Object Orientend programming

DATE : ... .

→ classes & objects

↓

entities in the real world

Group of these entities

class pen → attributes (properties) color
+
functions (behaviors)
changecolor()

Examples

pen — blue, yellow, red. } property ↓ color

↓

ChangeColor();

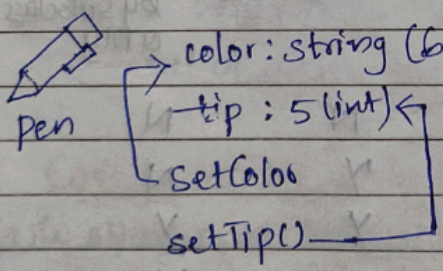
→ here the pen is the real world object, which has is property "color" and a function which changes the color

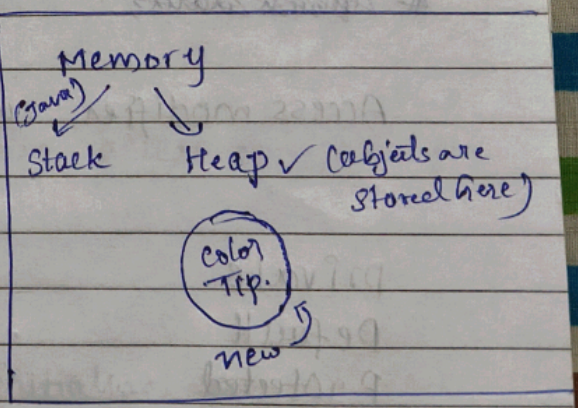
* Java, class names are started with capital and function names are started with lowercase. (it's not a rule, it's just a good programing skill).

Ex: Blueprint (M800) → class

m800
m800
m800
} obj's.

Explanation





code: public class oops{ → constructor.

Pen p1 = new pen(); // created a pen obj called p1

p1.setColor("Blue");

syso(p1.color); → TO access any property or function we use (.) dot operator.

DATE : .....

Keyword 'class'

```
class Pen {
    string color;
    int tip;

    void setColor(String newcolor){
        color = newcolor;
    }
    void setTip(int newTip){
        tip = newTip;
    }
}
```

Class Blueprint

Eg:

```
Class student{
    string name;
    int age;
    float percentage; //cgpa
    void calcpercentage (
        int phy, int che, int mat){
        percentage = (p+c+m)/3
    }
}
```

Access Modifiers / specifiers.

* It defines the access to the object

Access modifier	within class	within package	outside package by subclass only	outside package
private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
public	Y	Y	Y	Y

DATE : .....

Getters & setters

Get : to return the value
set : to modify the value

"this" : This keyword is used to refer to the current object

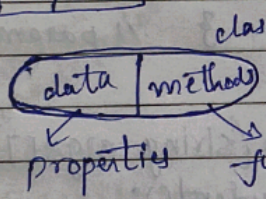
Encapsulation

- Encapsulation
- Inheritance
- Abstraction
- polymorphism

four main pillars of DSA

Encapsulation is defined as the wrapping up of data & methods under a single unit. It also implements data hiding :→ useless / sensitive → private / def

Encapsulation



→ with the help of Encapsulation we use access specifiers for the result of data hiding

Constructors → special function.

→ The objective of constructors is to initialize an object

Constructor is a special method which is invoked automatically at the time of object creation

- Constructors have the same name as class or structure
- Constructors don't have a return type (Not even a void)

DATE : .....

- constructors are only called once, at object creation
- memory allocation happens when constructor is called

Types of constructors

1) Non-parameterized
2) parameterized
3) copy constructor.

- Constructor overloading
→ This is an example of polymorphism
↓
multiple forms

Non-parameterized

```
class student{
    string name;
    int roll;
    student(){ // non-parameterized
    sysO("constructor is called");
    }.          // non.
```

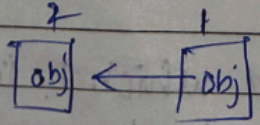
parameterized.

```
student(string name){
    this.name = name;
}
Student(int roll){
    this.roll = roll;
} // parameterized.
```

```
pablic static void main(string args[]){
    Student s1 = new student();
    student s2 = new Studen("peekshitha");
    Student s3 = new Studen("471");
```

Copy constructors.

→ the properties of one object is copied into another object



obj2 = copy(obj1).

DATE : ... .

Shallow & Deep copy "Lazy copy"

↓ ↓

References new memory.

→ Changes Reflect → Changes don't Reflect

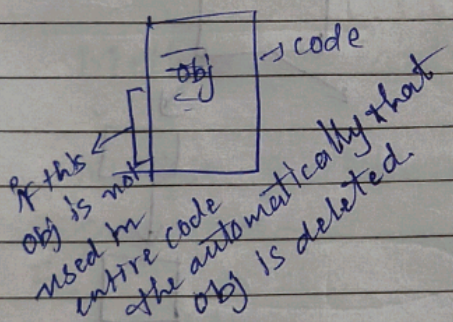
Destructors. Balance

↙ ↘ Destructor.

constructor

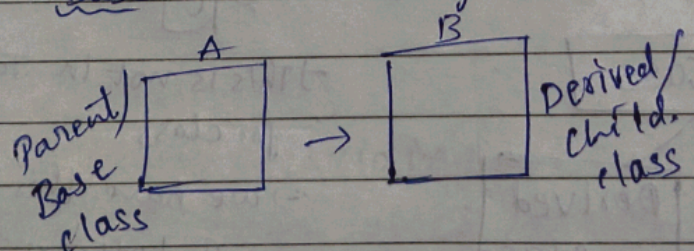
Garbage collectors in Java → delete the unwanted data automatically

So, In Java we no need to write Destructors.



Inheritance.

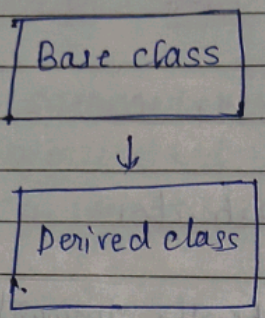
Inheritance is when properties & methods of base class are passed on to a derived class



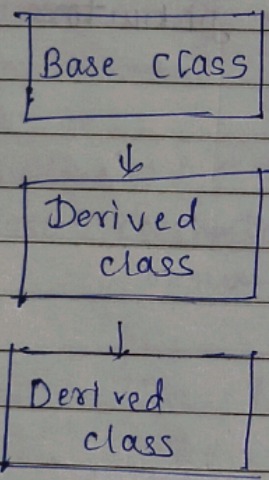
DATE : .....

Types of Inheritance.

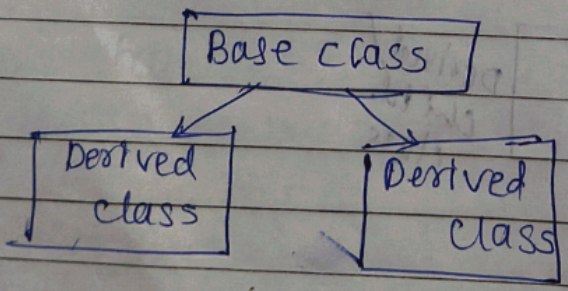
1) * single Level Inheritance



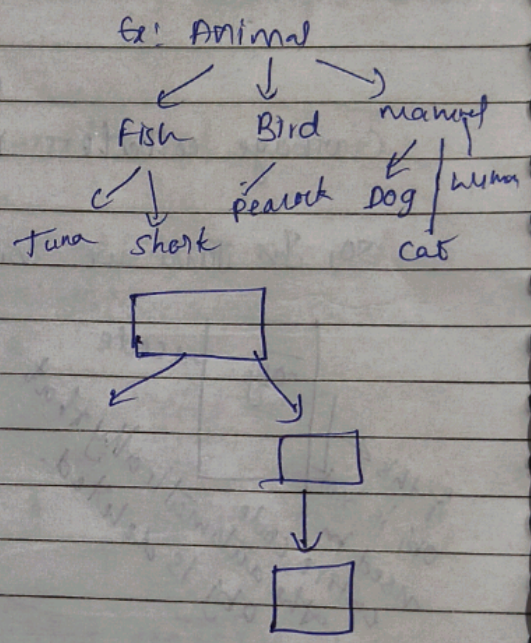
2) * Multi Level Inheritance



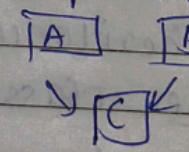
3) Hierarchial Inheritance.



4) Hybrid Inheritance



* Multiple inheritance



→ This is not in Java for class
→ we have this in Interfaces

Note: But in C++ we have this in classes too.

DATE : .....

Polymorphism.

* doing the same thing in an different forms.

Polymorphism
(many) (forms.)

Compile Time Polymorphism (static)
* method Overloading

Run Time Polymorphism (dynamic)
* method overriding

Method overloading.

Multiple functions with the same name but different parameters.

Ex:.

```
Calculator {
    sum (int a, int b)
    sum (float a, float b)
    sum (int a, int b, int c).
}.
```

Method overriding (Run time)

parent and child classes both contain the same function with a different definition

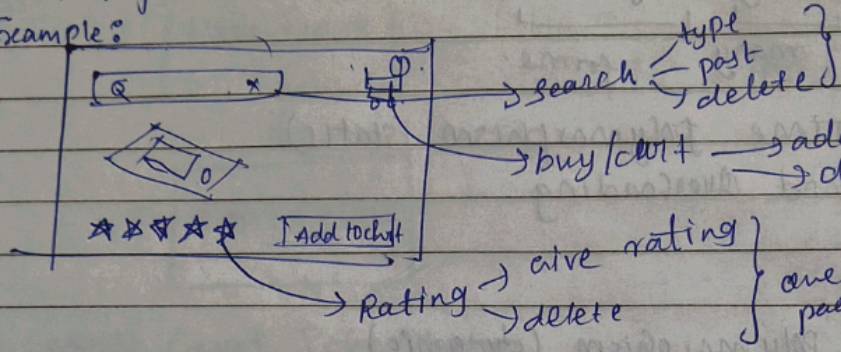
Ex: Animal
eat() → "eat anything"
↓
Deer
eat() → "eats grass"

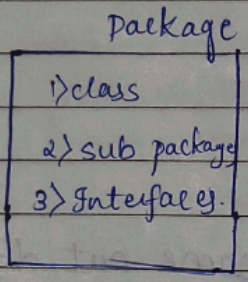
DATE : .....

packages in Java.

package is a group of similar types of classes, interfaces and sub-packages.

Example:





Types of packages.

i) In built

ii) user defined.

Example: Scanner(); → Scanner package "java.util.scanner"

→ Idea

Abstraction.

Hiding all the unnecessary details and showing only the important part to the user.

"Abstract classes" "Interfaces".

* Abstraction is mostly similar to the Encapsulation it terms of hiding the data.

→ idea ✓
→ Implementation ✗ } in Abstract

DATE : .....

Abstract class.

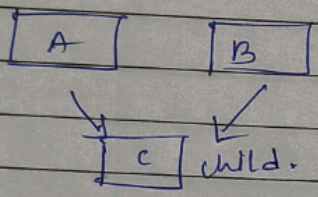
i) Cannot create create an instance of ~~obje~~ obstract class
(object)

ii) can have abstract/non-abstract methods

iii) can have constructors.

Interfaces.

Interface is a blueprint of a class.



like : class - blueprint → Obj
Interface - blueprint - class.

→ multiple Inheritance Implementation is done by Interfaces.

→ total abstraction.

Interface	class
↓	↓
Implements	extends

Properties of Interfaces

- All methods are public, abstract & without implementation
- used to achieve total abstraction.
- Variables in the interface are final, public and static
  ↙
  class → val → single

DATE : .....

Static keyword

static keyword in java is used to share the same variable or method of a given class.

* things we make static are :
- Properties
- Functions
- Blocks
- Nested classes.