SKILLTREE

設計模式 C#

Bill Chung

設計模式
Design Patterns

設計模式

- 設計模式是被用來解決特定的需求
- 如何在不重新設計下進行改變
- 組合多種設計模式

SKILLTREE

繼承與聚合

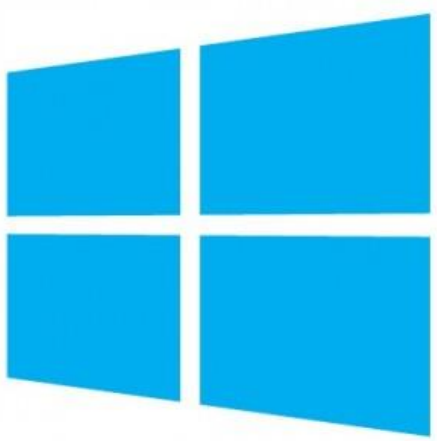
繼承的缺點

- 繼承是侵入性的
- 由於衍生類別必須具有基底類別的所有特性，會增加衍生類別的約束
- 衍生類別會強耦和基底類別，當基底類別被修改也會影響衍生類別

聚合 Aggregation

- Is-it → has-it

SOLID 六大原則



- 單一職責原則
  - **Single Responsibility Principle (SRP)**
  - 就一個類別而言，應該僅有一個引起它變化的原因
- 里式替換原則
  - **Liskov’s Substitution Principle (LSP)**
  - 軟體使用父類別的地方，一定也會適用於子類別

- 倚賴倒置原則
  - **The Dependency Inversion Principle (DIP)**
  - 高層模組不應倚賴低層模組，兩者都應該倚賴抽象
  - 抽象不應該倚賴細節，細節應該倚賴抽象
- 介面隔離原則
  - **The Interface Segregation Principle (ISP)**
  - 客戶端不應該倚賴它不需要的介面
  - 類別間的倚賴應建立在最小的介面上

- 開閉原則
  - **Open-Closed Principle (OCP)**
  - 對擴展開放，對修改封閉
- 最少知識原則 (迪米特法則)
  - **Law of Demeter (LOD)**
  - 一個物件應該對其他物件有最少了解

IoC
控制反轉

控制反轉

- 實現低耦合的最佳設計方式之一
- 控制反轉的設計原則，就是反轉這種在控制上的關係，讓通用的程式碼來控制應用特定的程式碼，不讓相較而言較多變的應用特定程式碼，去影響到通用的程式碼
- 相依於抽象而不倚賴實作

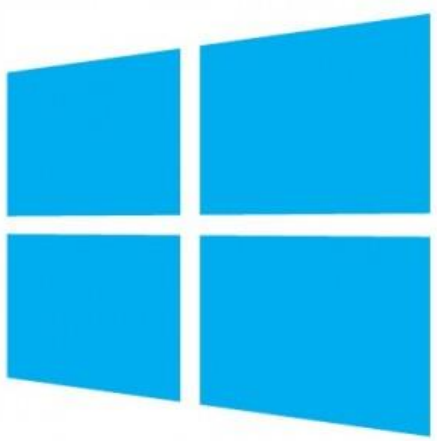
SKILLTREE

Dependency Injection

Dependency Injection

- Interface Injection
  - 使用介面實作注入
- Constructor Injection
  - 使用建構子注入
- Setter Injection
  - 使用屬性注入

單例模式
Singleton



單例模式

- 確保某個類別只有單一執行個體，而且自行建立執行個體並向整個系統提供這個執行個體
- 適用情境
- 多執行緒中的單例

屬性

作業

反射

SKILLTREE

Assembly

載入組件

- Assembly.Load
  - by AssemblyName
  - by Assembly name string
  - by Assembly byte[]
- Assembly.LoadFrom
- Assembly.LoadFile

建立執行個體

- AppDomain.CreateInstance
- AppDomain.CreateInstanceAndUnwrap
- Assembly.CreateInstance

SKILLTREE

利用反射存取成員

Member

- Type.GetMember
- Type.GetMambers

Method

- Type.GetMethod
- Type.GetMethods
- MethodBase.Invoke

Property

- Type.GetProperty
- Type.GetProperties
- PropertyInfo.SetValue
- PropertyInfo.GetValue

Interface

- Type.GetInterface
- Type.GetInterfaces

SKILLTREE

Activator

建立執行個體

- Activator.CreateInstance
- Activator.CreateInstanceFrom

使用反射建立泛型實體

全反射 BMI 範例

Atttribute

Attribute

- Attribute 是一種和一般命令程式不同的設計方式，通常被稱為『宣告式設計』
- 當一個 Attribute 被加入到某個元素時，該元素就被認為具有此特性的功能或性質
- Attribute 是被動的，無法存取目標物
- 要建立一個可以當作 Attribute 的類別, 必須繼承Attribute 類別
- 在執行階段可以使用反射來存取

自訂 Attribute 類別

```
internal class BoundaryAttribute : Attribute
{
    internal Double Max
    { get; set; }

    internal Double Min
    { get; set; }

    // 建構函式, 以便在套用 attribute 時初始化 Min, Max
    public BoundaryAttribute(int min, int max)
    {
        Max = max;
        Min = min;
    }
}
```

在列舉值中套用 Attribute

```
public enum GenderType
{
    [BoundaryAttribute(20, 25)]
    Man = 1,
    [BoundaryAttribute(18, 22)]
    Woman = 2
}
```

```
public enum GenderType
{
    [Boundary(20, 25)]
    Man = 1,
    [Boundary(18, 22)]
    Woman = 2
}
```

取得列舉值的 Attribute 資料

```
internal class EnumValueBoundryHelper
{
    internal Double Max
    { get; private set; }

    internal Double Min
    { get; private set; }

    public EnumValueBoundryHelper(GenderType gender)
    {
        FieldInfo data = typeof(GenderType).GetField(gender.ToString());
        Attribute attribute = Attribute.GetCustomAttribute(data,
typeof(BoundaryAttribute));

        BoundaryAttribute boundaryattribute =
(BoundaryAttribute)attribute;

        Min = boundaryattribute.Min;
        Max = boundaryattribute.Max;
    }
}
```

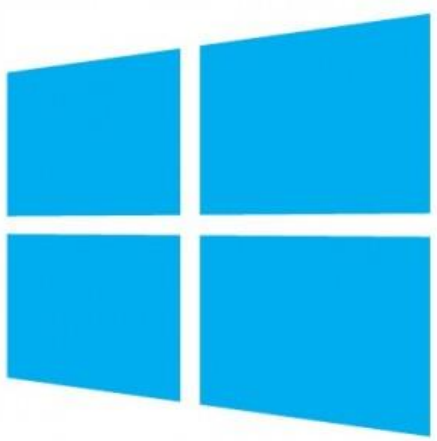
在型別上套用 Attribute

```
[BoundaryAttribute(0, 100)]
public class BoundryClass
{

}
```

取得型別的 Attribute 資料

```
internal class ClassBoundryHelper
{
    internal Double Max
    { get; private set; }

    internal Double Min
    { get; private set; }

    public void GetBoundry(Type type)
    {
        // 確認型別帶有 BoundaryAttribute
      if (type .IsDefined (typeof(BoundaryAttribute)))
      {
          Attribute attribute =
type.GetCustomAttribute(typeof(BoundaryAttribute), true);
          BoundaryAttribute boundaryattribute =
(BoundaryAttribute)attribute;
          Min = boundaryattribute.Min;
          Max = boundaryattribute.Max;
      }
    }
}
```

工廠模式
Factory



工廠模式

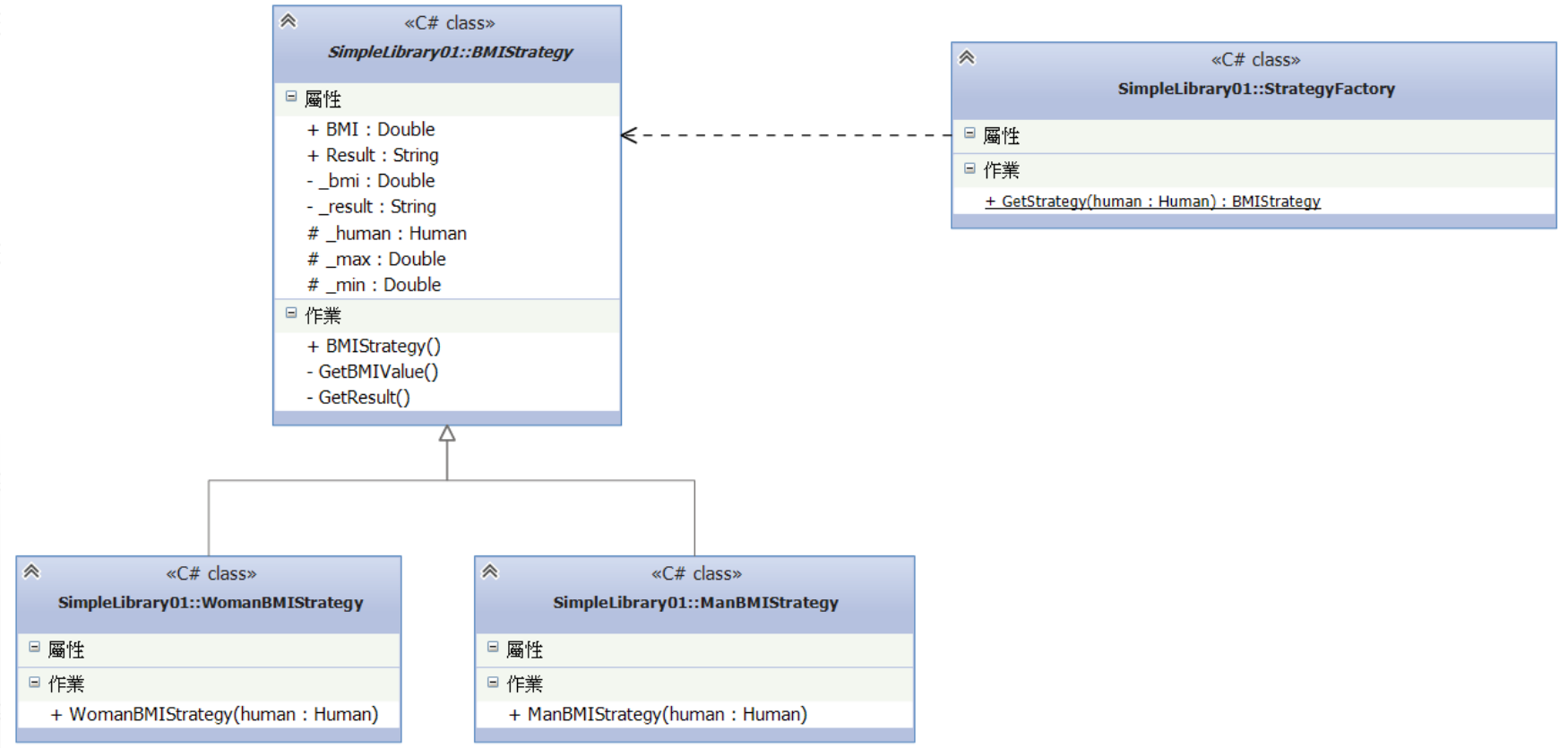
- 定義一個創建物件的介面
- 分離物件的使用與建構+管理
- 適用情境

SKILLTREE

簡單工廠

簡單工廠

- 利用分支運算 (if else, switch case) 決定實體
- 改善分支運算的問題
  - 使用資源字典
  - 使用 Attribute

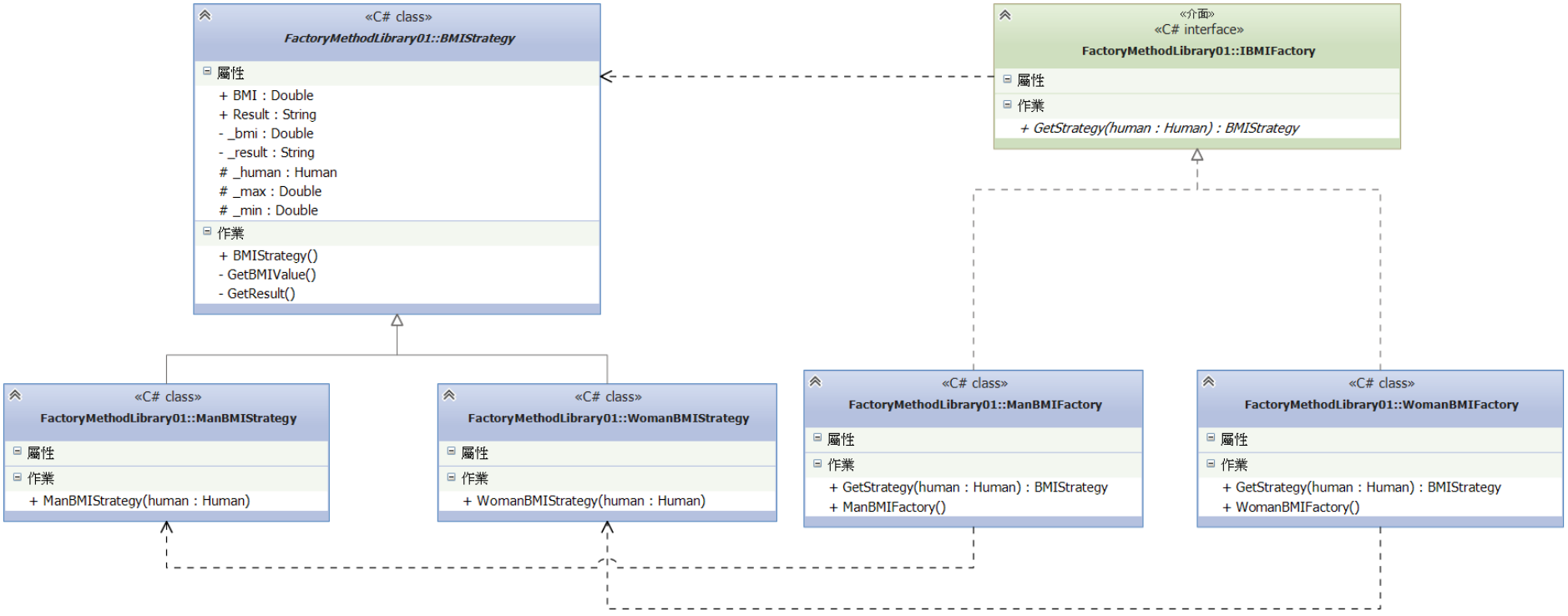

«C# class»
SimpleLibrary01::BMIStrategy
屬性
+ BMI : Double
+ Result : String
- _bmi : Double
- _result : String
_human : Human
_max : Double
_min : Double
作業
+ BMIStrategy()
- GetBMIValue()
- GetResult()
«C# class»
SimpleLibrary01::StrategyFactory
屬性
作業
+ GetStrategy(human : Human) : BMIStrategy
«C# class»
SimpleLibrary01::WomanBMIStrategy
屬性
作業
+ WomanBMIStrategy(human : Human)
«C# class»
SimpleLibrary01::ManBMIStrategy
屬性
作業
+ ManBMIStrategy(human : Human)

SKILLTREE

工廠方法

工廠方法

- 由不同的工廠, 決定不同的實體
- 定義一個用於創建物件的介面, 由此介面的子類別決定要實體化哪一個工廠
- 適用情境


«C# class»
FactoryMethodLibrary01::BMIStrategy
屬性
+ BMI : Double
+ Result : String
- _bmi : Double
- _result : String
_human : Human
_max : Double
_min : Double
作業
+ BMIStrategy()
- GetBMIValue()
- GetResult()
«介面»
«C# interface»
FactoryMethodLibrary01::IBMIFactory
屬性
作業
+ GetStrategy(human : Human) : BMIStrategy
«C# class»
FactoryMethodLibrary01::ManBMIStrategy
屬性
作業
+ ManBMIStrategy(human : Human)
«C# class»
FactoryMethodLibrary01::WomanBMIStrategy
屬性
作業
+ WomanBMIStrategy(human : Human)
«C# class»
FactoryMethodLibrary01::ManBMIFactory
屬性
作業
+ GetStrategy(human : Human) : BMIStrategy
+ ManBMIFactory()
«C# class»
FactoryMethodLibrary01::WomanBMIFactory
屬性
作業
+ GetStrategy(human : Human) : BMIStrategy
+ WomanBMIFactory()

SKILLTREE

泛型工廠

```
public class GenericFactory
{

    public static T CreateInastance<T> (string assemblyname,string
typename) where T:class
    {
        Object instance = Activator.CreateInstance(assemblyname,
typename).Unwrap();
        return instance as T;
    }
}

static void Main(string[] args)
{
    var c =
GenericFactory.CreateInastance<GenericLibrary01.Class1>("GenericLibra
ry01", "GenericLibrary01.Class1");
    Console.WriteLine(c.GetType().ToString());

    Console.ReadLine();
}
```

全反射 BMI 解決之道
分離抽象與實作

屬性

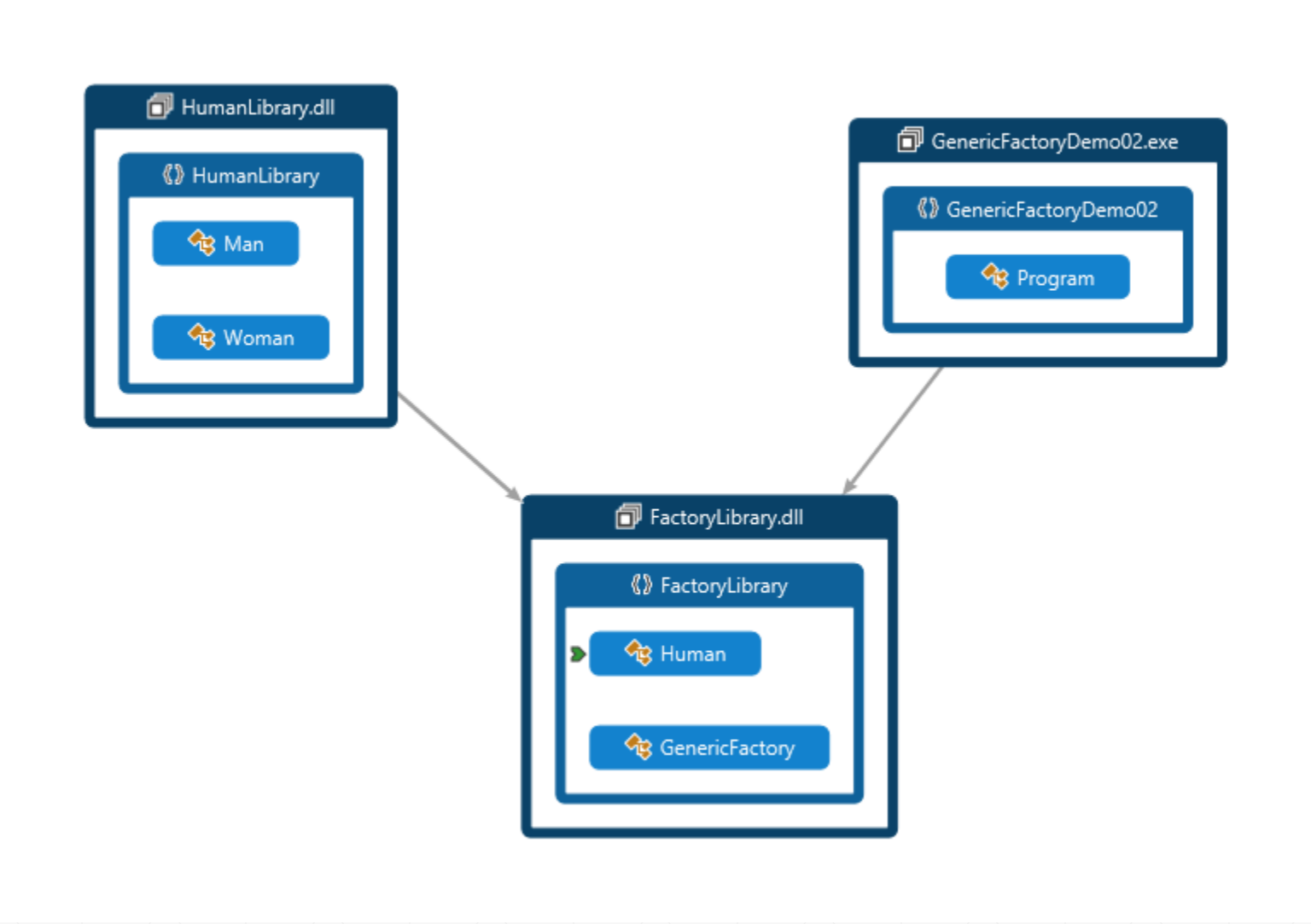
作業

屬性

作業

屬性

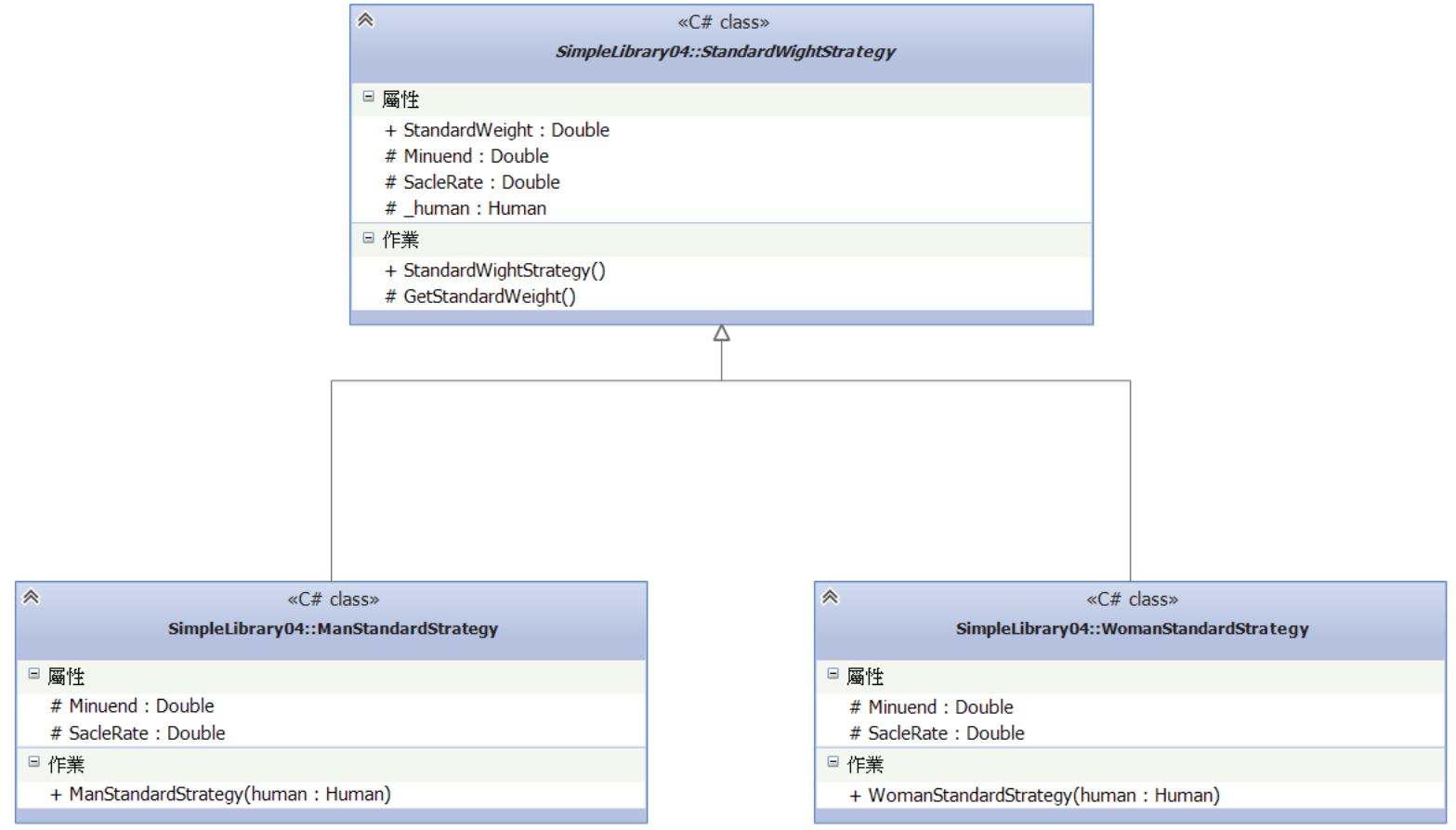
作業


HumanLibrary.dll
HumanLibrary
Man
Woman
GenericFactoryDemo02.exe
GenericFactoryDemo02
Program
FactoryLibrary.dll
FactoryLibrary
Human
GenericFactory

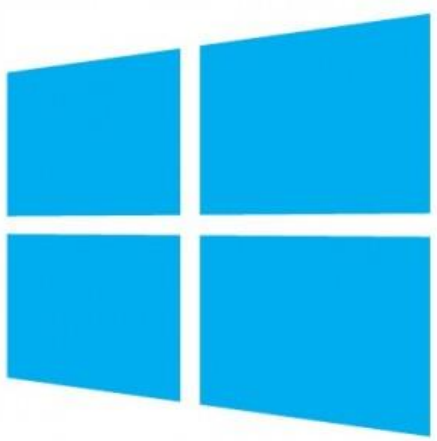
範本方法模式
Template Method

範本方法模式

- 減少多餘的程式碼
- 把通用實做放在基底類別
- 適用情境


«C# class»
SimpleLibrary04::StandardWightStrategy
屬性
+ StandardWeight : Double
# Minuend : Double
# SacleRate : Double
# _human : Human
作業
+ StandardWightStrategy()
# GetStandardWeight()
«C# class»
SimpleLibrary04::ManStandardStrategy
屬性
Minuend : Double
SacleRate : Double
作業
+ ManStandardStrategy(human : Human)
«C# class»
SimpleLibrary04::WomanStandardStrategy
屬性
Minuend : Double
SacleRate : Double
作業
+ WomanStandardStrategy(human : Human)

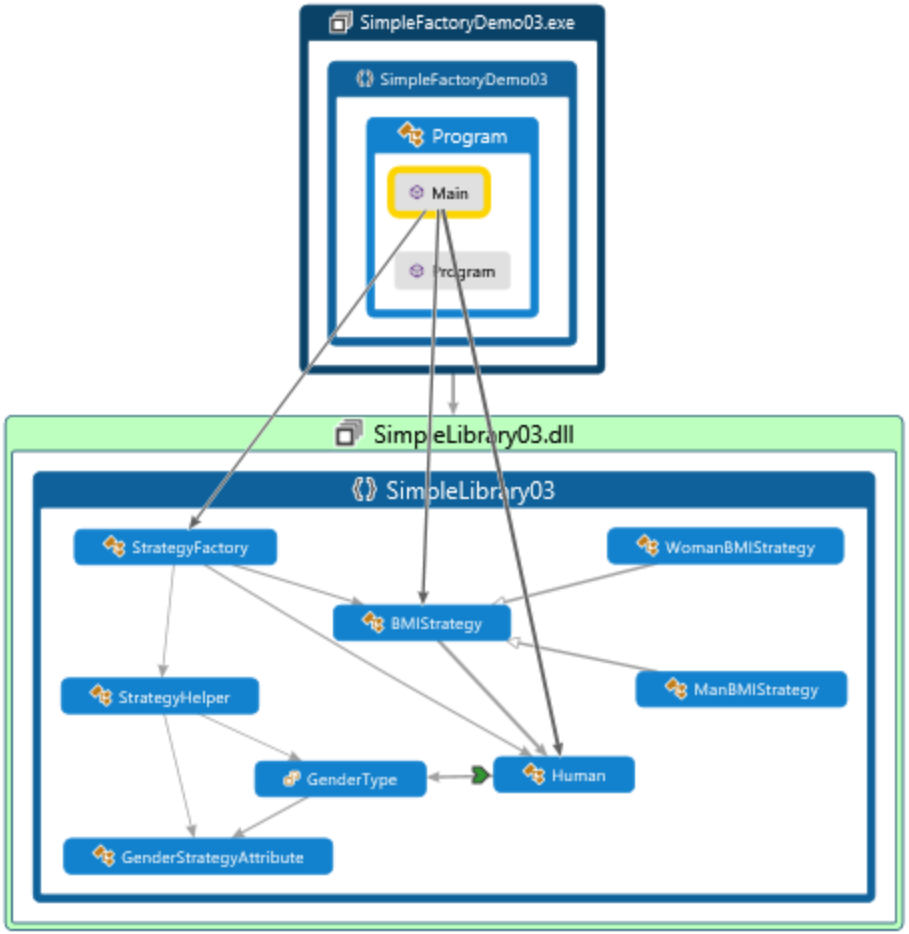
策略模式
Strategy



策略模式

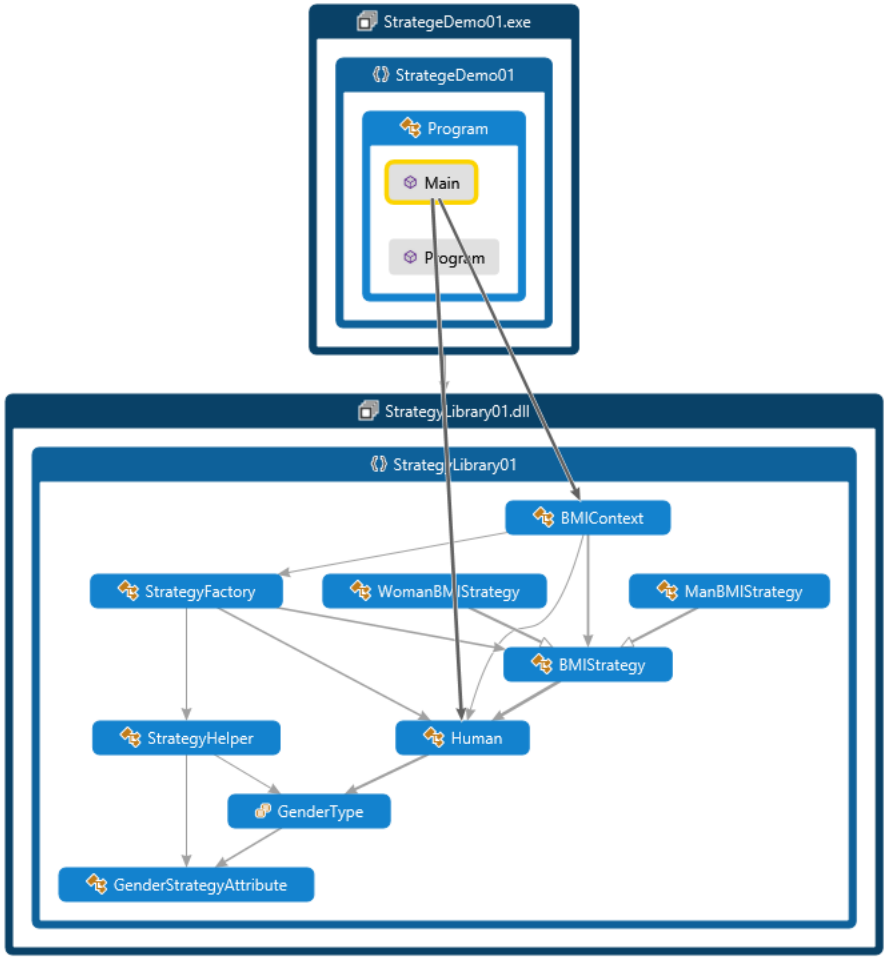
- 定義一組演算法, 將每個演算法封裝, 並且使它們可以互換
- 改善工廠的封裝

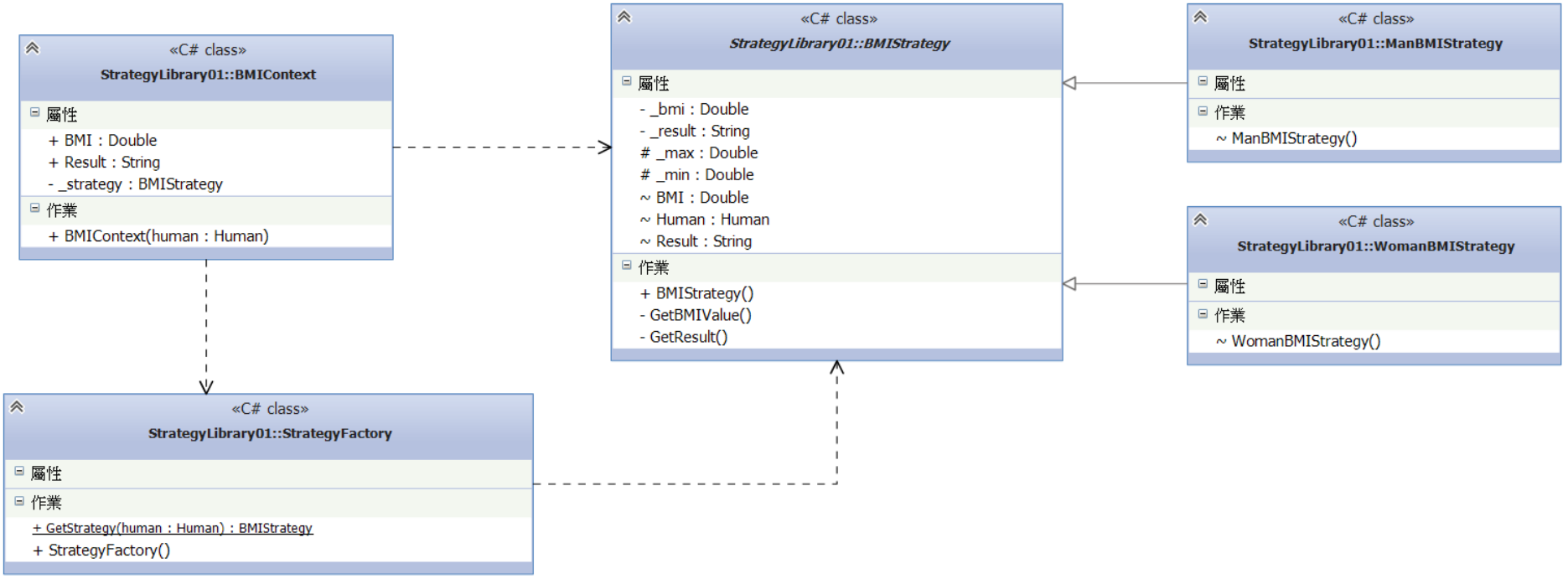
未使用Strategy Context封装前



```
 static void Main(string[] args)
 {
     Human human = new Human() { Age = 19, Gender = GenderType.Man,
Height = 1.72, Weight = 58 };
     BMIStrategy strategy = StrategyFactory.GetStrategy(human);

     Console.WriteLine(strategy.BMI.ToString());
     Console.WriteLine(strategy.Result);

     Console.ReadLine();
 }
```

使用 Strategy Context 封裝後

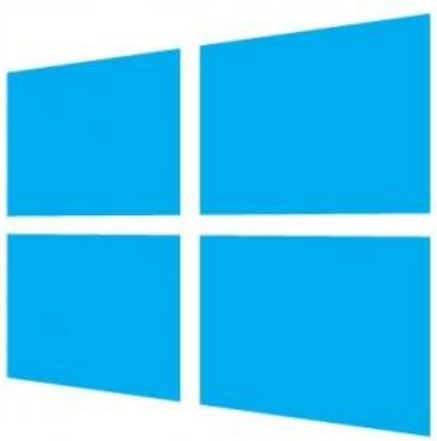



«C# class»
StrategyLibrary01::BMIContext
屬性
+ BMI : Double
+ Result : String
- _strategy : BMIStrategy
作業
+ BMIContext(human : Human)
«C# class»
StrategyLibrary01::BMIStrategy
屬性
- _bmi : Double
- _result : String
_max : Double
_min : Double
~ BMI : Double
~ Human : Human
~ Result : String
作業
+ BMIStrategy()
- GetBMIValue()
- GetResult()
«C# class»
StrategyLibrary01::ManBMIStrategy
屬性
作業
~ ManBMIStrategy()
«C# class»
StrategyLibrary01::WomanBMIStrategy
屬性
作業
~ WomanBMIStrategy()
«C# class»
StrategyLibrary01::StrategyFactory
屬性
作業
+ GetStrategy(human : Human) : BMIStrategy
+ StrategyFactory()

```
public class BMIContext
{
    BMIStrategy _strategy;

    public BMIContext(Human human)
    {
        //封裝 Factory 建立實體的過程
        _strategy = StrategyFactory.GetStrategy(human);
    }

    public Double BMI
    {
        get { return _strategy.BMI; }
    }

    public String Result
    {
        get { return _strategy.Result;}
    }
}
```

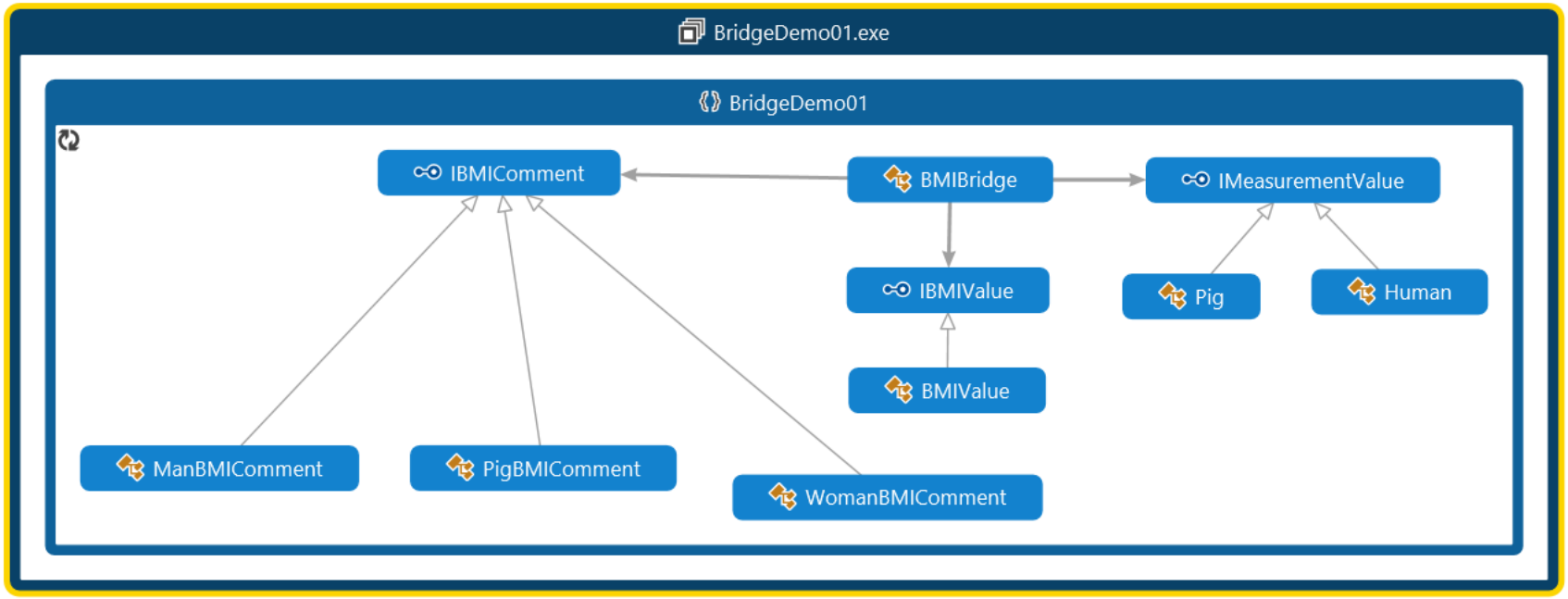
```
static void Main(string[] args)
{
    Human human = new Human() { Age = 19, Gender = GenderType.Woman,
Height = 1.72, Weight = 58 };
    BMIContext bmicontext = new BMIContext(human);
    Console.WriteLine(bmicontext.BMI);
    Console.WriteLine(bmicontext.Result);

    Console.ReadLine();
}
```

橋接模式
Bridge



橋接模式

- 將抽象和實作解耦合, 使其個別可以獨立變化
- 簡單說就是用組合/聚合替代繼承
- 適用情境


BridgeDemo01.exe
BridgeDemo01
IBMIComment
BMIBridge
IMeasurementValue
IBMIValue
Pig
Human
BMIValue
ManBMIComment
PigBMIComment
WomanBMIComment

SKILLTREE

謝謝各位

http://skilltree.my