OOP:

object: هو عنصر داخل الـ Project مثل: admin, Customer, Product

Advantages of OOP:

- → It is easy to under=tand & use.
- → Decreasing the Complexity.
- → Increasing Programmer Productivity.
- → It makes the code easier to maintain & debug.
- → Reduce redundancy in code.

features of OOP:

- → Class
- → object
- → encapsulation
- → inheritance.
- → Polymorphism.
- → Abstraction.

---

1 Class:

يبدأ بحرف capital

```
Class <name> { Variables implementation;
               methods implementation; }
```

ex:

```
Class User {

Variables { String ? username ;
            string ? PassWord ;

method { PrintName () {
           Print (Username); }

Default Constructor → User () { }

Constructor with Parameter → user ( string name) {
                               Username = name ; } }
```

(2) object:

لما نحتاج نستخدم الـ class اللى عملناه أو نستخدم أى حاجة جواه بنعمل object منه فالمكان ده وبيكن داخل الـ main

```
Void main () {

* classname + objectname = classname ();

User user = User ();
```

إنشاء ←

يتم استدعاء الـ Constructor الـ default

```
* class عشان استخدم اللى جوا الـ

→ user. Function ();

→ User. variable ;
```

Constructor:

ده عبارة عن method بتنفذ بشكل تلقائى اول ما بنعمل انشاء للـ object زى كده ← User user=User ();

وبما ان الشكل ده مافيش Parameter متبوت مع الاستدعاء فكده هيتم تنفيذ الـ Constructor الـ default وميزة الـ Constructor ده ان حتى لو احنا ما عملناهوش هو بيبقى معمول جاهز.

لكن لو:

```
User user = User ("mahmoud");
```

بالشكل ده هيتنفذ الـ Constructor اللى الـ Parameter تاعى string

الشكل ده ← مكان نستبدلى بـ

```
User (string name) {
    username = name; }

User (this.username);
```

الحالة دى لو انا عايز استقبل Variables بس ومش هاعمل اى عملية

لو

```
User ([this.username]);
```

معناها ان الـ Parameters هنا اختيارى

لو عايزة اعمل اكتر من ~~object~~ Constructor بنفس الاسم :

① User ( ) { } ← default.

② User. اي اسم تاني ( ) { }

User. Named Const ( ) { }

User. second Const ( ) { }

ولما اجي اعمل استدعاء لـ object :

في العادي ← User. user = User ( );

الجديد بقى ← User user = User. second Const ( );

↳ User user = User. Named Const ( );

---

[3] encapsulation :

* اي حاجة داخل اي class عشان اعرضها باستخدم لها setter اسمها method وعشان اعدل عليها باستخدم لها getter اسمها method

← الـ (_) قبل اسم المتغير بتخليه private بالنسبة للفايل اللي هو فيه وبكده ماينفعش استخدمه (اندهله) باسمه على طول User. variable ولازم استخدم اد getter واد setter

string ?_Password ;

⇒ getPassword ( ) { return _Password ; }

استدعاها User. getPassword ();

⇒ setPassword (string word) { _Password = word ; }

getter :

return type get propertyname { return _variable }

او

" " " => variable ;

استدعاها User. propertyname ;

setter:

```
set propertyname (value){

set Password (string value) {
    _Password = value;    }
```

الاستدعاء:

```
user.Password = "La Vie 666";
```

نمرر القيمة بعلامة (=) .

---

(4) Inheritance:

لو في متغيرات أو methods مشتركة بين أكثر من class منخليهم في class واحد اسمه Parent ونخلي المشتركين معاه children يورثوا منه

```
class Parent {
        string? name;
        string? email;
        int? Phone;
        getEmail() {
            return email; }
    }

class Child extends Parent {  }
```

اصبح محتوى كل ما بداخل الـ Parent

- لما نعمل object

```
Child ch = Child();
ch.getEmail();
```

هنقدر نستخدم اي حاجة جوا الـ Parent class .

- لو عملنا class تاني بيـ extend الـ Child هيبقى من خلال بيـ extend الـ Parent الأكبر وسيستخدم كل اللي فيه برضه .

Hierarchical Inheritance: Type of inheritance where multiple child classes (subclasses) inherit features from a single Parent class (superclass)

Constructor:

لو في Child class بيورث من Parent عند عمل object من الـ Child
بيتم استدعاء الـ Constructor بتاع الـ Parent ,
طب لو الـ Constructor ده ليه Parameter :
← لو احنا في الـ Parent نستخدم this.var
← لو احنا في الـ Child وعايزين نستخدم حاجة من الـ Parent نستخدم super

```
Class Parent {
    string ? name;
    Parent (this.name) {
        Print ("name : $name"); }
    }

Class child extends Parent {
    child (super.name);
```

(أو) `child ({super.name});` ← في الحالة دي هنستدعيها في الـ object كـ ("name:"الاسم")

المتغير موجود في الـ superclass

- ولو قبل لازم نمرر الـ Parameter عند عمل الـ object من الـ Child .

```
Child ch = Child ("mahmoud");
```

Child بيروح ينفذ الـ Constructor بتاع الـ Parent وبعده ينفذ بتاع الـ Child

note

* نستخدمها مع المتغيرات الموجودة داخل نفس الـ Class .
* super. نستخدمها مع المتغيرات اللي بتورثها من الـ Parent ومش موجودة معانا في الكلاس اللي احنا فيه (كتابة يعني)

لو عايزة استخدم method موجودة في الـ Parent في الـ Child
ممكن استدعيها على طول عادي (لكن الأفضل اعمل قبلها super)

ممكن → getname();

الأفضل → super.getname();

عشان تعرفنا انها جاية من
الـ Parent .