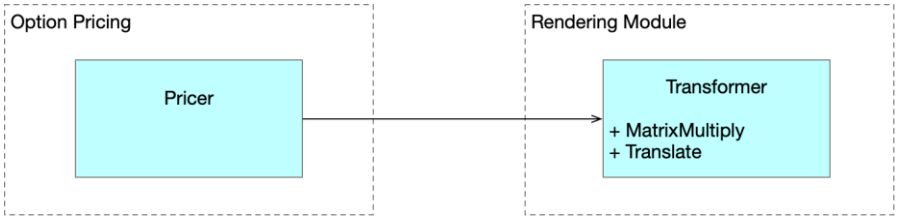
S.O.L.I.D

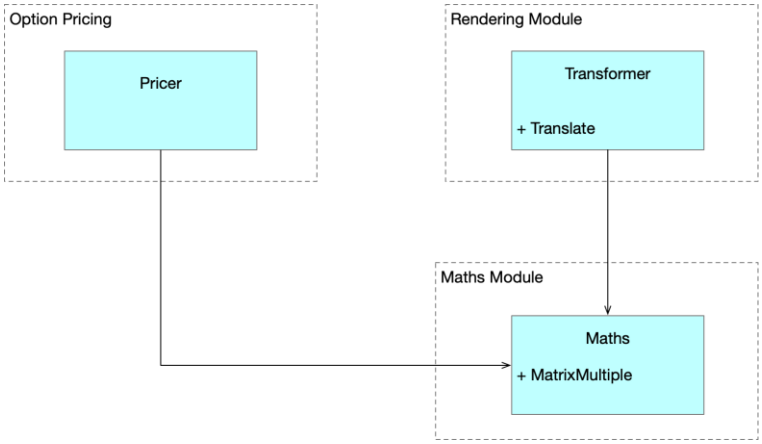
Overview

Single Responsibility Principle

A single class should have only one responsibility. If a class has multiple responsibilities, then changes to one responsibility can break the logic of the other responsibilities thus increasing brittleness. In cases where a module exposes a class that provides multiple responsibilities, clients of that module may be forced to recompile and redeploy when a responsibility on which they have no logical dependency changes.



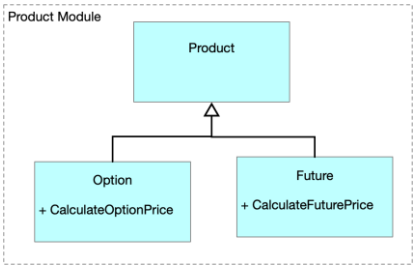
We can fix this as follows



Mixing business rules and persistence is a classic example of breaking the Single Responsibility Principle. Patterns such as Façade, Proxy and DAO patterns can be used in such situations

Open Closed Principle

Types should be open for extension and closed for modification. Consider the following simple type hierarchy and a piece of client code which breaks the Open/Closed Principle

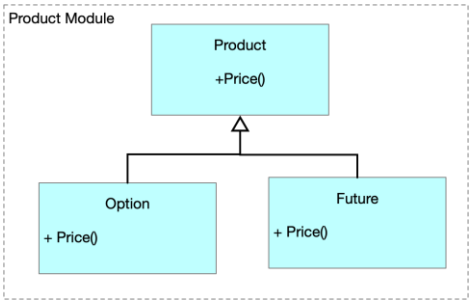



```
double PriceProducts(IEnumerable<Product> products)
{
    double total = 0.0;

    foreach (var product in products)
    {
                if (product is Option)
                total += ((Option)product).PriceOption();
          if (product is VarSwap)
                total += ((VarSwap)product).PriceVarswap();
    }

    return total;
}
```

The use of abstractions and polymorphism enable us to modify our hierarchy and `PriceProducts` method to obey the open closed principle.




```
double PriceProducts(IEnumerable<Product> products)
{
    double total = 0.0;

    foreach (var product in products)
          total += product.Price();

    return total;
```

Liskov Substitution Principle

Replacing objects of a base-class with object of a subclass should not impact the correctness of the program.

Interface Segregation Principle

Multiple interfaces are better than one big interface

Dependency Inversion Principle

Code against abstractions rather than concrete implementations

Questions

What is SOLID?

	Column Header
Single Responsibility	A class should have one responsibility
Open Closed Principle	Open for extension, closed for modification
Liskov Substitution Principle	Replacing objects with sub-type instances should not break the code
Interface Segregation	Multiple specific interfaces are better than one large one
Dependency Inversion	One should depend on abstractions, not concrete types

Why is SRP important?

If a class has multiple responsibilities, then changes to one responsibility can break the logic of the other responsibilities thus increasing brittleness.