SOLID Principles

ايه هي ال solid اصلا هي اختصار ل 5 حاجات و هما

1- single responsibility

2- Open Closed

3- Liskov substitution

4- interface segregation

5- dependency inversion

و هما عباره عن مبادئ بنطبقها علشان نخلي كودنا احسن و professional اكتر و هيخلي الدنيا اسهل لو حد اشتغل على الكود بعدنا

و هنتكلم عنهم واحده واحده

**Single Responsibility Principle**

و هو اني اخلي ال class بتاعي بيعمل حاجه واحده بس او مسؤول عن حاجه واحده فقط علشان لو هو مرتبط ب حاجتين ف انا لو عوزت اغير ف اي حاجه من الحاجتين لازم ارجع لنفس ال class و اغير فيه و دا عكس مبدأ ال single responsibility

ف لازم اخلي ال class بتاعي مسئول عن حاجه واحده بس علشان احقق المبدأ دا

و الكلام دا مش على ال class بس هو على ال methods برضو ان تكون كل method بتعمل حاجه واحده بس

**Open / Closed Principle**

و هو اني اخلي ال class بتاعي open لإن حد يزود عليه حاجه لكن مخليهوش يقدر يغير عليه

و المبدأ دا بوصله عن طريق اني اطبق ال polymorphism في الكود بتاعي

يعني انا لو عندي class circle و class triangle و عندي class AreaCalculator بيحسبلي ال Area بتاع الاتنين مع بعض ف دا كده انا مبطبقش عليه ال Open / Closed ف فالحاله دي هعمل interface اسمه Shape هيكون فيه فنكشن اسمها Area   
و هنعمل class circle بي implement ال interface دا و هنحسب ال Area بتاعة ال circle جوا كلاس ال circle

و هكذا مع ال triangle

و نيجي بقى ل class ال area calculator دا هنخليه يادوب بينادي على ال classes و بينادي على الفنكشن اللي جواهم

**Liskov substitution**

و هو لما يكون عندي class عامل inherit من class تاني لما احط ال child class مكان ال parent class ميأثرش ف حاجه

مثال لو انا عندي class اسمه bird فيه method اسمها eat و method اسمها fly

لو جيت اخلي class اسمه pigeon يعمل منه inherit  
و خليت class تاني اسمه penguin يعمل منه inherit ف كده يبقى احنا مبنحققش ال liskov substitution  
لإن كده مش هنقدر نخلي ال penguin يحل محل ال bird  
و هيبقى عندنا فنكشن زياده فال penguin مش هنستخدمها

لكن فالحاله دي لو خلينا class ال bird فيه method ال eat بس و خلينا ال fly ف ال pigeon بس يبقى احنا كده طبقنا ال liskov substitution

**Interface segregation principle**

معناه ان الاحسن ان يكون عندي interfaces كتير كل interface بيعمل حاجه معينه من اني يكون عندي interface واحد لكذا غرض علشان لما اجي اعمل implement من ال interface دا مضطرش اعرف method انا مش هستخدمها

و دا حله اني بعمل كذا interface كل interface له غرض واحد معين

و لما اجي اعمل implement اعمل لكذا interface عادي

**Dependency inversion principle**

و دا معناه اني مخلي class ف ال high level module معتمد على class تاني في ال low level module

لكن يعتمد على حاجه abstract عادي

يعني حلها اني مخليش class يعتمد على class تاني بعينه لكن ممكن يعتمد على interface مثلاا

و ساعتها اقدر استخدم اي class عامل implement لل interface دا