1/12/2023

Abdurahman Gamal Ahmed

no campany

Clean code

**Ch6 DS**

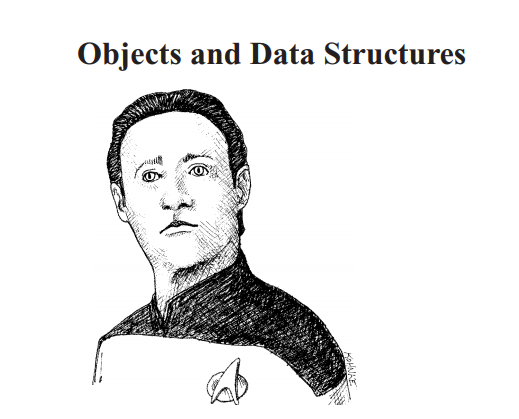


Table of Contents

[1-introduction : 2](#_Toc124471294)

[2- Data Abstraction: 2](#_Toc124471295)

[3-Data/Object Anti-Symmetry : 4](#_Toc124471296)

[4- The Law of Demeter: 6](#_Toc124471297)

[4.1- **Train Wrecks:** 6](#_Toc124471298)

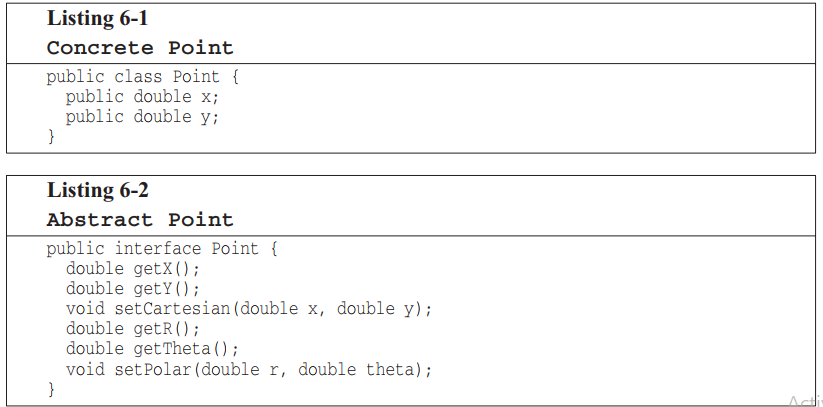
[4.2- Hybrids : 7](#_Toc124471299)

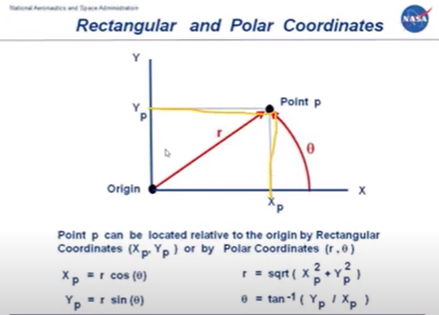
# 1-introduction :

ف سبب وراء تخلى ال variables تكون private .انك مثلا مش عاوز اى حد تانى يعتمد عليهم او يستخدمهم عشان يكون ف لينا دايما الحريه ان نغير فيهم ف اى لحظه من غير مناثر على حد تانى . طيب امال ليه بتعمل setter,getter وهم بيكشفه ال variables privateدول؟؟؟

عم بوب باشا طرح السوال العظيم دة ومجوبش عليه دلوقتى .

# 2- Data Abstraction:

عم بوب اول مبداء عطانى مثالين نقارن بينهم . والاتنين دول بيمثله او بيعبرة عن نقطه بس الاول كلاس دة كاشف كل حاجه جواه مخليها public يعنى وتانى واحد هو interface وخافى ال implementation بتاعه تماما .

الميزة ف التانى حاجة انك مستحيل تعرف هو ته تمثيل ل rectangular or polar coordinates . ولو حد ميعرفش ال rectangular or polar coordinates ف دول نعين لتمثيل النقطه ودة شكلهم الى معلم عليه بالون الاصفر دة ال rectangular والشكل الاحمر دة هو polar coordinates .

* ف هم بوب بيقلك كمان تانى شكل الى هو interface

ممكن ميكونش ولا polar coordinates ولا حتى

Rectangular

ومعل ذالك interface بيعبر عن DS بشكل جميل

ومش بس بيعبر عن DS دى كمان ال method الى

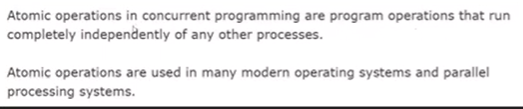
جواه بتفرض policy access.

يعنى انت ممكن تقره الاحدثيات x لوحدة من غير y او

العكس بس لازم لما تعمل set يكون للاتنين مع بعض.

X,Y وانك تعمل set لحجتين مع بعض ف

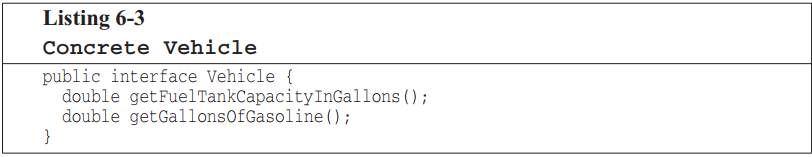
Statement واحدة او instruction واحد بنسميه Atomic operation .

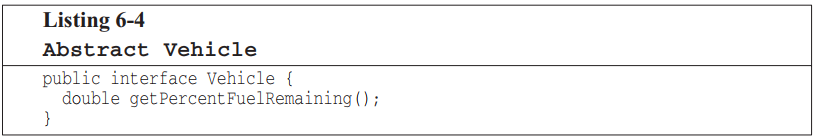
لو انت مش عارف يعنى اي Atomic operation ف شوف الاسكرين دى ومش هشرحها عشان انا عرفها .وعم بوب مش شرحها

وعلى العكس ف الclass الى رقمه 6-1 : باين خالص ان دة تمثيل لل rectangular coordinates وكمان نقدر تغير قيمه الx,y بشكل منفصل عن بعض عكس الى كان بيحصل ف ال interface . فكدة ال implementation بيكون واضح وحتى لو خلينا الx,y يكونوا private وعملنا setter,getter ف كدة بروض ال implementation هيكون واضح

وانك عشان تعمل Hiding implementation دة مش معناه انك تضيف layer من الfunction عشان توصل لل variables .

Hiding implementation هو بختصار ال abstractions و ان الكلاس مش بكل سهوله يخلنا نعملaccess لل variable عن طريق ال getter ,setter والافضل ان نعمل abstract interface الى بتسمح للuser ان يغير ف ال variable من غير ميكشف ال implementation بتاعها .

**بعد كدة عم بوب بيدنا مقارنه تانى :**

****

* الصورة الاوله بتحدد وحده معنيه gallons للتعمل مع الواقود او التانك ودة يعتبر concrete مش abstraction يعنى مش معنى انك عامل interface يبقا كدة انت شغال abstraction لا لازم كمان الحجات الى جوة الinterface تكون abstraction يعنى زى الماء كدة ملهوش طعم ولا لون ولا ريحه وا حد يقدر يشربه ف انت لما حددت وحدة كدة مش اى حد يقدر يستخدام ال interface ويعمل الlogic الخاص بيه . ف الحاله دى ال function هى مجرد accessor للvariable .
* الصورة التانيه دى مشconcrete ومعنديش اى معلومه عن شكل الداتا . ف الحاله التانيه هى الافضل
* ف احنا مش عاوزين نكشف عن تفاصيل الداتا بتاعتنا لكن عاوزين نعبر عن الداتا بطريقه abstract . وطبعا دة مش بيتحقق بانك تستخدام setter,getter او interface كمان ولاكن لازم تفكر عن احسن طريقه لتمثيل البينات بتاعك بobject .
* واسواء حاجة انك تستخدام setter,getterعمال على بطال بس استخدام ال accessor مش وحش بس استخدمها لوحدها وانت متخيل ان كدة بتعمل hide implementation دة مش صح .انت مفروض تفهم امتى هتستخدمها وتستخدمها صح وتستخدام الinterface عشان تعمل abstraction وتعمل عمليه abstractionبشكل صح .

انا عارف ان الكلام ممكن يبقا عايم بس ف لو حسيت كدة قدام هتلاقى الدنيا وضحه شويه ف كمل بس معايه . ولو وصلت للاخر وبردو مفهمتش للدرجه ف كدة ممكن يكون عندك مشكله ف الاساسيات .

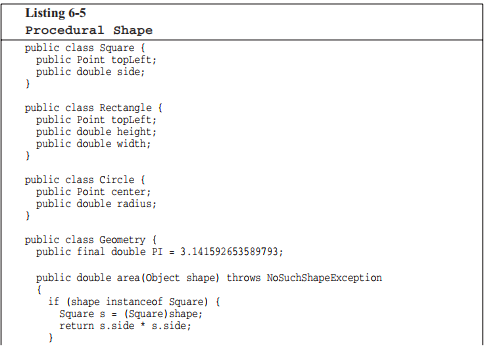
# 3-Data/Object Anti-Symmetry :

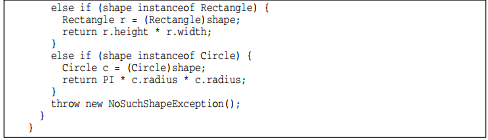
Anti-Symmetry معنها عدم تناسق ال object او الداتا .

عم بوب بيقلك الفرق بين ال object وال DS ان ال object ان :

* Object بيخفى الdata بتاعتها وره ال abstraction وتسيب function تتعامل مع الداتا
* DS دى بتكشف الداتا بتاعتها بتخلها public يعنى ومعنهاش function ليها مغزة

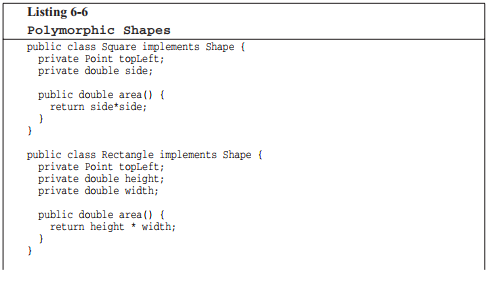
تعاله ناخد مثال عشان تفهم : والمثال هناخدة على كلاسين

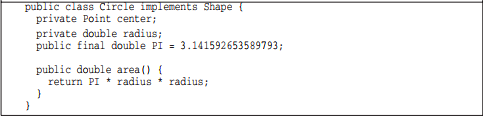


* **دة اول كلاس .**

ف الصورة انا عندى تلاته DS الى هم مربع ومستطيل ودايره وكلهم مفهمش اى action او function . وعندى كلاس geometry دة فيه ال function وف الحاله دى سهل خاالص اضيف function جوة ال geometry من غير ماثر على ال DS ودى ميزه بس لو ضفت شكل جديد هحتاج اعدل فى كل فانكشن ال geometry يعنى اضيف جوها else if تانى ف كل ال فانكشن ودة عيب يعتبر . ف الى فات دة عيوب ومميزات استخدام ال DS .

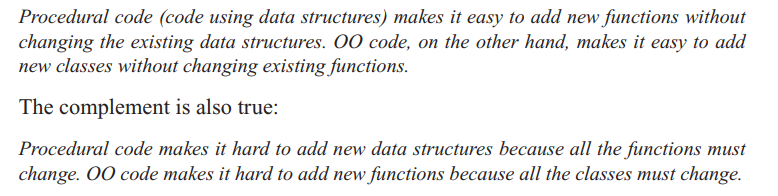
* ممكن بتوع ال oop يزعل من التصميم دة ويقلك دة procedure paradigm ولو مش عارف يعنى ايه دة انا شارحه ف كورس ال paradigm. ارجعله او سرش عليه .
* ف بتوع ال oop هيعمله او هيفضله التصميم الى جاى ومعاهم حق:

**ودة تانى كلاس :**

****

* هنا الى عندنا interface فيه الدوال الى انا عوزها اسمه shape وعندى شويه كلاس بتعمل implement للshape ف كدة كل كلاس هيعمل implement للدوال زى مايحب .
* ف حالتنا هنا الميزة ان لو ضفت ا كلاس جديد دى مش هياثر على اى كلاس موجود
* والعيب ان لو ضفت function ف ال interface دة هياثر على كل الكلاس الى موجود عشان لازم الكلاس الى موجودة تعمل implement لكل الدوال الى ف ال interface . وعم بوب ف الجزء دة بيقول ان ف طرق لحل العيب دة زى ال visitor DP وغيره بس دول مكلفين وممكن يرجعونا procedure paradigm .

ف مفروض كدة يكون وضح الفرق بين ال object وال DS وان هم عكس بعض اصلا .

* الاسكرين دى بتلخص كل الى قلته عن الفرق بين DS وال OOP

ومن الشرح الى فات لو انت عندك system كبير وعاوز تضيف Data type جديد يبقا استخدام ال object و ال oo . وعلى العكس

لو عاوز تضيف function جديدة فالطريقه الانسب ف الحاله دى هى procedural code and data structures .

والمبرمجين الى عندهم خبره عرفين ان فكرة ان كل حاجة objct دى خرافه ومش صح اوقات بنحتاج ال simple DS مع procedural

# 4- The Law of Demeter:

السكشن دة مهم خااالص .

Demeter دى اسم شخص والشخص دة عمل قانون بيقلك : ان ال module مينعفش يكون عارف الاجزاء الداخله بتاع ال object الى تخصه .

ذى ماشوفنا ف السكشن الى فان ان ال object بتخفى الداتا بتاعتها وتظهر ال operation وده معنا ان ال object مينفعش يكشف ال هيكل الداخلى بتاعه ف ال accessors بتكشف ال هيكل الداخلى للobject ف مفروض مش بيعمله اخفاء زى مانت فاهم .

* بس انا جاسس انك اتلغبط ف تعاله نشوف الموضع بشكل ادق :

لو انا عندى method اسمها f جوه كلاس اسمه c ---<ف ال method دى مفروض تعمل call لحاجه من الاmethod الى جايه بس :

1. تعمل call ل method تبع c
2. An object created by f
3. An object passed as an argument to f
4. An object held in an instance variable of C

* خلاصه الاربع حجات دول ان f هتعمل call بس
* لmethod جوة c او
* object اتبعتلها
* object هى عملته create او
* او تعمل call لobject جوة instance جوه ال c .
* يعنى f بتتعامل الا مع احاجة من تبع c او تبعها هى .
* *يعنى f مينفعش تعمل on object invoke method تكون راجعه من* allowed functions .لو مفهمتش السطر دة ف شوف الكود الى جاى وانت هتفهم .والسطر دة بيقلك متعملش الشكل الى ف الاسكرين



* الاسكرين دى بتنتهد قانون عم Demeter عشان حصل call لل getScratchDir من ال return بتاع ال getOptions وكمان حصل call لل getAbsolutePath من ال return بتاع getScratchDir. واحنا من الصبح بنقول ان الfunction مينفعش تعمل call غير لحاجة تبعها او تبع الكلاس بتاعها .
* هنعمل على الكود دة ف السكرين الى جايه متقلقش

## 4.1- **Train Wrecks:**

Train Wrecks معنها الحرفى القطار المدمر. وهو هنا قصدة على الاسكرن دى وهو سمها كدة عشان زى ما انت شايف ان الكود عامل زى سلسله القطر



والشكل دة يعتبر حاجة سيئه ومش صح انك تعملها . عشان بينتهك قانون Demeter لان كده ال module عارف ان ال ctxtبيحتوى على option وال option بيحتوى على Dir والDir بيحتوى على path .ودى معلومات كتير اوى على function واحدة انها تعرفها يعنى ال function الى بتعمل call عارفه تتنقل بين اكتر من object .

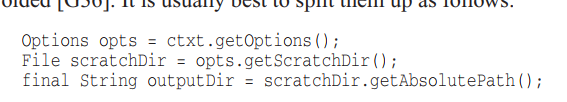
وكمان بنعتمد على دة انتهاك لقانون Demeter وله لا حسب هل الoprion ,dir,path دول object ولا DS .

1. لو كانوا object ف المفورض ال internal structure بتاعهم مفروض يبقا مخفى وبالتالى معرفه ما بداخلهم يعتبر انتهاك لقانون عموا Demeter

* وهنا قصدى ب ال internal structure يعنى ممكن تعتبرها ال filed الى جوه object .

1. اما لو كانوا DS ف الطبيعى انهم يكشفه ال internal structure ويبقا كدة قانوا Demeter اصلا مش هنطبقه هنا .

طيب تعاله نشوف حل للسكرين الى فاتت ومفروض تبقا عامله ازاى : دة هيكون الحل



## 4.2- Hybrids :