**Builder Design Pattern**

Намерение

Отделя изграждането на сложен обект от неговото представяне, така че един и същ процес може да се създадe различни представяния.

Проблем

Приложението трябва да създаде елементите на сложен агрегат. Спецификацията за агрегат съществува във вторично хранилище и едно от многото представяния трябва да се изгради в главното хранилище.

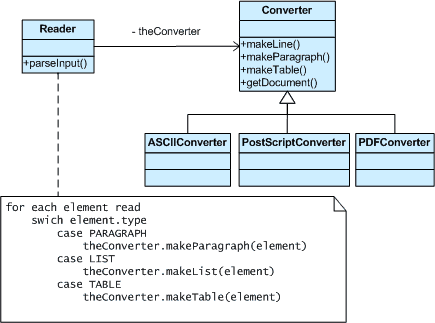
Дискусия

Отделяне на алгоритъм за интерпретиране (т.е. четене и писане) на някакъв механизъм (напр. RTF файлове) от алгоритъма за изграждане и презентиране на един от многото крайни продукти (например ASCII, TeX, text widget). В центъра на вниманието е създаването на сложни агрегати.

"Директора" се позовава на услугите на "Строителя", след като той интерпретира входните данни. "Строителя" създава част от сложен обект, всеки път когато се извика и поддържа всички междинни състояния. Когато продуктът е завършен, клиентът получава резултата от "строителя".

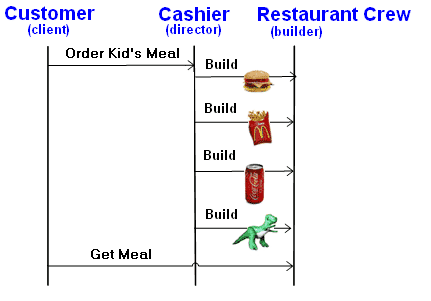
Позволява фин контрол върху процеса на строителството. За разлика от творческите „patterns“, които изграждат продуктите наведнъж, **Builder pattern** изгражда продукта стъпка по стъпка под контрола на "директора".

Структура



Пример

**„Builder”** модела разделя изграждането на сложен обект, от неговото представяне, така че един и същ строителен процес може да създаде различни представяния. Този модел се използва от ресторанти за бързо хранене да се изгради храна за децата. Детските ястия обикновено се състоят от основно ядене, допълнително, едно питие и играчка (например, един хамбургер, пържени картофи и кока-кола, и динозавър играчка). Имайте предвид, че може да има промяна в съдържанието на храната на децата, но процеса на приготвяне е един и същ. Дали клиентът поръчва хамбургер, чийзбургер, или пиле, процесът е един и същ. Служителят на гишето насочва на служителите да съберат основното ястие, допълнителното, и играчката. Тези елементи се поставят в торбичка. Напитката се поставят в чаша и оставя извън опаковката. Същият процес се използва при конкурентни ресторанти.



C# example

// "Director"

**class** Director

{

// Builder uses a complex series of steps

**public** **void** Construct(Builder builder)

{

builder.BuildPartA();

builder.BuildPartB();

}

}

// "Builder"

abstract **class** Builder

{

**public** abstract **void** BuildPartA();

**public** abstract **void** BuildPartB();

**public** abstract Product GetResult();

}

// "ConcreteBuilder1"

**class** ConcreteBuilder1 : Builder

{

**private** Product product = **new** Product();

**public** override **void** BuildPartA()

{

product.Add("PartA");

}

**public** override **void** BuildPartB()

{

product.Add("PartB");

}

**public** override Product GetResult()

{

**return** product;

}

}

// "ConcreteBuilder2"

**class** ConcreteBuilder2 : Builder

{

**private** Product product = **new** Product();

**public** override **void** BuildPartA()

{

product.Add("PartX");

}

**public** override **void** BuildPartB()

{

product.Add("PartY");

}

**public** override Product GetResult()

{

**return** product;

}

}

// "Product"

**class** Product

{

ArrayList parts = **new** ArrayList();

**public** **void** Add(string part)

{

parts.Add(part);

}

**public** **void** Show()

{

Console.WriteLine("\nProduct Parts -------");

foreach (string part in parts)

Console.WriteLine(part);

}

}