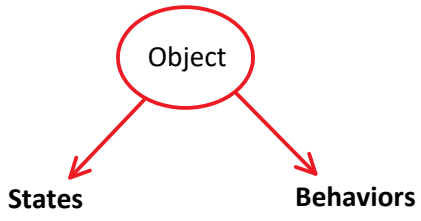
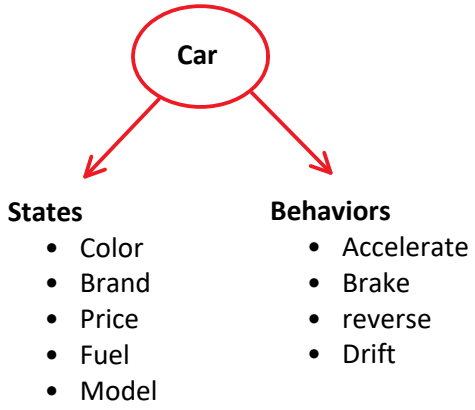
Objects

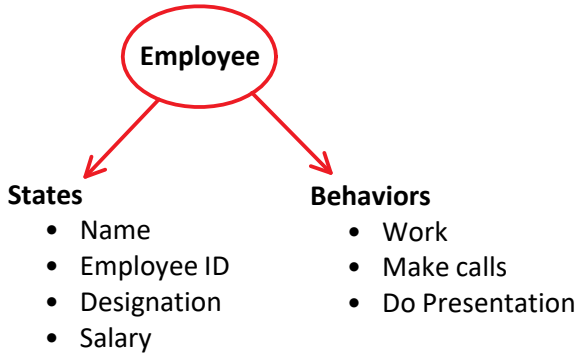
Object :

- **Object** is anything which has physical existence. in other words, **Object** is combination of **states** and **behavior**
- **Object** is a memory created in heap are to store **states** and **behaviors** which represent a real world object.
- If we want to create an **Object** a **Class** is necessary

ex .







States :
State is states or properties or attributes , data or information that describes an object.

Behavior :
Behavior is an action performed by object.

note : we can describe state of an object using global not-static variable
we can describe the behavior of an object using not-static methods

★ *note* : Static members are loaded in Class static area whereas non-static members are loaded into an object(which is in heap area).

Class : Class is a blueprint to an object.

Constructors

15 December 2022 13:59

Constructor :

- Constructor does not return anything not even **void**
- Constructor has the same name as the member Class
- Constructor is a non-static member of a **Class**.
- Constructor loads all the non-static member of a **Class** in an **object**
- Constructor is used to initialize the **states** of an **object**

*note : **At least one constructor** should be there **in a Class.***

- **There are two types of Constructors :**
  1. **Default Constructor :**
     - Default constructor exist in a class if there is no user defined constructor.
     - Default constructor has 0(zero) parameters and it is generated by compiler.

  2. **User-defined Constructor :**
     - Programmer written constructor (mostly to initialize the states) is called as user defined constructor.

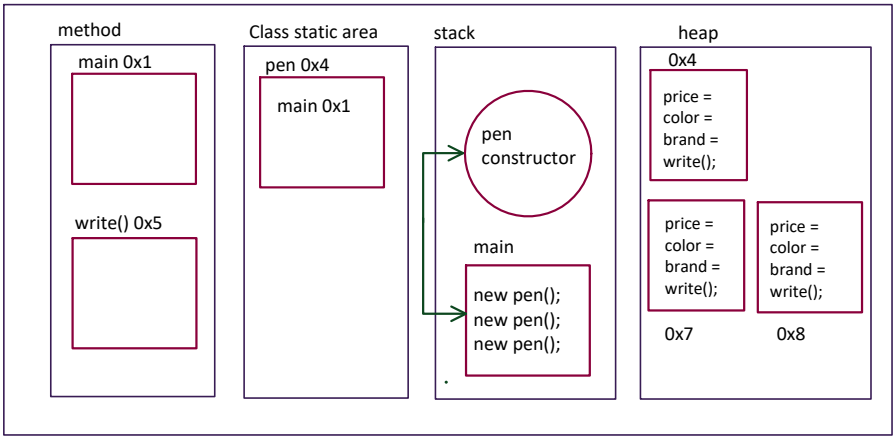
note : Default constructor & user defined constructor cannot exist at the same time.

Object Creation :

Syntax :

new ClassName();

-whenever JVM sees **new** keyword it creates an **object** in the **heap area**




```java
public class Pen {
    int price;
    String color,brand;
    //states

    public void write(){
        System.out.println("writing");
    }
    //behaviour

    public static void main(String[] args) {
        new pen();
        new pen();
        new pen();
    }

}
```

creating an object means calling a constructor

syntax : to create a constructor :

Class Name (parameters) ← *optional*

{

//statements

}

1. **Create a mobile object .**

```java
package com.TestPackage;

public class Mobile {
    int price,ram,rom;
    String color,brand,camerea;
    //states

    public void calling() {
        System.out.println("calling soomeone");
    }//behaviour



    public static void main(String[] args) {
        Mobile m = new Mobile();
        System.out.println(m); //address of object
        System.out.println(m.price);// 0
        m.calling(); //calling someone
    }
}
```

```
<terminated> Mobile [Java Application] C:\Program F
com.TestPackage.Mobile@34c45dca
0
calling soomeone
```

Object reference(an Identifier) **:**

- **Object reference** is the name given to identify an object
- not-static members can only be accessed with the help of object reference

1. **Write multiple Car Objects and initialize states of the object**

```java
package com.TestPackage;

public class Car{
    int price;
    String color,brand;
    //states

    public void drift(){
        System.out.println("Drifting");
    }
    //Behavior

    public static void main(String[] args) {
        Car car1 = new Car() ;
        car1.price = 20_00_000;
        car1.color = "White";
        car1.brand = "Tata";
        car1.drift();
        System.out.println(car1);

        Car car2 = new Car() ;
        car2.price = 60_00_000;
        car2.color = "White";
        car2.brand = "BMW";
        car2.drift();
        System.out.println(car2);
    }

}
```

```
<terminated> Car [Java Application] C:\Program Files\Java\j
Drifting
com.TestPackage.Car@34c45dca
Drifting
com.TestPackage.Car@52cc8049
```

Parameterised Constructor :

- The purpose of a constructor is to initialize the states of an object.
- A constructor is called Parameterized Constructor when it accepts a specific number of parameters. To initialize data members of a class with distinct values.
- In the example below we are passing an int, a double and two Strings to the constructor

```java
package com.TestPackage;

public class Car{
    int price;
    double milage;
    String color,brand;
    //states

    public void displayDeatails(){
        System.out.println(price);
        System.out.println(milage);
        System.out.println(brand);
        System.out.println(color);
    }
    //Behavior

    Car(int p, double m,String b, String c){
        price = p ;
        milage = m;
        brand = b;
```

```
<terminated> Car [Java Applicatio
750000
8.5
Kia
Red
670000
14.1
Tata
White
```

Test.java | Mobile.java | Car.java

```java
package com.TestPackage;

public class Car{
    int price;
    double milage;
    String color,brand;
    //states

    public void displayDeatails(){
        System.out.println(price);
        System.out.println(milage);
        System.out.println(brand);
        System.out.println(color);
    }
    //Behavior

    Car(int p, double m,String b, String c){
        price = p ;
        milage = m;
        brand = b;
        color = c;
    }
    //Parameterised Constructor

    public static void main(String[] args) {
        Car car1 = new Car(7_50_000,8.5,"Kia","Red");
        car1.displayDeatails();

        Car car2 = new Car(6_70_000,14.1,"Tata","White");
        car2.displayDeatails();

    }
}
```

Console

```
<terminated> Car [Java Applicatio
750000
8.5
Kia
Red
670000
14.1
Tata
White
```

note : Constructors unlike methods are called during Object declaration proceeding after the **new** *keyword*

This (keyword)

18 December 2022 10:27

This :

- **This** keyword is used to differentiate a local variable and a global non-static variable.
- **This** keyword refers to its current calling object
- **This** keyword cannot be used in a static method/block/context

Test.java Mobile.java Car.java Bottle.java

```java
package com.TestPackage;

public class Bottle {
     int price ;
     String color ;
     //states

     Bottle(){
         System.out.println(this + " printed from constructor");
     }

     public static void main(String[] args) {
        Bottle b1 = new Bottle();
        System.out.println(b1);
    }
}
```

Console

```
<terminated> Bottle [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (18-Dec-2022, 10:38:48 am – 10:38:49 am) [pid:
com.TestPackage.Bottle@34c45dca printed from constructor
com.TestPackage.Bottle@34c45dca
```

-**this** keyword is used to refer the current calling object's address you can differentiate a local variable and a global non-static variable using **this** keyword

Test.java Mobile.java Car.java *Bottle.java

```java
package com.TestPackage;

public class Bottle {
     int price ;
     String color ;
     //states

     Bottle(int price , String color){
         this.price = price;
         this.color = color;
     }

     public static void main(String[] args) {
        Bottle b1 = new Bottle(50,"Purple");

    }
}
```

Q. Create a Laptop Object and initialize the states using a constructor .

Test.java | Mobile.java | Car.java | Bottle.java | *Laptop.java

```java
package com.TestPackage;

public class Laptop {
    int price,ram;
    String brand,color;

    Laptop(int price,int ram,String brand,String color){
        this.price = price;
        this.ram = ram;
        this.brand = brand;
        this.color = color;
    }

    public static void main(String[] args) {
        Laptop l1 = new Laptop(45000,16,"HP","Grey");
        System.out.println(l1.price);
        System.out.println(l1.ram);
        System.out.println(l1.brand);
        System.out.println(l1.color);
    }
}
```

Console

<terminated> Laptop [Java Application] C

```
45000
16
HP
Grey
```

Constructor Overloading

19 December 2022 07:59

```java
package com.TestPackage;

public class Laptop {
    int price,ram,rom;
    String brand,color,model;
    String processor;
    boolean camera;

        Laptop(int price,int ram,int rom){
        this.price = price;
        this.ram = ram;
        }

        Laptop(String color,String brand,String model){
            this.brand = brand;
            this.color = color;
            this.model = model;
        }

        Laptop(String processor){
            this.processor = processor ;
        }

        Laptop(boolean camera){
            this.camera = camera;
        }

    public static void main(String[] args) {
        Laptop l1 = new Laptop(45000,16,512);

    }

}
```

- Having **multiple constructors** in the same class with **different signatures** is called as **constructor overloading**

Inheritance, Constructor Chaining & Diamond Problem

19 December 2022 08:05

Inheritance :

- Inheritance is the process of accessing of all the **states**/properties and **behaviours** from **parent class** to **child class**
- Inheritance is the process of one class acquiring properties and behaviours of another class

Parent/Super/Base class :

- Class which **gives** the properties and behaviour to another class is called as **Parent Class**

Child/Sub/Derived class:

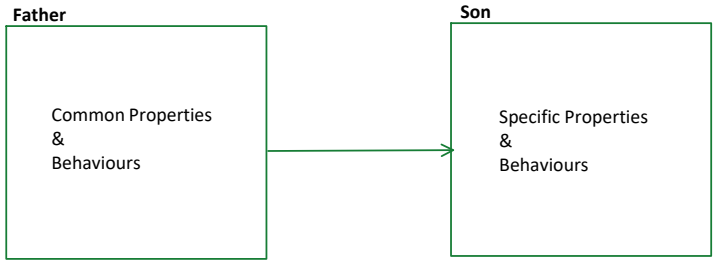
- Class which **takes** properties and behaviour from another class is called as **Child Class**

★ *note : In java we can achieve inheritance using **extends** keyword*
★ *note : Inheritance is unidirectional.*

notation used is

```
Class Father
{
//states //behaviours
}

Class Son extends Father
{
//
}
```

note : The parent-most Class, States and Behaviours are loaded first as properties and behaviours of the child-most class is loaded at the end.



Properties and behaviour **declared in parent class** are called as **common properties** or common behaviours
Properties and behaviour **declared in sub class** is called as **specific properties** or specific behaviours .

Test.java

```
package com.TestPackage;


class Test{
    public static void main(String arg[]){
        Son  s = new Son();
        s.age = 30; //common property
        s.name = "Mohit" ; //commoon property
        s.color = "Brown"; //common property
        s.gf = true ;//specific property
        System.out.println(s.age);//30
        System.out.println(s.color);//Brown
        System.out.println(s.gf);//true
        System.out.println(s.name);// Mohit
        s.work(); // Common  behavior//Father is working
        s.sleep();// specific behavior //Son is sleeping
    }
}
```

Console

```
<terminated> Test [Java Applicatic
30
Brown
true
Mohit
father is working
son is sleeping
```

Father.java

```
package com.TestPackage;

public class Father {
    int age;
    String name;
    String color;

    public void work() {
        System.out.println("father is working");
    }
}
```

Son.java

```
package com.TestPackage;

public class Son extends Father{
    boolean gf;

    public void sleep() {
        System.out.println("son is sleepin
    }
}
```

Constructor Chaining :

- Every Constructor should call another should constructor .
- We can call the parent Class constructor using **super()**
- You can call the same Class constructor using **this()**

note : **super** & **this** *are keywords whereas* **super()** & **this()** *are constructor calling statements*

- *rule 1: you can use either **this()** call statement or **super()** calling statement*
- *rule 2: one constructor can call only one another constructor*
- *rule 3: Constructor calling statements should be mandatorily be first line of conde in a constructor*

★ ***note :*** *It is always better to have a zero parameterized constructor in every class.*

Car2.java

```java
package com.TestPackage;

public class Car2 extends Vehicle{

    Car2(){
        System.out.println("Car constructor i
    }
}
```

Vehicle.java

```java
package com.TestPackage;

public class Vehicle {
    int price;
    String color,brand;

    Vehicle(){
        System.out.println("Vehicle Construct
    }


}
```

TestVehicle.java

```java
package com.TestPackage;

public class TestVehicle {

    public static void main(String[] args) {
        Car2 c = new Car2();
        System.out.println("Method is called last");
    }

}
```

Console

```
<terminated> TestVehicle [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (20-Dec-2022, 8:02:46 am – 8:02:46 am) [pid: 5180]
Vehicle Constructor is called first
Car constructor is called second
Method is called last
```

Car2.java

```java
package com.TestPackage;

public class Car2 extends Vehicle{

    Car2(){
        super();
        System.out.println("Car constructor zero parameter");
    }

    Car2(int x){
        this("hello");
        System.out.println("car constructor parameterized");
    }

    Car2(String s){
        super();
        System.out.println("car string parameterized");

    }
}
```

TestVehicle.java

```java
package com.TestPackage;

public class TestVehicle {

    public static void main(String[] args) {
        Car2 c = new Car2(10);

    }

}
```

Vehicle.java

```java
package com.TestPackage;

public class Vehicle {
    int price;
    String color,brand;

    Vehicle(){
        System.out.println("Vehicle Constructor zero parameter");
    }

    Vehicle(int x){
        super();
        System.out.println("Vehicle constroctor parameterized" + x);
    }


}
```

Console

```
<terminated> TestVehicle [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (20-Dec-2022, 8:32:3
Vehicle Constructor zero parameter
car string parameterized
car constructor parameterized
```

```java
package constructorChaining;

public class A {
    A(int x){//8th x=600
        this();
        System.out.println(x);//600
    }

    A(int x ,int y){//7th x=10,y=20
        this((x+y)*20);//600
        System.out.println("x+y");//x+y
    }

    A(){//9th
        super();
        System.out.println("BCQ");//BCQ
    }
}
```

```java
package constructorChaining;

public class B extends A {
    B(String x){//6th x="ok1"
        super(10,20);
        System.out.println(x);//ok1
    }

    B(String x,int y){//5th x="ok",y=1
        this(x+y);//"ok1"
        System.out.println(x+10+y);//ok101
    }

    B(int x){//uncalled ☹
        this("hey" + 10);
    }

    B(){//4th
        this("ok",1);
        System.out.println("Zero");//Zero
    }
```

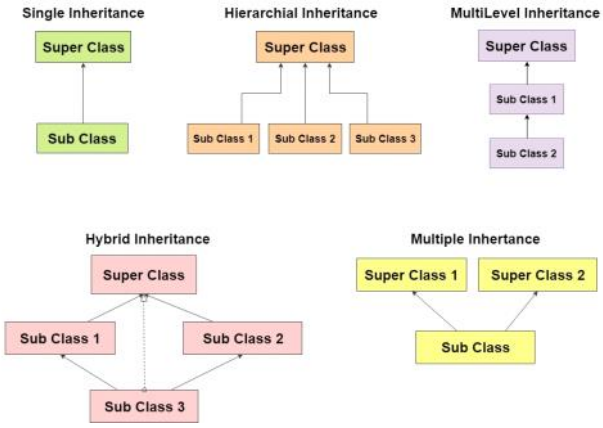
```java
package constructorChaining;

public class C extends B {
    C(int x, int y){//2nd x=10,y=20
        this(y);
        System.out.println(x+y);//30
    }

    C(int y){//3rd y=20
        super();
        System.out.println(y);//20
    }

    C(){ //1st
        this(10,20);
        System.out.println("ABC");//ABC
    }

}
```

Console

```
<terminated> TestChaining [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (20-Dec-2022, 1:24:04 pm – 1:24:04 pm) [pid: 8176]
BCQ
600
x+y
ok1
ok101
Zero
20
30
ABC
```

```java
package constructorChaining;

public class TestChaining {
    public static void main(String[] args) {
        C c = new C();
    }
}
```

write down types of inheritance (5 types) ;



1. **Single Level Inheritance**
   A Simple inheritance structure with one **parent** and one **child**
   A <- B

2. **Multilevel Inheritance**
   Chain of Single level inheritances
   A <-B <- C

3. **Hierarchical Inheritance**
   A **Parent** having multiple **children**
   A <-- B
   A <-- C
   A <-- D

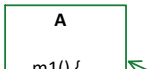
4. **Multiple Inheritance**
   Child having multiple **parents**
   A <-- C
   B <-- C

5. **Hybrid Inheritance**
   Mixture of all other inheritance levels
   A <-- C <-- B
   A <-- D
   A <- E

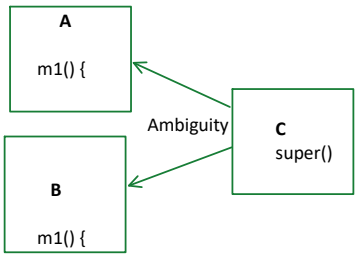
★ ***note IMP :***

Using Class we can achieve Single level, Multilevel and Hierarchical level inheritances only. We cannot achieve multiple level and Hybrid level inheritances using class due to diamond problem.

Diamond Problem :



Diamond Problem :



- When a Class is having multiple parents ,when object of child is created , parent Class constructors should be called. Since there are multiple parents, There is **ambiguity** for compiler to choose apparent constructor
- If there is a same method is present in two parent Classes there is **ambiguity** while calling the method, this kind of problem is called as **diamond problem**.
- due to diamond problem we cannot achieve multiple and hybrid level inheritance using Classes.
- This problem is sorted using **interfaces.**

note : Class is ***non-primitive data-type***

Non-Primitive Type Casting & Return Type

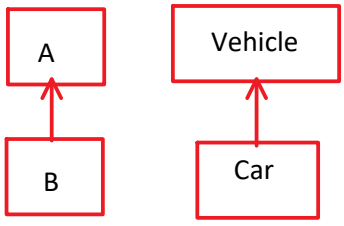
21 December 2022 08:20

Upcasting :

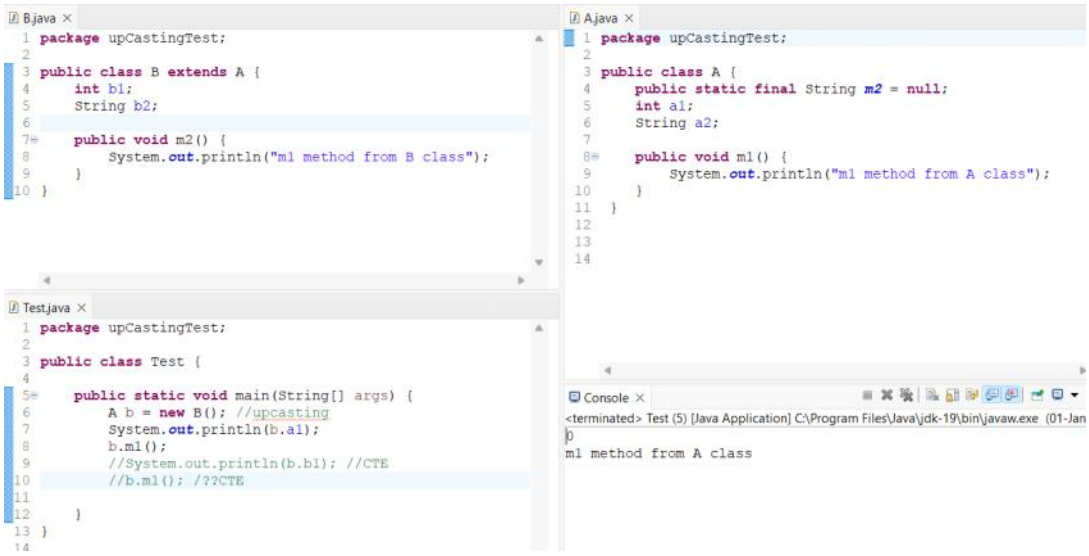
- Storing the **Child object** in **parent type reference**
- It is only possible in inheritance
- Disadvantage of upcasting we cannot access states and behaviours of the subclass
- Advantage of upcasting is **generalization**

eg.

1. **A a = new B();** *//upcasting*
2. **Vehicle v = new Car();**



// Upcasting a **new b**(child) object into **A**(parent) type reference



B.java

```
package upCastingTest;

public class B extends A {
    int b1;
    String b2;

    public void m2() {
        System.out.println("m1 method from B class");
    }
}
```

A.java

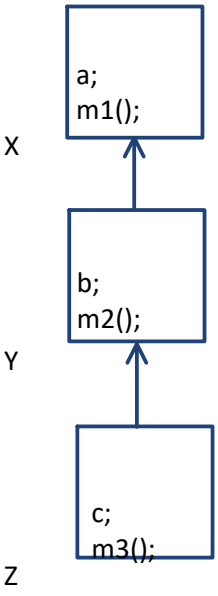
```
package upCastingTest;

public class A {
    public static final String m2 = null;
    int a1;
    String a2;

    public void m1() {
        System.out.println("m1 method from A class");
    }
}
```

Test.java

```
package upCastingTest;

public class Test {

    public static void main(String[] args) {
        A b = new B(); //upcasting
        System.out.println(b.a1);
        b.m1();
        //System.out.println(b.b1); //CTE
        //b.m1(); //??CTE

    }
}
```

Console

```
<terminated> Test (5) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (01-Jan
0
m1 method from A class
```




```
Y x = new Z();
System.out.println(x.b);
x.m2();
x.m1();
System.out.println(x.a);
System.out.println(x.c);//CTE
//x.m3(); //CTE
```

Console

```
<terminated> Test (1) [Java Application] C:\Program Files\Java\jdk-
0
m2 method from Y class
m1 method from Class X is called
0
```

Non-Primitive return Type Method

A.java

```java
package casting;

public class A {
    public static void main(String[] args) {
        int age = -1;
        if (age < 0) {
            System.out.println("invalid age entered");
            return;
        }else if (age > 17) {
            System.out.println("valid to vote");
        }else {
            System.out.println("valid age");
        }
        System.out.println("end");
        }
    }
```

Console

```
<terminated> A [Java Application] C:\Pro
invalid age entered
```

A.java

```java
package casting;

public class A {
    public static void main(String[] args) {
        int age = 15;
        if (age < 0) {
            System.out.println("invalid age entered");
            return;
        }else if (age > 17) {
            System.out.println("valid to vote");
        }else {
            System.out.println("valid age");
        }
        System.out.println("end");
        }
    }
```

Console

```
<terminated> A [Java Applicatio
valid age
end
```

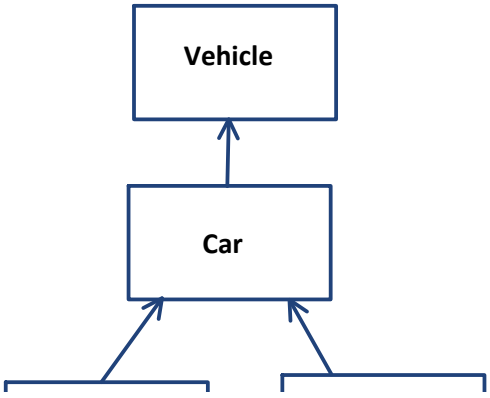
Non-Primitive Return type methods :

note : Any non-primitive datatype can store <u>three types of values</u>

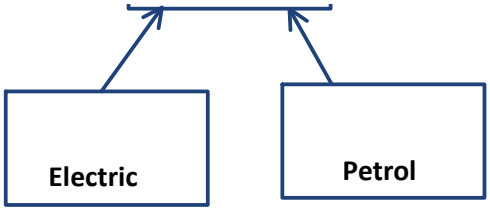
1. **null**
2. **object of its own**
3. **object of its child/subclasses**

syntax:

```
public Car discount( Car c )
{
        return new Car();
}
```

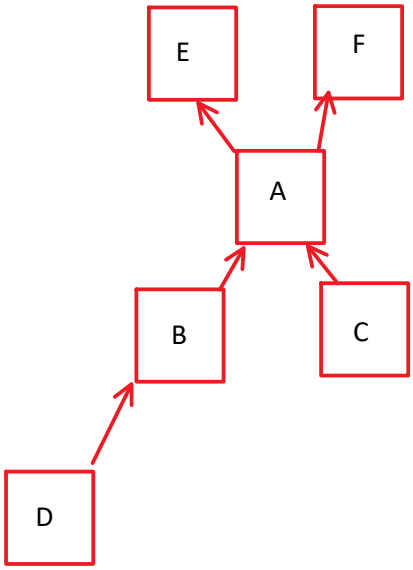



```
public Vehicle discount ( Vehicle() )
{
        return new Vehicle();
        return new Car();
        return new Electric();
        return new Petrol();
        return null();
}
```




```
}
```

Non-Primitive Method Binding :




```
public static void m1(A a)
{
//
}
public static void m1(B b)
{
 //
}
public static void m1(E e)
{
 //
}
```

1. Calling itself
2. Calling it's immediate parent
3. if itself method or parent are not present then its shows compile time error.

Down Casting :

- Converting a generalised object to its own type .
- The process of converting Parent type reference to child type reference.

Rules :

- Without upcasting you cannot downcast.

notes:

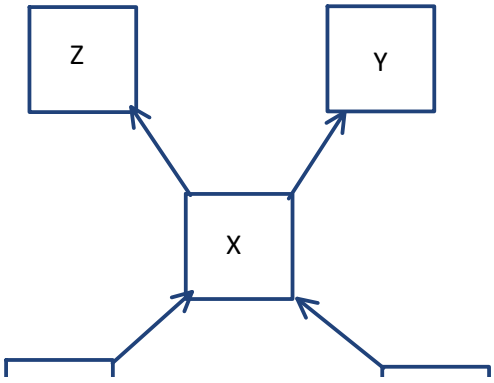
- *We downcast to access the properties and behaviour of child class. In other words down-casting is done to overcome the disadvantages of upcasting*
- *JVM throws you Class Cast Exception when you try to downcast without upcasting.*
- *An upcasted Object can be down-casted till its own type.*

eg.

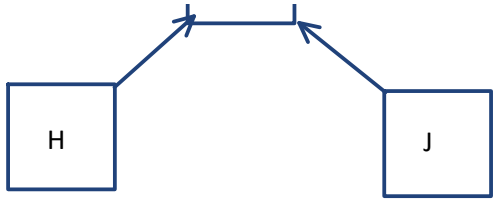
Z x = new J(); // Upcasting
H y = (H)x ; // Class Cast exception

Y y = new H();
H h = (H)y;

Y y = new J();



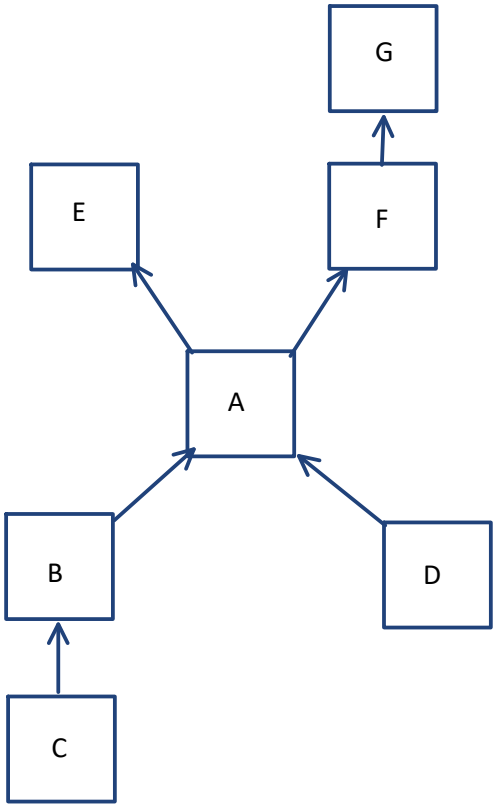
Y y = new J();
J j = (J)y;



eg2.

F f = new B() ;
A a = (A)f;
B b = (B)a;

G a = new A();
B b = (B)a ; //Runtime error



Method Overriding

Method Overriding :

- Method with same declaration but different implementation is called as method overriding
- Method overriding can be achieved only through inheritance.
- Method overriding means providing specific implementation in the child class for a parent behaviour

*Father.java

```java
package methodOverriding;

public class Father {
    public void drink(){
        System.out.println("Quarter");
    }
    public void eat() {
        System.out.println("VegS");
    }
}
```

Son.java

```java
package methodOverriding;

public class Son extends Father {
    public void  drink() {
        System.out.println("Full");
    }
    public void eat() {
        System.out.println("Non-Veg");
    }
}
```

Test.java

```java
package methodOverriding;

public class Test {
    public static void main(String[] args) {
        Father f = new Father();
        f.drink();
        f.eat();

        Son s = new Son();
        s.drink();
        s.eat();

        Father fs = new Son();
        fs.drink();
        fs.eat();


    }
}
```

Console

```
<terminated> Test (6) [Java Application] C:\Program Files\Java\jdk-19\bin\j
Quarter
Veg
Full
Non-Veg
Full
Non-Veg
```

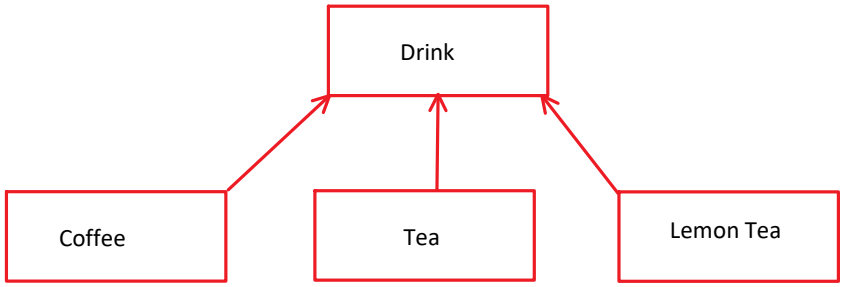
- overridden methods will be executed from child Class during upcasting

Generalization

26 December 2022 06:18

Generalization :

It is a Process of Providing common type for Child Class even during upcasting.



Here Coffee, Tea and Lemon Tea is also considered as Drink type. In other words Drink is common type for all other child types

Drink.java

```java
package generalization;

public class Drink {
 public void serve() {
     System.out.println("dink is served");
 }
}
class Coffee extends Drink{
    public void serve() {
        System.out.println("cofee is served");
    }
}
class Tea extends Drink{
    public void serve() {
        System.out.println("Tea is served");
    }
}
class LemonTea extends Drink{
    public void serve() {
        System.out.println("LemonTea is served");
    }
}
```

VendingMachine.java

```java
package generalization;

public class VendingMachine {
    public Drink getDrink(int option) {
        switch(option) {
        case 1:
        Coffee c = new Coffee();
        return c;
        case 2:
        Tea t = new Tea();
        return t;
        case 3:
        LemonTea lt = new LemonTea();
        return lt;
        }
        return null;
    }
```

Test.java

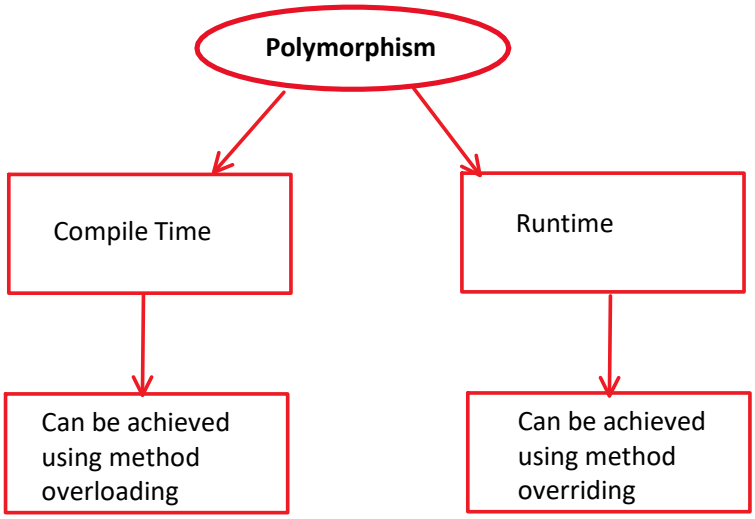
```java
package generalization;

public class Test {

    public static void main(String[] args) {
        int n = 1;
        VendingMachine vm = new VendingMachine();
        Drink d = vm.getDrink(n);
        d.serve();

    }

}
```

Console

```
<terminated> Test (2) [Java Application]
cofee is served
```

Polymorphism

26 December 2022 07:42



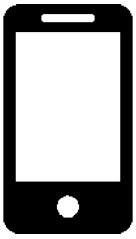
Polymorphism(general definition):

Anything taking multiple forms is said to be polymorphic in nature.

Compile Time Polymorphism / Early binding :

- The method binding process is happening at compile time.
- Compiler is going to bind the method statement with method implementation on the basis of arguments passed, hence the name compile time polymorphism .
- since, compilation happens first, it's also called as early binding.
- we can achieve compile time polymorphism using method overloading.

example :



1. pattern
2. pin
3. password
4. fingerprint
5. face

unlocking options ie. overloaded methods

```
private void unlock (int pin) {
        //some code;
}
private void unlock(String password){
        //some code;
}

Mobile m1 = new Mobile();
        m1.unlock(false); // Compile time error as there is no unlock method with a boolean parameter
        m1.unlock(1234)
```

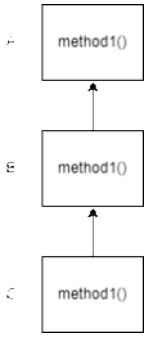
Run Time Polymorphism / Late Binding

- Method binding process is happening at runtime.
- The method call statement is associated with the respected method implementation on the basis of object

creation.

- Since object is created at runtime, it's also called as Runtime polymorphism.
- Since compilation happens first, and object is created later is called as **late binding.**
- We can achieve runtime polymorphism using method overriding.

example :




```
A a = new C();
a.method1() ;
```

Final keyword

26 December 2022 08:22

final :
final key word can be used with a variable , method and class.

Behaviour of final keyword with a variable.

- a variable declared with **final** keyword cannot change its value.
- in other words we cannot re-initialize the a final variable.
- final variable cannot have default value.
- in other words we have to initialize a final variable at the time of declaration only, else we get compilation error.

Final.java

```
public class Final {
    final static int b = 10;
    public static void main(String[] args) {
        final int a = 10;
        b = 1 ; //cte
        a = 30; // cte
    }
}
```

Console

```
<terminated> Final [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (26-Dec-2022, 8:30:54 am – 8
Error occurred during initialization of boot layer
java.lang.module.FindException: Error reading module: C:\Users\samwa\OneDr
Caused by: java.lang.module.InvalidModuleDescriptorException: Final.class
```

Behaviour of final keyword with a method.

- a method declared with **final** keyword cannot be re-implemented/overridden.
- final method cannot be overridden but can be overloaded.

*Final.java

```
public class A {
    public  final void m1() {
        System.out.println("A class");
    }
}

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

Behaviour of final keyword with a Class :

- a final class cannot be inherited.
- but a final class can inherit another class.

*B.java

```
public class B extends A  {

}
```

*A.java

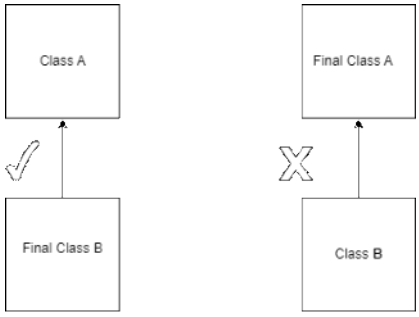
```
public final class A   {

}
```

*B.java

```
public class B   {

}
```

*A.java

```
public final class A extends B {

}
```



note : *a **final Class** members can only be accessed by its own objects .*

Access Modifiers

28 December 2022 08:52

- Access modifiers are keywords which defines the visibility of a method variable or a class.
- There are four important access modifiers
  1. **private**
  2. **default**
  3. **protected**
  4. **public**

1. Private :

- to make a constructor as **private** , we can create an object, only within the same class.
- private members can be accessed only within the same class.

2. Default

- Default access modifier is package level
- Class can also be default
- If any member of a class is default it can be accessed anywhere within the package
- If a member is not prefixed with any keyword it is set as a default
- if a class is default then it automatically becomes non-importable.

3. Protected :

- Protected members can be accessed within same Class, within the same package but no other class from
  different package can access it except for its child.
- We can access protected members in different packages only through inheritance.

example :

```
package groceries ;
public Class B {
                    protected static void m1(){
                            sopln("m1 method");
                    }
            }

package electronics ;
import grocerries.B;
public Class C extends B{
                    public static void main(String[] args){
                    B.m1();         //m1 method is called
                    }
            }
```

Public :

- Public is a project level element
- It can be accessed anywhere in the project.

	within class	within same package, different Class	different package , different Class
private	✓	x	x
default	✓	✓	x

protected	✓	✓	✓(with inheritance)
public	✓	✓	✓

Has-A relationship

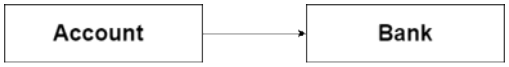
29 December 2022 08:07

Has - A relationship :

One Object having the dependencies on another object **Has - a** relationship can be achieved in two ways:

1. **Aggregation** : One object which has dependency on another object but also can exist/be functional without the other object.
   example1 : Mobile & sim
   here the Mobile object is dependent on sim but can still exist without the sim object
2. **Composition** : Dependency of two objects where one object cannot exist without another.
   example :
   a. Car & Engine
   b. Mobile & battery
   here the Car & Engine , and the Mobile & Battery Objects cannot exists without each other

   example :



Account.java

```java
public class Account {
    int AccountNumber,CustomerID;
    String AccountName,Address,Email;

    public void withdraw(int x){
        if(x < 0) {
        System.out.println("please enter a valid amount");
        }else{
        System.out.println(x + "Rs amount has been withdrawn");
        }
    }

    public Account(int accountNumber, int customerID, String accountName, String address, String email){
        super();
        this.AccountNumber = accountNumber;
        this.CustomerID = customerID;
        this.AccountName = accountName;
        this.Address = address;
        this.Email = email;
    }
```

Bank.java

```java
public class Bank {
    String name,branch,location;
    int pincode,IFSCcode;
    Account account;

    public Bank(String name, String branch, String location, int pincode,int IFSC, Account account) {
        super();
        this.name = name;
        this.branch = branch;
        this.location = location;
        this.pincode = pincode;
        this.IFSCcode = IFSC;
        this.account = account;
    }

    public void createAccount() {
        System.out.println("An account is created");
    }
```

Test.java

```java
package bank;

public class Test {

    public static void main(String[] args) {
        Account a = new Account(1234,402,"Sid","Goa","sid@mail.com
        Bank b = new Bank("YES","Goa","Goa",403,1441,a);
        b.account.withdraw(1200);
        b.createAccount();
    }

}
```

Console

```
<terminated> Test (3) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (29-Dec-2022, 4:14:29 p
1200Rs amount has been withdrawn
An account is created
```

Arrays

30 December 2022 08:21

main method with String[] as parameter will be invoked by JVM.

Q. What is use of String[] array as parameter in main method?
- To accept command line argument

Method Shadowing

27 December 2022 07:40

Static Method or Static Variable can be called

1. **Using Class name**
2. **Directly**
3. **Using Object reference**

note :

- Static methods cannot be overridden
- If there are methods in parent and child Class with same name/identifier but method belongs to class (static)
- Method binding happens on the basis of type of the reference and not upon the object created.
- Static methods are not polymorphic in nature(as they cannot be overridden).

note : **Object** is also called as **Instance**

note : if there are same static methods present in super class and sub class then method calling on the basis of type of object reference not on object creation.

example :

```
Class A{
        public static void m1(){
                //some implementation;
                }
        }

Class B{
        public static void m1(){
        //some implementation;
        }
}

Class Test{
        A a = new A();
        a.m1 // Class A method being called

        A a = new B();
        a.m1 //ClassA m1 method being called on the basis of the type of object reference
}
```

Packages

27 December 2022 08:13

Package :

- Package is nothing but a folder which is used to store java resources.
- Java resources are
  - Class
  - Abstract Class
  - Interface

Package in java can be categorized into two forms :

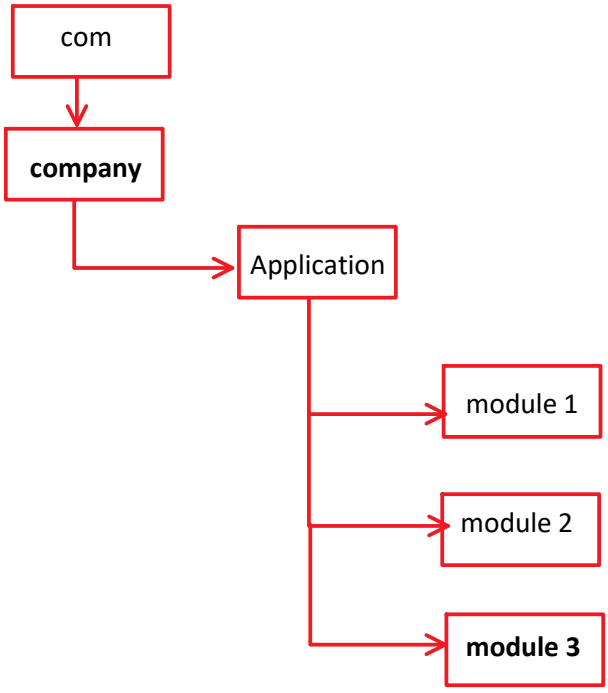
1. Inbuilt
2. User-defined

there are several inbuilt packages, util, lang etc

Advantages of Packages :

- Java package access protection
- java package removes naming collision
- we can achieve better maintenance of a project
- accessibility and searching becomes easy and fast
- we can achieve modularity

Standard package Structure :



note : Package should be the first line of the code of any .java filed

*ex . **package** shopping.footware.casuals ;*

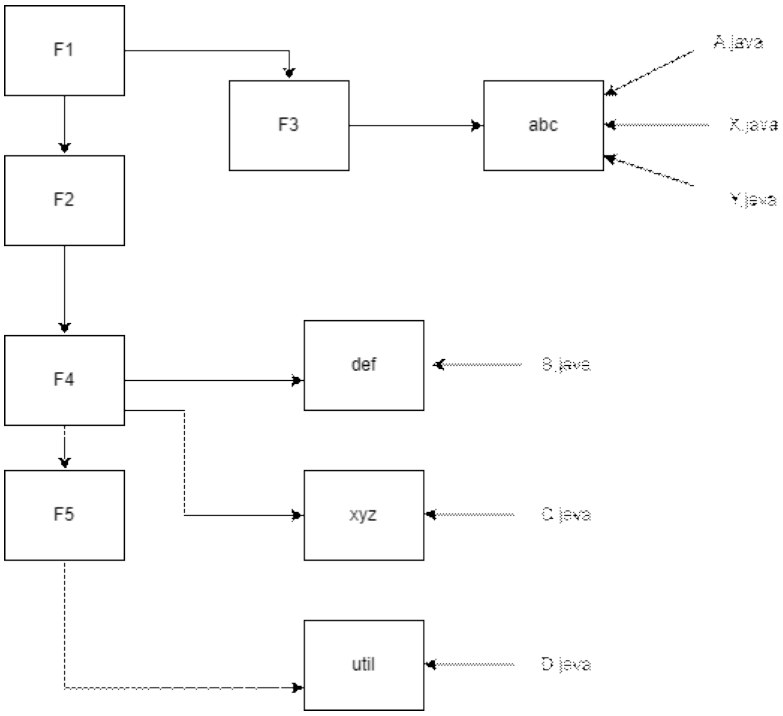
- Package declaration should be first line of code in any .java file
- *note : if we want to make any resource available in a different package then we have to write an **import** statement .*
  *ex. **import** grocerries.B*
- we can write **any number** of import statements.
- **Package declaration and import statement should start from the parent most package**

note

a. folders have slash(/) as a separator
b. packages have dot(.) as a separator

note

A fully-qualified class name in Java contains the package the class originated from.



note :
if you want to use all the 3 files from package abc then you can use * (asterisk) symbol.
ie. eg B Class package f1.f2.f4.def;
import f1.f3.abc.*

example :

```
package f1.f2.f4.f5.util;
import f1.f3.abc.x;
Class Any{
//
}
```

- package declaration and import statement should start from parent most package
- we cannot write class name while writing package
- In A.java file I import C file from package but I cannot use C file in y.java file

Scanner Class

30 December 2022 08:35

Scanner Class :

- it is a class present in java.util package.
- Scanner is an inbuilt class which is used to take input from the user.
- Scanner Class is a final Class.

byte	*nextByte()*
short	*nextShort()*
int	*nextInt()*
long	*nextLong()*
float	*nextFloat()*
double	*nextDouble()*
String	*next()*
String	*nextLine()*

all these methods are non-static methods

Scanner object is used to read input from the user at runtime

Test.java

```java
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter the data");
        byte b = sc.nextByte();
        System.out.println(b);
        System.out.println("program ended");
    }
}
```

Console

```
<terminated> Test (4) [Java Applicatic
enter the data
43
43
program ended
```

Difference between next() and nextLine() :

next() reads input till a space

nextLine() reads the input including space

in other words next() is used to read a word & nextLine() is used to read a sentence

Test.java

```java
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter the data");
        String b = sc.next();
        System.out.println(b);

    }
}
```

Console

```
<terminated> Test (4) [Java Applic
enter the data
Hello World
Hello
```

Test.java

```java
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter the data");
        String b = sc.nextLine();
        System.out.println(b);

    }
}

```

Console

```
<terminated> Test (4) [Java Applicat
enter the data
Hello World
Hello World
```

Q. WAP to take user input and print factorial of the input number.

Factorial.java

```java
import java.util.Scanner;
public class Factorial {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        int facto = 1;
        for(int i = 1 ; i <= num ; i++) {
            facto = facto * i ;
        }
        System.out.println("Factorial of " + num + " is " + facto);
    }

}
```

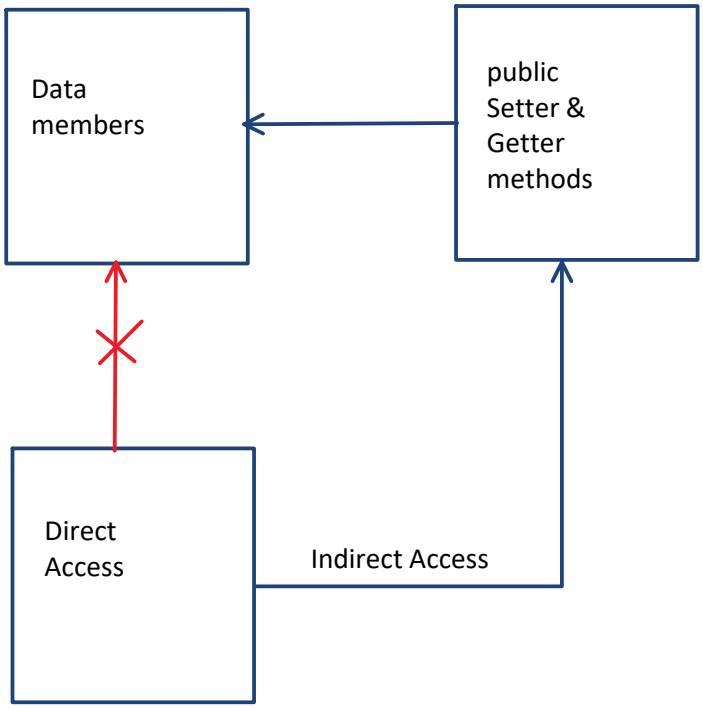
Console

```
<terminated> Factorial (1) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (30-Dec-2022, 1:58:45 p
Enter a number: 5
Factorial of 5 is 120
```

Encapsulation

03 January 2023 07:58

Encapsulation : Encapsulation is a data hiding process



Encapsulation :

- Encapsulation is the process of binding or wrapping up of data members along with it's data handler methods(getter & setter).
- Encapsulation is a data hiding process using public getter and setter methods.
- Encapsulation is process of binding states and behaviours of an object

*note: A class with data members as **private** and methods which are binded with these **private** data members, is called as **java bean** class*

- *A **java bean** class is a Class whose data members are protected.*

Advantages :

- Data security / we can protect the data from unauthorized access.
- We can achieve data validation.
- We can achieve data read only or write only.

example :

```java

public class BankAccount {
    private int balance ;
    private long accountNumber;
    private String ifsc;
    private int transactionAmount;
    //balance is made read Only
    public int getBalance() {
        return balance;
    }

    //read only
    public int getAccountNumber() {
        return accountNumber;
    }

    //both read and write
    public int getTransactionAmount() {
        return transactionAmount;
    }

    //both read and write
    public void setTransactionAmount( int transactionAmount ) {
        this.transactionAmount = transactionAmount ;
    }
}

```

Specifications for Java Bean Class:

1. Class should be public and non-abstract
2. Data members should be private
3. Each data member should have setter or getter methods
4. Java bean class should have default constructor

Concrete Method :

04 January 2023 07:47

Concrete Method/Non-Abstract Method : is a method with method declaration and method implementation

ex.

```
void main()
    {
        // some implementation
    }
```

note : we can have concrete method in class, abstract class, interface
in other words concrete method is a complete method or implemented method.

Abstract Method : Abstract method is a method with just declaration but not with implementation
In other words Abstract method is an incomplete method or unimplemented method , without method implementation.

ex.

```
void main()
```

Rules to Declare an Abstract Methods :

1. prefix **abstract** keyword
2. terminate the method with a semicolon (;)

   ex.

```
abstract void m1();
abstract String join();
abstract int add(int x, int y);
```

Abstract method is a normal method which can have a return type but not implementation.

- you can have an **abstract method** in abstract Class or interface only;

Q. Can we overload abstract methods ?

- Yes ,we can

Q. Can we override abstract methods ?

- Yes, it is mandatory

Concrete Class / Non-Abstract Class

04 January 2023 08:12

Concrete Class : Concrete Class is a complete class which has only implemented methods

eg

```
Class A {
        void m1(){

        }

        void m2(){

        }
}
```

Abstract Class : Abstract class is an incomplete class with unimplemented method or concrete methods or both.

ex.

```
abstract class A {
        abstract void m1();   //abstract method m1

public static void m2();   //concrete method m2
        {

        }
}
```

Points to remember :

- **abstract** keyword has to be prefixed to a class
- **abstract** Class can have abstract methods , concrete methods or both.
- we cannot create an Object of an abstract Class.

Abstraction

04 January 2023 08:31

Abstraction :

Abstraction is the process of hiding the implementation and providing necessary functionalities.

- we can achieve using abstract classes and interfaces

Advantage of Abstraction :

- Security
- Loose Coupling (change in implementation which does not affect users) ;

Abstract Class is an incomplete class which has to be completed(implemented completely using child class)

Animal.java

```java
package abstraction;

public abstract class Animal {
    public abstract void noise();
    public abstract void eat();
}
```

Lion.java

```java
package abstraction;

public class Lion extends Animal {
    public void noise() {
        System.out.println("roar");
    }
    public void eat() {
        System.out.println("Carnivore");
    }
}
```

Cow.java

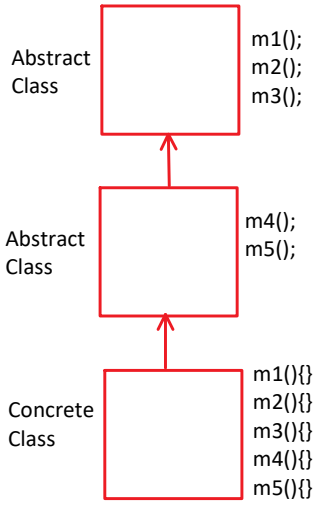
```java
package abstraction;

public class Cow extends Animal {
    public void noise() {
        System.out.println("Moo");
    }
    public void eat() {
        System.out.println("Herbivorous");
    }
}
```

Test.java

```java
package abstraction;

public class Test {
    public static void main(String[] args) {
        Animal a = new Lion();
        a.eat();
        a.noise();
        Animal c = new Cow();
        c.eat();
        c.noise();
    }
}
```

Console

```
<terminated> Test (8) [Java Application] C:\Program Files\Java\jdk-19\bin\
Carnivore
roar
Herbivorous
Moo
```



A & B are abstract classes and **C** is a concrete class.
We can complete all 5 methods in **C** class.
C can access all the methods of class A and class B.

Q. **Can a abstract method be final , static or private**

- No, because abstract method has to me overridden mandatorily
- Since we can have implemented/concrete methods in abstract class we cannot achieve 100% abstraction using abstract Class.

note- *abstract Class is an incomplete Class which is completed (implemented completely) in child class*

Rules :

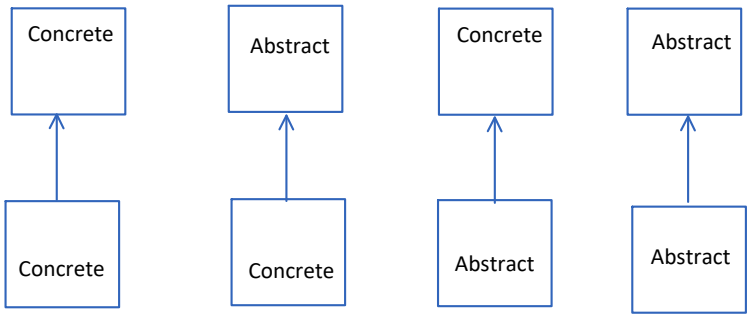
- Whenever a Class is extending to an abstract class, either all the unimplemented methods has to be implemented into the child/sub class or sub Class should also be abstract.
- Since we can have concrete methods in abstract class and it can be accessed by its child, a constructor in the abstract class should be there to load it.

- an abstract class can have static and final concrete methods.

Abstract Class	Concrete Class
Abstract Classes can have both abstract and concrete methods	Concrete Classes can have only concrete methods (even a single abstract method will make the class abstract
Abstract Class cannot be instantiated ie. we cannot create an object	Concrete Class can be instantiated/ ie we can create an object
Abstract Class cannot be final	Concrete Class can be final

Similarities between Abstract and Concrete Classes

- Both are Classes, which means non-primitive data types.
- Both the Classes can have constructor
- Both the Class can inherit another Class



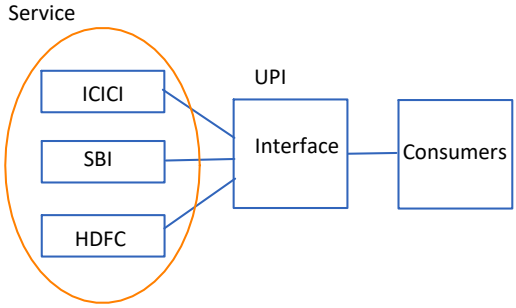
note :

- We can achieve partial abstraction using abstract class but we cannot achieve multiple inheritance
- We can achieve 100% abstraction using interface

Interfaces

04 January 2023 09:07

Interface : is not a class but a blue-print to a class



Interface is a medium between service and consumer

Syntax to create an interface :

```
interface InterfaceName{

}
```

User View vs Compiler implementation on Interfaces

```
1
2 public interface Switch {
3     int i = 10 ;
4     void on();
5     void off();
6
7 }
8
```

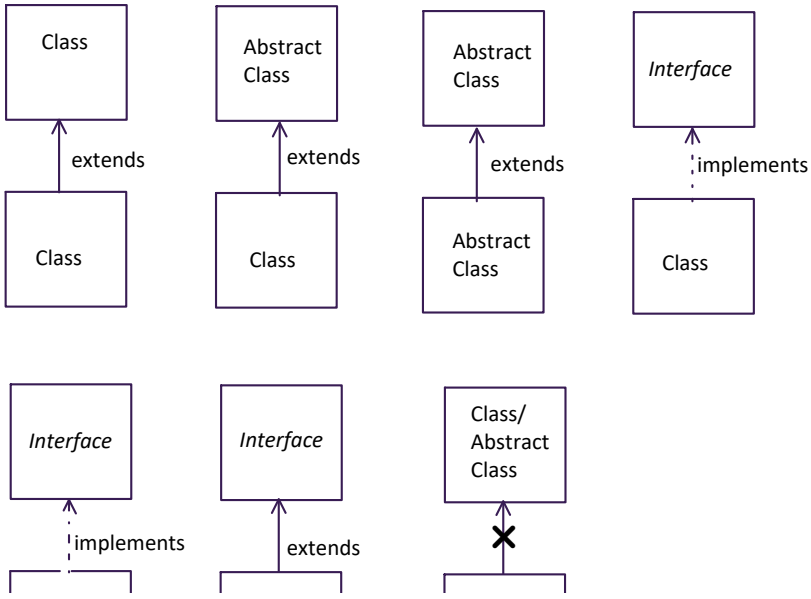
user view

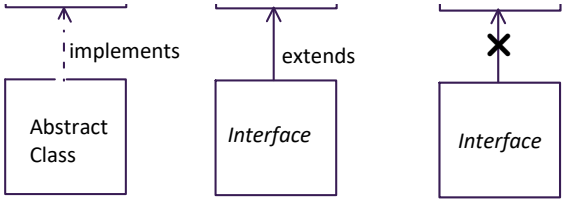
```
1
2 public interface Switch {
3     public static final int i = 10 ;
4     public abstract void on();
5     public abstract void off();
6
7 }
8
```

compiler view

- the default declaration of method is **public** and **abstract** in Interfaces
- the default declaration of a variable in an interface is **public**, **static** and **final**
- we cannot create an object of interface but interface is a datatype(non-primitive/user-defined)

- we can inherit any interface using **implements** key-word
- interface can only be implemented by a class
- a Class cannot extend an interface
- we represent implements using dotted arrows

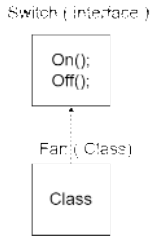




note : A class cannot extend to multiple Classes but interface can extend to multiple interfaces.

example 1 :

```java
public interface Switch {
    void on();
    void off();

}

class Fan implements Switch{
    public void on() {
        System.out.println("Fan on");
    }

    public void off() {
        System.out.println("Fan off");
    }
}
```



*Switch.java

```java
interface Switch {
    void on();
    void off();
}

interface Regulator {
    void incSpeed();
    void decSpeed();
}

class Fan implements Switch,Regulator{
    public void on() {
        System.out.println("Fan on");
    }

    public void off() {
        System.out.println("Fan off");
    }

    public void incSpeed() {
        System.out.println("Speed Increased");

    }
    public void decSpeed() {
        System.out.println("Speed decreesed");
    }
}
```

Test2.java

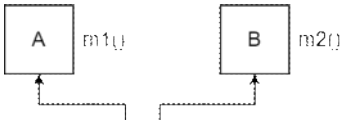
```java
public class Test2 {
    public static void main(String[] args) {
        Fan f = new Fan();
        f.on();
        f.off();
        f.decSpeed();
        f.incSpeed();
```

Console

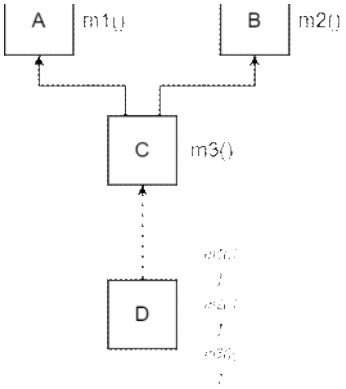
```
<terminated> Test2 [Java Appl
Fan on
Fan off
Speed decreesed
Speed Increased
```

example 2 :

```java
interface A {
    void m1();
}

interface B {
    void m2();
}
```

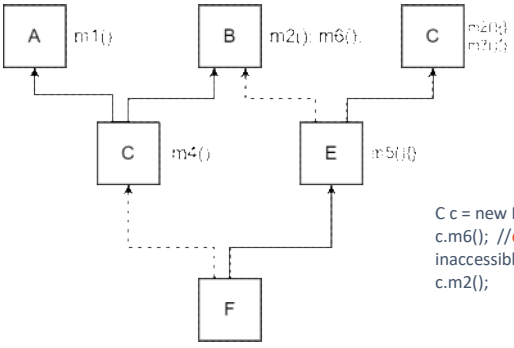


```java
}

interface B {
    void m2();
}

interface C extends A,B{
    void m3();
}

class D implements C {
    public void m1() {
        System.out.println("m1 method");
    }
    public void m2() {
        System.out.println("m2 method");
    }
    public void m3() {
        System.out.println("m3 method");
    }
}
```



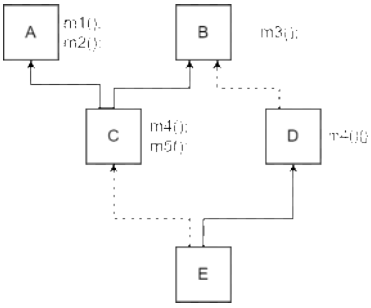
note : *A class cannot extend two classes(because of diamond problem) but an interface can extend two interface.*

example 3 :



C c = new F();
c.m6(); //compile time error as m6() is inaccessible due to upcasting
c.m2();

example 4 :



C c = new E();
c.m4();

Questions

06 January 2023 11:33

1. WAJP to find the absolute value, take the user input
2. WAJP which defines a method to determine whether a number is prime number or not
3. WAJP to print prime numbers from m to n
4. WAJP to define a method which returns true if a number is strong.
5. WAJP to print all the strong numbers from 1-100 .
6. WAJP to find if the number is armstrong number or not
7. WAJP to print armstrong number from 1-10000.
8. WAJP to find if a number is a disarium number or not.
9. WAJP to print disarium number from 1-1000.
10. WAJP to find if the number is xylem or not.
11. WAJP to print xylem numbers from 1-1000.
12. WAJP to print if a number is happy number or not.
13. WAJP to print happy numbers from 1-1000.
14. WAJP to find the number of digits in a number.
15. WAJP to calculate the average of the digits of the number.
16. WAJP to find how many times a digit is present in a number.
17. WAJP to convert decimal to binary.
18. WAJP to convert decimal to octal
19. WAJP to convert decimal to hexa-decimal
20. WAJP to convert binary to decimal
21. WAJP to convert octal to decimal
22. WAJP to convert binary to hexa-decimal
23. WAJP to convert octal to hexa-decimal
24. WAJP to convert octal to binary
25. WAJP to convert binary to octal
26. WAJP to find the remainder without using modulus operator
27. WAJP to reverse a number.
28. WAJP to check palindrome number
29. WAJP to print all 3 digit palindrome numbers
30. WAJP to print all 4 digit palindrome numbers