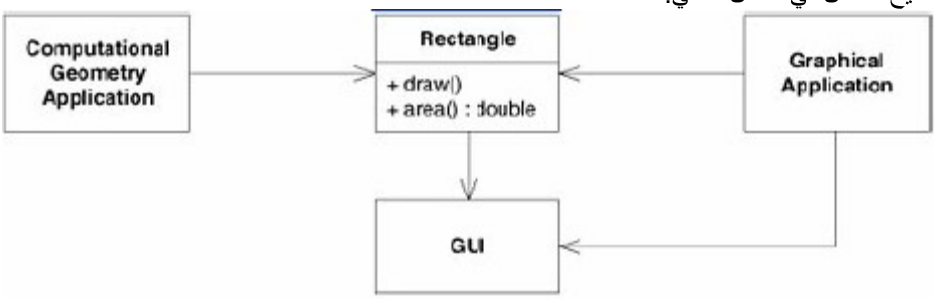
مبدأ المسئولية الواحدة

The Single Responsibility Principle (SRP)

يجب ان يكون هناك سبب واحد فقط لتغيير الكلاس

عندما نقوم بتصميم الفئات في النظام ، فإننا نقوم بتوزيع المسئوليات ، وعلينا خلال هذه المرحلة ان لا نضع اكثر من مسئولية في نفس الفئة لأن كل مسئولية تعتبر محورا من محاور التغيير . فإذا تغيرت المتطلبات فيما بعد ، فإن علينا ان ان نقوم بتغيير المسئولية في الفئة ، وبالتالي فإن الفئة التي تكون لها اكثر من مسئولية سوف يكون لها اكثر من سبب للتغيير.
إضافة إلى ذلك فإن المسئوليات داخل الفئة الواحدة يمكن ان تقترن فيما بينها ، وتعديل فئة من اجل مسئولية معينة يمكن ان يفسد قدرة الفئة على تحقيق بقية المسئوليات.

للتوضيح ، تأمل في المثال التالي:

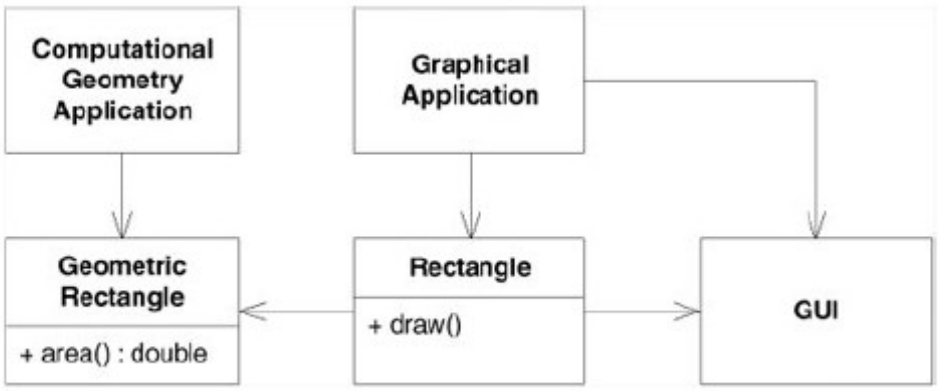


حيث قمنا بتعريف فئة المستطيل تحتوي على الدالة Area() لحساب مساحة المستطيل ، والدالة Draw لرسم المستطيل على الشاشة . ايضاً قمنا باستخدام تلك الفئة في تطبيقين مختلفين ، الأول ComputationalGeometryApplication يقوم بالحسابات الهندسية ويستخدم فئة المستطيل لهذا الغرض ، ولا يقوم اطلاقاً باي عملية رسم على الشاشة.
والتطبيق الآخر GraphicalApplication يقوم ايضاً ببعض الحسابات الهندسية لكنه اساساً مهتم برسم المستطيل على الشاشة.

واضح ان مثل هذا التصميم يخرق مبدأ SRP فالمستطيل له مسئوليتان الأولى القيام بالحسابات الهندسية ، والأخرى القيام بعمليات الرسم.

ومثل هذا الخرق يؤدي إلى العديد من المشاكل ، اولها انه لكي يقوم التطبيق ComputationalGeometryApplication باستيراد مكتبة المستطيل فإنه يجب ان يقوم ايضاً باستيراد المكتبة GUI حتى وان لم يستخدمها.
المشكلة الثانية ، انه إذا تغيرات المتطلبات في GraphicalApplication بالشكل الذي يتوجب معه تغيير فئة المستطيل فإن علينا ان نقوم ايضاً بإعادة بناء واختبار وتحزيم ComputationalGeometryApplication حتى نضمن انه سوف يعمل بالشكل السليم.

التصميم الجيد يكون عن طريق فصل المسئوليتين في فئتين مختلفتين كلياً كما هو موضح في الشكل التالي:



بهذا لم يعد ComputationalGeometryApplication بحاجة إلى استيراد GUI ولن يتأثر بالتعيرات المتعلقة بالرسم.

تعريف المسئولية:

المسئولية هي السبب في التغيير

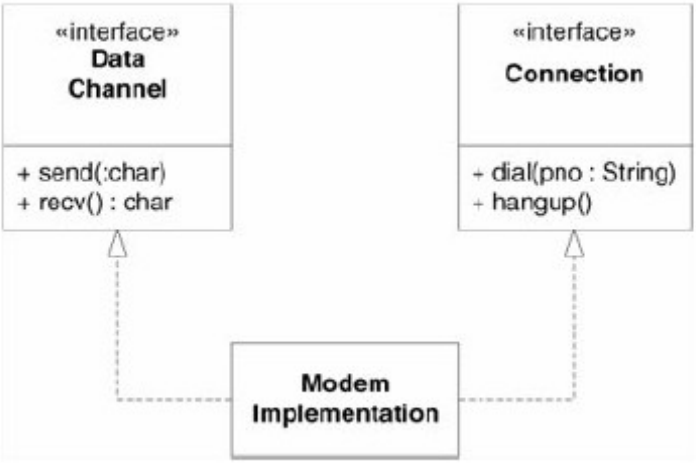
تعرف المسئولية في سياق مبدأ SRP، بإنها السبب في التغيير ، فإذا كنت تعتقد ان هناك اكثر من حافز لتغيير الفئة ، فإن لتلك الفئة اكثر من مسئولية.
احيانا يكون من الصعب رؤية هذا ، فنحن معتادون على مزج المسئوليات والتعامل معها على شكل مجموعات ، على سبيل المثال تأمل في واجهة المودم التالية:

```
[Code]
public interface Modem
{
public void Dial(string pno);
public void Hangup();
public void Send(char c);
public char Recv();
}
[/Code]
```

البعض قد ينظر اليها على انها واجهة ممتازة فجميع الدوال فعلا تنتمي إلى المودم .

في الحقيقة ان هذا التصميم يعرض مسئوليتين الأولى هي إدارة الأتصال (Dial و Hangup من اجل فتح واغلاق الأتصال) ، والثانية هي تراسل البيانات (Send و Recv من اجل استقبال وارسال البيانات).

والسؤال الذي يطرح نفسه هنا، هل يجب فصل المسئوليتين؟ إجابة هذا السؤال تتوقف على الكيفية التي يمكن ان يتغير بها التطبيق.
فإذا كان التطبيق يمكن ان ان يتغير بالشكل الذي يؤثر على توقيع الدوال المتعلقة بإدارة الأتصال ، فإن هذا سوف يؤثر على الزبائن المهتمة بتراسل البيانات ايضاً.
في هذه الحالة يجب الفصل بين المسئوليتين كما يوضح الشكل التالي:



من ناحية اخرى إذا كان تغيير في مسئولية يعني التغيير في الأخرى ، فلا حاجة إلى فصل المسئوليتين ، حيث ان مثل هذا الفصل لن يؤدي إلا إلى مزيد من التعقيد.

فصل المسئوليات المقترنة:

لو عدنا إلى الشكل الأخير، سوف تجد اننا وضعنا المسئوليتين في الفئة ModemImplementation ، ورغم ان هذا غير مستحب ، إلا انه ضروري احيانا ، فهناك العديد من العوامل المتعلقة بتفاصيل العتاد ونظام التشغيل والتي تجبرنا على القيام بمثل هذا الأقتران ، رغم هذا وعن طريق فصل الواجهات فإننا قمنا بفك هذا الأقتران بالدرجة المطلوبة تماماً ، وعليه فإن ModemImplementation هو عنصر مجهول لبقية العناصر لا يتم التعامل معه مباشرة ، باستثناء الدالة main في التطبيق.

من كتاب : Martin C. Robert (2006) Agile Principles, Patterns, and Practices in C#. Prentice Hall

المراجع

- SRP: The Single Responsibility Principle
- Single responsibility principle - Wikipedia, the free encyclopedia