes.
- One class inherits the features from a parent class and the newly created sub-class becomes the base class for another new class.
- As the name suggests, in the multi-level inheritance the involvement of multiple base classes is there.
- In the multilevel inheritance in java, the inherited features are also from the multiple base classes as the newly derived class from the parent class becomes the base class for another newly derived class.



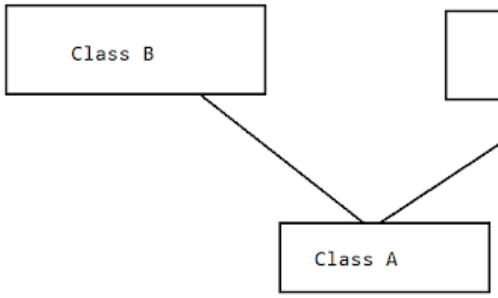
Hierarchical Inheritance

- The type of inheritance where many subclasses inherit from one single class is known as Hierarchical Inheritance.
- Hierarchical Inheritance a combination of more than one type of inheritance.
- It is different from the multilevel inheritance, as the multiple classes are being derived from one superclass.
- These newly derived classes inherit the features, methods, etc, from this one superclass.
- This process facilitates the reusability of a code and dynamic polymorphism (method overriding).



Multiple inheritance — not allowed in Java cuz it leads to ambiguity in compiler

- Multiple inheritance is a type of inheritance where a subclass can inherit features from more than one parent class.
- Multiple inheritances should not be confused with multi-level inheritance, in multiple inheritances the newly derived class can have more than one superclass.
- And this newly derived class can inherit the features from these superclasses it has inherited from, so there are no restrictions.
- In java, multiple inheritance can be achieved through interfaces. ☆



"One class can have multiple child class. But, a child class must have only one parent class"

also c/a (Diamond Shape Problem)

5☆ **Object Class**: When every a class is created without "extends" keyword. By default that class will refer to Object Class as its Parent Class.
Object class is an inbuilt JAVA Class (like String)

Hybrid Inheritance

- Hybrid inheritance is a combination of more than two types of inheritances, single and multiple.
- It can be achieved through interfaces only as multiple inheritance is not supported by Java.
- It is basically the combination of simple, multiple, hierarchical inheritances.

Note

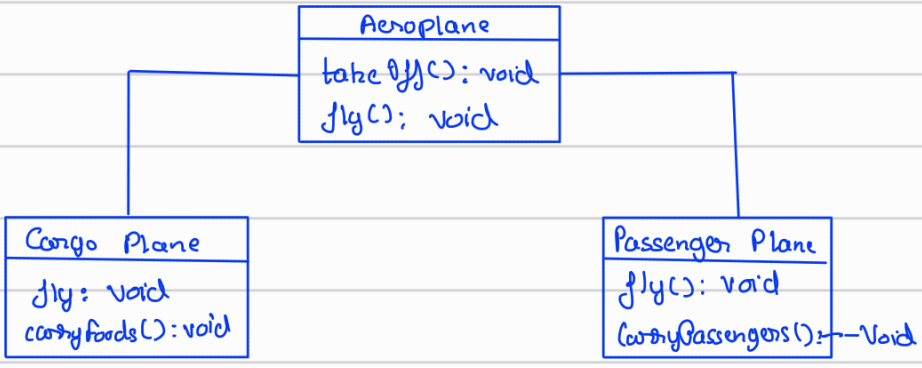
- All the common methods which every java class needs is a part of the Object class, because the Object class is the parent class for all the inbuilt classes in java.
- For all Exception and Error commonly required methods is a parent of "Throwable",hence Throwable is parent class for all "Exception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

Types of Methods in Inheritance

1. Inherited
2. Overridden
3. Specialized

① Inherited Method :-

U.M.L Diagram : Unified Modeling language Diagram provides pictorial representation of the program



So the methods which are directly inherited in the child class and are not modified in the child class are called as Inherited method.

Inherited method

- The method which would come from parent to child due to inheritance is called inherited method.

Example

```
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();//inherited method
                c.methodTwo();//Specialized method

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

Overridden Method

The method which is taken from Parent and changes the implementation as per the needs of the requirement in the class is called the "overridden method".

Example

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

             Child c=new Child();
              c.methodOne();//methodOne from child
        }
   }
```

Specialized Method

The methods which are specific to the particular class are called "Specialised method".

Example

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

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

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

overridden method

Specalised method

Rules to Override a method

- Whatever the Parent has by default available to the Child through inheritance, if the Child is not satisfied with Parent class method implementation then Child is allowed to redefine that Parent class method in Child class in its own way this process is called overriding.
- The Parent class method which is overridden is called the overridden method.
- The Child class method which is overriding is called the overriding method.
- In overriding method resolution is always takes care by JVM based on runtime object hence overriding is also considered as runtime polymorphism or dynamic polymorphism or late binding.
- The process of overriding method resolution is also known as dynamic method dispatch.

☆In overriding method names and arguments must be the same.
That is, method signatures must be the same.

- Until 1.4 version the return types must be same but from 1.5 version onwards covariant return types are allowed.
- According to this Child class method return type need not be same as Parent class method return type and Child types are also allowed.

Constructor Execution in case of inheritance

In case of Constructor the Parent class constructor would be executed followed by Child class constructor with the help of "super()".
It is basically used to make a call to the parent class constructor.
Internally jvm uses super() to promote constructor chaining at inheritance level.

Example

```
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 following things take place

a. Identification of instance variables,instance blocks from parent to child.
b. Execution of instance variables assignment,instance block only of parent class.
c. Execution of parent class constructor.
d. Execution of instance variables assignment,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()

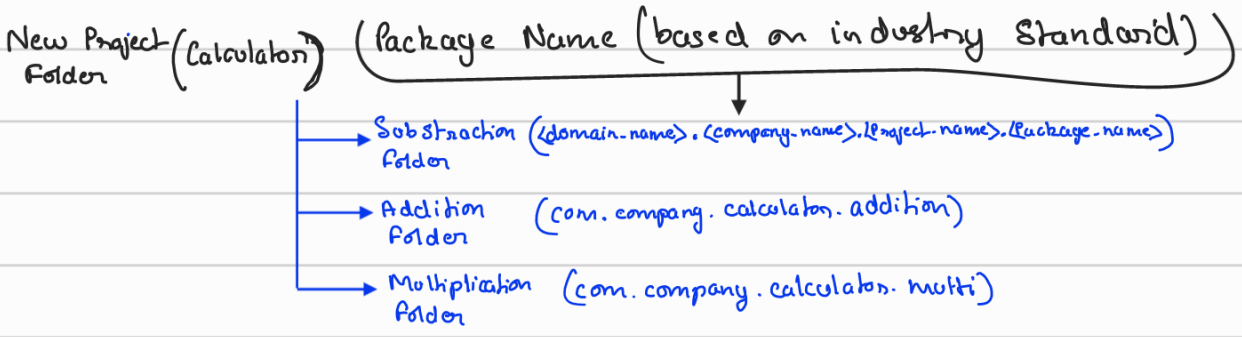
Output

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

Package and Access Modifier

WAP to create Calculator Program with addition & Substraction Packages.

Structure :- Project Structure



Access Modifier	with same Class	Outside Class Same Package	Outsid Package Subclass	Outside Package Non-Subclass
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
default	✓	✓	✗	✗
Private	✓	✗	✗	✗

Rules to Override a Child-Class;

① Decrease in visibility is not allowed:

```
Class Animal
{
  public void eat()
  {
    Sop("Animal eats everyday");
  }
}

Class Tiger extends Animal
{
  void eat()
  {
    Sop("Tiger hunts and eats");
  }
}
```

→ // While overriding method access modifier should be same as Parent method, can't decrease its visibility but increasing visibility is allowed.

② Same return type

```
class Parent
{
  pubic void age()
  {
    Sop("age is 20");
  }
}

class Child extends Parent
{
  public int age()        // not allowed
  {
    return 18;
  }
}
```

③ Same no. of argument as overridden method:-

```
class Parent
{
  public void age(int x)
  {
    Sop("Age is: " + x);
  }
}

class Child
{
  public void age()       // → not same method → it is considered as diff method wz diff parameters/argements as above method.
  {
    Sop("18");
  }
}
```

Constructor Execution in case of Inheritance

```
class Demo1
{  int a,b
   public Demo1()
   {
     Sop (" Parent Class Constructor")
   }
   Public Demo1(int x, int y)
   {
     Sop (" Parameterized Parent Class Const.")
      a=x;  b=y;
   }
}
class Demo2 extends Demo1
{
   int m, n;
   public Demo2()
   {
     Sop (" Child Class Const.");
   }
   public Demo2(int x, int y)
   {
     Sop (" Parameterised Child Class Const");
      m=x;  n=y;
   }
}
```

① if p.s.v.m{
Demo2 d1 = new Demo2 (10, 20)
O/p = ?

Soln → O/p = Parent class Constructor
Parameterized Child Class Const.

even though an objection creation the parameterized child constructor is called it prints zero Parameter Parent Class constructor cuz every constructor has "super()" method as default that calls Parent class method

Q) WAP to call Parameterize Parent Chid Constructor when Child Class is called.

```
class Demo1
{  int a,b
   public Demo1()
   {
     Sop (" Parent Class Constructor")
   }
   Public Demo1(int x, int y)
   {
     Sop (" Parameterized Parent Class Const.")
      a=x;  b=y;
   }
}
class Demo2 extends Demo1
{
   int m, n;
   public Demo2()
   {  Super(10, 20);
     Sop (" Child Class Const.");
   }
   public Demo2(int x, int y)
   {
     Sop (" Parameterised Child Class Const");
      m=x;  n=y;
   }
}
```

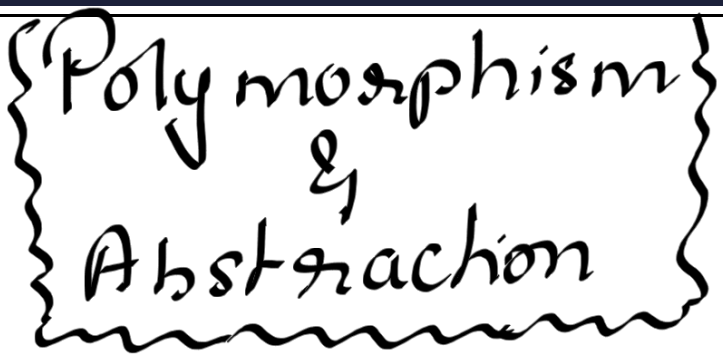
this methods Parent Class Constructor with 2 parameters.

O/p → Parameterized Parent Class Const
Child Class Const.

if instead of super(0, 20), this (10, 20) was written the Parameterized Child Class Constructor will be called such that:

O/p → Parameterized Child Class Const
Child Class Constructor

List of Concepts Involved:



- What is polymorphism?
- How to achieve polymorphism
- Runtime vs Compile time polymorphism
- Abstract keyword and Abstraction
- Abstract class and Abstract method
- final class
- final variable
- final method

What is polymorphism?

If one thing exists in more than one form then it is called Polymorphism.
Polymorphism is a Greek word, where Poly means many and morphism means structures or forms.

1.Static Polymorphism
2.Dynamic Polymorphism

1.Static Polymorphism:
If polymorphism exists at compilation time then it is called Static Polymorphism.
Ex: Overloading. — Eg- some method with diff. diff parameters

2.Dynamic Polymorphism:
If the polymorphism exists at runtime then that polymorphism is called Dynamic Polymorphism.
Ex: Overriding — Eg-

Method Overriding:

The process of replacing existing method functionality with some new functionality is called Method Overriding.

To perform Method Overriding, we must have inheritance relationship classes.
In Java applications, we will override super class method with sub class method.
In Java applications,we will override super class methods with subclass methods.
If we want to override super class method with subclass method then both super class method and sub class method must have the same method prototype. — Same parameter, Same access modifier, Same return type

Steps to perform Method Overriding:

1. Declare a superclass with a method which we want to override.
2. Declare a subclass and provide the same super class method with different implementation.
3. In the main class,in the main() method,prepare an object for the subclass and prepare a reference variable for super class[UpCasting].
4. Access the super class method then we will get output from the sub class method.

Example

```
class Loan{
	public float getIR(){
		return 7.0f;
	}
}
class GoldLoan extends Loan{
	public float getIR(){
		return 10.5f;
	}
}
class StudyLoan extends Loan{
	public float getIR(){
		return 12.0f;
	}
}
class CraftLoan extends Loan{
}
class Test{
	public static void main(String[] args){
		Loan gold_Loan=new GoldLoan();
		System.out.println("Gold Loan IR :"+gold_Loan.getIR()+"%");

		Loan study_Loan=new StudyLoan();
		System.out.println("Study Loan IR :"+study_Loan.getIR()+"%");

		Loan craft_Loan=new CraftLoan();
		System.out.println("Craft Loan IR :"+craft_Loan.getIR()+"%");
	}
}
```

NOTE:

To prove method overriding in Java, we have to access the super class method but JVM will execute the respective sub class method and JVM has to provide output from the respective sub classmethod, not from superclass method. To achieve the above requirement we must create reference variables for only super classes and we must create objects for subclasses.

```
class A{
	void m1(){
		System.out.println("m1-A");
	 }
}
class B extends A{
	void m1(){
		System.out.println("m1-B");
	}
}
public class Test{
	public static void main(String args[]){
		A a  = new B();
		  a.m1();
	 }
}
```

Rules to perform Method Overriding:

1.To override super class method with sub class, then super class method must not be declared as private.

Ex:

```
class A{
        private void m1(){
                System.out.println("m1-A");
        }
}
class B extends A{
      void m1(){
              System.out.println("m1-B");
      }
}

public class Test{
        public static void main(String args[]){
                    A a=new A();
                    a.m1();
        }
}
```

2.To override super class method with sub class method then sub class method should have the same return type of the super class method.

EX:

```
class A{
      int m1(){
              System.out.println("m1-A");
              return 10;
      }
}
class B extends A{
      void m1(){
              System.out.println("m1-B");
      }
}
public class Test{
       public static void main(String args[]){
               A a=new B();
               a.m1();
       }
}
```

3.To override super class method with sub class method then super class method must not be declared as final sub class method may or may not be final.

EX:

```
class A{
      void m1(){
              System.out.println("m1-A");
      }
}
class B extends A{
        final void m1(){
               System.out.println("m1-B");
        }
}
public class Test{
      public static void main(String[] args){
             A a=new B();
            a.m1();
      }
}
```

4.To override superclass method with sub class method either super class method or subclass method as static then compiler will rise an error.If we declare both super and sub class method as static in method overriding compiler will not rise any error,JVM will provide output from the super class Method.

NOTE: If we are trying to override superclass static method with subclass static method then super class static method will override subclass static method,where JVM will generate output from super class static method.

EX:

```
class A{
       static void m1(){
              System.out.println("m1-A");
       }
}
class B extends A{
       static void m1(){
              System.out.println("m1-B");
       }
}
public class Test{
       public static void main(String args[]){
                     A a=new B();
                     a.m1();
       }
}
```

Inherited method

- The method which would come from parent to child due to inheritance is called inherited method.

Example

```
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();//inherited method
                c.methodTwo();//Specialized method

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

Overridden Method

The method which is taken from Parent and changes the implementation as per the needs of the requirement in the class is called the “overridden method”.

Example

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

             Child c=new Child();
              c.methodOne();//methodOne from child
        }
   }
```

Inherited method

- The method which would come from parent to child due to inheritance is called inherited method.

Example

```
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();//inherited method
                c.methodTwo();//Specialized method

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

Overridden Method

The method which is taken from Parent and changes the implementation as per the needs of the requirement in the class is called the “overridden method”.

Example

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

             Child c=new Child();
              c.methodOne();//methodOne from child
        }
   }
```

5.To override super class method with subclass method,sub class method must have either the same scope of the super class method or more scope when compared with super class method scope otherwise the compiler will raise an error.

EX:

```
class A{
      protected void m1(){
         System.out.println("m1-A");
      }
}
class B extends A{
      public void m1(){
             System.out.println("m1-B");
      }
}
public class Test{
       public static void main(String args[]){
              A a=new A();
              a.m1();
       }
}
```

6.To override super class method with subclass method subclass method should have either same access privileges or weaker access privileges when compared with super class method access privileges.

CompileTime Polymorphism vs RunTime Polymorphism

What are the differences between method overloading and method overriding?

1. The process of extending the existing method functionality with new functionality is called Method Overloading.
   The process of replacing existing method functionality with new functionality is called Method Overriding.

2. In the case of method overloading, different method signatures must be provided to the methods
   In the case of method overriding, the same method prototypes must be provided to the methods.

3. With or without inheritance we can perform method overloading
   With inheritance only we can perform Method overriding

Abstract keyword and Abstraction

any method that has no implementation, body and have method signature (name) in java such methods are known as "Abstract Methods"

Abstract keyword is used before defining any abstract method

Ex: abstract public void takeoff();

Example

```
class DB_Driver{
       public void getDriver(){
               System.out.println("Type-1 Driver");
       }
}
class New_DB_Driver extends DB_Driver{
      public void getDriver(){
             System.out.println("Type-4-Driver");
      }
}
class Test{
      public static void main(String args[]){
                    DB_Driver driver=new New_DB_Driver();
                    driver.getDriver();
     }
}
```

- In the above example, method overriding is implemented, in method overriding, for super class method call JVM has to execute subclass method, not super class method.
- In method overriding, always JVM is executing only subclass method, not super class method.
- In method overriding, it is not suggestible to manage super class method body without execution, so that, we have to remove superclass method body as part of code optimization.
- In Java applications, if we want to declare a method without body then we must declare that method as "Abstract Method".
- If we want to declare abstract methods then the respective class must be an abstract class.

Example

```
abstract class DB_Driver{
        public abstract void getDriver();
}
class New_DB_Driver extends DB_Driver{
      public void getDriver(){
              System.out.println("Type-4 Driver");
      }
}

public class Test{
       public static void main(String args[]){
                     DB_Driver driver=new New_DB_Driver();
                     driver.getDriver();
       }
}
```

Down Casting

- In Java applications, if we declare any abstract class with abstract methods, then it is convention to implement all the abstract methods by taking sub classes.
- To access the abstract class members, we have to create an object for the subclass and we have to create a reference variable either for abstract class or for the subclass.
- If we create reference variables for abstract class then we are able to access only abstract class members, we are unable to access subclass own members.
- If we declare a reference variable for subclass then we are able to access both abstract class members and subclass members.

Example

```
abstract class A{
        void m1(){
                System.out.println("m1-A");
        }

        abstract void m2();
        abstract void m3();
}
class B extends A{
      void m2(){
              System.out.println("m2-B");
      }
      void m3(){
              System.out.println("m3-B");
      }
      void m4(){
              System.out.println("m4-B");
      }
}

public class Test{
        public static void main(String args[]){
               A a=new B();
               a.m1();
               a.m2();
               a.m3();
               //a.m4();---error

               B b=new B();
               b.m1();
               b.m2();
               b.m3();
               b.m4();
       }
}
```

In Java applications,it is not possible to create Objects for abstract classes but it is possible to provide constructors in abstract classes,because,to recognize abstract class instance variables in order to store them in the subclass objects.

EX:

```
abstract class A{
  A(){
      System.out.println("A-Con");
  }
}
class B extends A{
      B(){
             System.out.println("B-Con");
      }
}
public class Test{
       public static void main(String[] args){
              B b=new B();
       }
}
```

In Java applications, if we declare any abstract class with abstract methods then it is mandatory to implement all the abstract methods in the respective subclass.
If we implement only some of the abstract methods in the respective subclass then compiler will rise an error, where to come out from compilation error we have to declare the respective subclass as an abstract class and we have to provide implementation for the remaining abstract methods by taking another subclass in multilevel inheritance.

EX:

```
abstract class A{
      abstract void m1();
      abstract void m2();
      abstract void m3();
}
abstract class B extends A{
       void m1(){
             System.out.println("m1-A");
       }
}
class C extends B{
      void m2(){
              System.out.println("m2-C");
     }
     void m3(){
              System.out.println("m3-C");
     }
}
public class Test{
       public static void main(String[] args){
              A a=new C();
              a.m1();
              a.m2();
              a.m3();
       }
}
```

In Java applications, if we want to declare an abstract class then it is not at all mandatory to have at least one abstract method, it is possible to declare abstract class without having abstract methods but if we want to declare a method as an abstract method then the respective class must be an abstract class.

```
abstract class A{
      void m1(){
              System.out.println("m1-A");
      }
}
class B extends A{
        void m2(){
                System.out.println("m2-B");
         }
}
public class Test{
       public static void main(String args[]){
              A a=new B();
              a.m1();
              //a.m2();---------->Error

               B b=new B();
               b.m1();
               b.m2();
      }
}
```

In Java applications, it is possible to extend an abstract class to concrete class and from concrete class to abstract class.

```
class A{
      void m1(){
             System.out.println("m1-A");
      }
}
abstract class B extends A{
        abstract void m2();
}
class C extends B{
        void m2(){
                System.out.println("m2-C");
        }
        void m3(){
                System.out.println("m3-C");
        }
}
public class Test{
       public static void main(String args[]){
              A a=new C();
              a.m1();
              //a.m2();  error
             //a.m3();  error
```

```
            B b=new C();
            b.m1();
            b.m2();
            //b.m3(); error

            C c=new C();
            c.m1();
            c.m2();
            c.m3();
    }
}
```

Note:
In Java applications, it is not possible to extend a class to the same class, if we do the same then the compiler will raise an error like "cyclic inheritance involving".

```
class A extends A{
}
Status: Compilation Error: “cyclic inheritance involving".

class A extends B{
}
class B extends A{
}
```

Status: Compilation Error: “cyclic inheritance involving".

final class: don't Participate in inheritance.

- If a class is marked as final, then the class won't participate in inheritance,if we try to do so then it would result in “CompileTime Error”.

Eg: String, StringBuffer,Integer,Float,.....

final variable : can not change/reassign the variable

- If a variable is marked as final, then those variables are treated as compile time constants and we should not change the value of those variables.
- If we try to change the value of those variables then it would result in “CompileTimeError”.

final method : can not override.

- If a method is declared as final then those methods we can't override, if we try to do so it would result in “CompileTimeError”.

Note
Every method present inside a final class is always final by default whether we are declaring or not. But every variable present inside a final class need not be final.

The main advantage of the final keyword is we can achieve security.
Whereas the main disadvantage is we are missing the key benefits of oops:
polymorsim (because of final methods), inheritance (because of final classes) hence if there is no specific requirement never recommended to use final keyboard.