Министерство транспорта Российской Федерации

Федеральное агентство железнодорожного транспорта

Федеральное государственное образовательное учреждение высшего профессионального образования

«Дальневосточный Государственный Университет Путей Сообщения»

Кафедра «Информационные Системы и Технологии»

Лабораторная работа

На тему: «Паттерны»

Выполнил: Рудь О.М.

группа 220

Проверил: Водолажский А.А.

Хабаровск, 2013

## Что такое паттерны

При создании программных систем перед разработчиками часто встает проблема выбора тех или иных проектных решений. В этих случаях на помощь приходят паттерны. Дело в том, что почти наверняка подобные задачи уже решались ранее и уже существуют хорошо продуманные элегантные решения, составленные экспертами. Если эти решения описать и систематизировать в каталоги, то они станут доступными менее опытным разработчикам, которые после изучения смогут использовать их как шаблоны или образцы для решения задач подобного класса. Паттерны как раз описывают решения таких повторяющихся задач.

Концепция создания программного обеспечения с использованием паттернов, несомненно, очень важная, но относительно молодая, быть может, поэтому до сих пор нет четкого определения, что же такое паттерн. Об этом свидетельствуют непрекращающиеся дискуссии в популярной литературе и на соответствующих форумах в сети.

Например, следует ли считать алгоритмы и структуры данных паттернами? По этому вопросу существуют противоположные мнения. Согласно одному из них, алгоритмы являются вычислительными паттернами, а хорошо известная фундаментальная монография Дональда Кнута "Искусство программирования" по сути, представляет собой каталог таких паттернов. Согласно другому мнению, алгоритмы не являются паттернами, так как решаемые ими проблемы слишком малы (оперируют такими понятиями как вычислительная сложность и потребление ресурсов), а область решения хорошо очерчена. Паттерны же решают проблемы большего масштаба, при этом паттерн дает не конкретное решение, а некий путь к решению, причем, выбор правильного паттерна - задача нетривиальная, предполагающая от архитектора наличие интуиции, опыта, определенного творчества.

## Что же такое вообще паттерн в ООП?

Паттерн (от англ. Pattern) — образец, шаблон.  
Представьте, что вы хотите сделать новый автомобиль, но вы никогда этим не занимались. Сколько колес и почему вы спроектируете для него? Сейчас вы уже скорее всего скажете что 4, однако почему не 3, 5, 10, 20? Потому-что практикой использования уже было выяснено, что обычные автомобили лучше всего делать на 4-х колесах — это шаблон проектирования сформированный временем. Именно такому же подходу и служат паттерны в ООП и вы не столкнетесь с ними в разработке до тех пор, пока вам не потребуется «сделать автомобиль». Однако иногда случается так, что вы создаете «трицикл», и только потом, набив несколько шишек с его устойчивость и неудачным вписыванием в колею на дороге, узнаете что существует паттерн «автомобиль», который значительно упростил бы вам жизнь, знай вы про него ранее.  
  
*Примечание:*  
Паттерны не привязаны к какому-либо конкретному языку программирования. Это просто подход к проектированию чего-либо. Если смотреть глубже, то многие паттерны ООП были созданы на основе реальных жизненный ситуаций в проектировании вполне себе осязаемых объектов нашего мира. Именно на таких метафорах и описаниях и будет построено дальнейшее изложение.

## Структурирующие паттерны

Данные паттерны помогают внести порядок и научить разные объекты более правильно взаимодействовать друг с другом.

## Adapter или wrapper (адаптер, обертка)

Данный паттерн полностью соответствует своему названию. Чтобы заставить работать «советскую» вилку через евро-розетку требуется переходник. Именно это и делает «адаптер», служит промежуточным объектом между двумя другими, которые не могут работать напрямую друг с другом.

##### Bridge (мост)

Представим ситуацию, когда вам требуется работать на разных автомобилях, однако садясь в новый автомобиль вам уже желательно знать как им управлять. Таким образом вы сталкиваетесь с паттерном «мост». С одной стороны вы имеете множество различных автомобилей (разные модели и марки), но среди все них есть общая абстракция (интерфейс) ввиде руля, педалей, коробки передач и так далее. Таким образом мы задаем как-бы правила изготовления автомобилей по которым мы можем создавать любые их виды, но за счет сохранения общих правил взаимодействия с ними, мы можем одинаково управлять каждым из них. «Мостом» в данном случае является пара двух «объектов»: конкретного автомобиля и правил взаимодействия с этим (и любым другим) автомобилем.

##### Composite (компоновщик)

Довольно интересный паттерн суть которого заключается в минимизации различий в управлении как группами объектов так и индивидуальными объектами. Для примера можно рассмотреть управление солдатами в строю. Существует строевой устав, который определяет как управлять строем и согласно этого устава абсолютно не важно кому отдается приказ (например «шагом марш») одному солдату или целому взводу. Соответственно в устав (если его в чистом виде считать паттерном «компоновщик») нельзя включить команду, которую может исполнить только один солдат, но не может исполнить группа, или наоборот.

##### Decorator (декоратор, оформитель)

Как понятно из названия, данный паттерн чаще всего используется для расширения исходного объекта до требуемого вида. Например мы условно можем считать «декоратором» человека с кистью и красной краской. Таким образом, какой бы объект (или определенный тип объектов) мы не передали в руки «декоратору», на выходе мы будем получать красные объекты.

##### Facade (фасад)

Паттерн «фасад» используется для того, чтобы делать сложные вещи простыми. Возьмем для примера автомобиль. Представьте, если бы управление автомобилем происходило немного по-другому: нажать одну кнопку чтобы подать питание с аккумулятора, другую чтобы подать питание на инжектор, третью чтобы включить генератор, четвертую чтобы зажечь ламочку на панели и так далее. Всё это было бы очень сложно. Для этого такие сложные наборы действий заменяются более простыми и комплексные как «повернуть ключ зажигания». В данном случае поворот ключа зажигания и будет тем самым «фасадом» для всего обилия внутренних действий автомобиля.

##### Front controller (единая точка входа)

Если проводить аналогии с реальными миром, то «единая точка входа» это то, через что вы сейчас читаете данную статью (например броузер). Она служит «единой точкой входа» для всего интернет пространства. То есть вы используете один интерфейс (броузер) для получения доступа к разным объектам большой системы (сайтам в интернете). Данный паттерн в целом сильно похож на «фасад».

##### Flyweight (приспособленец)

Самым лучшим примером (который я смог найти в реальной жизни) для метафорического сравнения паттерна «приспособленец» является театральная постановка. Представьте что нам требуется поставить пьесу. Однако по сценарию в этой пьесе задействованы несколько десятков людей, которые по своей сути выполняют одинаковые действия, например участвуют в массовках различных сцен в разные промежутки времени, но между ними всё же есть какие-то различия (например костюмы). Нам бы стоило огромных денег нанимать для каждой роли отдельного актера, поэтому мы используем паттерн «приспособленец». Мы создадим все нужные нам костюмы, но для каждой массовки будем переодевать небольшую группу актеров в требуемые для этой сцены костюмы. В результате мы имеем возможность ценой малых ресурсов создавать видимость управления большим количеством казалось бы разных объектов.

##### Proxy или surrogate (прокси, заместитель, суррогат)

Данный паттерн позволяет создавать какие-либо специальные механизмы доступа к объекту, что чаще всего направлено именно на улучшение производительности отдельных частей программы. В реальной жизни можно привести следующий пример: сотрудникам одного из подразделений фирмы регулярно требуется получать информацию о том, какого числа бухгалтерия планирует выплатить зарплату. С одной стороны каждый из них может индивидуально и регулярно ездить в бухгалтерию для выяснения этого вопроса (полагаю такая ситуация нередко встречается во многих организациях). С другой стороны, при приближении планируемой даты подразделение может выбрать одного человека, который будет выяснять эту информацию у бухгалтерии, а в последствии уже все в подразделении могут выяснить эту информацию у него (что значительно быстрее). Вот именно этот человек и будет реализованным «прокси» паттерном, который будет предоставлять специальный механизм доступа к информации из бухгалтерии.

**Задание:**

1. Описание паттернов
2. Диаграмма классов
3. Исполняемый код работающих приложений

## Назначение паттерна Adapter

Часто в новом программном проекте не удается повторно использовать уже существующий код. Например, имеющиеся классы могут обладать нужной функциональностью, но иметь при этом несовместимые интерфейсы. В таких случаях следует использовать паттерн Adapter (адаптер).

Паттерн Adapter, представляющий собой программную обертку над существующими классами, преобразует их интерфейсы к виду, пригодному для последующего использования.

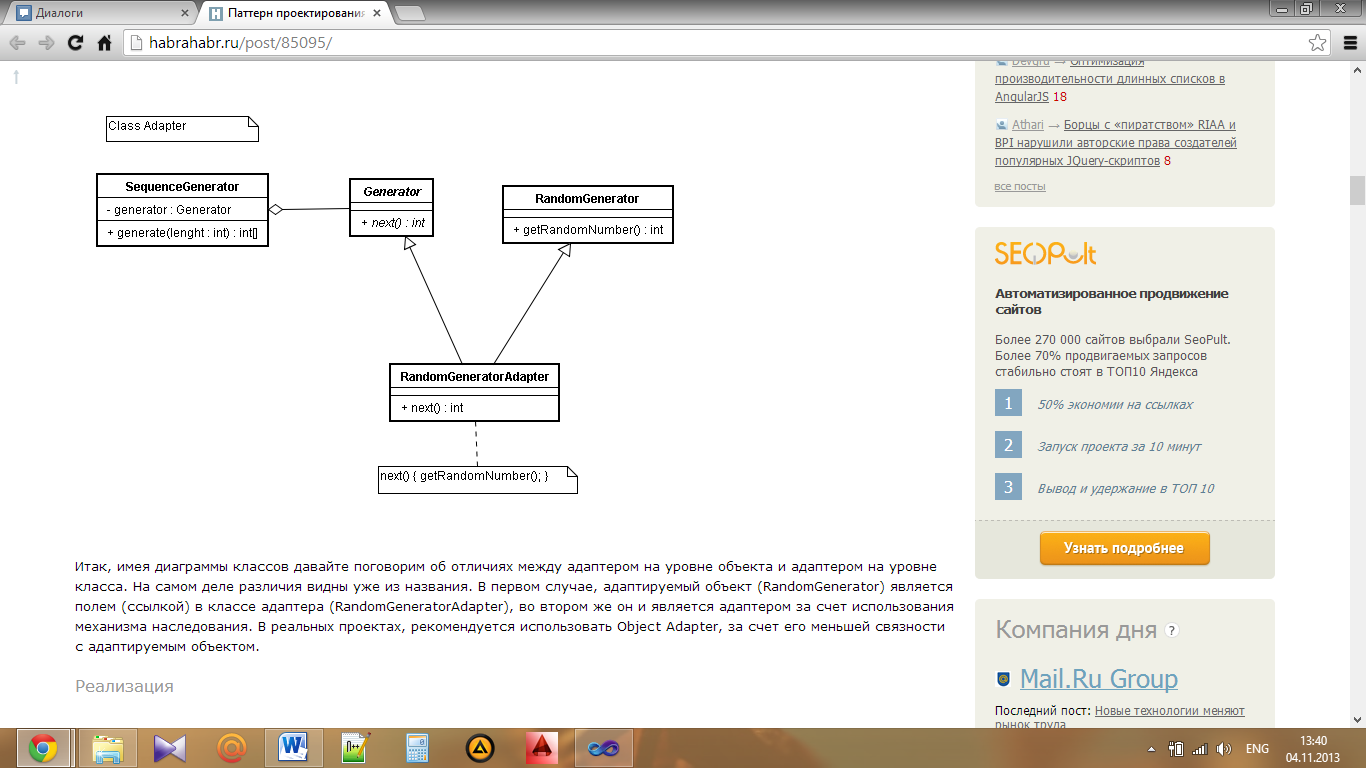
Рассмотрим простой пример, когда следует применять паттерн Adapter. Пусть мы разрабатываем систему климат-контроля, предназначенной для автоматического поддержания температуры окружающего пространства в заданных пределах. Важным компонентом такой системы является температурный датчик, с помощью которого измеряют температуру окружающей среды для последующего анализа. Для этого датчика уже имеется готовое программное обеспечение от сторонних разработчиков, представляющее собой некоторый класс с соответствующим интерфейсом. Однако использовать этот класс непосредственно не удастся, так как показания датчика снимаются в градусах Фаренгейта. Нужен адаптер, преобразующий температуру в шкалу Цельсия.

Контейнеры queue, priority\_queue и stack библиотеки стандартных шаблонов STL реализованы на базе последовательных контейнеров list, deque и vector, адаптируя их интерфейсы к нужному виду. Именно поэтому эти контейнеры называют контейнерами-адаптерами.

## Описание паттерна Adapter

Пусть класс, интерфейс которого нужно адаптировать к нужному виду, имеет имя Adaptee. Для решения задачи преобразования его интерфейса паттерн Adapter вводит следующую иерархию классов:

* Виртуальный базовый класс Target. Здесь объявляется пользовательский интерфейс подходящего вида. Только этот интерфейс доступен для пользователя.
* Производный класс Adapter, реализующий интерфейс Target. В этом классе также имеется указатель или ссылка на экземпляр Adaptee. Паттерн Adapter использует этот указатель для перенаправления клиентских вызовов в Adaptee. Так как интерфейсы Adaptee и Target несовместимы между собой, то эти вызовы обычно требуют преобразования.



Код Adapter:

**interface Generator**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Adapter

{

interface Generator

{

int Next();

}

}

**RandomAdapter.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Adapter

{

class RandomAdapter:Generator

{

Random d = new Random();

public int Next()

{

return d.Next();

}

}

}

**Sequence\_Generator.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Adapter

{

class Sequence\_Generator

{

public Generator Generator1;

public int[] Generate(int length)

{

int[] m=new int[length];

for (int i = 0; i < length; i++)

m[i] = Generator1.Next();

return m;

}

}

}

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Adapter

{

class Program

{

static void Main(string[] args)

{

Sequence\_Generator SG = new Sequence\_Generator();

SG.Generator1 = new RandomAdapter();

int[] m1 = SG.Generate(5);

for (int i = 0; i < m1.Length;i++ )

Console.WriteLine(m1[i]);

Console.ReadLine();

}

}

}

# Паттерн Proxy (заместитель, surrogate, суррогат)

## Назначение паттерна Proxy

* Паттерн Proxy является суррогатом или замеcтителем другого объекта и контролирует доступ к нему.
* Предоставляя дополнительный уровень косвенности при доступе к объекту, может применяться для поддержки распределенного, управляемого или интеллектуального доступа.
* Являясь "оберткой" реального компонента, защищает его от излишней сложности.

## Решаемая проблема

Вам нужно управлять ресурсоемкими объектами. Вы не хотите создавать экземпляры таких объектов до момента их реального использования.

## Обсуждение паттерна Proxy

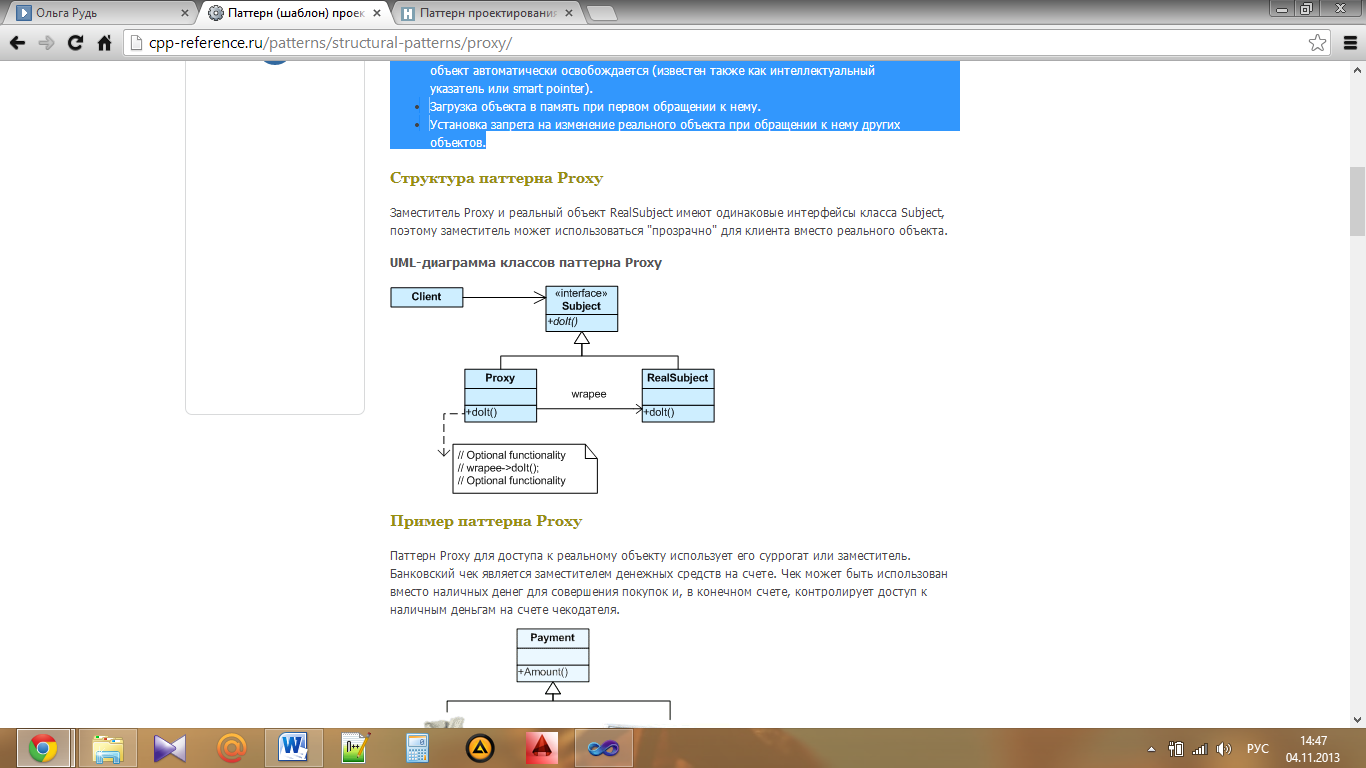
Суррогат или заместитель это объект, интерфейс которого идентичен интерфейсу реального объекта. При первом запросе клиента заместитель создает реальный объект, сохраняет его адрес и затем отправляет запрос этому реальному объекту. Все последующие запросы просто переадресуются инкапсулированному реальному объекту.

Существует четыре ситуации, когда можно использовать паттерн Proxy:

* Виртуальный proxy является заместителем объектов, создание которых обходится дорого. Реальный объект создается только при первом запросе/доступе клиента к объекту.
* Удаленный proxy предоставляет локального представителя для объекта, который находится в другом адресном пространстве ("заглушки" в RPC и CORBA).
* Защитный proxy контролирует доступ к основному объекту. "Суррогатный" объект предоставляет доступ к реальному объекту, только вызывающий объект имеет соответствующие права.
* Интеллектуальный proxy выполняет дополнительные действия при доступе к объекту.

Вот типичные области применения интеллектуальных proxy:

* Подсчет числа ссылок на реальный объект. При отсутствии ссылок память под объект автоматически освобождается (известен также как интеллектуальный указатель или smart pointer).
* Загрузка объекта в память при первом обращении к нему.
* Установка запрета на изменение реального объекта при обращении к нему других объектов.



Код Proxy

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Proksi

{

interface Subject

{

int Request();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace Proksi

{

class RealSubject:Subject

{

public RealSubject() { Thread.Sleep(2000); }

public int Request() { return 10; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Proksi

{

class ProxySubject:Subject

{

private RealSubject rs = null;

public int Request()

{

if (rs==null)

rs=new RealSubject();

return rs.Request();}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Proksi

{

class Program

{

static void Main(string[] args)

{

Random e = new Random();

Subject[] m=new Subject[5];

Console.WriteLine(DateTime.Now);

for (int i = 0; i < m.Length; i++)

{

//m[i] = new RealSubject();

m[i] = new ProxySubject();

}

int a = e.Next(0, m.Length);

Console.WriteLine(m[a].Request());

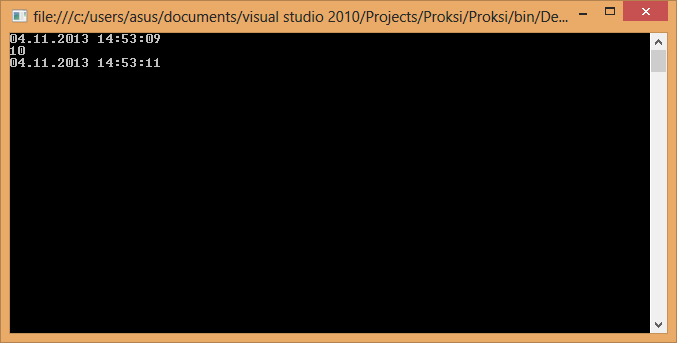
Console.WriteLine(DateTime.Now);

Console.ReadLine();

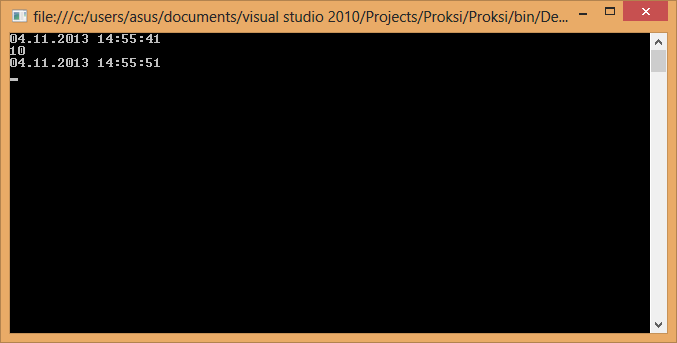
}

}

}



m[i] = new RealSubject();



Список литературы:

1. <http://cpp-reference.ru/patterns/introduction/>
2. <http://habrahabr.ru/post/136766/>
3. <http://cpp-reference.ru/patterns/structural-patterns/proxy/>
4. <http://cpp-reference.ru/patterns/structural-patterns/adapter/>
5. <http://habrahabr.ru/post/85095/>