Учреждения образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра экономической информатики

Лабораторная работа № 6

«Использование паттернов проектирования и SVN

при разработке ПО»

по курсу

«Современные информационные технологии»

Выполнил: студент гр.172304

Жеребицкий В.В.

Проверила:

Волошко Е.А.

Минск 2013 г

**Цель работы**

Изучить понятие паттерна, виды паттернов, их назначение и реализовать паттерн «Команда» для своего приложения.

**Краткие теоретические сведения**

Subversion (SVN) – это централизованная система для совместной работы. В ее основе лежит хранилище, которое содержит данные в форме дерева файловой системы – обычной иерархии файлов и папок. Клиенты подключаются к хранилищу и читают или изменяют эти файлы. Записывая данные, клиент делает информацию доступной для остальных; читая данные, клиент получает информацию от других.

TortoiseSVN – это бесплатный Windows-клиент с открытыми исходным кодом для системы управления версиями Apache Subversion. То есть он управляет файлами и директориями во времени. Файлы хранятся в центральном хранилище. Хранилище больше похоже на обычный файловый сервер, кроме того он запоминает каждое изменение когда-либо сделанное в ваших файлах и директориях. Это позволяет вам восстановить старые версии ваших файлов и проверить историю изменений: как, когда и кто изменил ваши данные. Вот почему многие думаю о SVN и, вообще, о системах управления версиями, как о своего рода «машине времени».

Шаблон проектирования или паттерн проектирования – повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код. Это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориетированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

Виды паттернов:

* Порождающие паттерны
* Поведенческие паттерны
* Структурные паттерны

Перейдем к Порождающим паттернам. Они предназначены для создания новых объектов в системе.

1. **Строитель**

Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.

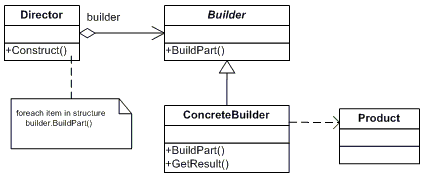


Рисунок 1 – Структура паттерна Строитель (Builder)

1. **Абстрактная фабрика**

Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

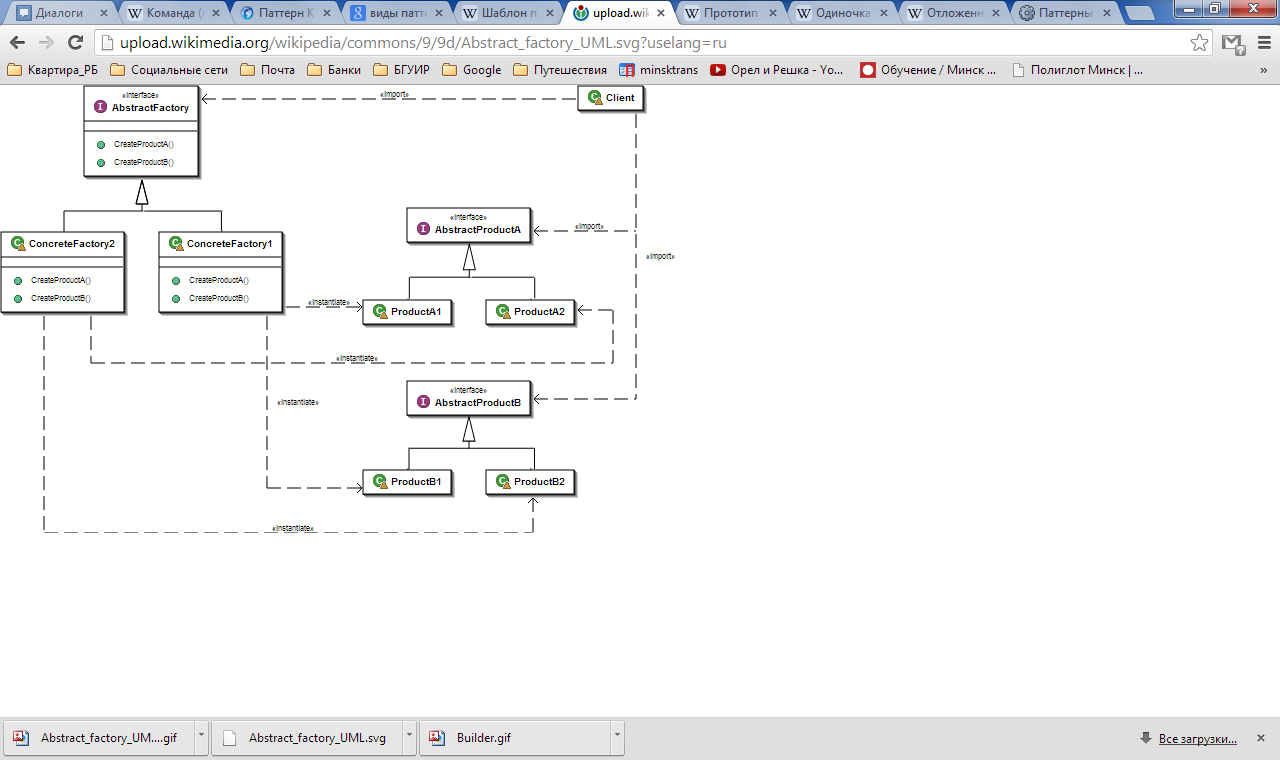


Рисунок 2 – Структура паттерна Абстрактная фабрика (Abstract Factory)

1. **Прототип**

Задаёт виды создаваемых объектов с помощью экземпляра-прототипа и создаёт новые объекты путём копирования этого прототипа. Проще говоря, это паттерн создания объекта через клонирование другого объекта вместо создания через конструктор.

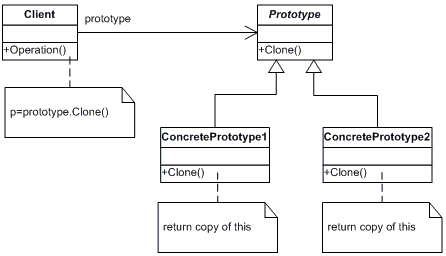


Рисунок 3 – Структура паттерна Прототип (Prototype)

1. **Одиночка**

Гарантирует, что у [класса](http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) есть только один экземпляр, и предоставляет к нему глобальную точку доступа. Существенно то, что можно пользоваться именно экземпляром класса, так как при этом во многих случаях становится доступной более широкая функциональность. Например, к описанным компонентам класса можно обращаться через [интерфейс](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), если такая возможность поддерживается языком.

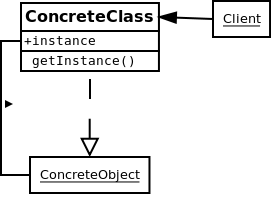


Рисунок 4 – Структура паттерна Одиночка (Singleton)

1. **Отложенная (ленивая) инициализация**

Создание объекта в момент обращения к нему. Как правило, он используется в сочетании с такими шаблонами как Фабричный метод, Одиночка и Заместитель.

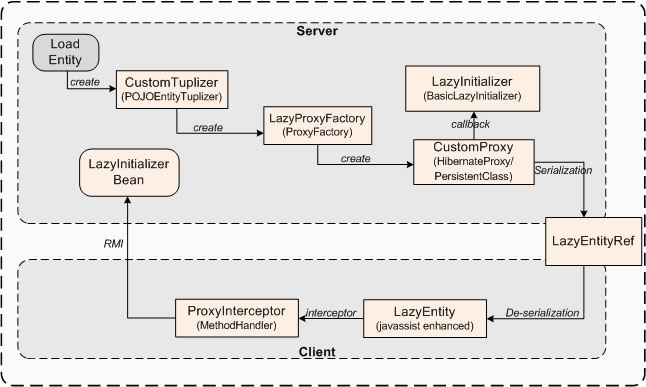


Рисунок 5 – Структура паттерна Отложенная инициализация (Lazy initialization)

Перейдем к Поведенческим паттернам. Они предназначены для распределения обязанностей между объектами в системе.

1. **Посредник**

Обеспечивает взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга.

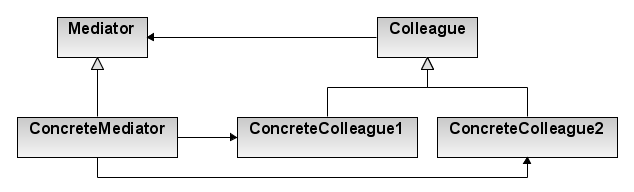


Рисунок 6 – Структура паттерна Посредник (Mediator)

1. **Посетитель**

Описывает операцию, которая выполняется над объектами других классов. При изменении Visitor нет необходимости изменять обслуживаемые классы.

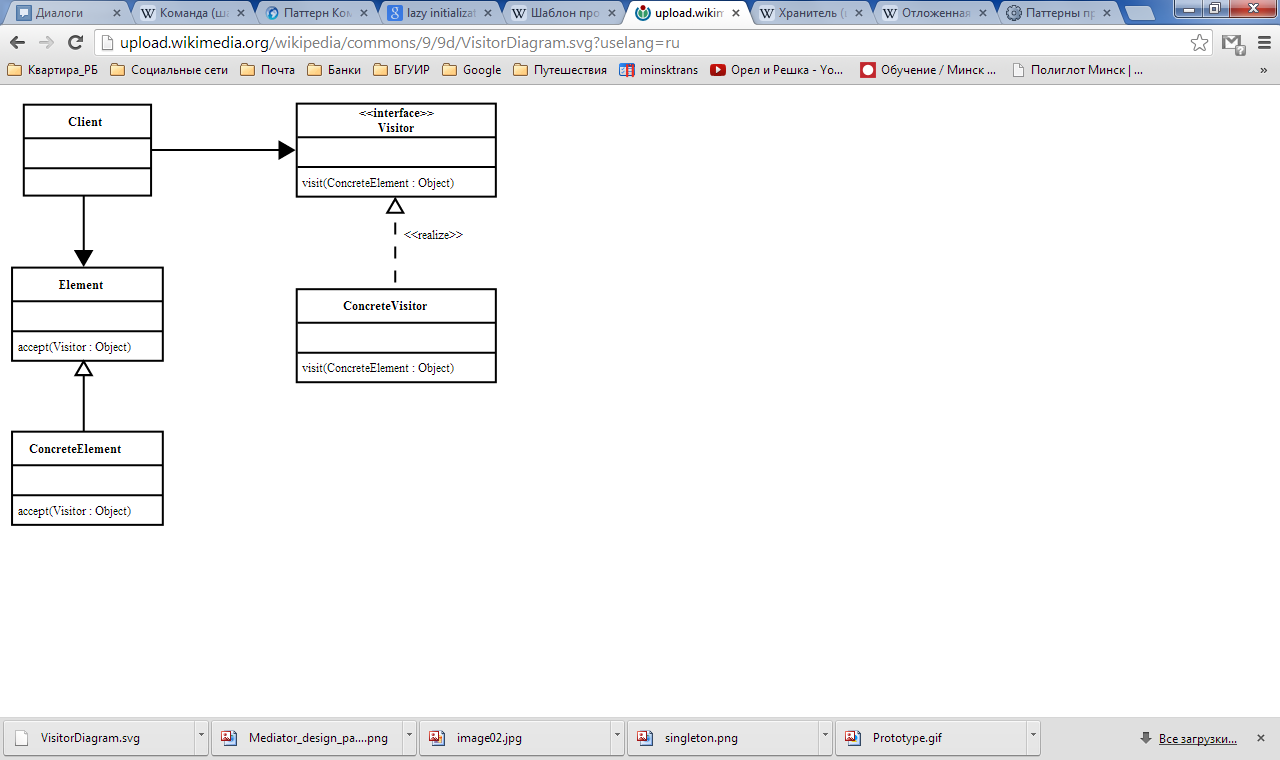


Рисунок 7 – Структура паттерна Посетитель (Visitor)

1. **Хранитель**

Задаёт виды создаваемых объектов с помощью экземпляра-прототипа и создаёт новые объекты путём копирования этого прототипа. Проще говоря,

Позволяет, не нарушая инкапсуляцию, зафиксировать и сохранить внутреннее состояние объекта так, чтобы позднее восстановить его в это состояние.

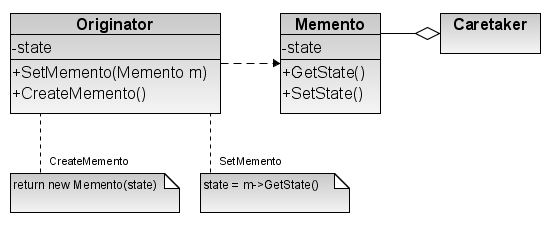


Рисунок 8 – Структура паттерна Хранитель (Memento)

Перейдем к Структурным паттернам. Они решают задачи компоновки системы на основе классов и объектов.

1. **Мост**

Используется, чтобы «разделять абстракцию и реализацию так, чтобы они могли изменяться независимо». Шаблон инкапсулирует агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами.

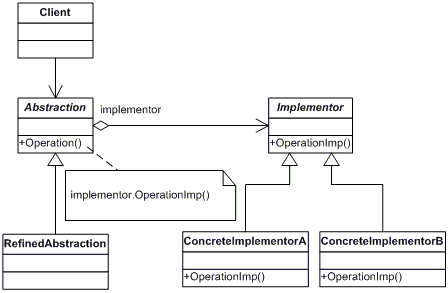


Рисунок 9 – Структура паттерна Мост (Bridge)

1. **Компоновщик**

Объединяет объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково.

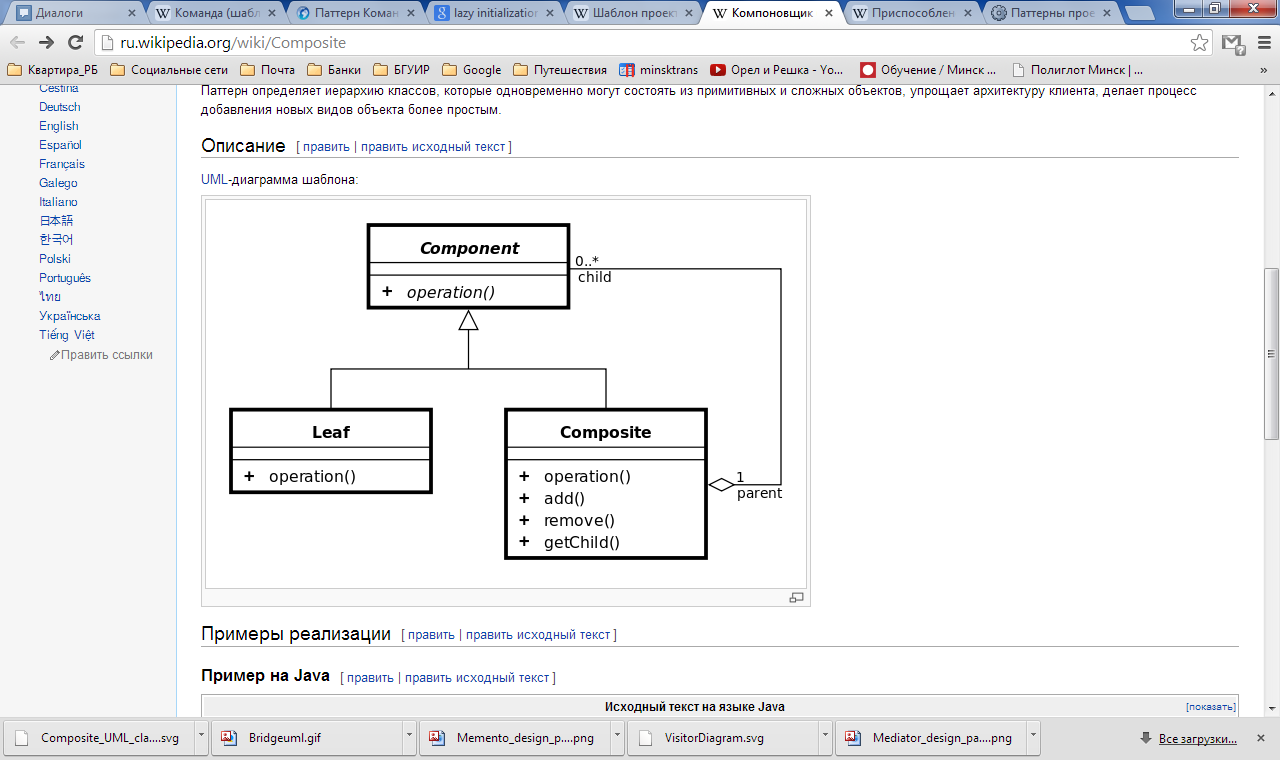


Рисунок 10 – Структура паттерна Компоновщик (Composite)

1. **Приспособленец**

Используется, когда объект, представляющий себя как уникальный экземпляр в разных местах программы, по факту не является таковым.

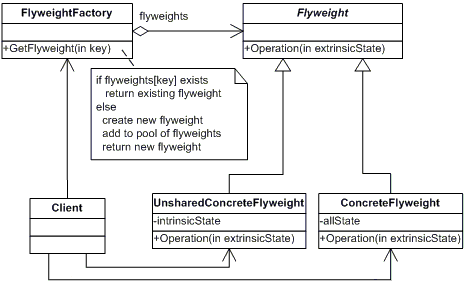


Рисунок 11 – Структура паттерна Приспособленец (Flyweight)

Рассмотрим подробнее поведенческий паттерн **Команда** (Command).

Реализация паттерна Команда включает в себя создание структуры, в которой класс-отправитель и класс-получатель не зависят друг от друга напрямую. А также организацию обратного вызова к классу, который включает в себя класс-отправитель.



Рисунок 11 – Структура паттерна Команда (Command)

****

Рисунок 12 – Диаграмма классов