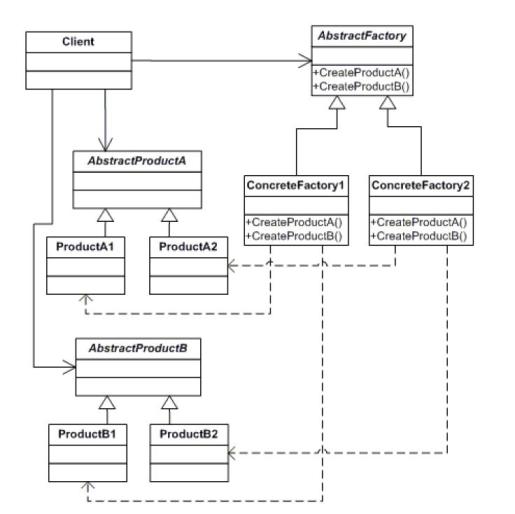
1. Създаващи шаблони за дизайн – Abstract Factory pattern

Шаблона за дизайн Abstract Factory се използва, когато е необходимо на клиента да създава обекти, които са свързани по някакъв начин. Този шаблон е най-полезен, когато трябва да се направи набор от такива свързани обекти и клиента не знае точно кой тип трябва да създаде.

Абстрактната фабрика съдържа описание на методите с които се създават обектите, без да има конкретна имплементация. Нейните наследници, които могат да се инстанцират имплементират тези методи и създават съответните обекти, като може да има много на брой такива наследници създаващи различни типове обекти. Шаблона е особено полезен при създаването на Графичен интерфейс при създаването на свързани компоненти.

Подобни шаблони са Simple Factory, Factory Method и Builder Pattern



Пример за Abstract Factory pattern:

public interface AbstractFactory

{

AbstractProductA CreateProductA();

AbstractProductB CreateProductB();

}

public class ConcreteFactoryA : AbstractFactory

{

public AbstractProductA CreateProductA()

{

return new ProductA1();

}

public AbstractProductB CreateProductB()

{

return new ProductB1();

}

}

public class ConcreteFactoryB : AbstractFactory

{

public AbstractProductA CreateProductA()

{

return new ProductA2();

}

public AbstractProductB CreateProductB()

{

return new ProductB2();

}

}

public interface AbstractProductA { }

public class ProductA1 : AbstractProductA { }

public class ProductA2 : AbstractProductA { }

public interface AbstractProductB { }

public class ProductB1 : AbstractProductB { }

public class ProductB2 : AbstractProductB { }

public class Client

{

private AbstractProductA \_productA;

private AbstractProductB \_productB;

public Client(AbstractFactory factory)

{

\_productA = factory.CreateProductA();

\_productB = factory.CreateProductB();

}

}

1. Структурни шаблони за дизайн – Facade pattern (Фасада)

Фасадата се използва за скриване на сложността на определена система, като предоставя унифициран интерфейс към функционалностите й, а вътрешно се грижи за правилното използване на различните класове, които я съставят.

Фасадата спомага за разскачване на системата от многобройните и клиенти, но от друга страна лимитира функционалностите предоставяни на тези клиенти.

Фасадата дефинира нов интерфейс за работа с цялостната функционалност.

Подобен шаблон за дизайн е Adapter pattern.



Пример за Façade pattern:

class MainApp

{

public static void Main()

{

Facade facade = new Facade();

facade.MethodA();

facade.MethodB();

// Wait for user

Console.ReadKey();

}

}

class SubSystemOne

{

public void MethodOne()

{

Console.WriteLine(" SubSystemOne Method");

}

}

class SubSystemTwo

{

public void MethodTwo()

{

Console.WriteLine(" SubSystemTwo Method");

}

}

class SubSystemThree

{

public void MethodThree()

{

Console.WriteLine(" SubSystemThree Method");

}

}

class SubSystemFour

{

public void MethodFour()

{

Console.WriteLine(" SubSystemFour Method");

}

}

class Facade

{

private SubSystemOne one;

private SubSystemTwo two;

private SubSystemThree three;

private SubSystemFour four;

public Facade()

{

one = new SubSystemOne();

two = new SubSystemTwo();

three = new SubSystemThree();

four = new SubSystemFour();

}

public void MethodA()

{

Console.WriteLine("\nMethodA");

one.MethodOne();

two.MethodTwo();

four.MethodFour();

}

public void MethodB()

{

Console.WriteLine("\nMethodB");

two.MethodTwo();

three.MethodThree();

}

}

1. Поведенчески шаблони за дизайн – Strategy Pattern

Strategy Pattern се използва за дефиниране на набор от алгоритми, като ги енкапсулира и ги прави взаимно заменяеми. Това позволява клиентите да ползват различни алгоритми в зависимост от необходимостта чрез единен интерфейс.

Strategy pattern предоставя много добър механизъм за спазване на Open/closed принципа, чрез изнасяне на поведението на даден клас в набор от други класове, които чрез общия си интерфейс могат да бъдат подавани на този клас и да определят съответното му поведение.

Strategy Pattern се прилага в много различни ситуации, примерно при разработка на изкуствен интелект за игри, или при правене на йерархия от графични компоненти.

Подобен шаблон е Bridge pattern.



Пример за Strategy Pattern:

class MainApp

static void Main()

{

Context context;

// Three contexts following different strategies

context = new Context(new ConcreteStrategyA());

context.ContextInterface();

context = new Context(new ConcreteStrategyB());

context.ContextInterface();

context = new Context(new ConcreteStrategyC());

context.ContextInterface();

Console.ReadKey();

}

}

abstract class Strategy

{

public abstract void AlgorithmInterface();

}

class ConcreteStrategyA : Strategy

{

public override void AlgorithmInterface()

{

Console.WriteLine(

"Called ConcreteStrategyA.AlgorithmInterface()");

}

}

class ConcreteStrategyB : Strategy

{

public override void AlgorithmInterface()

{

Console.WriteLine(

"Called ConcreteStrategyB.AlgorithmInterface()");

}

}

class ConcreteStrategyC : Strategy

{

public override void AlgorithmInterface()

{

Console.WriteLine(

"Called ConcreteStrategyC.AlgorithmInterface()");

}

}

class Context

{

private Strategy \_strategy;

public Context(Strategy strategy)

{

this.\_strategy = strategy;

}

public void ContextInterface()

{

\_strategy.AlgorithmInterface();

}

}