

OOPS PRINCIPLES

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

ENCAPSULATION

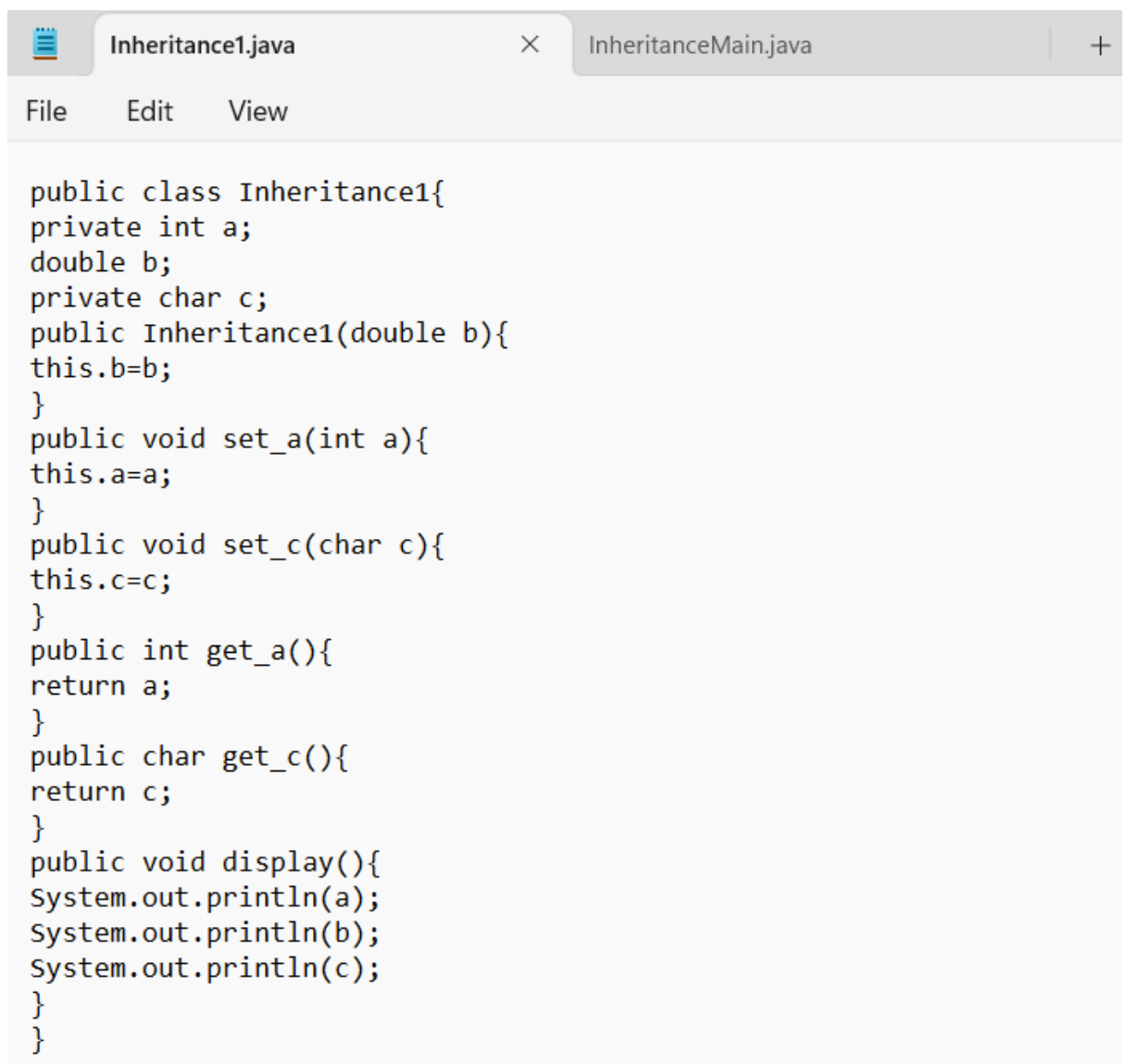
- It is a process of wrapping the properties and behavior of an object is called encapsulation.
- We use encapsulation for data hiding and security to data. (eg:ATM)
- Steps to design encapsulation:
 - 1.Declare a data private because to hide the data and to provide the security.
 - 2.Design a constructor for data which is not private.
 - 3.Design getter and setter method for the data which is private.

```
class Inheritance{
private int a;
double b;
private char c;
public Inheritance(double b){
this.b=b;
}
void set_a(int a){
this.a=a;
}
void set_c(char c){
this.c=c;
}
int get_a(){
return a;
}
char get_c(){
return c;
}
public void display(){
System.out.println(a);
System.out.println(b);
System.out.println(c);
}
public static void main(String []args){
Inheritance i= new Inheritance(2.56);
i.set_a(9);
i.set_c('a');
i.display();
}
}
```

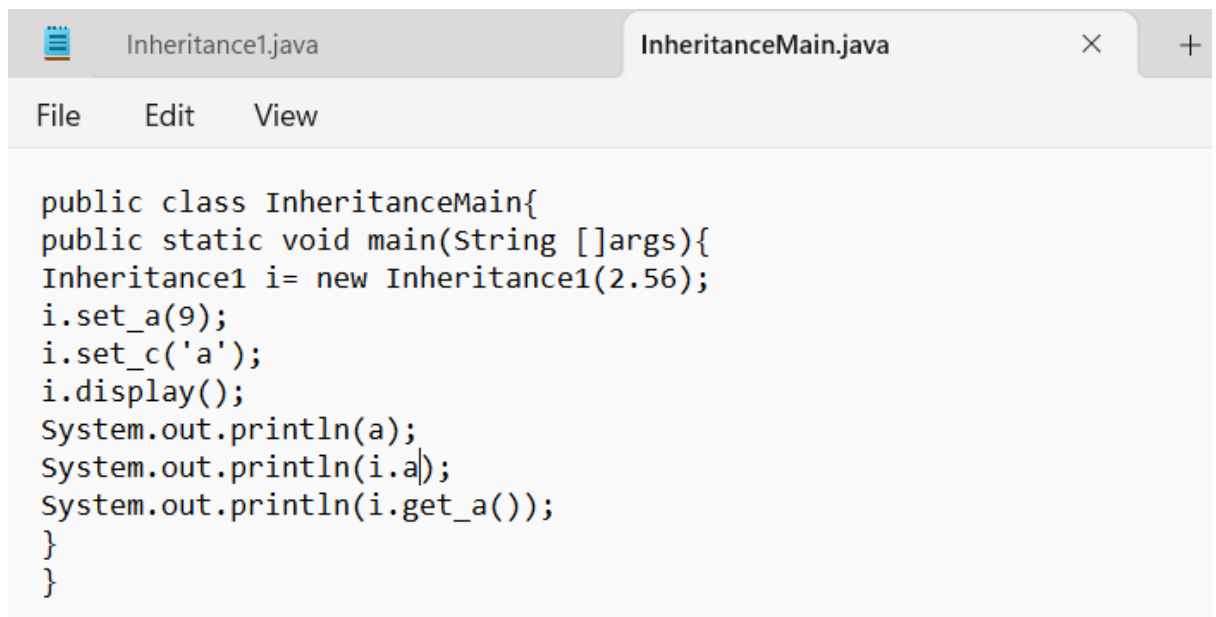
- Here in display method, we directly accessing the variables because we set the variables before using them.
- We can use a or this.a or get_a() to get the output.
- By the way the output of above program is

```
C:\Users\ganieswar\Desktop\java_programs>java Inheritance1.java
9
2.56
a
```

- But the problem comes when we are using them from different classes like:



```
public class Inheritance1{
    private int a;
    double b;
    private char c;
    public Inheritance1(double b){
        this.b=b;
    }
    public void set_a(int a){
        this.a=a;
    }
    public void set_c(char c){
        this.c=c;
    }
    public int get_a(){
        return a;
    }
    public char get_c(){
        return c;
    }
    public void display(){
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```



```
public class InheritanceMain{
public static void main(String []args){
Inheritance1 i= new Inheritance1(2.56);
i.set_a(9);
i.set_c('a');
i.display();
System.out.println(a);
System.out.println(i.a);
System.out.println(i.get_a());
}
}
```

```
C:\Users\ganieswar\Desktop\java_programs>javac InheritanceMain.java
InheritanceMain.java:7: error: cannot find symbol
System.out.println(a);
                   ^
    symbol:   variable a
    location: class InheritanceMain
InheritanceMain.java:8: error: a has private access in Inheritance1
System.out.println(i.a);
                   ^
2 errors
```

- So, we can't access the private variables directly or by using object, the way is to use getter method.

```
public class InheritanceMain{
public static void main(String []args){
Inheritance1 i= new Inheritance1(2.56);
i.set_a(9);
i.set_c('a');
//i.display();
System.out.println(i.get_a());
System.out.println(i.b);
System.out.println(i.get_c());
}
}
```

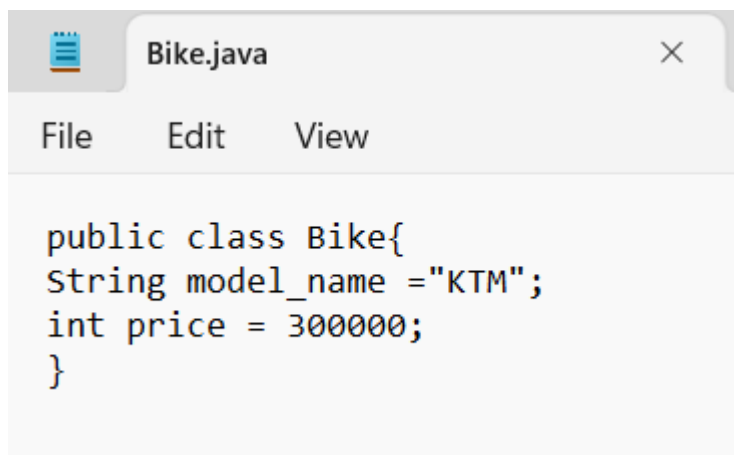
```
C:\Users\ganieswar\Desktop\java_programs>java InheritanceMain.java
9
2.56
a
```

- Here we can access the b using the object and before that please check that if b is public or not.

- IF AN VARIABLE IS PRIVATE AND WE WANT TO USE IT FROM ANOTHER CLASS, WE NEED TO HAVE SETTER AND GETTER METHODS FOR THAT.
 - LOGIN PROGRAM
-

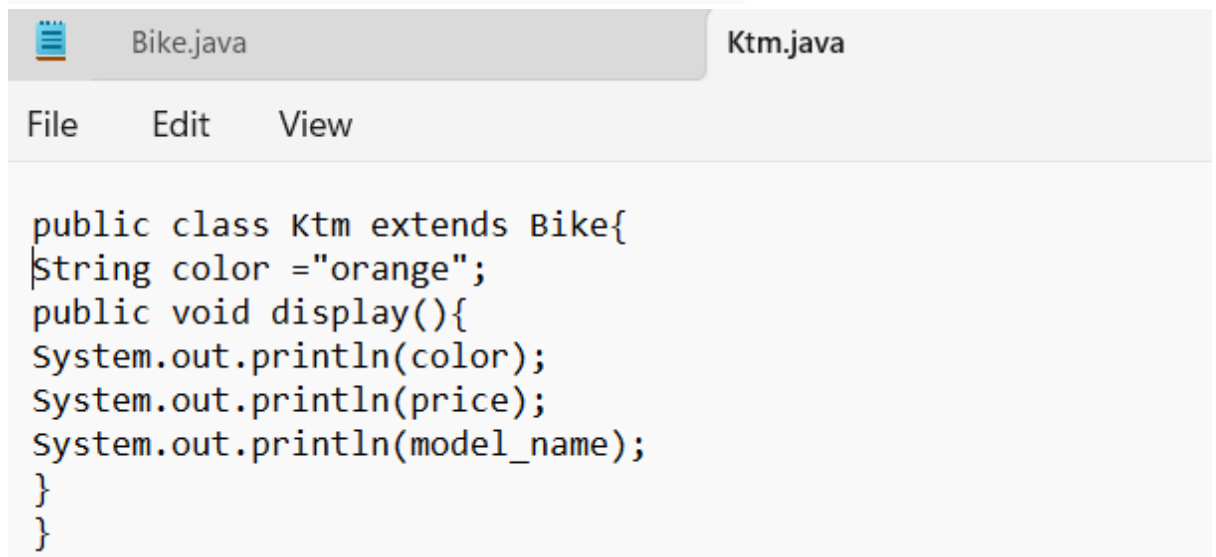
INHERITANCE

- It is a process of acquiring properties and behaviors from one class to another class.
- There are 2 ways: using class; using interface
- Keywords: extends and implements
- Child class inherits all the parent class properties, if parent has variable: a and method: display (); child has variable: b and method: print (); parent contains only 2 but child contains all 4.
- Static members will be hidden from the child class.



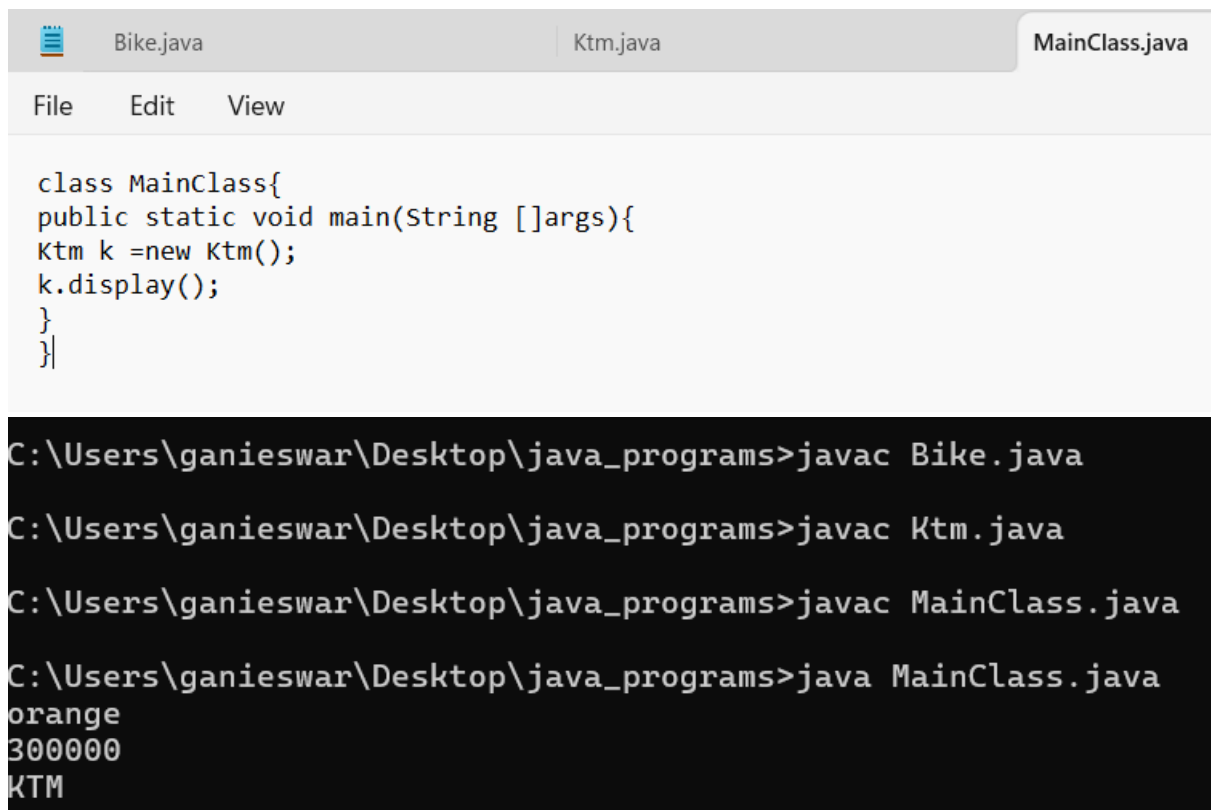
A screenshot of a code editor window titled "Bike.java". The window has a menu bar with "File", "Edit", and "View". The code inside is as follows:

```
public class Bike{  
    String model_name ="KTM";  
    int price = 300000;  
}
```



A screenshot of a code editor window showing two tabs: "Bike.java" and "Ktm.java". The "Ktm.java" tab is active. The window has a menu bar with "File", "Edit", and "View". The code inside is as follows:

```
public class Ktm extends Bike{  
    String color ="orange";  
    public void display(){  
        System.out.println(color);  
        System.out.println(price);  
        System.out.println(model_name);  
    }  
}
```



```
class MainClass{
public static void main(String []args){
Ktm k =new Ktm();
k.display();
}
}
```

```
C:\Users\ganieswar\Desktop\java_programs>javac Bike.java
C:\Users\ganieswar\Desktop\java_programs>javac Ktm.java
C:\Users\ganieswar\Desktop\java_programs>javac MainClass.java
C:\Users\ganieswar\Desktop\java_programs>java MainClass.java
orange
300000
KTM
```

- This is the example for single level inheritance: one parent and one child.
- Multi-level Inheritance: parent, child, grandchild.
public class Flower {};
public class Rose extends Flower {};
public class Redrose extends Rose {}.
- Hierarchical Inheritance: one parent and two children.
public class Bank {};
public class Hdfc extends Bank {};
public class Icici extends Bank {};
- Multiple Inheritance: two parents and one child.
It is not possible using class.
- We can achieve multiple inheritance by only using interface in java.

CONSTRUCTOR CHAINING

- It is a process of calling or invoking a constructor inside another constructor by using constructor calling statement.
- There are 2 constructor calling statements. – this () – super ().
- Rules of constructor calling:
 1. It should be first statement.

2. A constructor can call or invoke only one constructor if we try to invoke more than 1 constructor we will get compile time error.

3. Calling itself is not possible.

- **This** is used to call or invoke one constructor inside another constructor **in the same class**.

```
public class ThisUsage{
    String name;
    int age;
    char gender;
    public ThisUsage(){
        System.out.println("no argument constructor");
    }
    public ThisUsage(String name){
        this();
        this.name = name;
        System.out.println("string argument constructor");
    }
    public ThisUsage(String name, int age){
        this(name);
        this.name = name;
        this.age=age;
        System.out.println("string and age argument constructor");
    }
    public ThisUsage(String name, int age,char gender){
        this(name,age);
        this.name = name;
        this.age=age;
        this.gender=gender;
        System.out.println("String,age and gender argument constructor");
    }
    public static void main(String []args){
        ThisUsage tu = new ThisUsage("ganesh",20,'M');
    }
}
```

```
C:\Users\ganieswar\Desktop\java_programs>java ThisUsage.java
no argument constructor
string argument constructor
string and age argument constructor
String,age and gender argument constructor
```

- **Super** calling statement is used to invoke or calling parent class constructor inside child class constructor.
- While invoking parent class properties to child class, it is mandatory to invoke parent class constructor implicitly or explicitly.
- If it is a no argument constructor, it is not required to invoke the constructor explicitly, because compiler invoke implicitly.
- If it is parameterized constructor, it is mandatory to invoke a constructor explicitly by developer.

```
public class A{
    int a;
    A(int a){
        super();
        this.a=a;
    }
}
```

```
public class B extends A{
    int b;
    public B(int a,int b){
        super(a);
        this.b=b;
    }
    public void display(){
        System.out.println("a :"+a);
        System.out.println("b :"+b);
    }
}
```

```
public class ABMainclass{
    public static void main(String []args){
        B nb = new B(10,20);
        nb.display();
    }
}
```

```
C:\Users\ganieswar\Desktop\java_programs>javac A.java
C:\Users\ganieswar\Desktop\java_programs>javac B.java
C:\Users\ganieswar\Desktop\java_programs>javac ABMainclass.java
C:\Users\ganieswar\Desktop\java_programs>java ABMainclass.java
a :10
b :20
```


- In class is it mandatory to call by super ()?
- No, it is **not mandatory** to explicitly call super () in a class constructor unless:
The superclass has no default (no-argument) constructor.
 If the superclass defines constructors but does not have a no-argument constructor, then you must explicitly call one of its constructors using super () or super(args).
You want to call a specific constructor in the superclass.
 When the superclass has multiple constructors, you might want to call a particular one using super(args).
- We cannot invoke this () calling statement and super () calling statement in a single constructor because constructor calling statement should be the first statement and only one statement in a constructor either this () or super ().

CLASS TYPECASTING

- The process of converting one type of class into another type of class is known as class typecasting or non-primitive typecasting.
- To perform type casting IS-A relation (INHERITANCE) is mandatory.
- There are 2 types: -Up Casting - Down Casting
- **UP CASTING:** The process of converting child class properties into parent class properties is known as Up Casting. It is also known as generalization.
- While upcasting only parent class properties can be accessed because child class started acting like a parent class, so child class lose/hide its own properties.
- While upcasting if there are any override properties present in child class then (only override properties are accessed) they can be accessed because override method belongs to parent class, it does not belong to child class.
- Example if there are no overridden properties:

```
public class Demo5{
    public int a=10;
    public void display(){
        System.out.println("Demo5 class display() method");
    }
}
```

```
public class Demo6 extends Demo5{
    public int b=20;
    public void print(){
        System.out.println("Demo6 class print() method");
    }
}
```

```
public class MainClass5{
    public static void main(String []args){
        Demo5 d= new Demo6();
        System.out.println("a :" + d.a);
        d.display();
    }
}
```

```
C:\Users\ganieswar\Desktop\java_programs>java MainClass5.java
a :10
Demo5 class display() method
```

- We cannot access child class properties.
- Example if there are overridden properties:

```
public class Demo5{
    public int a=10;
    public void display(){
        System.out.println("Demo5 class display() method");
    }
}
```

```
public class Demo6 extends Demo5{
public int a=20;
public void display(){
System.out.println("Demo6 class display() method");
}
}
```

```
public class MainClass5{
public static void main(String []args){
Demo5 d= new Demo6();
System.out.println("a :" + d.a);
d.display();
}
}
```

```
C:\Users\ganieswar\Desktop\java_programs>java MainClass5.java
a :10
Demo6 class display() method
```

- We cannot override variables, the overridden method displayed here.
- **DOWN CASTING:** It is a process of converting parent class properties into child class properties. It is also known as Specialization.
- We cannot perform downcasting implicitly, we can perform only explicitly by using typecasting operator.
- We cannot achieve downcasting directly (without upcasting), if we try the compiler will throw classcast Exception.
- While performing downcasting we can access both the parent and child class properties.

```
public class Demo5{
public int a=10;
public void display(){
System.out.println("Demo5 class display() method");
}
}
```

```
public class Demo6 extends Demo5{
    public int b=20;
    public void print(){
        System.out.println("Demo6 class print() method");
    }
}
```

```
public class MainClass5{
    public static void main(String []args){
        Demo5 d= new Demo6();
        Demo6 d1=(Demo6)d;
        System.out.println("a :" + d1.a);
        System.out.println("b :" + d1.b);
        d1.display();
        d1.print();
    }
}
```

```
C:\Users\ganieswar\Desktop\java_programs>java MainClass5.java
a :10
b :20
Demo5 class display() method
Demo6 class print() method
```

- Here we can see upcasting and downcasting in MainClass5 and also while downcasting we use (Demo6) d to avoid compatibility error.
- We use the object of upcasting while doing downcasting.
- **IS – A RELATIONSHIP**

■ HAS-A RELATIONSHIP

- The dependency between two or more objects is called HAS A relationship, it is classified into two types:
 - Aggregation
 - Composition
- Aggregation: The dependency between two or more objects such that an object can exist without that dependent object. It is also known as loose coupling.
Eg: cab and ola, food and Zomato.
- Composition: The dependency between two or more objects such that an object cannot exist without that dependent object. It is also known as tight coupling.
Eg: human and heart, phone and battery, car and engine.

```
public class Address{
    String doorNo;
    String street;
    String city;
    String district;
    String state;
    String country;
    int pincode;
    public Address(String doorNo, String street,String city,
    String district,String state,String country,int pincode){
        this.doorNo = doorNo;
        this.street = street;
        this.city = city;
        this.district = district;
        this.state = state;
        this.country = country;
        this.pincode = pincode;
    }
}
```

```

public class Person{
    String name;
    int age;
    String gender;
    long mobno;
    Address addr; //HAS-A RELATIONSHIP
    public Person(String name, int age, String gender, long mobno, Address addr){
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.mobno = mobno;
        this.addr = addr;
    }
    public void Details(){
        System.out.println("***Person Details***");
        System.out.println("name: "+name);
        System.out.println("age: "+age);
        System.out.println("Gender: "+gender);
        System.out.println("mobile number: "+mobno);
        System.out.println("***person address***");
        System.out.println(addr.doorNo);
        System.out.println(addr.street);
        System.out.println(addr.city);
        System.out.println(addr.district);
        System.out.println(addr.state);
        System.out.println(addr.country);
        System.out.println(addr.pincode);
    }
}

public class HasA{
    public static void main(String []args){
        Address addr = new Address("15-160", "dpt", "jami", "vzm", "ap", "india", 535250);
        Person p = new Person("ganesh", 20, "M", 9949201457L, addr);
        p.Details();
    }
}

```

```

C:\Users\ganieswar\Desktop\java_programs>java HasA.java
***Person Details***
name: ganesh
age: 20
Gender: M
mobile number: 9949201457
***person address***
15-160
dpt
jami
vzm
ap
india
535250

```

■ Initializers in Inheritance:

```
public class A1{
    static{
        System.out.println("A class static block");
    }
    {
        System.out.println("A class non-static block");
    }
}
```

```
public class B1 extends A1{
    static{
        System.out.println("B1 class in static block");
    }
    {
        System.out.println("B1 class in no-static block");
    }
}
```

```
public class Initializers1{
    public static void main(String []args){
        A1 a= new A1();
        B1 b= new B1();
        A1 a1= new B1();
    }
}
```

```
C:\Users\ganieswar\Desktop\java_programs>java Initializers1.java
A class static block
A class non-static block
B1 class in static block
A class non-static block
B1 class in no-static block
A class non-static block
B1 class in no-static block
```

- Static blocks execute only once when the class is loaded into memory, regardless of how many objects are created.
- Non-static blocks execute every time an object of the class is instantiated, in the order of inheritance (parent to child).

- Even though the reference type is A1, the object type is B1. Java must fully initialize the B1 object, which includes executing the non-static blocks of both A1 and B1.
- INSTANCE OF keyword: It is a keyword or operator which is used to check the given object is an instance of specific type (sub class) or It is used to check the child class ref var belong to parent class or itself.
- It returns Boolean type.
- By using instance of keyword, we can compare child class reference to parent class or itself but we cannot compare parent class reference to child class.

```
public class Flower{  
}
```

```
public class Rose extends Flower{  
}
```

```
public class RedRose extends Rose{  
}
```

```
public class Instance{  
    public static void main(String []args){  
        Flower f=new Flower();  
        Rose r= new Rose();  
        RedRose rr= new RedRose();  
        System.out.println(f instanceof Object);  
        System.out.println(r instanceof Object);  
        System.out.println(rr instanceof Object);  
        System.out.println(f instanceof Flower);  
        System.out.println(r instanceof Flower);  
        System.out.println(rr instanceof Flower);  
        System.out.println(r instanceof Rose);  
        System.out.println(rr instanceof Rose);  
        System.out.println(rr instanceof RedRose);  
        System.out.println(f instanceof Rose);  
        System.out.println(r instanceof RedRose);  
        System.out.println(f instanceof RedRose);  
    }  
}
```



```
C:\Users\ganieswar\Desktop\java_programs>java Instance.java
true
true
true
true
true
true
true
true
true
false
false
false
```

METHOD OVERLOADING

- It is a process of defining or declaring multiple method in same class with same name but should be different in parameter length/ type/ order.
- We can perform method overloading for both the static and non-static methods.

```
public class Addition{
    public static int add(int a, int b){
        return a+b;
    }
    public static int add(int a, int b, int c){
        return a+b+c;
    }
    public static void main(String []args){
        System.out.println(add(2,3));
        System.out.println(add(2,3,5));
    }
}
```

```
C:\Users\ganieswar\Desktop\java_programs>java Addition.jav
5
10
```

- If we try to overload a non-static method, we need to access it by using an object.
- CAN WE OVERLOAD A MAIN METHOD?

- Yes, we can overload main method but parameters should be different.
- Main method of String argument will get invoked by JVM to start the execution.

```
public class MainMethod{
    public static void main(String []args){
        System.out.println("string argument main method(default)");
        int a[] = {1,2,3,4};
        main(a);
    }
    public static void main(int []args){
        System.out.println("int argument main method");
    }
}
```

```
C:\Users\ganieswar\Desktop\java_programs>java MainMethod.java
string argument main method(default)
int argument main method
```

METHOD CHAINING

- It is a process of invoking/ calling one method from another method
- We can invoke a method in the same class or in the child class method. We can invoke more than one method in a method.

```
public class Demo7{
    public static int add(int a, int b){
        return a+b;
    }
    public static int sub(int a, int b, int c){
        return add(a,b)-c;
    }
    public static void main(String []args){
        System.out.println(sub(10,20,5));
    }
}
//25|
```

METHOD OVERRIDING

- It is a process of inheriting parent class method into child class and provide implementation to the inherited method in the child class.

- IS-A relationship is mandatory.
- CAN WE OVERRIDE A STATIC METHOD?
- No because while we inheriting parent class static method, it will be hidden from the child class.
- CAN WE OVERRIDE MAIN METHOD IN JAVA?
- NO, because main method is static method and it will be hidden from childclass and cannot be overridden.

```
public class Demo8{  
    public void display(){  
        System.out.println("display method in parent class");  
    }  
}
```

```
public class Demo9 extends Demo8{  
    public void display(){  
        System.out.println("display method in child class");  
    }  
}
```

```
public class MethodOverriding{  
    public static void main(String []args){  
        Demo9 d= new Demo9();  
        d.display();  
    }  
}
```

```
C:\Users\ganieswar\Desktop\java_programs>javac Demo8.java
```

```
C:\Users\ganieswar\Desktop\java_programs>javac Demo9.java
```

```
C:\Users\ganieswar\Desktop\java_programs>javac MethodOverriding.java
```

```
C:\Users\ganieswar\Desktop\java_programs>java MethodOverriding.java  
display method in child class
```

POLYMORPHISM

- Poly - Many; Morphism – Form (One object many forms)
- An object shows the different behaviors in different situation in its life cycle is known as polymorphism.
- For Example, a person can be son, father, husband, brother all at a time.
- In java, nextInt () ---1. Scanner (To read integer value)
---2. Random (To generate random integer value)

```
import java.util.Scanner;
import java.util.Random;
public class Polymorphism{
public static void main(String []args){
Scanner sc = new Scanner(System.in);
System.out.println("enter a number");
System.out.println("entered number is "+sc.nextInt());
Random r= new Random();
System.out.println(r.nextInt());
}
}
```

```
C:\Users\ganieswar\Desktop\java_programs>java Polymorphism.java
enter a number
10
entered number is 10
-1699078804
```

BINDING

- It is a process of providing connection between method calling statement and method declaration.
- Types: 1. Statis binding 2. Dynamic binding
- STATIC BINDING: The binding which happened between method calling statement and method declaration by compile time is known as static binding.
- Static binding is also known as Early Binding (because binding happens at compile time)

- **DYNAMIC BINDING:** The binding which happened between method calling statement and method declaration by JVM during runtime is known as Dynamic Binding.
- It is also known as late binding.
- **POLYMORPHISM is classified into two types:**
 - 1.Compiletime polymorphism:** It is a process of binding a method during compile time by compiler is known as compile time polymorphism. Also known as static binding. Best example is **METHOD OVERLOADING.**
 - 2.Runtime polymorphism:** It is a process of binding a method during run time by JVM is known as Runtime polymorphism. Also known as Dynamic binding. Best example is **METHOD OVERRIDING.**

ABSTARCTION

- It is the process of hiding the internal implementation and providing only functionality to the user.
- A member which is prefix with abstract keyword is known as abstract members. There are **abstract method** and **abstract class**.
- We cannot declare a variable as abstract because variable contains initialization or re-initialization but abstract keyword is used to hide the implementation.
- We can make inly non static methods abstract because we cannot override static method to provide implementation.
- **CONCRETE MEMBERS:** not prefix with abstract keyword are known as concrete members.
- In abstract class we can declare:
 - Static and non-static variables
 - Constructor
 - Static and non-static initializers
 - Static and non-static concrete methods
 - Abstract method
- If a class in abstract class, it is not mandatory to declare the method as abstract method. If a class contains any abstract method, it is mandatory to declare the class as abstract class.

- We cannot make abstract method final because we cannot override a final method and final method can't be empty.
- How to provide implementation to the abstract method:
REFER SOME MORE INFORMATION ABOUT THE ABSTRACT CLASS.
- We cannot achieve 100 percent abstraction using class.

INTERFACE

- It is a keyword.
- Interface is a class definition block (blueprint of a class) which it contains only abstract methods. It is used to achieve 100 percent abstraction and also can achieve multiple inheritance.
- Interface contains:
 - Variables
 - Abstract methods
 - Static methods
 - Private methods (because only with these we can achieve 100%)
- There is no constructor in interface and also won't allow initializers.
- Object creation for an interface is not possible.
- Class extends class
- Interface extends interface
- Class implements interface (reverse is not possible)

```
public interface Demo11{
    public void test();
    public void display();
}
```

```
public class Demo12 implements Demo11{
    public void test(){
        System.out.println("test method");
    }
    public void display(){
        System.out.println("Display method");
    }
}
```

```

public class Demo13{
public static void main(String []args){
Demo12 d=new Demo12();
d.display();
d.test();
}}

```

```

C:\Users\ganieswar\Desktop\java_programs>java Demo13.java
Display method
test method

```

```

public interface Demo11{
public void test(){
System.out.println("test method");
}

public void display();
}

```

```

C:\Users\ganieswar\Desktop\java_programs>javac Demo11.java
Demo11.java:2: error: interface abstract methods cannot have body
public void test(){
        ^
1 error

```

- Here we can observe that, we cannot have body to the methods in interface.

- Types of inheritance in interface:

- Single level

```

interface parent_interface

```

```

{
}

```

```

interface child extends parent interface

```

```

{
}

```

```

Class implemented_class implements child_interface

```

```

{
//override and provide implementations
}

```

■ Multi-level

```
interface A {  
}  
interface B extends A {  
}  
interface C extends B {  
}  
Class implemented_class implements C {  
    //override and provide implementations  
}
```

■ Hierarchical Inheritance

```
interface A {  
}  
interface B extends A {  
}  
interface C extends A {  
}  
Class Demo implements B, C {  
}
```

■ Multiple Inheritance

```
interface A {  
}  
Interface B {  
}  
Interface C extends A, B {  
}  
class sampleclass implements C {  
}
```

■ Hybrid Inheritance

It is the combination of one or more inheritance like single, multi-level, hierarchical, multiple.

- A class can extends a class and then same class implements the interface, we can inherit both of them in a single class.

```
class A {}  
interface B {}    class C extends A implements B {}
```