OOPS! → work with real life entities
→ divides entities into Modules

Class
- State (Variables)
- Function (Methods)

Adv
1.) Modularity
2.) Scalability
3.) Data hiding
4.) Real life Scenario

Class → representation specifying characteristics and behaviour of an object.

Object → instance of class

```
class Customer
{ public String customerID;
}
```

→ Each customer would have different values for each instance. ∴ variable is called as instance variable

ACCESS MODIFIER

private → accessible only inside class
public → all other classes
protected → via protected inheritance in classes
(Same package or subclasses in different package)

Methods → provide behaviour of an entity

Que → If no object is made of a class. will it take memory?

YES, Small amount

1.) Class Meta data → Structure, methods, fields etc.
2.) Static Member → takes memory even if no object is made.
3.) Method code → Byte code of Methods are stored.

Customer customer = new Customer();

↑ reference variable — ↑ new keyword is responsible for creation.

default value is given into instance variable

∴ boolean flag
↓ default
false

Method → public void Bill(String username)

public: access modifier; void: return type; Bill: Method Name; String username: Parameter

values passed → actual Parameter
arguments in header → formal Parameters

{ . . .

return ——;

} ↑—— only one value is returned

(local Parameter = formal parameter)

Variables declare inside Methods ⇒ local variable

Pass By Value

passed value is copied from actual Parameter → formal Par.

∴ different memory location

⟷

Value changed in Method

↓

No impact

Pass by Reference

Object is Passed as Parameter

∴ formal ⊕ actual → same location

⟷

Change is visible.

Constructor

If there are n customer initializing each instance variable will be very difficult.

∴ Use constructor

↓

Special Java Method → initializes class variable at-time of object creation.

<access modifier> <name> (parameter)
{
  //Body
}

public Customer()
{
}

> parameterless constructor ⟶ no arguments
↓
for initializing default values

User create नहीं करेगा तो Compiler करेगा !!

> Parameterized Constructor ⟶ • accept Parameter.
• assigned to instance Variables

Ex → Customer obj = new Customer("C103", "Jacob

* Multiple constructors can be made in an Class
  > Parameterized
  > No Parameter
  > Partial Parameter
  } Polymorphism.

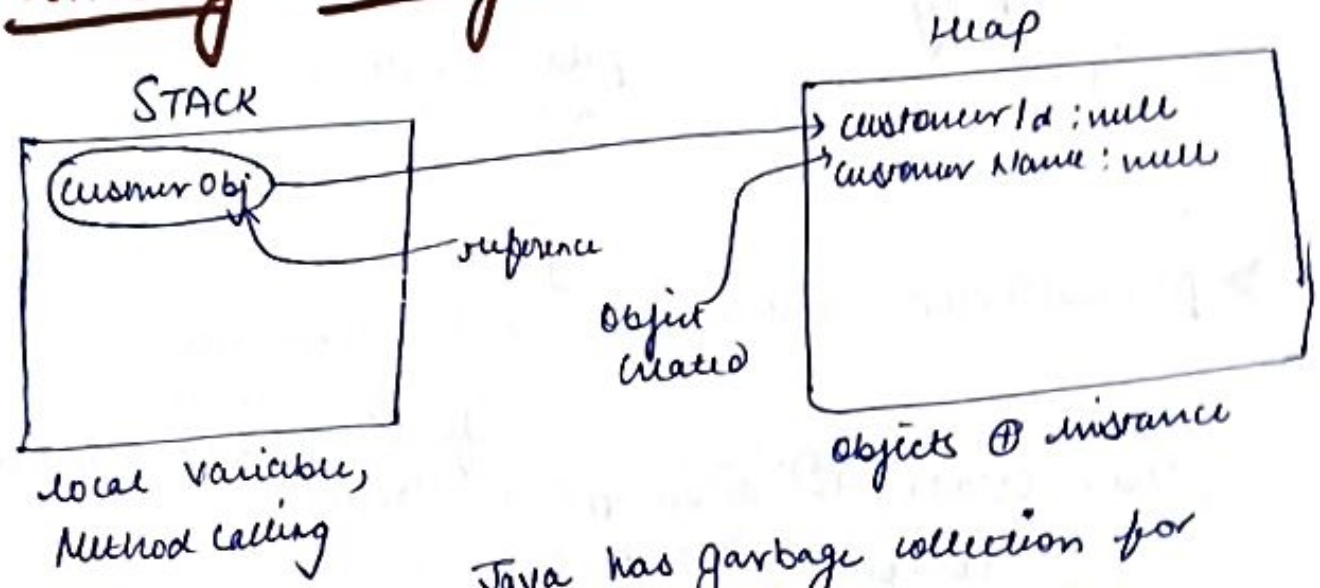
this → reference variable that refers to current obj.

this.customerID = customerID
↑ reference of obj ∴ instance
↑ local

* can also call constructor and Method.

this.() ← No Parameter ∴ parameter less constructor will be called.

Memory Management



Java has garbage collection for having unused memory.

object with no reference → eligible for memory locat

→ reference variable initialized to → null

→ reference variable initialized to new obje

Encapsulation

AKA

Data hiding

Making variables and methods together

Locking the data

→ Variables as private

→ using getter ↓ accessor setter ↓ mutator to filter values

Abstraction

Knowing only relevant details

Ex → ATM

hides internal implementation

Know → methods, paramdtrs to pass

Acess Modifier

public → Accessible from any other class

private → Accessible only inside its own class.

protected → " inside same package and sub classes in different Package.

default → Accessible inside same Package.

String

Store sequence of characters (class)

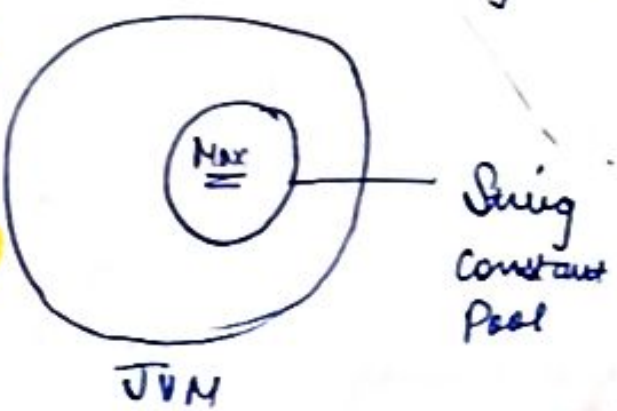
literal → String msg = "Hi";

new() → String customerName = new String("Hi");

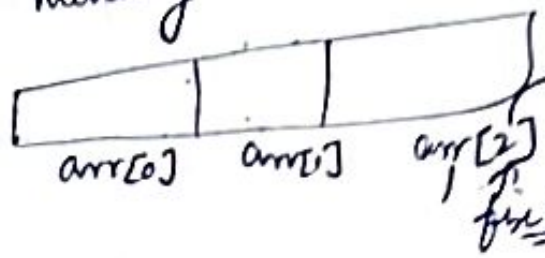
Every time u use a String literal अगर वो already exist करता होगा तो उसका reference return होगा

else

नया instance बनेगा !!



Array → Same data type in contiguous memory



→ datatype[] arrayName = {Separated by comma};

→ datatype[] arrayName = new datatype[size];

→ for (datatype name : array)

Multi Dimensional Array ⇒ arrays of arrays each element holding reference of other Array.

runtime → int[][] obj = new int[3][];

obj[0] = new int[5]; → 5 columns

obj[1] = new int[3]; → 3 columns

obj[2] = new int[7]; → has 7 columns.

public static void main (String[] args)

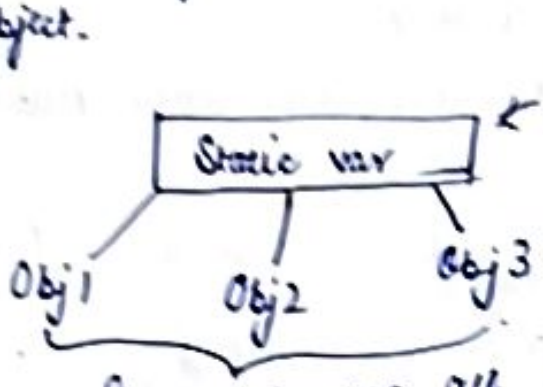
↑ array of Strings

Passed using command lines

Static → If a variable is common for all objects keep it as static.

↓

Static variable Belongs to entire class not to a specific object.



* Memory allocation is given once.

AKA → class level variables

[Don't need an object to access]
क्योंकि उसको already memory मिल चुकी है।।

1) Static variable → single copy of variable is created for all objects.

2) Static block → to initialize static variable.
Gets executed at least once.
[called automatically when class loads.]

3) Static method → (ex. main() ; can be access without any object.
- directly call static Methods
- " access static Data
- can't use this or super().

Belongs to current instance

Static Method doesn't belong to specific instance

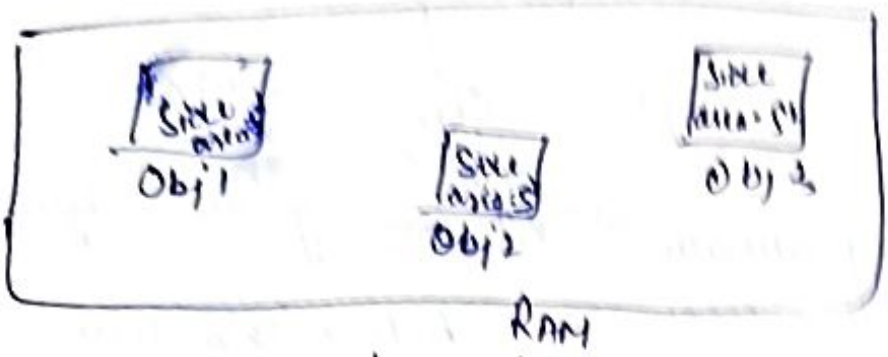
Immediate Parent static allow to use stuff

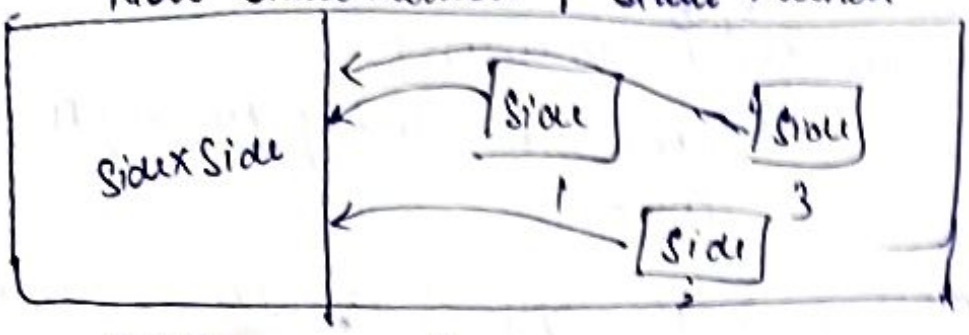
KANAK

4) Static classes : Object of a nested class, unless it is referenced by outer class.

* Static Member can not access Non Static Members.

Non static Member, Non static Method

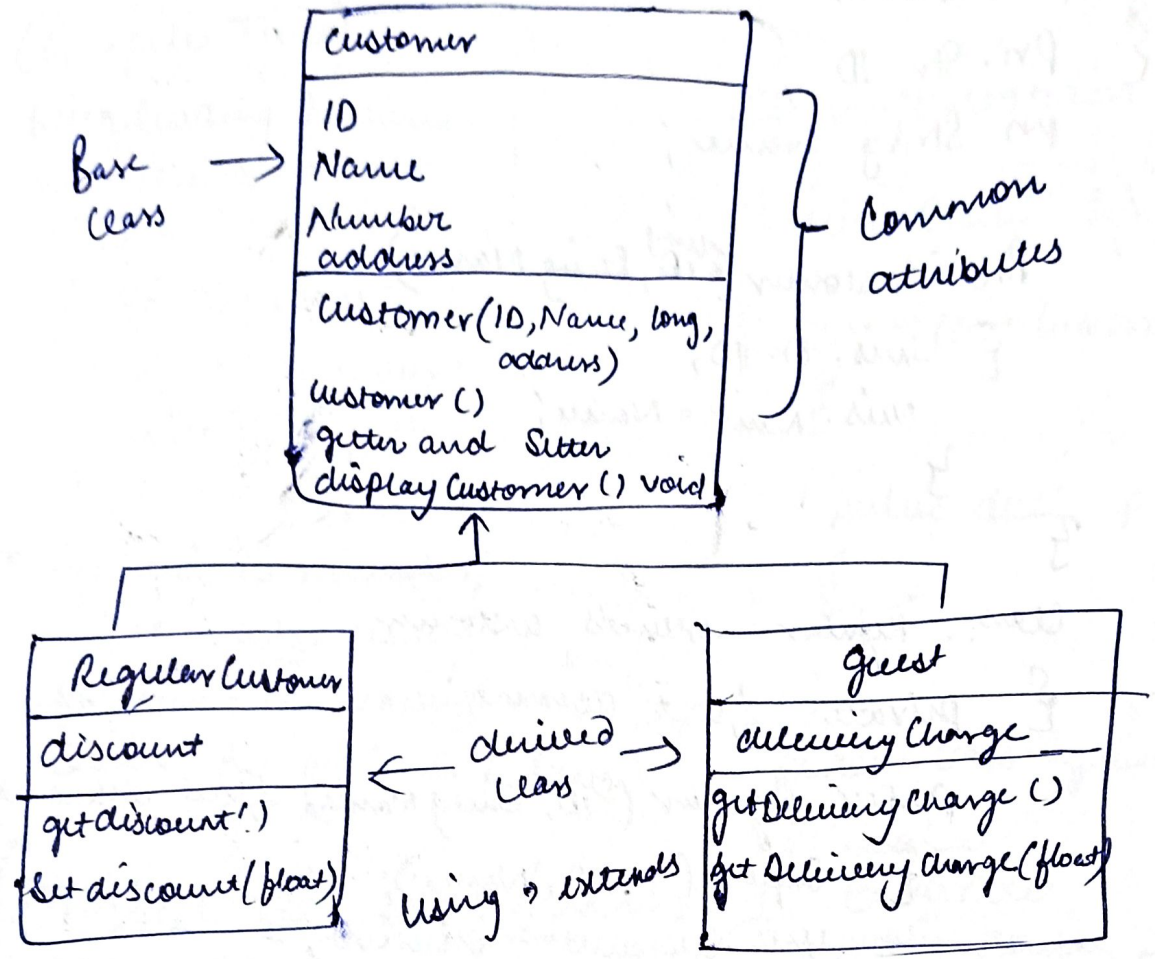




Logical Reasons: Non static variables belong to objects and static method has no object reference.

∴ reference has to be made first.

Inheritence



One class inherits the features of another class.

→ making new classes based on existing ones.

(Non Private)

- Code Reusability
- Method Overriding
- Abstraction

Constructor → invoked while creating objects

child class → invokes parent constructor.

∴ First Parent Constructor

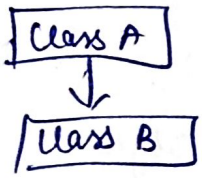
Second child constructor.

Calling Parameterized Constructor

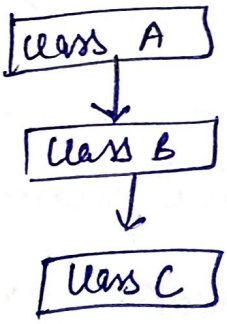
```
Class Customer
{  Pri. Str. ID
   Pri String Name;

   Public Customer (String ID, String Name)
   {  this. ID = ID;
      this. Name = Name;
   }
}
Class Regular extends customer
{  private float discount;
   Public Regular (String ID, String Name, float discount)
   {  Super (custID, Name);
      this. discount = discount;
   }
}
```

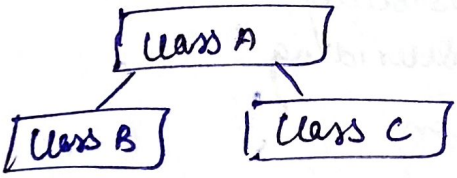
Single



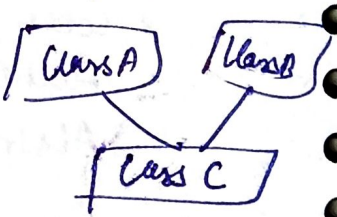
Multilevel



Hierarchical



Multiple



(Not in Java)
Not in OOPS

Polymorphism → many forms

Static (Compile Time)

Overloading Methods

multiple methods with same name (diff. Signature)

- no. of Parameter
- data type of Para.
- order of Parameter

* no need to remember/ create diff. Names for Methods having same functionality

Dynamic

Runtime → overriding Parent Method

Diff implementation for Parent & child

with same Signature

```
public class Customer
{
    public double payBill(double total)
    {
    }
}

public class Regular Extends Customer
{
    @Override
    public double payBill(double total)
    {
        return ...
    }
}
```

Now Dynamically it can override the Method. Using Same Signature

But New Methods of child can't be accessed first Overridden ones

Constructor Overloading

multiple constructors in a class

Parameterized

No Parameters

Diff Parameters

```
Costumer regularCustomer = new RegularCustomer();
regularCustomer.payBill();
```

Actual object sitting in memory is RegularCustomer().

Dynamic Binding → JVM checks real obj type and decides the Method of child that is overridden

Override → not required but recommended
- with → compiler checks
- clarity essential

Annotation → @

- information for compiler / JVM → provides additional meaning
- helps to prevent errors

super → invoking Parent Class method from child class method
→ accessing parent constructor
→ accessing parent class instance variable in child class (if same name)

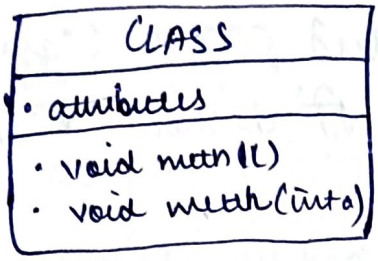
```
public double payBill (double totalPrice)
{ double priceAfter = super.payBill(totalPrice)
  return priceAfter
}
```
calls parent method

```
public void calculateSalary()
{ this.Salary = super.Salary + bonus;
}
```
calls parent instance.

Method Overloading

a class → multiple methods

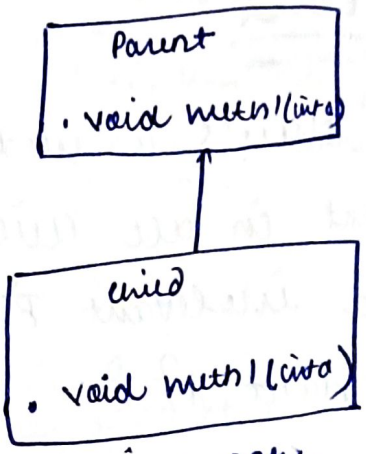
Same name ← different Signature



Compile Time poly.

Method Overriding

subclass → Same name, Same Signature



runtime poly.

toString() → method returns textual representation of object

Name of object @ hashcode → food @ af7d876

can be used to give meaning full information.

Wrapper Classes → Integer, Boolean, Double etc.
(for representing Primitive as objects)

final keyword → to make components remain constant. i.e → π → 3.14

final

- Variables
  private final float a = 3.14f
  value can't be changed now
  ∴ constant
- Classes
  final class can't be extended.
  public final class MyFinal1()
  {
  }
- Methods
  Can't be overridden in sub classes
  public void MyFinalMethod() final

If final variable is not initialized → blank final variable

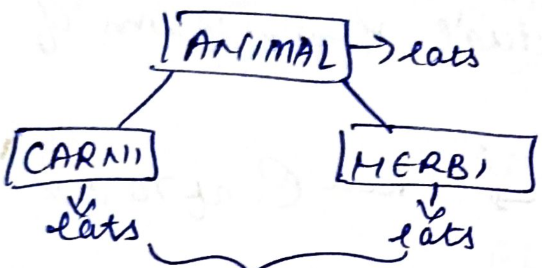
↓

Can be initialized in constructor only (once only)

Abstract

When there's a need of a new method to be different in all child classes. and parent class को method irrelevant है But child को declare करना important भी है.

then → use abstract.



दोनो अपने हिसाब से use करेंगे

∴ → abstract void eats();

↓ just declare it in the Parent.

define in child.

∴ अब यह class incomplete है और it needs a child to complete.

∴ Keep class as abstract.

abstract class Animal

∴ Any class extending this class must provide methods implementation

Abstract → Something is incomplete

Classes
incomplete and can't create object directly

Methods
has no definations
∴ must be preceded with abstract.

Interface
used to provide generic template

- Contains method signatures and constant declarations.
- 1 Interface extend Multiple Interface
  1 Class implement Multiple Interface] Multiple Inheritance
- 1 Class extend 1 Class but implements Multiple Interface
- Class should implement all methods OR declared as abstract.
- Can have default & static methods

~~default & public~~ Methods → public & abstract
attribute → Static & final & public

कोई method को
फिर वैसा ही रखना।

default public void display()
{
}
(Can be overridden)

Static
- Can be defined
- can't override

[Can be directly used with Interface name]