Class

↳ defines the structure and working of object

object - real instance

→ It is used to access the class

→ every obj has state and behaviour (wt knows, wt does)

for that we use Class as blueprint

---

Final Keyword

→ It can be used with variable, method, class

* final int a = 5; || this value not be changed

(or)

* final int a;

a = 5; || only once we can assign the value

→

```
class main
  {
PSV  main( string arg[])
   {
     B obj = new B ();
      obj.show();
     }
   }
```

→ now we create object for B so actualy, hello can be printed But when use Final keyword in A then hi can be printed and also we not overwrite the same fn name in B.

```
Class A
  {
final public void show()
    {
      System.out.println("hi");
    }
   }
```

```
Class B extends A
 {
   public void show()
    {
    system.out.pr("hello");
    }
 }
```

```
final class A    - we not inherit this class with other.
  {


  }
```

now

→ whenever printing an object

O/P

object . Student @ 15 db97

object class — class name — hexcode (or) hashvalue — alphanumeric

so here wt happen

whenever object is created it automatically calls

ie) every class in Java is the child class of object class

→ So that In object class

```
public String toString() {      - this automatically called
      (returntype)                if we not known and write the
                                  method
return getClass(). getName () + "@" + Integer.toHexString (ha

}
```

→ ie) we get string representation of object:

→ So that

```
Class B          here we override                 Class A
                 that method                       main ()
{                                                  {
public String toString() <------------------------ B b = new B()
  {                                                printf(b);
  return "hello";
                                                   }
  }
}
```

So here we also

return name+" "+ID;

else In this object answer we return

Note
package object;

~~Abstract Class~~

, not specifiers

## Access Modifiers in Java

i) public, protected, private, Abstract, final

```
class A
{

}
```

→ It be default class i.e) it is visible only to that package while

```
Public Class A
{

}
```

→ It be visible to anywhere i.e) outside the package and within that package

```
Private class A
{

}
```

→ It is not be used But private - specific class only be used

Protected —— i.e) protected be used by subsiding class.

```
Public Class A
{
  Protected int i;
}
```

```
Class B extends A
{
public void show()
{
System.out.println(i);
}
}
```

So that

Class not be private and protected ~~mostly~~ ~~not~~ not used

# Abstract modifier

```
Public class A
{
  Public static void main (String arg[])
  {
   D ob = new D();
     ob.call();
     ob.cook();
     ob.dance();

   }

}
```

- where the class be abstra
  but also the method is
  not be abstract

- within class the method
  is abstract if it is
  declared but not defi

```
abstract class B
(4)      {
         Public void call()
          {
           System.out.println(" calling ");

          }
          Public abstract void move();
          Public abstract void dance();
                       (2)
          }
```

.. note
- in abstract class we
  not create object
  (not possible)

(1)
so here we only
declare not define
so it be abstract

whenever
the
methods
inside a
class is
abstract
the class
also abstract
(3)

```
abstract class C extends B
   {
    Public void move()
      {
     S.O. print ("move");
      }
    }
```

Class B and C
are abstract
class.

```
class D extends C
  {
   public void dance ()
    {
    printf ("dance");
    }
  }
```

— It has all the
methods with its
definition

It is called
Concreate class

(15)

```
public class A
{
public static void main (String arg[])
 {
   B b = new B( );   show (b);
   C c = new C( );   show (c);
 }
 public static void void show ( B obj)
  {
    obj . print ( );
  }
}
```

if it is static
it can access directly

i) here we create object of both class and pass the object as parameter to both.

```
public ~~static~~ static void show (C obj)
  {
    obj . print ( );
  }
```

how we write in single method

Note
These two are different because of parameter

```
public class B extends D
{
  public void print ( )
   {
     printf (" Hi ");
   }
}
public class C extends D
{
  public void print ( )
   {
     printf ("Hello");
   }
}
```

• for that we create a abstract class

```
abstract class D
{
  public abstract void print ( );
}
```

2.

In main

```
public static void show (D obj)
 {
   obj . print ( );
 }
```

object of super class

here the object is passed

show( b )
can be passed

## Interfaces in Java

→ It is similar to abstract class

→ difference is here

i) No atleast one ~~method~~ can be defined inside

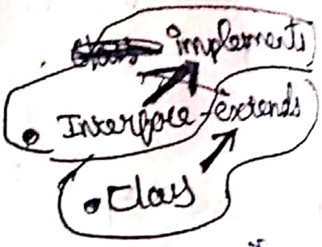
ii) when we inherits the Interface we should define for all m present in Inte

→ object for Interface not can be created.

↳ but Indirectly possible (i.e) we ~~gave~~ memory only for that can declare in the interface

→ In Java multiple Inheritance is not directly supported for that Interface we go

Note



Inherit

Note

→ Interface can only extends

e.g)

```
Class C implements A, B
{

}
```

both are Interface

by default so no need to mention

also we can write

Syntax

```
interface A
{
    int a=10;
    ~~Public~~ void Show();
}
```

public abstract ~~int a;~~

It by default public static Final

~~also int a;~~ assign only once

This is also security features.

Note:

```
interface A
{
    Public void Show();
}
class B implements A
{
    Public void Show();
    {
        Print("hi");
    }
    Public void hi()
    {
        Print("hi")
    }
}
```

So now in B

it can able to a both ie) show() hi()

```
B b=new B();

A b=new B();
```

here we give only the memory for the variables methods present in Interface

# multipleinheritance

i)

```
interface A          interface B          Class C implements A,B
{                    {                    {
                                          ~~[illegible]~~
                     }                    }
}
```

ii) Two ways of creating object for an interface indirectly

i) way we see before

2) way ii , creating annomious class
i.e) class have no name
ie)

```
interface A          Class B Implements A
{                    {
  void show()          A a = new A() {
}                        Public void show()
                         {

                         };
                       a.show();
                     }
```

This is called annomous class

i.e) whenever give definition that possible only in class, so there is no name for that class so called annomous class

Types of Interface

1) Marker Interface - with no methods

2) SAM - Single Abstract Method (or) functional Interface
with one method

If only one method then we directly use lambda method

i.e)

```
interface A          Class B
{                    {
  void show()          A a = () -> { System.out.println("in show")};
}                      a.show();
                     }
```

only like so we no need this.

```
interface A
{

}
interface B
{

}
interface C extend A, B  ✓
{

}
```

Note

- In main method having class in that only one class i.e) main class should be public
- also interface is Not public in class of same package

~~Constructors~~

## object Creation detail

The reference is stored in stack memory

A obj = new A(); — constructor — it how much memory we need

new — keyword to create object in heap memory

In Java

object is Called instance

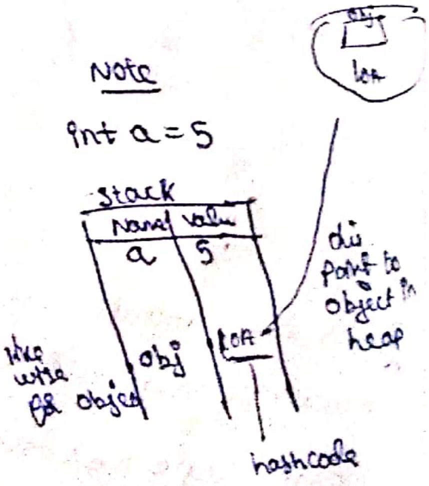
- obj = new A();

again this wt happen

new — it create new memory in heap with different hashvalue and old obj get goes to Garbage collection (deleted)

Note

int a=5



- obj=null; — also obj get deleted

## Constructor

Note

→ In Java there is no Destructor instead Carbage collector automatic.

```
Class A
{
 int a;
 int b;

}
```

```
Class B
{
 Psvm()
 {
  A a = new A();
 }
```

it is constructor But in A class where is constructor?
↓
(e)
By default Java create constructor. Called default constructor.
On creating the memory

Notes:

→ Constructor name same as class name
→ It does not return any value
→ It does not call, it automatically called while object is creating.
(by us)

```
Class A
 {
 int a;
 int b;
Public A()
 {
  a=1;
  b=2;
  System.out.println(" ");
 }
 Public B(int a)
 {


 }

 }
```

By the parameter we differentiate constructor during the object creation

Note

1) If the parametrized constructor is we create once, then we should create the default constructor also, then only we able to call.

2) If we have variable a in class also as a parameter of a fn in same class

```
Class A
{
int a;
void fn(int a)
{
this.a = a
}
```

return type important (should specify)

this is a class variable

## Inheritance

## Note

IS-A - inheritance

has-A - The class when relationship object created

→Inheritance provide reusability of code.

```
Interface A
{
  int a=1;
}
Interface B extend A
{
  int a=2; ——  —— this is the value when a is displayed in another class (e)
}
```

In Interface the value inside that interface is final in also it is inherited

```
✓ Class C implements B
  {
     prtf (Ent.) —— 2
  }
(but) Class C implements A,B
  {
  }
```

- so now it is not possible to display a

1) singlelevale Inheritance

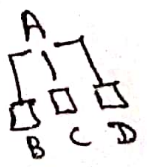
```
Class A          — super class (or) parent (or) Base class
{
}
Class B extends A
{
}
```

sub class (or) child class (or) derived

2) Multilevel Inheritance

```
class C extends B
{
}
```

3) hierachical Inheritance (ie) one class serve as the superclass for more than one class)



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

# Setter and Getter

**Note:** just a normal method like but used to access private variable

```
publicClass main {
public static void main(string arg[])
  {
     A obj= new A();

     obj. SetName (5)

     System.out.println(obj.getName())

  }
```

**Note**
if private
extend also
not work

```
Class A
 {
  private int a;


  public void setName(int a)
   {
     this.a = a;
   }
   public ~~void~~ int getName ()
     {
      return a;

     }

  }
```

— so it is private we use only within this class.
i.e) we create object of this we not able to access a.

## Encapsulation

→ Getting and setting the value using methods.

→ object knows → variable
object does - function

## Abstraction

Just create the abstract and implement using that abstract

eg) Phone — Iphone, Samsung } having same functionality e.g
| abstract

ie) we pass the spectic object but the function parameter is abstract

## Polymorphism

→ many forms

→ Same with different behaviour like

Note

By defaul
all variabl
in class is
0
except
in inte

Note

when we
combine b
word
here
capital