SOLID Principles :

كنا بنكتب اكواد عادي وهوب لقينا الدنيا اتطورت وقالك لا عايزين نستخدم ال

ونعمل كلاس ونعمل فانكشنز ونستخدم ميثودز جواه ونرتب الدنيا , وبدل ما oop

احنا شغالين تاسكات صغيره يلا بينا نجمعهم في بروجيكت ومن هنا دخلنا دوامه دا ايرور عايز يتصلح ونصلح من حته يبوظ من حته , ودا جزء عايز يتعدل ودا جزء عايز يتضاف ودا جزء عايز يتمسح ودا كلاس معتمد علي

الكلاس دا فمش عارفين نعدل اوي ولا نتعامل .... و هوب الدنيا كبرت مننا ف كان لازم نحط زي قواعد كدا نأخذ بالنا منها قدر الامكان عشان نسهل علي نفسنا كل البروسيسيز اللي فوق دي

ومن هنا جه مصطلح

* SOLID Principles 🡨

كلها كابيتال ودا معناه اي ؟! ..... ان كل حرف بيشير SOLID مبديا كدا نلاحظ ان حروف كلمه

لحاجه معينه .... طب الكلام علي ايي ! .

**S 🡪 Single Responsibility**

اول مبدا معانا بيقول لما تييجي تعمل كلاس وتحط جواه فانكشنز وميثودز , الافضل انك تخلي لكل حاجه كلاس مش كلاس واحد يعملك كل حاجه يعني بمعني تاني جزأ المهام واعمل لكل حاجه كلاس بحيث لما تحب تعدل حاجه تعدل علطول عليها بسهوله ومنغير متأثر علي حاجه تانية زي مثلا:

لو عندك بتسجل الناس الجديده في الشركه وكمان بتحسب الضرايب الشركه ومرتبات الموظفين لو عملت فانكشن تعمل كل دا مع بعض كدا كل ما احب اعمل حاجه واحده بس هبقي مضطره استدعي الباقي ولو حبيت اعدل عن حاجه واحده احتمال كبير بيقي في لخبطه والتعديل كمان ممكن يأثر علي حاجه تانيه ف كده ممكن يحص منك مشاكل كتير في الأفضل انك تفصل كل حاجه لوحها للوضوح وتقليل المشاكل الممكن تظهرلك وتسهل عليك في الاستدعاء والتعديل والوقت

Class should have one responsibility

**O 🡪 Open-Closed**

هنا بقى بقولك ان الكود بتاعك مفتوح ومغلق في نفس الوقت , اكيد استغربت ازاي مفتوح ومقفول في نفس الوقت اقولك انا بص هنا انا قصدي مفتوح بالنسبه للتعديل عليه كحاجه اضافيه ومغلق انك تلعب في الاساس الموجود اصلا بمعني تاني انك تقدر تعدل عليه وضيف عليه تعديلات جديده بس بشرط متغيرش في الكود القديم هتقولي بردو مش فاهم أقول مثال:

انت مثلا وانت بتلعب لعبه زي بابجي او أي لعبه تانيه أيا يكن بينزلك تحديث لها تقريبا كل شهر التحديث ده بيكون عباره عن إضافة شخصيه جديده في اللعبه او سلاح جديد (ده المقصود بمفتوح)بس مبتجيش تلاقي اللعبه اتغيرة والاكواد القديمه بتاعتها اتغيرت وبقت اللعبه فيفا (ده المقصود بغلق)

L**🡪Liskov Substitution**

Inheritance دا يخص شويه فكره ال

ودا كل الفكره اللي فيه ان عندك كلاس الاب وكلاس الابن المفروض ان الابن قادر يعمل كل الحاجات اللي الاب بيعملها ومش بس كدا دا ممكن يزود عليها كمان دا انا كمان ممكن استغني عن الاب واخلي الابن هو اللي يقوم بالدورين فكل الفكره

ان لو عندي كلاس للاب وكلاس للابن اقدر ابدل بنهم من غير مبوظ حاجه هتقولي يعني ايه الكلام ده أقول هدى على رزقك وانا اقولك بص يعم هنا قصده لو عندك كلاس الاب وفيه مهام معينه بيقوم بيها وعندي كلاس للابن بيقوم بمهام معينه تانيه هتقولي يعني ايه كلاس اب وكلاس ابن بص الاب ده الكلاس الأساسي والابن ده هي بيعمل نفس الحاجات الفي كلاس الاب وزياده عليه بعض المهام الخاصه بيه زي مبيحصل في ال

طيب ازاي بقى ده بيعملInheritance

Tهو الاب

Sهو الابن

هنا انا عملت اوبجيكت من الاب وابجيكت من الابن وحجزت لهم مكان في الميموريT t = new T( ); S s = new S( );

ممكن اقول

T t = new T( ); Or T t = new S( );

هنا ان بقوله ان اوبجيكت ال انا عامله من الاب هيساوي وبجيكت الاب بعني لو الاب مش موجود يقدر الابن يعمل نفس الدور.

I**🡪Interface Segregation**

دا بقي بعد ما عملت كلاس وفانكشنز وظبطت البرنامج محتاج دلوقت تعمل انترفيس علشان تسهل علي اليوزر الاستخدام ف هنا بيقولك خلي لكل مهمه انترفيس لوحدها متحطش كل الفيتشرز ف انترفيس واحده

لو دلوقت عندك وجهة واحده بتعمل طباعه وتخزين دا احسن ولا لو جزء للطباعه لوحدها وجزء للتخزين لوحده ؟ افرض انت دلوقت عايز تطبع بس او تخزن بس فعلشان كده الأفضل انك تعمل لكل حاجه كلاس او وجهة يعني هنا هيكون واحده للتخزين ووحده تانيه منفصله عنها للطباعه هو الافضل

D**🡪Dependency Inversion**

**اولا المعني الحرفي ليها هو الاعتماديه المعكوسه**

**هو عايز يقول ان الطبيعي ان الاب بيعتمد علي الابن لانه فيه كل خصائص الاب وزياده بس دا هيوديني لمشكله ان الاب ممكن يكون بيعتمد علي الابن في بعض الحاجات ف انا بقا مش عايز الاب ( كلاس عالي المستوي ) يعتمد علي الابن ( كلاس منخفض المستوي ) خالص وهعالج دا لو استخدمت ال**

**Abstraction or interfaces**

**وتكون حاجات بسيطه مفيهاش تفاصيل**

 لو عندك كلاس بيدير قاعدة البيانات، المفروض ما يشتغلش مباشرة مع قاعدة بيانات معينة، لكن يشتغل مع واجهة عامة

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

بتأثر على الكود:Solidدلوقت هنعرف ازاي

أولا:

بيكون الكود اكتر قابليه للصيانه والتعديل عليه لانه بيكون اكتر تنظيما ولو محتاج تضيف او تعدل هتلاقس الكود مترتب ومفصول بطريقه منطقيه

ثانيا:

بيكون اكثر قابليه للتوسيع يعني الكود بيكون قابل للتطوير من غير ميحصل مشاكل او ايرور وتقدير تضيف مميزات جديده من غير متعدل على الكود القديم

ثالثا:

الكود بيكون اقل عرضه للمشاكل لان كل جزء في الكود بيكون مسؤول عن حاجه واحده منفصله فلو حصل مشكله تقدر توصل لها وتعدل عليها في اسرع وقت

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

دلوقت اقولكم بعض الامثله الواقعيه :

1-زي مقولة الألعاب تقدر تعدل وضيف شخصيات ومهارات جديده من غير متغير في الكود القديم

2-في الويب أي كود للويب بيكون مقسم لاجزاء علشان تقدر تعدل على كل حاجه جوه وتغير لونها مثلا لوحدها من غير متبوظ أي حاجه تانيه

**لانها أساس البرمجه الكائنيه وبتساعدك انك تكتب كود نضيف ومنظم خالى من المشاكل قابل اللاضافه فلازمSolidلازم نهتم**

**تتعلمه جيدا وبتدريج والوقت هتلاقي نفسك بتكتب كود افضل**

***By eng : Mostafa***

***Reviewed by eng: Martina Mina***