12/12/2022

Abdo Gamal Ahmed

no campany

Design pattern

Intro



Table of Contents

[1 –references : 2](#_Toc121934747)

[2 –Goal of course : 2](#_Toc121934748)

[3 – what is Design pattern “DP” ? 2](#_Toc121934749)

[4- What does the pattern consist of ? 3](#_Toc121934750)

[5 – who invented DP ? 4](#_Toc121934751)

[6 – Why should learn pattern ? 4](#_Toc121934752)

[7 – classification DP : 4](#_Toc121934753)

[8- Future of good Design: 4](#_Toc121934754)

[9 - Design principle : 5](#_Toc121934755)

# 1 –references :

* تانى كتاب الى اسمه head first دة من احسن واسهل الكتب الى هتلاقيها بتشرح ال design patterns او اى محتوى عموما .
* اما اول كتاب دة حلو لو انت عندك خلفيه شويه .

# 2 –Goal of course :

1. نفهم يعنى ايه DP .
2. بنستخدام ال DP فى ايه .
3. ليه ال DP اتوجدت .
4. هو ليه ال DP مفيد بالنسبالنا .
5. ازاى نطبقه وايه المشكال الى بيحلها ال pattern المختلفه .
6. ايه المشاكل الى بيحلها ال pattern المختلفه .
7. اخر حاجة ال implementation بتاع ال pattern . والنقطه دى مش هتهمنا للدرجه , عشان اهم حاجة ف ال course انك تفهم المشاكل الى بيحلها pattern معين .
8. ف المهم بانسبالنا اكتشاف ال pattern الى ممكن يحل المشكله الى وقعنا فيها . ف مش هتعرف ال pattern الى محتاجه الا لو كنت فاهم كويس ال pattern .

# 3 – what is Design pattern “DP” ?

* ال DP هى مجموعه من الحلول لمجموعه من المشاكل الى دايما بتقبلنا واحنا بنعمل design للSW بتاعنا .

بمعنى ان ف مشاكل بنقبلها على المستوى الشخصى لنيا as developer واحنا شغلين ف design بتاعنا ف الى بنعمله ان بنبداء نبحث عن مجموعه من الحلول ونجربها لحاد منوصل للحل المظبوط ف بنبداء نعتمد الحل دة ولما تقبلنا نفس المشكله بنستخدام نفس الحل جربناه قبل كدة .

* ف فى مجموعه من المشاكل الى لحظنا انها بتتكر مع معظم الcommunity بتاعه ال developer .ف بدءنا ندرو على حلول للمشكال دى لحاد ملقينا انسب حل للمشكله الى بتقبلها . ف بداءنا نقول لما تقابلك المشكله الكذا استخدام الحل الكذا .
* نقدر نقول على ال Design pattern انها template او انها blue prints او انها road maps بتقلك لما تقابل المشكله الفلانيه المفروض تمشى على الطريق دة عشان تحل المشكله .
* ال DP as a template مش بيقدرملك الحل السحرى لحل المشكله .او مش بيدخل ف تفاصيل الحل هو يدوب بيوضحلك الطريق الى هتمشى عليه .حتى لو بنتكلم ان ال DP دة بيكون ليه implementation بس مش هو دة ال implementation الوحيد للDP يعنى ممكن ال DP الواحد يتعمله اكتر من implementation لاكن ف النهايه هى road map واحدة بتمشى عليها عشان توصل للحل .
* وخلى بالك ف فرق كبير بين ال algorithm وال Dp .يعنى ممكن تقلى ال algorithm بيديك حل للمكشله صح ؟لا مش صح ال DP دة اعم واشمل من ال algorithm .عشان ال algorithm ف الاول وف الاخر بيديك مجموعه من الخطوات الى مفروض تمشى عليها عشان تحل مشكله معينه .ولو فيه اكتر من واحد شافوا خطوات ل الجروزم ف الاخر كلهم هتلقيهم كتبه نفس الكود تقريبا .

امال ف ال DP احنا بنتكلم عن General solution بنقول فيه المفروض يكون عندك كذا وكذا والمفروض توفر مذا وكذا .

يعنى مثلا ف الاخر الكذا دة اعمله class ولا interface بارحتك بس كل واحد هيكون ليه مميزات وعيوب . لكن ف الاخر ال DP مش هيجبر يكون عندك الحاجة الفلانيه ف الكود .

* الالجروزم دة باضبط زى الوصفه بتاعه اى اكله هيكون فيه المقادير والخطوات الى هتمشى عليها عشان تطلع طبخه معينه .
* اما ال DP دة حاجة اعم واشمل الى مثلا هتقلك ازاى توفر ال environment المناسبه عشان توفر الطبخه الفلانيه .ف ال Dp ف الاخر هيديك مثال على ال implementation لاكن انت مش مجبر تعمل implement للpattern بشكل دة.

وعشان كدة كمان هتلاقى ف two pattern كل واحد فيهم بيحل مشكله غير التانى لكن لما تيجى تبص على ال implementation بتاعهم جايز جدا تلاقى ال implementation ده واحد ؟ّ! يعنى مثلا انا بحل المشكله x وانت بتحل المشكله y كل واحد فينا هعمل implement للحل للمشكله لوحدة لكن ف الاخر ممكن تلاقينا مطلعين نفس الكود . عشان زى مقلت قبل كدة ال pattern مش بيقلك انت هتكتب كود عامل ازاى لاكن بيقلك الطريق الى هتمشى فيه عامل ازاى .

# 4- What does the pattern consist of ?

* اى pattern ف الدينا لما تيجى نتكلم عليه او تقراه من كتاب هتلاقيه دايما مكون من اربع حجات .

1. : Internالمشكله الى بيحلها ال. Pattern يعنى هيقلك ف جمله واحدة المشكله والحل بتاعها .والجمله دى لو طلعنا بيها وفهمنها ف كل pattern كدة زى الفل مش عوزين حاجة تانى, طيب ليه عشان كونك فهمت الpattern بيحل انهى مشكله دة هيخلينا ان شاء الله لما نقع ف المشكله المشابه لدى هنعرف حل المشكله ف اروح ابحث عن ال implementation بتاعها .
2. : Motivation تانى حاجة بنتكلم باتفصيل شويه عن المشكله وواسبابها والحلول المتاحه وليه اخترنا الحل الفولانى وساب باقى الحلول ليه اكيد هيكون فيه اسباب ؟
3. Structure for pattern: يعنى هيقلك لما تشوف المشكله دى المفوض تعمل structure بتاع ال class بالشكل معين عشان تحل المشكله.
4. Code example: :هيقلك implementation بتاع المشكله زى مقلنا مش ملزم تلتزم ب ال implementation دة .بس المهم تلتزم بحل المشكله as concept .

* وعاوز تتخل لما بنتكلم عن ال pattern زى بالضبط لما كنا بنتكلم عن ال Data structure ف الجزء بتاع ان هى مش مرتبطه بلغه معينه عشان كدة هتلاقى ال DP مش بتتكلم معاك ف تفاصيل كتير .

ف مثلا ممكن تكون ف programing language مفهاش interface ف هتعمل ايه ؟ ف عشان كدة ال DP مش مرتبطه بلغه معينه .

# 5 – who invented DP ?

* مفيش حاجة امسها مين اخرت ال DP عشان ال DP تم اكتشفها مش اخترعها . زى مقلنا مش نشويه ان ال Dp هى عباره عن حلول لمجموعه من المشاكل الى ظهرت معانا . ف احنا مخترعناش ال pattern احنا اكتشفنا ان دى انسب طريقه بحل مشكله يعد التجربه وكدة يعنى .ولما اكتشفنا الطريقه الى هنمشى بيها لحل المشكله دى بدانا نديلها اسم .
* اول ناس اتكلمت عن ال DP كانوا اربعه جمعه شويه مشاكل عامه وقالوا يجمعاه احنا لقينا الناس لما بتيجى تواجه المشاكل دى لقينا ان الحلول دى هى الانسب . والكتاب بتاعهم اتسمه the gang of four
* ومن اشهر الناس الى اتكلمت عن الموضع هم uncle Bob و reboot see marted انا شخصينا شوفت كتاب clean كود ل uncle bob وكان عظيم الصراحه . "تانى اسم شكلى كاتبه غلط "

# 6 – Why should learn pattern ?

* عشان ديما هتقابل مشاكل كتير ودايما هتكون الحلول قدامك ليه بنقول هتقابل مشاكل كتير عشان ف ناس جربت قبلك ووقعت ف مشاكل انت هتقع فيها بعدين
* الناس دى من كتر موقعه ف مشاكل معينه حطه حلول للمشاكل دى .
* والحاجة المهم جدا ان تعلم ال DP بيسهل عمليه ال communication بينا كتيم . يعنى لما نقع ف مشكله معينه واحنا بنعمل design للSW ف بدال محد يشرحلك خوات حل مشكله يعنى بدال ميقلك اعمل 2 class زيادة واعمل implement للinterface الخ ف يقلك استخدام ال DP الفلانى ف انت تفهم الحل من غير شرح كتير ملهوش لزمه ف دة هيسهل ال communication بين التيم يخويا .

# 7 – classification DP :

لما شفنا المشاكل بتعتنا لقينا ان احنا بنواجه 3 مشاكل اساسيه . بيتفرع منهم شويه مشاكل كتير .

1. creation pattern :لقينا ان احنا بنواجه مشاكل واحنا بنعمل create لobject ف عشان كدة طلع مجموعه من الحلول مجموعه من ال pattern صنفنها تحت category اسمه creation pattern .ف ال creation pattern من اسمها هى عباره عن مجموعه من ال pattern الى هتلحنل المشاكل الى هتقبلنا واحنا بنعمل create لobject

وايوه ف مشاكل هتقبلها وانت بتعمل create ل object وفايه الpattern هنا ان هيسعدنا لحلها من غير متفتح الكود وتبداء تغير فيه حاجه .

1. : Structure pattern ودى هتساعدنا ف حل مجموعه مشاكل هتقبلنا واحنا بنبنى ال class class وعلاقه ال class ببعض.
2. Behavioral pattern: دى مجموعه ال pattern الى هتسعدنا ف حل مشاكل ال communication بين ال object. طبعى جدا ف اى app ان احنا بنبنى class وبطلع منها object عشان ال object دى ف الاخر تكلم بعض .

# 8- Future of good Design:

* قبل منبداء ف ال pattern هنتكلم على مجموعه من ال concept وال principle لازم تكون موجودة ف ال app بتاعى . يعنى هنشوف مجموعه من ال concept وال principle الى ال pattern جايه عشان تاكد عليها .
* يعنى كل منخلص pattern هنقول الpattern دة قدر يطبق ال principle كذا وكذا .
* خلى بالك من ال principle الى جايه عشان هنتكلم عليهم ف كل ال pattern وكمان مفيش اى SW هيطلع غير لما نتاكد ان هو ماشى على مجموعه من ال principle الى جايه مش لازم يكون ماشى على الكل بس على الاقلى يكون متبع مجموعه من الى هنقولهم .
* ايه الحجات الى بحكم بيها ان ال SW بتاعى الى طالع دة كويس ولا لا وهنا **مش** بتكلم من وجه نظر ال client . احنا هنا بنتكلم عن ال good design من وجه نظر ال Developer

يعنى هل انت بنيت ال SW صح ؟ لو بعد فترة من لوقت ال requirement اتغيرت ف بناءا عليه عاوز تغير حاجة فى الكود عندك ف هل هتقدر تعمل دة بسهوله ولا هتهد حاجة موجودة عندك ؟ولا هتشيل كود وتحط كود تانى .

* **ف الحجات الى اقدر احكم بيها على ال good design هو كويس ولا لا هى :**

1. Code reuse: :يعنى الكود الى كتبته دة لو حبيت بعد شويه هل تقدر تستخدامه ف app تانى من غير مشاكل ولا هتحتاج تهد وتبنى الدنيا من الاول ولا هتغير حجات بسيطه ولا هترمى الكود اصلا وتبنى من الاول خاالص ؟ وهيكون ف معاير نقدر نحكم بناء عليها هل ف code reuse متحقق ف الكود الى كتبته ولا لا ؟
2. Extensibility :الحاجة الوحيدة الثابته عندنا ان ف change على مستوى ال requirement .فدايما هيحصل تغير ف بالتالى انت مطالب دايما تغير ف الكود بتاعك ف ازاى اخلى ال app بتاعى قادر ان اغير فيه ازود فيه future و requirement من غير ماثر ف الكود الى كان مكتوب قبل كدة . ودة برضو ف مجموعه من المعاير اقدر احكم من خلالها على هل الكود الى انت كاتبه دة لو حولت بعد شويه ان اغير او ازود future بنسبه قد ايه انت هتغير ف الكود بتاعك دة .

* ف من اهم الحجات الى بنتكلم عليها واحنا بنتكلم عن ال good design من وجه نظر المطور هى ال code reuse , extensibility والنسبه الى هتغير بيها الكود بتاعك لما ال requirements تتغير .

# 9 - Design principle :

* هنتكلم عن مجموعه من ال principle الى لازم انت as developer تاخد بالك وانت بتعمل design لاى SW حتى لو حجمه صغير . عشان "من شب على شئ شاب عليه ".ف لازم تتعود من دلوقتى تراعى ال principle دى .
* "قصدى ب client ف الجزء الى هيجى developer تانى فخلينا متفقين ال client بالنسبه لل developer هو developer تانى "

1. **Encapsulate what varies :**

يعنى وانت شغال هتكتشف ان ف اجزاء من الكود بتتغير بستمرار يعنى معتمده مثلا على input من ال client ف هتكتشف ان الكود بتاعك بيتغير كل لما بغير input جايلنا من بره ف هتلاقى مثلا ان بداءت تخلى الكود بتاعك مجموعه من ال if else . بمعنى ان لو ف ال app بتاعى عندى مجموعه من ال employ وكل employ ليه role معينه زى admin , Receptionist, .. ف هتلاقى نفسك عامل جوة func .

If (role ==”admin ”)

If else (role ==” Receptionist”)

.

.

.etc.

* ف كل معدد ال role الى موجودة تزيد هتزود عدد ال if else ف دة مش صح او مش احسن حاجة افرض لو الrequirements اتغيرت وبقى عندك 50 role مثلا .فهتلاقى ان الكود بتاعك تحول لمجموعه من ال if else condition . وممكن كمان يكون اى action هتاخدة جوه func معتمد على ال role ف هتلاقى نفسك بتكرر كمان ال if else condition جوه كل ال code .
* **ف الحل لما نلاقى ال action الى هتعمله بيتغير بناء على role بتاعه الموظف ف عشان كدة سعتها هنتقلك انت لازم تعمل encapsulate يعنى لازم تفصل الجزء دة ف class لوحدة او ف function لوحدة سواء كنا بنتكلم عن مستوى function او على مستوى class .** هشرح كل ال concept دى بالتفصيل بعد شويه .

1. **Program to an interface not to an implementation :**

قبل مبداء عوزين نتفق على حاجة اول منقول كلمه interface بنسيه 99% مش قصدنا interface بتاع ال c# ,وبنكون بنقصد بكلمه interface يعنى layer بينك وبين حد تانى .

وعلى مستوى ال implement ال interface ده ممكن يكون abstract class او ممكن يكون interface .

ودة عشان ف لغات مش بيكون فيها interfacesزى ال c++ مفهاش interface ف لما اقلك انت مفروض توفر interface بين ال two class دول ف ال c++ مش قصدنا انك تعمل interface قصدنا layer بين ال two class يعنى هتوفر abstract class بين ال two class .

* ف الخلاصه ال interface بيكون قصدنا بيه as a implementation لما interface او abstract class . نرجع بقا لموضوعنا
* طيب كدة ايه قصدنا ب Program to an interface not to an implementation :

يعنى on concrete class depend on abstractions not :هشرحلك الجمله دى بص احنا عرفين يعنى ايه concrete class يعنى class ينفع اخد منه object بمان اقدر اعمل منه object يعنى كاتب implementation لشويه ال function الى جوه ف عشان كدة هو concrete ف اى كلاس ف الدنيا مكتوب جواه implementation هو concrete class

وال interface او abstract class مش بكون كاتب implement للfunc الى جواهم .

ف الجمله دى on concrete class depend on abstractions not بتقلك وانت بتعمل design for SW اعتمد اكتر على ال abstraction class او interface اكتر متعتمد على concrete class وهنعرف ليه بالتفصيل اكتر بعد شويه .

* ومعظم الامثله والمشاريع الى عملتها انا و الى هتشوفها بعد كدة كمان هتلاقى دايما لو عندى abstract class و concrete class هتلاقى دايما ال concrete class بيورث من ال abstract class طيب هو كدة ايه لزمه ال abstract class ف لزمه ال abstract class ان يوضحلك الطريق الى هتمشى عليه .ف من خلال ال function الى جوة ال class هتفهم وصف ال class او هو ايه ال class دة.

او لما تشوف function الى جوة abstract class هتعرف اجابه السوال what is shape لو فرضا sheep دة abstract class . ف بستخدام ال interface اقدر اعمل represent لسوال ال what is the shape?

* طيب وبعدين لما جيبت عملت inherit من ال abstract class او جيت عملت implementation للinterface بداءت تحط implementation لل func انت كدة بتجاوب على السوال التانى how ؟ انا قلتلك ان اى “abstract class” emp ف الدنيا عندة function اسمها calculate salary دة what ف بعد كدة عملت class اسمه accountant ورث من abstract class اسمه employee . ف هتعمل implementation للfunc الى اسمها calculate salary طيب ليه عشان تجاوب على ال سوال التانى

How the salary is calculate لمين لclass accountant .يعنى السوال بتاع How the salary is calculate يخص ال class accountant ؟

* نفس الكلام لو خليت rectangle class يورث من shape abstract ويعمل implemented لداله calcArea دى عشان تجاول على سوال how area of rectangle will be calculated ?
* ف دايما لو لقيت نفسك عاوز توصف ال concept نفسه يبقا انت كدة بتفكر ف abstraction "وقصدى ب abstraction اما abstract class او interface على مستوى ال implementation "

و لو لقيت نفسك عاوز توصف ال implementation يبقا انت كدة بتجاوب على سوال ازاى او how يبقا انت كدة رايح لناحيه concrete class

* وهنتكلم عن الموضع باتفصيل تانى بعدين وكدة النقطه رقم اتنين خلصت .

1. **Favor composition over inheritance :**

احنا بنفضل نعمل علاقه composition بين 2 class بدال منعمل علاقه inheritance بين ال 2 class وهنشرح الاسباب بعد شويه . بس قبل مخلص الintro هفكرك بمعنى ال **composition**

**بص يغالى** احنا عندنا 3 انواع من العلاقات بين 2 class :

1. **Assassination**  :

ودى علاقه لحظيه يعنى class **بيستخدام** object من class تانى ف وقت معينى .

خد مثال عشان تفهم علاقه ال instructor وال pen وجود كل واحد فيهم مش معتمد على وجود التانى بس طول م ال instructor بيشرح على الصبورة ف علاقه بين ال instructor وال pen والعلاقه دى لحظيه .

مثال تانى : واحد بيلبس نضارة لما يقراءه دى برضو علاقه Assassination عشان انا بستخدام النضاره لحظه القراءه .

الخلاصه : لما تلاقى العلاقه زى واحد بيستخدام التانى ف وقت معين وبعد كدة يسيبه "وال life cycle بتاعه كل واحد مش بتاثر على التانى يعنى كل class اقدر اعمل منه object لوحدة " دة اسمه **Assassination** .

- الموضوع as a code بنعمله ازاى او بيكون ازاى ؟ استخدام ال object بيكون على مستوى function بس وعمليه ال create للobject بيتكون مش معتمدة على اى object تانى وبعد مخلص ال function دى علاقتى بتتقطع مع الobject دة

-طيب لو كان قلنا ان ال instractor لازم يكون معاه قلم ف هعمل ال object برة اى داله والعلاقه دى مش Assassination وهنشوف العلاقه دى اسمها ايه كمان شويه .

1. **Aggregation** :

دى علاقه يحتوى على يعنى class يحتوى على object من class تانى. وهنا العلاقه دى شبه الى فاتت بس الفترة هنا بتكون اكبر من الى فاتت .

* لو احنا ف غرفه نقدر نقول الغرفه تحتوى على طالب و instructor .ف كدة العلاقه بين ال طالب والغرفه هى علاقه aggregation

**ولاحظ** ان ال غرفه وال student محدش فيهم معتمد على التانى كل واحد يقدر يعيش من غير التانى يعنى لو مفيش طالب ف عادى ف غرفه والعكس . و علاقه الطالب ب الغرفه طويله شويه عكس ال Assassination .

ازاى هحقق دة as a code كل الى هعمله هعمل reference from class student جوة كلاس room ف كدة مش انا الى مسئول عن عمل create للobject وكمان مش انا الى مسئول عن موت ال object .

Class room {

Student student ;

Public void Room (student \_student){

Student=\_ student;

}

}

1. تلخيص الاتنين الى فاته عشان لو اتلغبط .
2. ف الحاليتن two class not dependent on each other وكمان ف الحاليتين من انا الى بخلق الobject مش بعمله new جوة منى ولا انا الى بموته .
3. Assassination : العلاقه بتكون على مستوى function معينه ولمدة صغيرة
4. **Aggregation** : العلاقه بتكون على مستوى ال class ولفترة اطول

**3-composition :**

تالت واخر علاقه وهى consist of

ودى معنها object بيتكون من object تانى وال life cycle بتاعه بتعتمد على وجود object تانى .

ناخد مثال علاقه الحيطه ب الroom هى composition يعنى ال room مينفعش تكون موجودة الا بوجود الحيطه .

Class Room {

Wall [] walls;

Public void Room(){

Walls=new walls [4](){new wall(),new wall(),new wall(),new wall()}

}

}

والمثال الى فات بيوضح ان لما ال Room Object تموت كمان ال wall هتموت معاها عشان عملت new لل wall جوة ال room .

* لو خدنا مثال سريع كدة :
* علاقه العربيه والمتور او علاقه العربيه والكرسى الى فيها هى علاقه يحتوى على عشان كل يقدر يعيش من غير التانى .
* جسد الانسان --الايد والجسد علاقه يحتوى على عشان الانسان مش هيموت من غير ايدة

علاقه الدماغ والجسد علاقه يتكون من عشان مفيش انسان من غير دماغ ولسه عايش .