✅ ايه هي OOP وايه هي المبادئ الأساسية لها؟

- **OOP = Object Oriented Programming**:
  - أسلوب برمجة بيعتمد على الكائنات (Objects) والكلاسات (Classes)
  - بيخلي الكود منظم، سهل الصيانة وإعادة الاستخدام.
  - **المميزات**:
    - يسهل إعادة الاستخدام (Reusability).
    - يقلل تكرار الكود (Code Duplication).
    - يسهل الصيانة والتوسع.
  - **المبادئ الأساسية (Pillars 4):**

1. **Encapsulation**
2. **Inheritance**
3. **Polymorphism**
4. **Abstraction**

---

2. إيه هي الـ 4 مبادئ الأساسية في OOP؟

1. **Encapsulation (التغليف):**
2. حماية البيانات وربطها بالوظائف اللي بتتعامل معاها.
3. **Abstraction (التجريد):**
4. إخفاء التفاصيل الغير مهمة وإظهار الجزء الأساسي.
5. **Inheritance (الوراثة):**
6. إعادة استخدام كود من Class آخر (توريث خصائص ووظائف).
7. **Polymorphism (تعدد الأشكال):**
8. نفس الميثود ممكن يكون ليها سلوك مختلف حسب السياق (Overriding/Overloading).

---

3. يعني إيه Class؟

- ** هو عبارة عن قالب (Blueprint)** بيُستخدم لإنشاء كائنات (Objects).
  بيحتوي على **Properties** (بيانات/Attributes) و **Methods** (سلوكيات/Functions).

يعني إيه Object؟ وإزاي بنعمله من Class؟

- **Object:**
  - نسخة (Instance) من الكلاس.
  - بيتعمل باستخدام الكلمة `new` .

الفرق بين Class و Object؟

- **Class:**
  - هو **التصميم/القالب (Blueprint)**.
  - بيحدد الخصائص (Properties) والدوال (Methods) اللي أي Object معموله هيحتوي عليها.
  - مش حاجة ملموسة، مجرد وصف.
- **Object:**
  - هو **النسخة (Instance)** اللي اتعملت من الكلاس.
  - الكائن ده هو اللي بيبقى موجود فعليًا في الذاكرة وتقدر تتعامل معاه.

---

6. إيه هو Constructor؟

- هو **دالة خاصة** موجودة داخل الكلاس.
- بتتندَه أوتوماتيك أول ما نعمل **Object (instance)** من الكلاس.
- الغرض الأساسي: **تهيئة الخصائص (initialize properties)** بالقيم الأولية.
- كل كلاس ممكن يكون فيه **Constructor واحد بس**.

---

7. إيه هي Properties و Methods في الكلاس؟

- **Properties (الخصائص):**
  - هي **المتغيرات** اللي بتتخزن جوه الكائن (object).
  - بتمثل **البيانات أو السمات** الخاصة بالكائن.
- **Methods (الوظائف/الدوال):**
  - هي **دوال** موجودة داخل الكلاس.
  - بتمثل **السلوكيات أو الأفعال** اللي الكائن يقدر يعملها.

🎤 سؤال: ما الفرق بين Constructor و Method؟

> **Constructor**
>
> دالة خاصة تتنفذ تلقائي أول ما يتعمل Object من الكلاس وهدفها تهيئة القيم أو حجز الموارد.
>
> **Method**
>
> دالة عادية بنستدعيها يدويًا لتنفيذ كود أو عملية معينة، ممكن ترجع قيمة، واسمها أي حاجة مناسبة.

✅ ايه هي الـ Abstraction؟

- الـ **Abstraction** هو **إخفاء التفاصيل المعقدة (Implementation Details)** وإظهار فقط **الوظائف المهمة (What the object does)** للمستخدم.
- الهدف منه إنك **تتعامل مع الكود بطريقة أبسط** من غير ما تشيل هم التفاصيل الداخلية.

---

3. الفرق بين Abstraction و Encapsulation؟

- **Abstraction:**
  - إخفاء التفاصيل وإظهار المهم.
- **Encapsulation:**
  - حماية البيانات نفسها وربطها مع الدوال.

يعني إيه Abstract Method ؟

- دالة مُعرّفة من غير Body (من غير كود).
- لازم الكلاس الابن يكتب الكود الخاص بيها (Implementation).

✅ 1. Abstract Class

📌 التعريف:

- الـ **Abstract Class** هو كلاس عادي لكن فيه **على الأقل دالة واحدة Abstract (من غير Implementation)**.
- الكلاس ده **غير مكتمل** ← يعني مينفعش نعمل منه Object مباشرة.
- لازم كلاس تاني يرث منه ويكمّل تنفيذ الدوال الـ Abstract.

📌 الخصائص:

1. ممكن يحتوي على:
   - **دوال عادية** (Implemented Methods).
   - **دوال Abstract** (بدون تنفيذ).
   - **متغيرات** عادي جدًا.
2. الهدف منه:
   - يكون **Base Class / Blueprint** للكلاسات اللي هترثه.
   - يفرض على الكلاسات اللي ترثه إنها **تنفذ (Override) الدوال الـ Abstract**.
3. في بعض اللغات:
   - في **C++** ← اسمها **Pure Virtual Function**.
   - في **C# / Java** ← فيه كلمة مفتاحية `abstract` .

✅ 2. Interface

التعريف:

- الـ **Interface** هو عبارة عن **عقد (Contract)** بيحدد مجموعه دوال (methods) لازم أي Class يطبقها يوفر ليها **Implementation**.
- بيقول لأي كلاس: "لو عايز تطبقني، لازم تنفذ كل الدوال اللي جوايا".
- يعني 100% دوال Abstract (مفيش أي كود تنفيذي).
- في **C# / Java** فيه كيان رسمي اسمه `interface` .
- في **C++** مفيش `interface` ككلمة محجوزة، لكن بنعمل نفس الفكرة باستخدام **Abstract class كلها دوال pure virtual**.
-
- لأ، ما ينفعش تعمل **Object مباشر** من الـ **Interface** لأنه **مش فيه Implementation** (مفيهوش كود يتنفذ). هو مجرد **Contract** بيحدد Signatures للدوال.

✅ الفرق بين Interface و Abstract Class

- **Interface**:
  - كل الدوال مجردة بالكامل (Full Abstract).
  - مفيهوش أي Implementation.
  - بيدعم **Multiple Inheritance** (الكلاس يقدر يرث أكتر من Interface).
- **Abstract Class**:
  - ممكن يحتوي على دوال مجردة + دوال فيها Implementation.
  - مينفعش أعمل منه Object.

- الوراثة فيه بتكون من كلاس واحد (Single Inheritance).

✅ ايه هي الـ Encapsulation؟

- **التعريف**: هو مبدأ من مبادئ الـ OOP، معناه إنك "تغلف" البيانات (variables) والدوال (methods) اللي بتتعامل معاها في كيان واحد (Class)، وتتحكم إزاي الناس توصل للبيانات دي.
- هو إني أخفي البيانات الحقيقية للكلاس وأخلي التعامل معاها بس عن طريق دوال زي الـ Getter والـ Setter. الهدف منه حماية البيانات والتحكم في الوصول ليها.

الفكرة الأساسية

- إنك تخلي المتغيرات (Attributes) **private** بحيث محدش يقدر يوصلها أو يعدل عليها مباشرة من برّه الكلاس.
- وتسمح بالوصول أو التعديل عليها عن طريق **public methods** زي الـ **getters** و **setters**.

الهدف من الـ Encapsulation

1. **حماية البيانات** (Data Hiding) ← محدش يقدر يغير القيم بطريقة غلط.
2. **سهولة الصيانة** ← لو غيرت طريقة التخزين، مش هتأثر على اللي بيستخدم الكلاس.
3. **التحكم في الوصول** ← تقدر تحط شروط قبل ما تقبل أي تعديل.

---

3. الفرق بين getters و setters؟

- **Getter:**
  - ميثود بيُستخدم عشان **نقرأ/نرجّع قيمة خاصية (Property)** من الكلاس.
- **Setter:**
  - -ميثود بيُستخدم عشان **نعدّل/نغيّر قيمة خاصية (Property)**.

بيستخدموا للتحكم في طريقة الوصول والتغيير.

✅ 1. تعريف Access Modifiers

- **Access Modifiers =**

- "مستويات الوصول" أو "صلاحيات الوصول".
- وظيفتها إنها **تحدد مين يقدر يوصل (يستخدم) إلى المتغيرات والدوال بتاعت الكلاس**.
- يعني ببساطة: بتتحكم في "إيه اللي مسموح يظهر لبرا الكلاس" وإيه اللي يفضل مستخبي.

✅ 2. الأنواع الأساسية في Java / C++:

🔹 Public

- أي حاجة **public** تقدر توصل لها من أي مكان (جوه أو برة الكلاس).
- مثال: زي "واجهة الكلاس" اللي أنت عايز الناس تستعملها.

🔹 Private

- متاح بس جوه الكلاس نفسه.
- دي بتستخدمها علشان تعمل **Encapsulation** (إخفاء البيانات).

🔹 Protected

- أي حاجة **protected**: متاح جوه الكلاس والـ subclasses اللي وارثاه.

✅ 7. يعني إيه Virtual Function؟

- دالة بتتعرّف في الكلاس الأب وبتتعملها override في الكلاس الابن.
- بتمكّننا من تحقيق **Polymorphism (Runtime)**.

more information

الـ **Virtual Function** هو **دالة في الكلاس الأب (Base Class)** بنعلن إنها "افتراضية"، يعني:
لو الكلاس الابن (Derived Class) عمل **Override** لها ← هيتنفذ الكود بتاع الابن.
لو ماعملش Override ← هيتنفذ الكود الافتراضي اللي في الأب.

✅ تعريف Friend Function

- **Friend Function =**
- دالة **مش عضو (Not a member) في الكلاس،**
  لكنها بتاخد صلاحية خاصة تقدر توصل للـ **private** و **protected members** جوا الكلاس.

- بيتعملها تعريف باستخدام الكلمة المحجوزة `friend` جوه الكلاس.

✅ Static Method

📌 التعريف:

- الـ **Static Method** هي دالة بتتكتب جوه الكلاس بكلمة مفتاحية `static` .
- **مش مرتبطة بأي Object** ← يعني مش محتاجة تنشئ Object علشان تستدعيها.
- ممكن استدعائها مباشرة باستخدام **اسم الكلاس** بدل ما تناديها من خلال Object.

---

10. يعني إيه Polymorphism؟

📌 التعريف:

- الـ **Polymorphism** هو مبدأ من مبادئ الـ OOP.
- معناه: **نفس الميثود ممكن يتنفذ بشكل مختلف حسب الكلاس أو حسب السياق.**

✨ الأهمية:

- بيخلي الكود **مرن (Flexible)**.
- بيسهل **إعادة الاستخدام (Reusability)**.
- بيسمح بإضافة **توسعات (Extensibility)** من غير ما نكسر الكود القديم.

عندنا نوعين:

- Compile Time Polymorphism زي الـ Function Overloading.
- Runtime Polymorphism زي الـ Function Overriding باستخدام Virtual Functions."

---

11. الفرق بين Overloading و Overriding

- **Overloading**:
  - نفس اسم الدالة لكن اختلاف في عدد/نوع البرامترز – بيحصل في نفس الكلاس.

- **Overriding**:
  - نفس الدالة في الكلاس الأب لكن بتتعمل override في الكلاس الابن.

---

12. يعني إيه Early Binding و Late Binding؟

- **Early Binding (Static Binding)**:
  - تحديد أي دالة هتتنفذ بيتم في وقت الـ Compile (زي overloading).
  - هو إن المترجم يحدد أي دالة هتتنفذ وقت الـ **Compile**. وده بيحصل في **Function Overloading** أو أي دوال عادية مش معمولة Virtual
- **Late Binding (Dynamic Binding)**:
  - تحديد الدالة بيتم وقت التشغيل باستخدام الـ Virtual Function (زي overriding).
  - هو إن القرار يتأجل لحد وقت التشغيل (**Runtime**) علشان نعرف أي نسخة من الدالة تتنفذ. وده بيحصل باستخدام **Virtual Functions** مع **Overriding**.

---

13. يعني إيه Constructor و Destructor والفرق بينهم؟

1 Constructor (المُنشئ)

- دالة خاصة (special member function).
- **بيتنفذ أول ما يتعمل Object** من الكلاس.
- بيستخدم **لتهيئة القيم (initialize variables)** أو لحجز موارد (مثل الذاكرة أو فتح ملفات).
- له **نفس اسم الكلاس**.
- **مفيهوش return type** حتى `void` .
- ممكن يكون عنده **باراميترات** ويدعم **Overloading**.

2 Destructor (المُدمر)

- دالة خاصة بيتنفذ **لما الـ Object يتدمر** (يخرج من الـ scope أو يعمل له `delete`).
- وظيفته: **تحرير الموارد** زي الذاكرة، الملفات المفتوحة، أو الاتصالات بالشبكة.
- له نفس اسم الكلاس **مع علامة** `~` في البداية.
- **مفيهوش باراميترات**.
- **مفيهوش overloading**.

- الفرق: الـ constructor بيتهيأ القيم، الـ destructor بيمسح/ينظف.

---

14. ينفع يكون الـ Constructor Private؟ وليه؟

- آه، ينفع.
- بيستخدم مع **Singleton Design Pattern** (لما عايز أسمح بوجود Object واحد بس من الكلاس).

✅ هل الـ private members بتتورّث؟

- أيوة، **الـ private members بيتورّثوا** تقنيًا (بيكونوا موجودين في الـ object اللي اتعمل من الـ subclass).
- لكن ❌ **مينفعش يتعاملهم access مباشرة** من الكلاس الابن.
- دوال `public` أو `protected` موجودة في الكلاس الأب.

---

16 ✅ Inheritance (الوراثة)

📌 التعريف:

هو مبدأ من مبادئ الـ OOP.
بيخلي كلاس الابن (Child Class) يرث الخصائص (Properties) والدوال (Methods) من كلاس الأب (Parent Class).

✨ الأهمية والفوائد:

إعادة استخدام الكود (Code Reusability): مش محتاج تكتب نفس الكود مرتين.

توسيع الكلاس الأب (Extensibility): تقدر تضيف دوال جديدة أو تعدل السلوك في الكلاس الابن.

---

17. ينفع أورث من أكتر من كلاس في نفس الوقت؟

- في C++: آه، ممكن (Multiple Inheritance).

- في Java/C#: لأ، لكن فيه Concept اسمه **Interface** لتحقيق نفس الهدف.

---

4. الفرق بين Single و Multiple Inheritance؟ وهل JS بيدعم؟

- **Single:**
  - كلاس يرث من كلاس واحد.
- **Multiple:**
  - كلاس يرث من أكتر من كلاس (مش مدعوم في JS).

---

18. يعني إيه Object ويعني إيه Class؟ والفرق بينهم؟

"الكلاس Class هو عبارة عن قالب أو تصميم بيحدد الخصائص والدوال.
الأوبجكت Object هو نسخة فعلية من الكلاس بتتخزن في الذاكرة وتقدر تستخدم الخصائص والدوال دي."

✅ الفرق بين Class و Struct في C++

- **Struct**:
  - الـ members الافتراضية بتكون **public**.
- **Class**:
  - الـ members الافتراضية بتكون **private**.
  - غير كده الاتنين تقريبًا بيعملوا نفس الدور (ممكن يبقوا فيهم دوال، وراثة، Constructors).

20.اي الفرق بين Stack vs Heap

- **Stack**:
  - تخزين البيانات الصغيرة والمؤقتة (local variables). أسرع.
- **Heap**:
  - تخزين الـ objects الكبيرة والديناميكية. أبطأ لكنه مرن.

21. Pass by Value vs Pass by Reference

- **Pass by Value**:
  - بيبعت نسخة من القيمة، التعديل ميأثرش على الأصل.
- **Pass by Reference**:
  - بيبعت العنوان، التعديل بيأثر على القيمة الأصلية.

---

22. What is the meaning of Copy Constructor? Why we use it?

- **Copy Constructor**:
  - وConstructor بياخد Object من نفس الكلاس كـ parameter وبيعمل نسخة منه.
- **Use**:
  - لما نحب نعمل نسخة عميقة (deep copy) أو ننسخ Object لـ Object تاني.

إزاي تطبق OOP في مشروع حقيقي؟

- تقسم المشروع لـ Classes (Users, Products, Orders).
- كل Class فيه البيانات والدوال الخاصة بيه.
- تستخدم الوراثة لما في علاقة "is-a"، والـ Composition لما "has-a".

✅ هل الـ Destructor ليه Overload؟

- لأ، الـ Destructor **ما ينفعش يتعمله Overload** لأنه:
  - ما بياخدش Parameters.
  - بيكون واحد بس لكل Class.
- استدعاؤه بيكون أوتوماتيكي لما الكائن يخرج من الـ Scope أو يتم تحريره