OOP summary (in dart)

Constructors

What is a Constructor?

a Constructor is a special type of function *named with the same name of the class* that is *called by default* when an object of the class is defined.

Why use a constructor?

when defining an object of a class that has some variables in it , like this:

```
class items{
  String? name;
  int? price;
  int? discount;
}

void main(){
  items x= items();
  x.name="Laptop";
  x.price= 100;
  x.discount=20;
}
```

you notice how you declare the value of the items manually one by one after defining the object....
do you really think your code needs more lines 😔

now see how a Constructor work with this problem :

```
class items{
  String? name;
  int? price;
  int? discount;

  items(String n, int p, int d){
    name=n;
```

```
    price=p;
    discount=d;
  }

}

void main(){
  items x= items("Laptop", 100, 20);
}
```

here , instead of defining every variable in that class one by one , we was able to initialize the values of them while defining the object it self

Ways to write the Constructor and the difference between them:

- first is to write it as a function that receive data and then assign them to the properties in the class, that make the initializing for the variables required with the same order is is written in the function:

```
class items{
  String? name;
  int? price;
  int? discount;

  items(String n, int p, int d){
    name=n;
    price=p;
    discount=d;
  }

}

void main(){
  items x= items("Laptop", 100, 20);
}
```

- A shorter version of the first way , called the one line assigning , also make the initializing required and in the same order as the function:

```
class items{
  String? name;
  int? price;
  int? discount;

  items(this.name, this.price, this.discount);

}

void main(){
  items x= items("Laptop", 100, 20);
}
```

- some time , we don't need to initialize those variables but we want to keep the constructor ,just in case we want to initialize it , in summary , i want to be able to choose whether to initialize my variables or not , we put the constructor parameter in square braces [ ] , but we also still have to write it in order of the parameters in the constructor:

```
class items{
  String? name;
  int? price;
  int? discount;

  items([this.name, this.price, this.discount]);

}

void main(){
  items x1= items("Laptop", 100, 20);
  items x2= items();
}
```

- and now we come to my favorite way , which solve most of the problems in the previous ones , we can write the parameter in a shape of a map , (so what does it do when we write it in a shape of a map?) this will make you call the parameter by name , without having to memorize where it was , and also you have the option of initializing the parameter :

```
class items{
  String? name;
  int? price;
  int? discount;

  items({this.name, this.price, this.discount});

}

void main(){
  items x1= items(price: 100, name: "Laptop", discount: 20);
  items x2= items();
}
```

- but what if i want one of the parameters to be required , like i want the name of the item to be non optional , but the price and the discount can be optional .
  no problem at all , all you have to do is to type the word required before the parameter , that will make it non-optional:

```
  items({required this.name, this.price, this.discount});
```

Named Constructors

```
class items{
  String? name;
  int? price;
  int? discount;

  items(this.name);
  items.const1( {this.name, this.price, this.discount});

}

void main(){
  items x2= items("Laptop");
  items x1= items.const1(price: 100, name: "Laptop", discount:
```

```
20);
}
```

Encapsulation

```
class items{
  String? name;
  int? price;
  int? _discount;

  items({this.name, this.price});

  setDiscount(int d){
    _discount=d;
  }

  getDiscount(){
    return _discount;
  }

}

void main(){
  items x= items(name: "Laptop", price:100);
  x.setDiscount(20);
  print(x.getDiscount())
}
```

Set - Get

```
class items{
  String? name;
  int? price;
  int _discount=0;

  items({this.name, this.price});

  int get disc => _discount;

  set disc(int value){
```

```
    _discount=value;
  }

}

void main(){
  items x= items(name: "Laptop", price:100);
  x.disc=20;
  print(x.disc);
}
```

inheritance

Single-level inheritance :

```
class Parent{
  String? name;
  String? age;

  Parent([this.name, this.age]);

  printName(){
    print(name);
  }

}

class Child extends Parent {}

void main(){

  Parent p1= Parent("Ahmed", "40");

  Child ch1= Child();

  ch1.printName();

}
```

Multi-level inheritance :

```dart
class Parent{

  String? name;

  String? age;


  Parent([this.name, this.age]);


  printName(){

    print(name);

  }

}


class Child extends Parent {

  String? job;

}


class GrandChild extends Child {}


void main(){

  Parent p1= Parent("Ahmed", "40");

  Child ch1= Child();
```

```
  GrandChild ch2= GrandChild();
  ch2.printName();
  print(ch2.job);
}
```

Hierarchical inheritence :

```
class Parent{
  String? name;
  String? age;

  Parent([this.name, this.age]);

  printName(){
    print(name);
  }
}

class Child1 extends Parent {
  String? job;
}
```

```dart
class Child2 extends Parent {
  String? state;
}


void main(){
  Child1 ch1= Child1();
  Child2 ch2= Child2();


  ch1.name= "Ali";
  ch2.name= "Ahmed";


  ch1.printName();
  ch2.printName();
}
```