

OOP

Object Oriented Programming



Mohammed Bin Siddico

Whatsapp: 01227897361

OOP == OBJECT ORIENTED PROGRAMMING

الـ OOP هي حاجة لو درسناها بلغة وحيث تغير اللغة بتدرس بيها مش لازم
تدرسها تاني

ترجمتها الحرافية هي البرمجة الكائنية

أي اللي يخليني أبدأ أدرس حاجة زي كده ؟؟

إن البرامج اللي إحنا كنا بنعملها كانت لسه بسيطة كانت قائمة على الـ variables مثل هذن name, age وهكذا كنت بستخدم المتغيرات اللي بنشأها علشان أخزن المتغيرات مفيش مشاكل !!!!

بس إحنا لحد دلوقتي بنتكلم على البرامج البسيطة مثل جمع رقمين وهذا متكلمناش عن البرامج المعقدة الكبيرة معرفش أعمل معاهها كده لو المشاريع الكبيرة اللي بتكون على أرض الواقع جيت أعمل معاهها كده مش هنلاقي المكان اللي نقدر نبدأ منه كتابة الكود .

فلنفترض مثل إن أنا عايز أعمل football game فأننا هنا الطبيعي هتعامل مع football players لو عايز أمثل محمد ضلاح مثل !!!

هنا أنا محتاج أتحكم في الـ player نفسه بالكامل علشان أقدر أديله أوامر "يتحرك، يقف، يجري ، وهكذا"

فأننا هنا مثل هدي لللاعب name, speed , and other attributes بس هنا كده الموضوع مش عملي "مش أحسن حاجة" لأن الـ attributes دي هي مدددة مرتبطة ببعضها وكمان بأنهـي لاعب بالظبط !!!

وكمان لو عندي عدد لاعبين كتير "100 لاعب مثل" هفضل كده اعرف المتغيرات دي كلها للعدد دا كلـه إزاـي !!! حاجة مش منطقية

لكن بما إن الموضوع بدأ في التعامل مع كائن كامل زي كده هنلـأجـأ لـ OOP

لو حبينا نخزن كائن عندنا في البرمجة نقدر نعمل دا
من خلال الـ `oop` بكل سهولة
لان الـ `concept` بتاع الـ `oop` اتعمل علشان نقدر نحط أي حاجة على أرض الواقع عندي
جوا الكود بتاعك

”أي حاجة انت شايفها على أرض الواقع تقدر تحولها لـ `code` بـ استخدام الـ `oop`“

إزاى أقدر أستخدم الـ `oop` ؟؟؟؟؟ أو إزاى أقدر أمثل الكائن باستخدام `code`؟؟؟؟؟
الموضوع هنا مختلف شويتين عن إنشاء متغير عادي لأن إحنا هنا بننشأ `object` ”كائن“
إزاى أقدر أبدأ أمثل `object` بـ `code` بـ أقدر أستخدمه ؟؟؟؟؟
علشان نقدر نعمل كده عندنا خطوات ثابتة هنمشي عليها علشان الدنيا تكون
بسيئة

1. هنحدد نوع الفصيلة ”بشري، جماد، حيوان، عربيات“
2. هنحدد الخصائص أو السمات ”الحاجات اللي معظم كائنات الفصيلة دي بتشرك فيها“ الخاصة بالفصيلة اللي انت اخترتها دي

مثلاً الـ `Object` بتاعي اسمه `siddiq` فدا فصيلته `Human` ”كائن بشري“، والسمات الخاصة بيها مثلاً `skinColor`, `Height`, `Weight`, `NumberOfArms`



CLASSES



Create Class

```
class + className "فصيلة الكائن"  
"Attributes" "السمات الخاصة بالكائن اللي عندي"  
{}
```

دا
الشكل
الأساسي
الخاص
بإنشاء
CLASS



Example 1

```
class Human{  
String skinColor;  
double height;  
double weight;  
int armsOfHuman;  
}
```

في الوقت الحالي إحنا أنشأنا CLASS من نوع الكائن HUMAN وأضفنا السمات الخاصة بيها على هيئة متغيرات ، لازم ندي قيم ملموسة للسمات اللي أنا عرفتها بيتدي أدي قيم لما يكون عندي كائن "OBJECT"

في الوقت الحالي ممكن نلاقيه عاطينا error
لأن أنا هنا عندى القيمة non nullable
فلازم علشان أدل المشكلة دي أضيف "?" null check operator
علشان تكون القيمة nullable في الوقت الحالي وبكله هيختفي الـ error



Example 1

```
class Human{  
String? skinColor;  
double? height;  
double? weight;  
int? armsOfHuman;  
}
```

TRUE CODE

OBJECTS

بشووف في البداية أنا هخزن أي ؟؟؟ كائن بشري يبقى من نوع Human + اسم الكائن "اللي هخزنه يكون اسم معبر عن الكائن وبعمل assign مع نوع الكائن نفسه"

Human Siddiq = Human();

Human == class name, Siddiq == object name

لو جيت كتبت siddiq. هلاقيه بدأ يعملي suggestion بالـ attributes اللي داخل الـ class علشان أخزن فيها قيمة بدل ما تفضل قيمتها بـ null

siddiq.height = 187;



```
void main()
{
    Human Siddico = Human();
    Siddico.skinColor = 'Black';
    Siddico.height = 178;
    Siddico.weight = 78;
}
class Human{
    String? skinColor;
    int numberOfWorks = 2;
    double? height;
    double? weight;
}
```

في معظم شغلنا في الـ **class** الـ **attributes** مبتكونش واحدة قيمة ودا لأن الـ **object** بتاع كل **attributes** بتكون مختلفة عن الثاني بس في حاجات هي بتبقى مشتركة في **objects** كثيرة من الـ **class** دا فممكّن نديها قيمة وكده كده انت ممكّن تغير في قيمة الـ **attribute** دا من خلال الـ **object** اللي أنشأته

مثلا هنا عدد أيدي أي إنسان في الطبيعي بتكون 2 فممكّن نديها قيمة ولو في حالات exception ممكّن object that في الـ **exception only**.

السمات == الفضيلة == Attributes == Class

METHODS

لما بنجي نمثل كائن مش طبيعي إنه بيكون عنده شوية سمات فقط أكيد في مهام هو بيقدر يقوم بيها ويعملها "سفرأو يكتب، يجري، يقف، وهكذا....."

"Functions == Methods" فالازم نمثل برضو الحاجات دي

```
void read( ) { print('this man is read !'); }
```



```
void main()
{
Human Siddico = Human();
Siddico.skinColor = 'Black';
Siddico.height = 178;
Siddico.weight = 78;
Siddico.read();
// print any thing in read method
}
class Human{
String? skinColor;
int numberOfWorks = 2;
double? height;    double? weight;
void read(){
print('This Man Is Read !');
}
}
```

مفيش فرق بين ال Function و Method
إن ال Method هي عبارة عن معرفة بداخل class
إنما دي لو Function معرفة برا class هي بقى اسمها عادي Function

ال Method اللي معرفة بداخل class هي اللي بتقولي class دا يقدر يقوم بأي بالضبط

CONSTRUCTORS

ليه في البداية كنا بنعرف ال object كده بدلا من

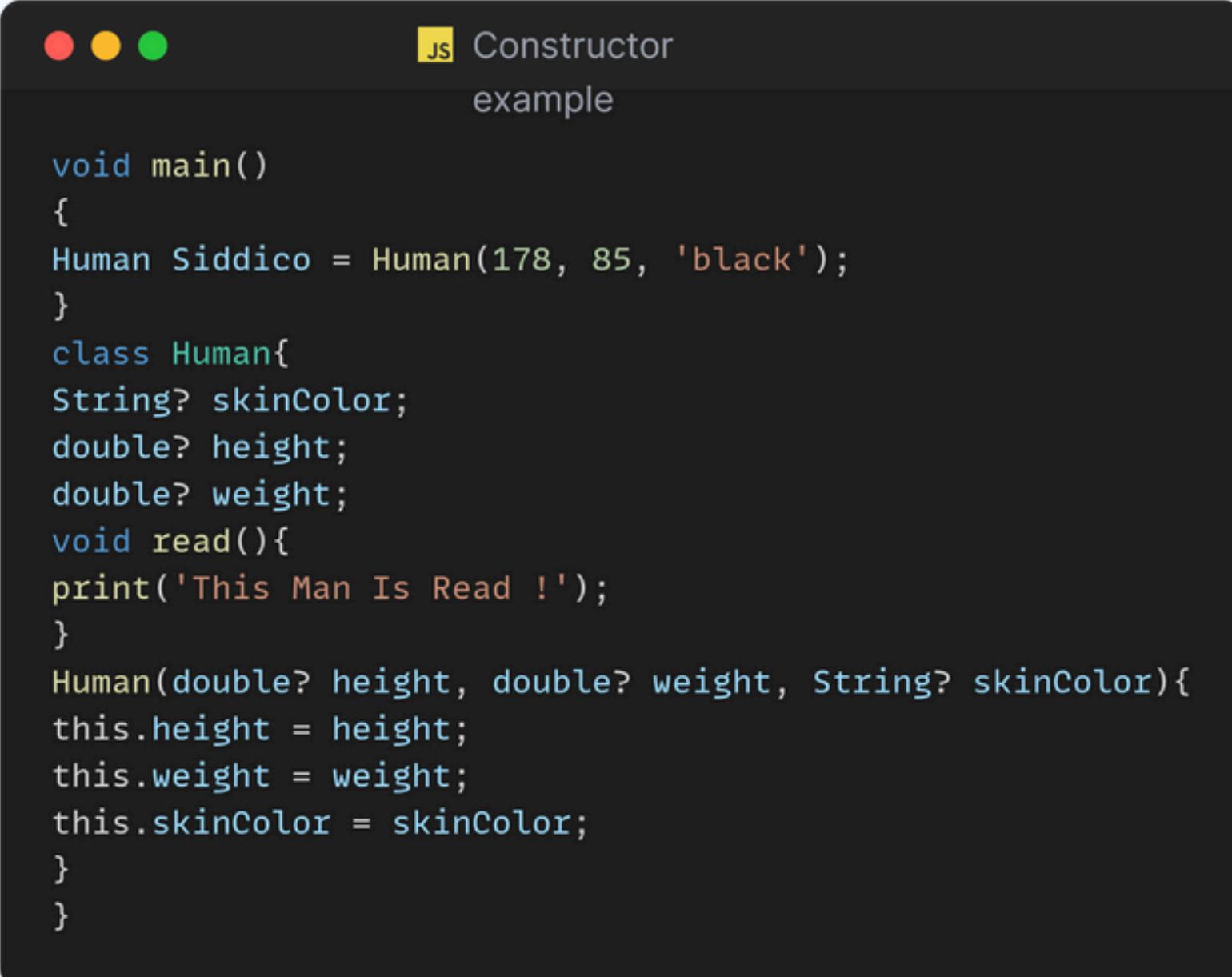
عن دا بيسمى ال constructor ودا المسئول عن إنشاء ال object أصلًا بيعمل عملية "عملية بناء الكائن" construct

هي دي أصلًا حاجة بتبقى موجودة by default بمعنى ان الـ class اللي إحنا بنعمله بيكون فيه constructor بس مبيبقاش ظاهر بس موجود وعلشان كده في كل مرة بننشأ فيها الكائن بنستدعي فيها الـ constructor

طب هل أقدر إني أتحكم في طريقة إنشاء الـ constructor بحيث إني بدل ما كل شوية أكتب: `Human siddiq = Human();` وأقعد أعمل access لكل attribute لوحدها !!!!!!!
أيوه أقدر أعمل كده لأن الـ constructor ما هو إلا method بتتأثر في كل مكان بنستدعيه فيه عن طريق:

`Human siddiq = Human(178, 85, 'black', 2);`

فكده أنا عملت object وعطيته attributes بتعاتته مرة واحدة



The screenshot shows a code editor with a dark theme. At the top, there are three colored dots (red, yellow, green) followed by a yellow square icon containing the letters 'js' and the text 'Constructor example'. The code itself is written in Dart:

```
void main()
{
    Human Siddico = Human(178, 85, 'black');
}

class Human{
    String? skinColor;
    double? height;
    double? weight;
    void read(){
        print('This Man Is Read !');
    }
    Human(double? height, double? weight, String? skinColor){
        this.height = height;
        this.weight = weight;
        this.skinColor = skinColor;
    }
}
```

بس لازم تكون حافظ أماكن ال attributes بالترتيب
علشان ميدصلش أي مشاكل ويديك error أو إنك تخلي parameters عبارة عن
named parameters وتخزن كل قيمة في مكانها الصحيح



js Constructor Named Parameters example

```
void main()
{
Human Siddico = Human(height: 178, weight: 85, skinColor: 'black');
}
class Human{
String? skinColor;
double? height;
double? weight;
void read(){
print('This Man Is Read !');
}
Human({double? height, double? weight, String? skinColor}){
this.height = height;
this.weight = weight;
this.skinColor = skinColor;
}
}
```

ENCAPSULATION

Is concept in OOP

معناه أي ؟؟؟ ، إزااي أقدر أستخدموه كـ code ، ليه محتاجه ؟؟؟
هنعرف إجابات الأسئلة دي بالتفصيل واحدة واحدة

لو أنا عندب مثلا class اسمه human فيه مجموعة من الـ attributes وردت أنسأت
مجموعة objects من الـ class دا وكل واحد يقدر يعمل علىـ access ويقدر
يديها قيم عادي جدا مع العلم ان القييم دي ممكن متكونش منطقاً بس هتتزرن
عادى جدا .

هنا بقى تظهر المشكلة وهي ان في بعض الأوقات مش لازم الشخص يكون ليه
علىـ access علىـ attribute دا مثلاً لازم يكون ليـ control اكتر عليه !!

أقدر أتحكم في القيمة اللي الـ attribute هيأخذها لأن مينفعش أكتب كود صح في
حين إن الـ logic غلط ومن هنا يبدأ يظهر مفهوم

Encapsulation

وهو ببساطة إني بعمل تغليف للقيمة دي بمعنى ان مفيش اي دد من برا يقدر
يعمل علىـ access عليها

مثلاً siddico حاول بعمل access علىـ attribute بتاعته مش هيعرف يعمل كده
وبالتالي مش هيعرف يدي قيمة غلط وبكلده هيبكون عندي control اكتر علىـ القيمة
اللي داخلة للمتغير دا

كمان مش ضوري إني أعمل encapsulation علىـ attributes كلها لاء هو بيكون
لـ attribute معينة بحيث إنها متاخدش قيم غلط في المستقبل

التغليف دا بيتعمل عن طريق إني بخلي الـ attribute دي private بحيث إني
مقدرش استعيها برا الصفحة اللي هيا فيها إنما لو في الصفحة يستدعها
عادى وبكلده عن طريق إضافة _ قبل الـ attribute

Example: int ? _numberOfArms; ----> Private Attribute

Mohammed Bn Siddico



Encapsulation

```
class Human{  
    int _numberOfArms = 2;  
    String? skinColor;  
    double? height;  
    double? weight;
```

File1.dart

```
Human({double? height, double? weight, String? skinColor}){  
    this.height = height;  
    this.weight = weight;  
    this.skinColor = skinColor;  
}  
}
```



Encapsulation

main.dart

```
void main()  
{  
    Human Siddico = Human(height: 178, weight: 85, skinColor: 'black');  
    Siddico._numberOfArms = 10; // will give you an error  
}
```

أنا كد خلاصت عملية ال encapsulation

بس مش دا اللي أنا عايزة أنا كمان عايزة ال Object يكون ليه

على ال attribute دى ويديها قيمة منطقية

return return نقدر نعمل عن طريق إني أعمل function مش بتعمل "more control"

لها حاجة هي مسؤولة عن إنها بتدي value لل attribute فقط

Example:

```
void setNumberOfArms(int numberOfArms){
```

this.attribute "encapsulated" = value it take from function

example: this._numberOfArms = numberOfArms ;

}

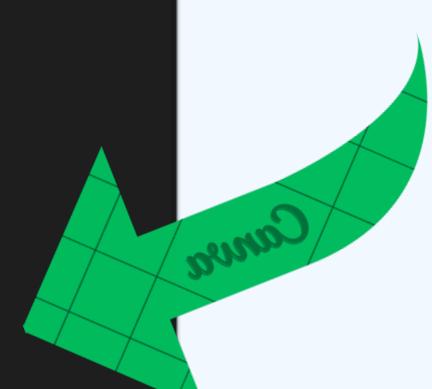
وبكلده إحنا نقدر نديها قيمة عادي جداً عن طريق إني أروح على ال void main() من object.functioncreated(value) خلاص:

example: Siddico.setNumberOfArms(2);

```
class Human{  
int _numberOfArms = 2;  
String? skinColor;  
double? height;  
double? weight;
```

File1.dart

```
Human({double? height, double? weight, String? skinColor}){  
this.height = height;  
this.weight = weight;  
this.skinColor = skinColor;  
}  
void setNumberOfArms(int numberOfArms){  
this._numberOfArms = numberOfArms;  
}
```



```
void main()
{
Human Siddico = Human(height: 178, weight: 85, skinColor: 'black');
Siddico.setNumberOfArms(2); // will set value for attribute
}
```

set
 value
 for
 attribute

بس كده برضو كأني معمليش حاجة لأن أنا كده أقدر أحط أي قيمة بس هنا الاختلاف بيرجع عن طريقة الـ `method` وأقدر أنا من خلال الـ `method` فيها أكثر وأكثر عن طريق `if`

```
if(numberOfArms <=2 && numberOfArms >0){
this._numberOfArms = numberOfArms;}
```

معنى كدا ان لو اتحط اي قيمة خارج الرينج دا مش هتخزن في الـ `attribute`

```
Siddico.setNumberOfArms(10); // will not return any value
```

```
Siddico.setNumberOfArms(1); // will return value 1 because it's in range
```

بس حالياً لو حبينا نطبع القيمة اللي اتخزنـتـي مش هعرف فلازم أعمل `method` تانية أقدر أطبع من خلالـهاـ الـ `attribute` دي هنا لازم نحدـدـ الـ `return type`

Example:

```
int getNumberOfArms{
return this._numberOfArms;
} and call here in void main
print(Siddico.getNumberOfArms());
```

لو قيمة غير اللي في الشرط هيـتجـاهـلـهاـ ويروح يجيـبـ لـوـ فيـ قـيـمةـ byـ deـfauـlـtـ غيرـ كـداـ هـيرـجـعـ الـ قـيـمةـ عـادـيـ وفيـ الطـبـيـعـيـ أيـ priـvـateـ attributeـ بيـكـونـ مـحـتـاجـ عـلـشـانـ نـقـدـرـ نـديـهاـ قـيـمةـ وـنـاخـدـ الـ قـيـمةـ نـطـبـعـهـاـ مـثـلاـ"ـ getterـ



Encapsulation

File1.dart

```
class Human{  
    int _numberOfArms = 2;  
    String? skinColor;  
    double? height;  
    double? weight;  
  
    Human({double? height, double? weight, String? skinColor}){  
        this.height = height;  
        this.weight = weight;  
        this.skinColor = skinColor;  
    }  
    void setNumberOfArms(int numberOfArms){  
        this._numberOfArms = numberOfArms;  
    }  
  
    int getNumberOfArms(){  
        return this._numberOfArms;  
    }  
}
```

Set

Get

funcs



Encapsulation

main.dart

```
void main(){  
    Human Siddico = Human(height:178, weight: 87, hairColor:"Black");  
    Siddico.setNumberOfArms(2); // setter  
    print(Siddico.getNumberOfArms()); // getter  
}
```

Call

Funcs

SETTERS

هي مش حاجة جديدة هي حاجة بنسخدمها لتحسين طريقة كتابة الـ **method** اللي من خلالها بندي قيمة للـ **attribute** ليس إلا

طريقة الكتابة:

```
set + attributeName(variable that i will store value in){  
if(my conditions){  
this._numberOfArms = variableName that you get to use ;}  
}
```

Previous Image Example:

```
set numberOfArms(int numberOfArms){  
if(numberOfArms<=2 && numberOfArms>=0){  
this._numberOfArms = numberOfArms;  
}}
```

الفرق بين القديمة والجديدة هي كلمة **set** فقط اللي بتدل على إعطاء قيمة **ليس إلا**"Setters"

setterName == nameOfAttribute

وطريقة استخدامه تكون أفضل من الـ **method** العادي لما بنيجي نعمله **سيكون** **Siddico.numberOfArms = 2;**

مش بنحط القيمة بين أقواس زي الـ **method** السابقة تعاملك هنا وكأنك **بتعامل مع attribute عادي**

ARROW FUNCTION

دي بنستخدمها لما ال function او ال method اللي عندي
بتنفذ statement واحدة فقط فمبقاش مضطر أحط السطر اللي هيتنفذ بداخل {}
الاقواس دي بنستخدمها لما يكون في blockOfCode هيتنفذ
طالما سطر واحد بقى بنستخدم ال arrow <=> بجانب اسم ال method واكتب
ال statement اللي هيتنفذ

Example: int getNumberOfArms() => this._numberOfArms;

وبكدا هي هتفهم وكمان مش لازم أكتب return لأنها متعرفة بداخل ال arrow دا
إنها هترجع الحاجة اللي بعده فهو هيعرف انه هيعمل return

GETTERS

ما هي إلا طريقة من خلالها بنعرض القيمة اللي خزناها من خلال ال setter هروح أنشأ
ال getter لازم احدد ال datatype علشان اعرف ال data اللي راجعة على عكس ال setter
مكتناش بنبقى محتاجين ال datatype

طريقة الكتابة:

datatype + get + attributeName {
block of code that will be run; }

Or

datatype + get + attributeName => statement that will execute;

هذا مبندطش () بجوار attributeName علشان انا مش محتاجها اصلاً لأن ال method دي
هترجع قيمة مش هتاخذ قيمة

Example: int get numberOfArms => this._numberOfArms;

MORE ABOUT CONSTRUCTORS

هل انا لازم في ال constructor اخزن القيمة الراجعة في متغير
وبعد كده أبعتها لل attribute ؟؟؟؟؟

حاسها لفة طويلة أوي في وبطيئة في نفس الوقت حين إن أنا ممكن ابعث
القيمة دي للattribute مباشرة

وهي اني بغير في ال constructor وبخليه:

Human({this.attribute1, this.attribute2, this.attribute3});

هناك `this.attribute` اللي معرف في الـ `object` اللي جاية لـ `attribute` دي ويخزنها في الـ `attribute`

Example:

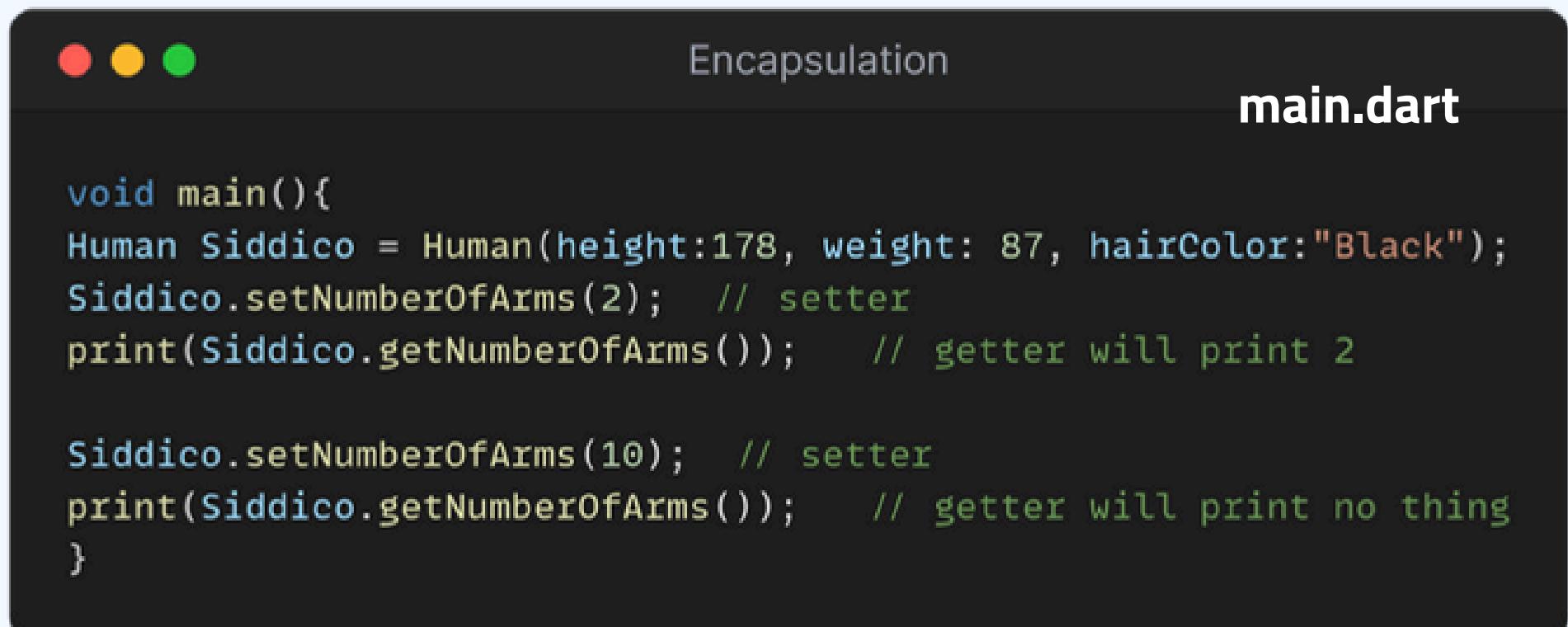
Human({this.height, this.weight, this.hairColor});

والقيمة التي في الـ `object` موجودة عادي بدون تغيير

void main(){

```
Human Siddico = Human(height: 178, weight: 87, hairColor : 'Black');
```

}





Encapsulation

File1.dart

```
class Human{  
    int _numberOfArms = 2;  
    double? height;  
    double? weight;  
    String? hairColor;  
  
    // Constructor  
    Human({this.height, this.weight, this.hairColor});  
  
    // Setter  
    set numberOfArms(int numberOfArms){  
        if(numberOfArms≤2 && numberOfArms ≥0){  
            this._numberOfArms = numberOfArms;      }  
    }  
  
    // Getter  
    int get numberOfArms => this._numberOfArms;  
}
```

INHERITANCE

لازم ترکزو في ال concept الجديد دا يا شباب لانه مهم جداً وهنفهم
مع بعض بمثال

لو أنا حبيت أمثل مجموعة من الكائنات مثلا scar عبارة عن cat, kittyg, lion وطبعا عملت class بيمثل الفصائل دي "dog, cat, lion" فلنفترض إن أنا حبيت أدي كل كائن بعض الصفات والمهام هكتشف انهم ممكن يكونو بيشركون في حاجات كتير مثلا عدد لاطراف, النوم, الاكل, الشرب يبقى هعمل كالتالي

```
class Cat{  
int numberOflimsb = 4;
```

```
eat(){  
print('eating');}  
}
```

```
class Lion{  
int numberOflimsb = 4;
```

```
eat(){  
print('eating');}
```

```
class Dog{  
int numberOflimsb = 4;
```

```
eat(){  
print('eating');
```

طب وأنا ليه أفضل أكتر الصفات والمهام دي كلها في classes كلها !!!

طب ليه الصفات والمهام دي إتكررت في الفصائل كلها ؟؟؟

لأنهم بيشركون في نفس فصيلة ال Animal شيء طبيعي انهم يتشاركون في بعض
الصفات والمهام اللي بيقومون بها وبالتالي طبيعي التكرار.

فطبيعي انك تقول دلوقتي أنا ليه معملش class اسمه Animal مثلا واحظ فيه
الصفات كلها والمهام اللي في ال classes الصغيرة ويكون لها access عليها مباشرة
منه !!!! لازم بقى تخلي بالك من حاجة هنا وان الحيوانات بتشارك في بعض الصفات
والمهام معنى كده ان في صفات ومهام مختلفة فلو حطيناها في Animal class
ممكن أنا أتلدي و access مهمه معينة لحيوان بالغلط هي مش بتاعته أصلًا وهذا
بقى يظهر مفهوم ال inheritance

مفهوم الـ inheritance ببساطة إني في بعض الأوقات لما بنجي
نمثل بعض الفصائل ممكن يكون فيه حاجات كثيرة مشتركة بينهم
تدرج تحت فصيلة أكبر منها مثل

فالحل هنا عندي عن طريق اني بنشأ class باسم الفصيلة
الأخير واحد فيه الحاجات المشابهة كلها سواء صفات او مهام والـ classes الصغيرة
هندست فيها الصفات والمهام اللي تكون مختلفة من كائن لل الثاني ولما هنجي ننشأ
عن "animal" الكبير class "dog, cat, lion" هو يرث من الـ class الكبير طرق extends + الـ class الكبير

```
class Animal{           class Lion extends Animal{    class Dog extends Animal{  
int numberOflimsb = 4; void roar(){  
eat(){}          print('roaring');}  
print('eating');}      }  
}  
 وكذلك الأمر مع cat وهنلاقي الـ objects اللي انت بتنشأها  
يكون ليها access على اللي جوا اللـ class Animal عادي جداً
```

أنا كد بقوله أنا هنشأ فصيلة lion مثل وبقوله خللي بالك ان هو Animal بمعنى انه
هيرث كل الصفات والمهام الموجودة في الـ class اللي اسمه Animal نقدر نعمل
نكرر كلام كثير

```
void main(){  
Dog jack = Dog();  
Cat kitty = Cat();  
Lion scar = Lion();  
scar.eat();        kitty.eat();        jack.eat();  
jack.bark();      scar.roar();  
}
```



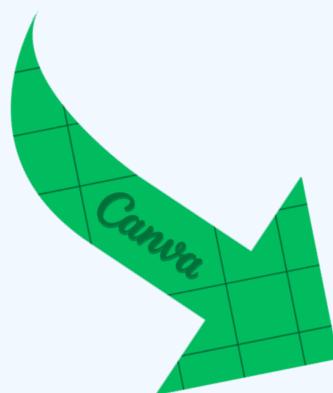


Inheritance

main.dart

```
void main(){
Lion scar = Lion();
Cat jack = Cat();
Dog kitty = Dog();

scar.eat(); // eating
jack.eat(); // eating
kitty.eat(); // eating
scar.roar(); // roaring
jack.bark(); // barking
kitty.meo(); // meoing
}
```



Inheritance

File1.dart

```
class Animal(){
    int numberofLimbs = 4;
    eat(){
        print('eating');
    }
    sleep(){}
    walk(){}
}

class Cat extends Animal{
    void meo(){
        print('meoing');
    }
}

class Dog extends Animal{
    void bark(){
        print('barking');
    }
}

class Lion extends Animal{
    void roar(){
        print('roaring');
    }
}
```

SUPER CONSTRUCTOR

طبعا كلنا عارفين ان مع اي class بننشأه بيكون فيه constructor فاضي او فيه متغيرات بنسخدمه علشان نبني ال object بتاعنا بشكل سريع مثل لو أنا عندي متغيرات متعرفة في ال class الكبير "Animal" وعايز أعمل access للمتغيرات دي في ال classes الصغيرة "cat, dog, lion" في ال constructor بتاعها مش هعرف لأن ببساطة المتغيرات دي مش متعرفة في ال classes الصغيرة علشان نحل المشكلة دي عندي طرفيتين

الطريقة الأولى

وهي عن طريق اني بعمل constructor في ال class الكبير "Animal" علشان أقدر أعمل create object بسرعة اللي هنثر منه الكلام دا

String? skinColor; double? weight; Animal(this.skinColor, this.weight);

وبالتالي ال classes الصغيرة "cat, dog, lion" بقت مجبرة عند إنشاء الكائن إنها تستقبل القيم دي وعلشان القيم اللي هنخزنها في اي class من الصغارين لازم نروح نخزن ال attributes الموجودة في ال Animal فيه نقدر نعمل كده عن طريق super constructor وهو عبارة عن ال constructor بتاع ال class الكبير مع تغيير ال إلى super فقط وبكده نكون خلمنا

```
class Animal{  
    int numberOflimsb = 4;  
    String skinColor;  
    double weight;  
    Animal(required this.skinColor, required this.weight);  
    eat(){  
        print('eating');}  
}
```

```
class Dog extends Animal{  
    Dog(required super.skinColor, required super.weight);  
}
```

الطريقة الثانية

وهي عن طريق اني بنشأ متغيرات في ال constructor
بتاع ال classes الصغيرة وبيعث قيم ليها من ال attributes الموجودة في ال class
الكبير عن طريق إستدعاء ال constructor بتاعها

```
class Animal{  
    int numberOflimsb = 4;  
    String skinColor;  
    double weight;  
    Animal(required this.skinColor, required this.weight);  
    eat(){  
        print('eating');}  
}  
  
class Dog extends Animal{  
    Dog(required String skinColor, required double weight).super  
        (skinColor: skinColor,  
         weight: weight);  
}
```

هنا أنت بقوله انت هتستقبل القيم بتاع ال Dog مثلا وبعد كده تستدعي ال constructor
بتاع ال Animal class skinColor وهقوله خزنلي ال skinColor في ال Animal class
بتاع ال Dog هنا وكذلك الأمر مع ال weight هقوله خزنلي ال weight بتاع
ال Animal class في ال weight بتاع ال constructor هنا بس كده أتمعني تكون وضحت
النقطة دي

كانت بتقول أنا عايزه أستقبل القيم بتاع المتغيرات دي هنا في ال constructor دا
للمتغيرات اللي عندي .

كانت بتقول أنا عايز أستقبل المتغيرات بتاعتي وهديها لـ attributes بتاع الكائن اللي
أنا برض منه .

First Way Of Super Constructor



Super Constructor

```
class Animal(){
    int numberofLimbs = 4;
    String skinColor;
    double weight;
    Animal(required this.skinColor, required this.weight);
    eat(){
        print('eating');
    }
    sleep(){}
    walk(){}
}
class Dog extends Animal{
    Dog(super.skinColor, super.weight);
    void bark(){
        print('barking');
    }
}
```

Second Way Of Super Constructor



Super Constructor

```
class Animal(){
    int numberofLimbs = 4;
    String skinColor;
    double weight;
    Animal({required this.skinColor, required this.weight});
    eat(){
        print('eating');
    }
    sleep(){}
    walk(){}
}
class Dog extends Animal{
    Dog({required String skinColor, required double weight}).
super(skinColor: skinColor, weight: weight);
    void bark(){
        print('barking');
    }
}
```

override

لو أنا عندي class اسمه Animal فيه مجموعة attributes, methods فيه مجموعة methods كلها وطبعاً أي class يرث منه هيرث الحاجات دي كلها ويقدر ينفذ كلها بنفس الشكل بالضبط

لكن في بعض الأوقات بيكون عندي صفات مشتركة بين كائنات لكن بيتم تأديتها بشكل مختلف فمش طبيعي اني أكررها في كل class على حسب اللي بيتم فيه بنلاقي مثلاً في حيوان بيأكل بطريقة مختلفة عن باقي الحيوانات نحلها إزاي !!!!
نقدر نحل حاجة زي كده عن طريق إنشاء method بنفس الاسم اللي في الـ class الكبير في النوع دا ونديله الأوامر اللي هي عملها بالضبط

```
class Animal{                                class Dog extends Animal{  
    int numberOflimsb = 4;                  void eat(){  
    eat();                                print('dog is eating');  
    print('eating');                      } }  
} }
```

وهنا لو استدعيينا الدالة دي هنلاقيه بينادي على اللي في الـ class الصغير كأولوية وينفذ اللي جواه

الـ override بإختصار شديد إني برت بعض المهام زي الـ method اللي اسمها eat مثلًا ولكن حابب أنفذها بشكل مختلف "الكود اللي بين {} يكون مختلف" وبكده اللي جوا الـ class الصغير يكون قدر انه يعمل override للـ اللي جوا الـ class الكبير وبنجي فوق الـ method اللي بتعمل override وبنكتب فوقها `@override` للتوضيح بحيث ان لو دد قرأ الكود بعدى يقدر يفهم حاجة زي كده + أنها مش بتضيف اي حاجة للكود إنما توضح فقط



@override

```
void main(){
Dog jack = Dog();
jack.eat(); //will print 'dog is eating' not 'eating'
}
```



@override

```
class Animal(){
    int numberofLimbs = 4;
    eat(){
        print('eating');
    }
    sleep(){}
    walk(){}
}
class Dog extends Animal{
    @override
    void eat(){
        print('dog is eating');
    }
    void bark(){
        print('barking');
    }
}
```

POLYMORPHISM

بمعنى متعدد الأشكال

يعني أي !!! ونقدر نستخدمه إزاي !!!!! وليه أستخدامه أصلًا !!!! هنعرف كل دا بالتفاصيل واحدة واحدة

مثلاً فلنفترض ان أنا عايز أنشأ حديقة حيوانات "zoo" طبعاً هيكون فيها مجموعة حيوانات "Animal" ممكن أعمل دا عن طريق

```
List<Animal> zooAnimals = [scar, jack, kitty];
```

وبكلده أنا أكون طبقت مفهوم ال polymorphism من غير ما تأخذ بالك أصلًا !!!
أنا مخزن مجموعة من الحيوانات كل واحد فيه type مختلف "Dog, Lion, Cat" مع
إن أنا أصلًا محدد ال type بتاع ال list هو

هنا هيقبل حاجة زي كده بسبب ال polymorphism لأن ال Dog, Lion, Cat Classes
من ال Animal class فبكل بساطة أقدر أقول إن أي class يرث منه هو شكل مختلف
ليه

معنى كده إن أي class يرث منه انت عباره عن شكل مختلف منه

هو تجميع مجموعة من الأشكال المختلفة تحت شكل واحد : polymorphism
علشان نقدر نخزنها بشكل سليم وعلشان يتنفذ لازم تكون عاملين inheritance





Polymorphism

```
void main(){
Dog jack = Dog();
Lion scar = Lion();
Cat kitty = Cat();
List<Animal> zooAnimals = [scar, jack, kitty];
}

class Animal{
    int numberofLimbs = 4;
    eat(){
        print('eating');
    }
}

class Dog extends Animal{}
class Cat extends Animal{}
class Lion extends Animal{}
```

ABSTRACT CLASS

we will understand with example

فلنفترض ان انا عندي مجموعة من الكائنات eat method عندها ولكن الحيوانات دي كل واحد بيأكل بطريقة مختلفة عن الثاني فبقيت متضرر اني اروح لكل class واعمل فيه Override علشان انا بتعامل حاليا مع الكود انما انا لو واحد اول مرة اتعامل مع الكود ومعرفش ان كل كائن ليه طريقة اكل مختلفة وعملت classes جديدة هعرف ازاي اني لازم اعمل override for eat method !!!!!

الفكرة بقى هنا ان لازم اي class ننشأه يكون يرث من ال class اللي أكبر منه انه يعمل على override for eat method

.1 ان ال class اللي احنا بنرث منه هنخليه abstract

.2 اروح على ال method اللي انا عايز يتعملاها override اجباري واشيل method implementation بتاعها "{}" وبكده انت بقى مجبوا ت عمل override لل class اللي في ايه class يرث من ال class اللي هي فيه

الـ abstract class بتديني إمكانية اني معملش implementation للـ method فـ أنا مقولتش طريقة تنفيذـ الـ method دي وبالتالي في كل class يرث من الـ class اللي هي فيه لازم نعمل override ليها ونحطـ الـ implementation الخاصـ بيـها

Notice: لازم نخلي بالـ انـ الـ abstract classes معرفـش أعملـ منها object نقدر نستخدمـها كـ typeـ والـ الكـائنـاتـ المـتـخـزـنةـ هـنـلاـقـيـهاـ معـمـولـهـ بـ تـرـثـ منـ الـ abstract classـ دـاـ لـكـنـ مشـ منـ الـ classـ نـفـسـهـ





Abstract Class

First Step

```
abstract class Animal(){  
    int numberofLimbs = 4;  
    eat();  
}
```

Secind Step

```
class Dog extends Animal{  
    @override  
    eat(){  
        print('dog is eating');  
    }    }  
class Cat extends Animal{  
    @override  
    eat(){  
        print('cat is eating');  
    }    }  
class Lion extends Animal{  
    @override  
    eat(){  
        print('lion is eating');  
    }    }
```

این لاملاً بیان می‌کند که این کلاس از کلاس Animal ارث می‌کند
کلاس Dog از کلاس Animal ارث می‌کند
کلاس Cat از کلاس Animal ارث می‌کند
کلاس Lion از کلاس Animal ارث می‌کند

IMPLEMENTATION

لما جينا نخلی class بيرث من class تاني استخدمن كلمة اسمها extends بس مش دي الكلمة الوحيدة اللي بتستخدم للتوريث في كلمة كان اسمها implements تقدر تعمل كده برضو

في فرق واحد وهو في طريقة التنفيذ بتاعتي implements هي طريقة مختلفة من الوراثة

```
class Lion Implements Animal {}
```

أنا مش بقوله الـ class اللي اسمه lion بيرث من الـ class اللي اسمه Animal فقط لا أنا بقوله خللي بالك دا بيرث منه ولازم ينفذ كل حاجة اتزكرت في الـ class اللي اسمه Animal بالشكل الخاص بي

معنى إن أي حاجة انت بتترثها من Animal إبدأ ونفذها في بشكل مختلف في Lion سواء methods أو attributes

بنستخدم Implements لما يكون عندي class فيه مجموعة methods وعايز أرثهم كلهم بشكل مختلف سواء كان الـ class اللي انت بتترث منه او لاء





```
class Animal(){  
    int numberofLimbs = 4;  
    eat(){}
    sleep(){}
    walk(){}
}  
class Lion implements Animal {  
    @override  
    int numberofLimbs = 2;  
  
    @override  
    void eat(){  
        print('lion is eating');  
    }  
    @override  
    void walk(){  
        print('lion is walking');  
    }  
    @override  
    void sleep(){  
        print('lion is sleeping');  
    } }
```

هنلليظ هنا ان

Lion class

يعمل

implements

For Animal class

فلارزم كل حاجة في
Animal class انفذها

بسكل مختلف في

المثالي زي Lion class

المثال دا يا شباب

NAMED CONSTRUCTORS

انا هنا عايز أعمل لعبة رسم circle وعايزين نمثل الدائرة دي كـ
دا هعمله عن طريق oop أعمل class علشان أقدر أمثلها كـ

```
class Circle{ }
```

دلوقي أنا مهتم أضيف الـ attributes, methods
radius, إللي اي دائرة تحتاجها مثلًا:
placeOfX, placOfY

علشان أقدر أعرف مكان الـ CenterOfCircle

```
void main(){
```

```
Circle circle1 = Circle(radius: 5, x: 3, y: 4);
```

```
circle1.draw();
```

```
}
```

```
class Circle{
```

```
double radius;      int x;          int y;
```

```
Circle({required this.radius, required this.x, required this.y});
```

```
void draw(){
```

```
print ("draw circle at x: $x , and y: $y , with radius: $radius");
```

```
}
```

في بعض الأوقات بنكون محتاجين بنبني الكائن بشكل مختلف شوية لأن أنا ممكن
إنشأ أكثر من constructor كل واحد بيشتغل بطريقة مختلفة عن الثاني ممكن ابني
وادي بنستخدمها علشان ابنى الكائن بشكل متميز ومختلف عن
الـ constructor العادي وبتدي إمكانية رسم الـ object بشكل أسرع

أنا مثلًا عايز أديه إمكانية إنه يرسم الـ circle في الـ "x= 0, y = 0" نقدر
نعمل دا عن طريق الـ named constructor يعني انه يبني الكائن في المكان اللي

הבדבבם

Circle. + NameOfConstructor (attributes) { values of attributes that you will give }

Example: Circle.origin({required this.radius}){ x = 0; y = 0; }



Named Constructor

```
void main(){
Circle circle1 = Circle.origin(radius: 5);
circle1.draw();
}

class Circle{
double radius;
int x;
int y;
Circle({required this.radius, required this.x, required this.y});

draw(){
print('draw circle at x: $x, y: $y with radius: $radius');
}
// Named Constructor
Circle.origin({required this.radius}){
x = 0;
y = 0;
}
}
```

LATE KEYWORD

هي كلمة بنسخدمها علشان تساعدي في حل مشكلة ال null safety
بستخدمها طالما أنا متأكد إن المتغير دا هترجع فيه قيمة يبقى مش لازم أخليه late لأن كده انت تقدر تخزن فيه قيمة null بالفعل إنما نستخدم بقى كلمة nullable قبل الـ data type بتاع المتغير دا

Example: late int x;

لان ببساطة كلمة late بتقول للattribute خللي بالك ان x هتاخذ قيمتها بس مش دلوقتي ولكن اكيد هتاخذ قيمتها قبل ما يتم استخدامها
بستخدمها لما اكون متأكد ان المتغير كده كده هيابد قيمة قبل ما يتم استخدامه لحل مشكلة ال null safety

MIXINS

عندي Animal class فيه attributes, methods عملنا كده علشان نقدر نمثل Animal class الحيوانات بس الوضع الحالي دا مش أحسن حاجة لأن أنا لو رحت عملت مثلا Snake class فهـي هترث attributes, methods من الـ Animal class فلنفترض مثلا إن في walk method في الـ Animal class بس الـ Snake مبتعملاش الـ Animal method دي فـهـضرـانـيـ اـشـيلـهـاـ منـ الـ Animal وابتـهاـ فـيـ الـ classesـ الـ يـتـعـمـلـ walkـ وـ crawlـ مشـ أـحـسـنـ حـاجـةـ بـرـضـوـ لأنـ كـداـ هـيـكـونـ فـيـ اـكـواـدـ بـتـتـكـرـ كـتـيرـ فالـحلـ اـنـيـ اـقـسـمـ الحـيـوـانـاتـ مـثـلـ لـنـوـعـ بـيـمـشـيـ وـنـوـعـ بـيـزـحـفـ وـاـخـلـيـ الـحـيـوـانـاتـ تـرـثـ مـنـهـمـ بـجـانـبـ وـرـاثـتـهمـ منـ الـ Animalـ classـ

class Animal{}

class Mammals{walk(){print('Animal is Walking'); } }

class Reptiles{crawl(){print("Animal is Crawling"); } }

بس المشكلة ان ال class اللي هنشأه

لازم يرث كده من Animal , Mammals or Reptile Classes

بس هنا في مشكلة وهي ان مش مش متاحة عندنا في dart

ان class يرث من اكتر من class ممنوع

وهنا تظهر أهمية ال Mixins هي شبيهة بـ class بتشيل

عند class بناء على mixin اغيرها لـ Mammals, Reptiles واحد جواها اللي اناحتاجه

وبكده ال class اللي هنشأه يقدر يرث من Animal, Reptile or Mammals من خلال:

class Dog extends Animal with + anotherClass{}

```
void main(){
```

```
Dog jack = Dog();
```

```
jack.walk();
```

```
}
```

```
class Animal{}
```

```
mixin MammalsMixins{
```

```
walk(){
```

```
print('Animal is Walking');
```

```
}
```

```
mixin ReptileMixins{
```

```
crawl(){
```

```
print('Animal is Crawling');
```

```
}
```

```
class Dog extends Animal with MammalsMixins {}
```

1. وقدر بكده من خلال ال mixins انه ترث من اكتر من class واحد

2. ولو صادفت انه يترااث من 2 classes فيهـم method واحدـة انت بترااث منها ليـها الاولـوية

3. ودا بيـديـنا control بـشكل رـهـيـب وـاـنـ الـكـودـ بتـاعـيـ يكونـ شـغـالـ بـشـكـلـ سـلـيمـ



Mixins

```
void main(){
Dog jack = Dog();
jack.walk(); // pet is walking
Snake s1 = Snake();
s1.crowl(); // animal is crowling
}

class Animal{}

mixin mammalsMixin(){
void walk(){
print("animal is walking");
} }

mixin reptileMixin(){
void crowl(){
print("animal is crowling");
} }

mixin petMixin(){
void walk(){
print("pet is walking");
} }

class Dog extends Animal with mammalsMixin, petMixin{}
class Snake extends Animal with reptileMixin{}
```

OBJECT CLASS

لو رحت أنشأت Lion class

```
class Lion{}
```

ورحت على main وعملت منه object

```
void main(){
```

```
Lion scar = Lion(); }
```

ورحت كتبت scar. هلاقي ظهرلي حاجات أنا مش عارف أطلها هي جاية منين !!!!

"toString, runtimetype, hashCode"

الموضوعة بسيطة جدا لأن أي class انت بتعمله هو بيكون by default extends من Object class و الـ Object class

ممكن نحتاجه في بعض الاوقات زي في بعض الـ methods اللي ممكن اضطر اعملها
Object class منها الـ equality override مثل:

```
Lion scar1 = Lion();
```

```
Lion scar2 = Lion();
```

```
print(scar1 == scar2);
```

المفروض هنا يطبعلي true بس لاء هلاقي طبع لأن هنا انت مش بتقارن
بسيطة "int, string" لاء هنا بنقارن Object بـ Object تاني الموضوع اكتر
تعقيد فـ method اللي بتبقى راجعة من الـ Object class علشان تقدر تقارن object
بـ object تاني لو عايز احل المشكلة دي هضر اني عمل override للـ method اللي
راجعي من الـ Object class علشان اقارن بين object والـ object

ANONYMOUS OBJECTS

لو انا عندي مثل Cat, Dog classes و Animal class بيرثو منه
أقدر أعمل objects من الـ list الـ cat, dog objects فيها الـ Animal اللي عملتها

```
void main(){  
Dog jack = Dog();  
Cat kitty = Cat();  
List<Animal> animals = [jack, kitty];  
}  
  
class Animal{}  
class Dog extends Animal{}  
class Cat extends Animal{}
```

الـ line ده ; Dog jack = Dog() بينشأ object اسمه jack وطبعاً الجزئية اللي بعد = هي
الـ reference اللي بتنشأ لإنها الـ constructor وبعد كده عطيناه الـ name jack كـ reference
 حاجة بتشاور على الـ object عندي في الـ zakerة
 منقدرش نقول ان jack هو الـ object انما بنسخدمه علشان نجيب بيانات من الـ object
 وعملت كده علشان اوصل للبيانات بسهولة من خلال الـ name

يعنى ان انا لو عملت Dog(); كده أنا أنشأت object بدون اسم فمش هعرف أنا دي
عليه واجيب البيانات اللي فيه لأن الـ name هو حاجة مهمة جداً جداً
لأنه يكون reference للبيانات اللي أنا مخزنها وهنا يظهر الـ object anonymous
عبارة عن object بدون اسم

ENUMS

فلنفترض ان ادنا جايين نعرف متغير اسمه gender

String gender = "male";

الموضوع بسيط بس لازم اخلي بالي ان المتغير دا مميز شوية لأن ليه قيمة محددة "اللي ممكن يشيلها بس الفكرة بقى ان انت تخزن اي حاجة غير القيمتين دول بس دا مش logic

علشان كده في طريقة أقدر أخزن من خلالها قيمة محددة وهي الEnums بستخدمها لما ألاقي نفسي هخزن مجموعة من القيم محددين "قيم أقدر أعدها"

1. يعرف الenum + بديها اسم

2. تخزن فيها القيم بتاع المتغير اللي ممكن يشيلها

Example: enum Gender{ male, female }

بنخزن القيم كـ text مش

ولما اجي اعرف المتغير ; gender1 = Gender.male وبكده تكون لحقت نفسي

من اللخبطة احسن اكتب القيمة غلط بدلا من ;

وفي التعامل كمان خصوصا لو كانت مع if statement, or switch

```
if(gender1 == Gender.male){
```

```
print('this is male');
```

instead of

```
if(gender2 == 'male'){
```

```
print('this is male');
```

الحاجة اللي بتنكتب باليد تكون عرضة انها تكون خطأ علشان كده بنستخدم

الenum لتفادي الأخطاء الكتابية

وكمان استخدامها مع الswitch case بيكون لذيذ جدا لأن الـ switch اصلاً بيستخدم

علشان تعمل check على قيمة محددة



Enum

```
void main(){
Gender gender1 = Gender.male;
switch(gender1){
    case Gender.male:
print('this is male');
break;

    case Gender.female:
print('this is female');
break;
}
}

enum Gender {
male, female
}
```



وکانت هي دي نهایة کورس ال OOP معانا يا خباب فرمان
أوي إتنا غطينا الجزئيات دي کلها

**“Encapsulation, Inheritance, Polymorphism,
Abstraction, Override, Mixins, implement,
Named Constructors, Super Constructor”**

وصابات تانية أتنى تكونو کلام استغرتسم ولو صد وقف معاه
صاحبة يقدر يعطاي Whatsapp وإن شاء الله أقدر أساعدك

Whatsapp: 01227897361

ال OOP طالا اتعلمه بـ لغة ايا كانت من هستاج تراکرها بلغة
تاني إن شاء الله وهلاقو هنا صابات زيادة لو انت من
متخصص dart من هستاجها هلاقيها في الأضـ
وفـقـنا اللـهـ وـإـيـاكـمـ لـاـ بـعـدـ وـيرـضاـهـ