Учреждения образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра экономической информатики

**Лабораторная работа № 8**

«Использование паттернов проектирования и SVN

при разработке ПО»

по курсу

«Современные информационные технологии»

Выполнила ст. гр.172303

Воронович А.И.

Проверила:

Волошко Е.А.

Минск 2013

**Краткие теоретические сведения**

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Классификация паттернов:

– порождающие паттерны;

– структурные паттерны;

– паттерны поведения;

Порождающие паттерны:

1. **Абстрактная фабрика (англ. Abstract factory).** Паттерн предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

Данный паттерн позволяет изменять поведение системы, варьируя создаваемые объекты, при этом сохраняя интерфейсы. Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение.

Плюсы: изолирует конкретные классы, упрощает замену семейств продуктов, гарантирует сочетаемость продуктов.

Минусы: сложно добавить поддержку нового вида продуктов.

**2. Строитель** – отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.

**3**. **Отложенная (ленивая) инициализация** **(англ. Lazy initialization).** Приём в программировании, когда некоторая ресурсоёмкая операция (создание объекта, вычисление значения) выполняется непосредственно перед тем, как будет использован её результат. Таким образом, инициализация выполняется «по требованию», а не заблаговременно. Аналогичная идея находит применение в самых разных областях: например, компиляция «на лету» и логистическая концепция «Точно в срок».

Частный случай ленивой инициализации — создание объекта в момент обращения к нему — является одним из порождающих шаблонов проектирования. Как правило, он используется в сочетании с такими шаблонами как Фабричный метод, Одиночка и Заместитель.

**4. Одиночка (Singleton).**

Гарантирует, что у класса 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, если такая возможность поддерживается языком.

Плюсом паттерна является контролируемый доступ к единственному экземпляру. Минусами – глобальные объекты могут быть вредны для объектного программирования, в некоторых случаях приводят к созданию немасштабируемого проекта; усложненность написания модульных тестов и следование TDD.

**5. 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.

Паттерн используется чтобы избежать дополнительных усилий по созданию объекта стандартным путем (имеется в виду использование конструктора, т. к. в этом случае также будут вызваны конструкторы всей иерархии предков объекта), когда это непозволительно дорого для приложения; избежать наследования создателя объекта (object creator) в клиентском приложении, как это делает паттерн abstract factory.

Данный паттерн используется, когда система не должна зависеть от того, как в ней создаются, компонуются и представляются продукты.

*Структурные паттерны:*

**1. Декоратор (англ. Decorator) –** структурный шаблон проектирования, структурирующий объекты.

Динамически добавляет объекту новые обязанности. Является гибкой альтернативой порождению подклассов с целью расширения функциональности.

Иногда возникает необходимость добавить некую функциональность не для всего класса, а лишь для конкретного объекта этого класса..

**2. Приспособленец (Flyweight)** используется для уменьшения затрат при работе с большим количеством мелких объектов. При проектировании приспособленца необходимо разделить его свойства на внешние и внутренние. Внутренние свойства всегда неизменны, тогда как внешние могут отличаться в зависимости от места и контекста применения и должны быть вынесены за пределы приспособленца.

Flyweight дополняет паттерн Factory таким образом, что Factory при обращении к ней клиента для создания нового объекта ищет уже созданный объект с такими же параметрами, что и у требуемого, и возвращает его клиенту. Если такого объекта нет, то фабрика создаст новый.

3. **Фасад.**

Фасад – паттерн, структурирующий объекты.

Предоставляет унифицированный интерфейс вместо набора интерфейсов некоторой подсистемы. Фасад определяет интерфейс более высокого уровня, который упрощает использование подсистем.

Разбиение на подсистемы облегчает проектирование сложной системы в целом.

Общая цель всякого проектирования – свести к минимуму зависимость абстракций друг от друга и обмен информацией между ними. Один из способов решения этой задачи – введение объекта Фасад, предоставляющий единый упрощенный интерфейс к более сложным системным средствам.

*Поведенческие паттерны:*

**1. Команда (англ. Command)** – паттерн, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.

Паттерн создается c целью создания структуры, в которой класс-отправитель и класс-получатель не зависят друг от друга напрямую, а также для организации обратного вызова к классу, который включает в себя класс-отправитель.

**2. Посредник.** "Посредник" определяет интерфейс для обмена информацией с объектами "Коллеги", "КонкретныйПосредник" координирует действия объектов "Коллеги". Каждый класс "Коллеги" знает о своем объекте "Посредник", все "Коллеги" обмениваются информацией только с посредником, при его отсутствии им пришлось бы обмениваться информацией напрямую. "Коллеги" посылают запросы посреднику и получают запросы от него. "Посредник" реализует кооперативное поведения, пересылая каждый запрос одному или нескольким "Коллегам".

**3. Хранитель (англ. Memento)** Хранитель — паттерн поведения объектов.

Не нарушая инкапсуляции, фиксирует и выносит за пределы объекта его внутреннее состояние так, чтобы позднее можно было восстановить в нем объект.

Паттерн Хранитель используется в тех случаях, когда необходимо сделать “снимок” состояния объекта, и, возможно, позже воспользоваться этим “снимком”, для восстановления объекта в прежнее состояние. Это может пригодиться, например, при создании точек отката системы или операций последовательной отмены действий.

**Реализация паттерна «Команда»**

****

Рисунок 1 – Диаграмма классов паттерна «Команда»