PAGE NO.:
DATE: / /

OOPs in Dart.

1) - Object Oriented Programming
- Ayushi Taught petty much of Dart.
- You so you all know Dart is an object oriented Programming language just like Java & C++.

2) - Object Oriented Programming
so what is OOP? ↑
This word will haunt and follow you throughout ~~the~~ if you are starting with development. So let get into it & and know what it is before it haunts you:
- What is use of OOP?
- Do we really need OOP?
Dw we will cover and try our best to get answer of this questions.

3) So Before OOP we had Something called Procedural oriented Program (POP). we all must have done this in our basics in C.

- Here we had a progam defined ~~consisting~~ in set of functions with variable and function that operate on data

- This style of programming is very simple.

- As a programs grow when you tackle many functions and then its very difficult to. You just copy paste line of code here and there and then you might entangle yourself.

* - OOPs is great application but writing POP code is better if we're doing something small like solving problems.

4) Mems

5) Here we can see how it structure is
- POP follows Top down Approch
- OOP follows Bottom up approch

6) OOP
- Associated with class & objects
- Objects
- Simply the complex codes

7) Basic 4 pillars of OOP

8) Class (Imp)

- OOPs concept revolves around class
- ~~Blue print~~ class is just like a Blueprint and object is an actual implementation of that Blue print

5) OOP > POP
- Access specifiers provided (private acess specifiers)
- Data sharing
- Code reusibility (Inheritance)
- Modfication
- Easy to maintain
- clean structure

8) ~~Class dog~~
Eg.
```
class TestClass {
void display() {
    print("Hello World");
}
}
void main() {
TestClass c = new TestClass();
c.disp();
}
```

Eg. Dog class

```
void main (){

var dog1 = Dog();
dog1.age = 9;
dog1.breed = "chihuahua";
print ("${dog1.age} and${dog1.breed}");
dog1.walk();
dog1.bark();

var dog2 = Dog();
dog2.

}

// creating a class named "Dog"
class Dog {
// Define properties of a Dog.
int? age; // instance variable.
String? breed; // ——— "———

// These are the method which
facilitate communication b/w objects.

void walk(){

print ("${this.breed} is now walking");
}

void bark(){
  print ("${this.} is now barking");
}
}
```

Ctrl + V.

12) Inheritance

class Parent Grandparent

class Parent
Mom × Dad Dad Uncle

Class Children (Meme)

- Dog example

```
void main(){

Bike().twowheels();
print(Bike().color);
}


class Vehicle{
String color = "Red";
void twowheels(){
  print("Two Wheels");
}
}

  class Bike extends Vehicle{
    void engine(){
  print("Engine");
  }
  }

class cycle extends Vehicle{
  void paddles(){
  print("paddles");
  }
}
```

14) Type of inheritance.

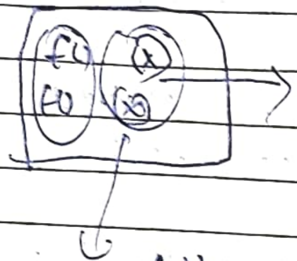
(*) Abstraction.
(Big long fancy word)

```
void main()
{
  Bike().twowheels();
}
// creating an Abstract class
abstract class Vehicle
{
  void twoWheels(); // Abstrat functn
  void color()
  {
    print("Red");
  }
}

class Bike extends Vehicle{
  void twowheels(){
    print("Two wheels");
  }

}
```

Ex - DVD player (object)
↓inside ↓outside
logic board buttons → click (Play)
(Hidden)
[I dont care abt] [I care abt]

- we can hide some properties + methods

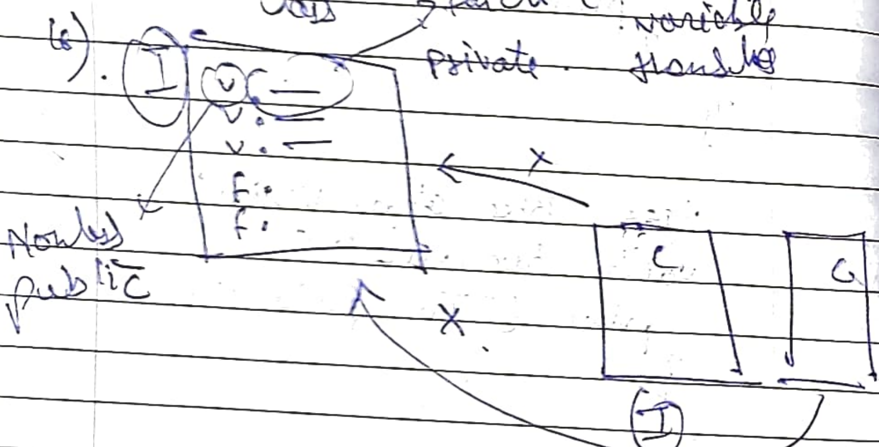
- Simple Interface, Reduce the Impact of change



Change
Delete
Not impact the outside code

(No code touches outside
Hidden (Only using Parameters)
(Private)

6). class → function to change variables
private. Handling



f() {
i++
}
← scopping →
j() {
i = 0
}

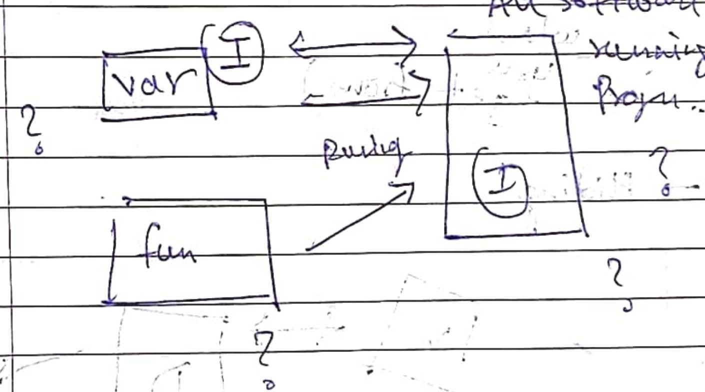
- In capsulation Not Allows other that shouldn't be part modified by other of your code

13) Poly morphism
Many Forms

c) All this object should have ability to send

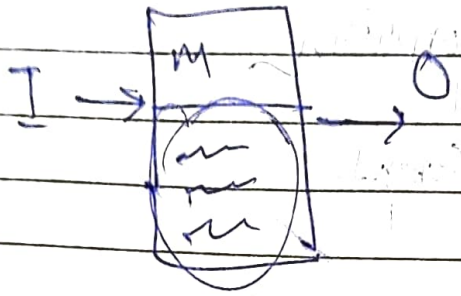
√) render method
- To get rid of long if else + switch case

16) Encapsulation - Reduce complexity
+ increase reusibility
(safer to ex

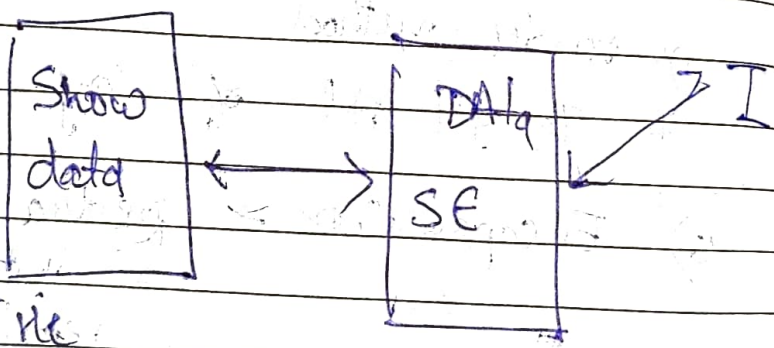


Untrasable Bugs

(*) Abstraction



Hiding
Abstraction



(He
doesn't
care
what
you
get from)

- Hiding