Лабораторная работа № 4

«Паттерны проектирования»

Краткие теоретические сведения:

Шаблоны проектирования (паттерн, англ. design pattern) — это многократно применяемая архитектурная конструкция, предоставляющая решение общей проблемы проектирования в рамках конкретного контекста и описывающая значимость этого решения. Паттерн не является законченным образцом проекта, который может быть прямо преобразован в код.

1. **Абстрактная фабрика** (англ. Abstract factory) — порождающий шаблон проектирования, позволяющий 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. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки). Затем пишутся наследующиеся от него классы, реализующие этот интерфейс.

   Цель

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

   Плюсы

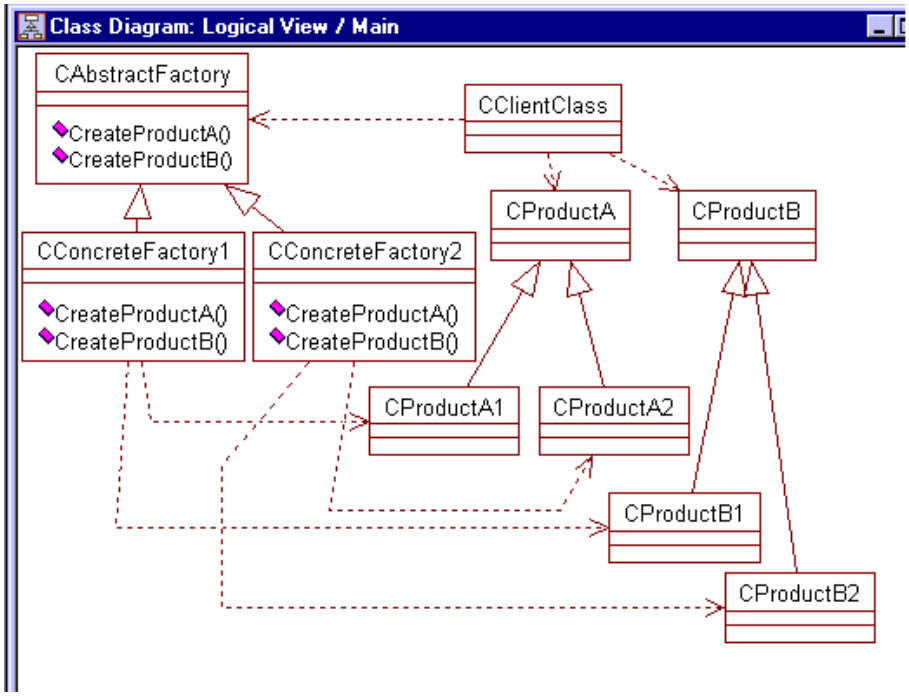
   - изолирует конкретные классы;
   - упрощает замену семейств продуктов;
   - гарантирует сочетаемость продуктов.

   Минусы

   - сложно добавить поддержку нового вида продуктов.

   Применимость

   - Система не должна зависеть от того, как создаются, компонуются и представляются входящие в нее объекты.
   - Входящие в семейство взаимосвязанные объекты должны использоваться вместе и вам необходимо обеспечить выполнение этого ограничения.
   - Система должна конфигурироваться одним из семейств составляющих ее объектов.
   - Требуется предоставить библиотеку объектов, раскрывая только их интерфейсы, но не реализацию.



2. **Строитель, (англ. Builder)** порождающий шаблон проектирования. в нем акцентируется пошаговое конструирование объекта - в отличие от фабрики, где конструируется семейство классов.

   Цель

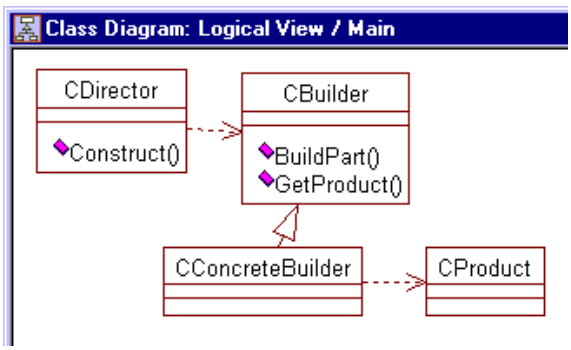
   Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.

   Плюсы

   - позволяет изменять внутреннее представление продукта;
   - изолирует код, реализующий конструирование и представление;
   - дает более тонкий контроль над процессом конструирования.

   Применимость

   - алгоритм создания сложного объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой;
   - процесс конструирования должен обеспечивать различные представления конструируемого объекта.



3. **Фабричный метод (англ. Factory Method)** — порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс инстанциировать. Иными словами, Фабрика делегирует создание объектов наследникам родительского класса. Это позволяет

использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне. Также известен под названием виртуальный конструктор.

Цель

Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанциировать. Фабричный метод позволяет классу делегировать создание подклассам.

Плюсы

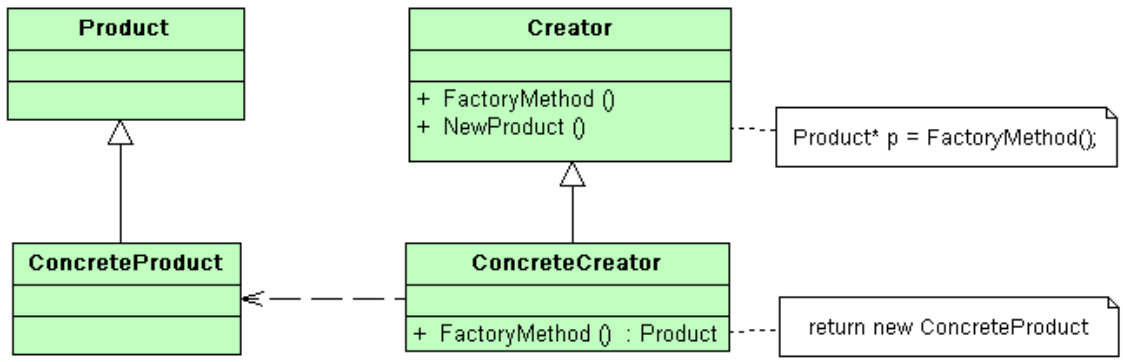
- позволяет сделать код создания объектов более универсальным, не привязываясь к конкретным классам (ConcreteProduct), а оперируя лишь общим интерфейсом (Product);
- позволяет установить связь между параллельными иерархиями классов.

Минусы

- необходимость создавать наследника Creator для каждого нового типа продукта (ConcreteProduct).

Применимость

- классу заранее неизвестно, объекты каких подклассов ему нужно создавать.
- класс спроектирован так, чтобы объекты, которые он создаёт, специфицировались подклассами.
- класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и планируется локализовать знание о том, какой класс принимает эти обязанности на себя.

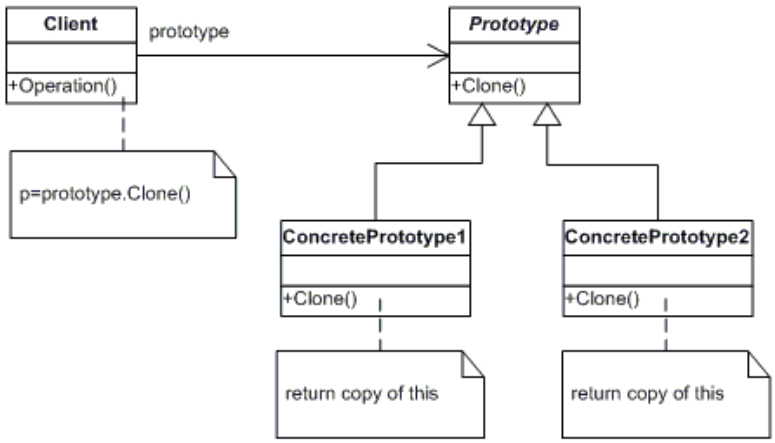


4. **Прототип** (англ. *Prototype*) — порождающий шаблон проектирования. Задаёт виды создаваемых объектов с помощью экземпляра-прототипа и создаёт новые объекты путём копирования этого прототипа.

Применимость

Используйте этот шаблон проектирования, когда система не должна зависеть от того, как в ней создаются, компонуются и представляются продукты:

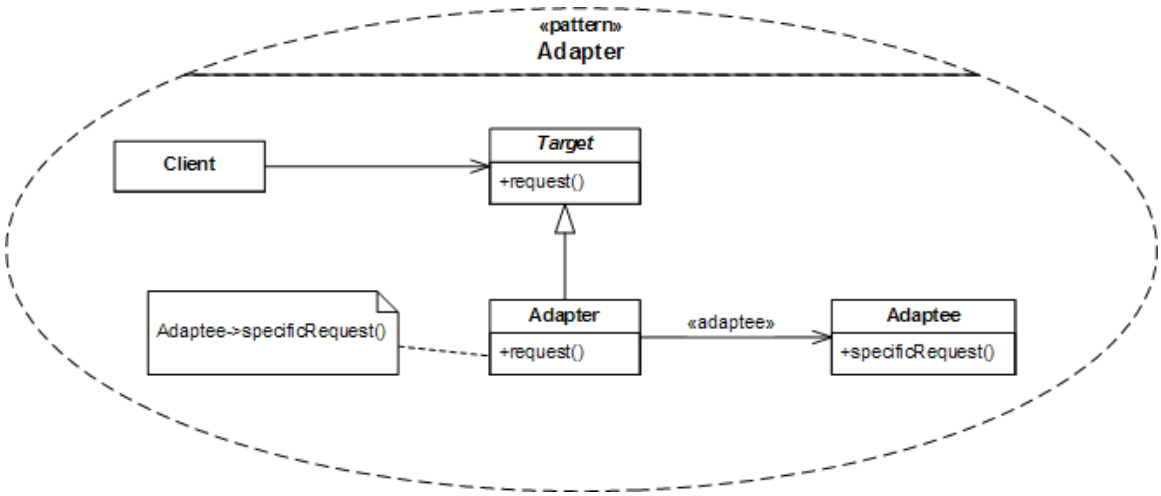
- инстанцируемые классы определяются во время выполнения, например с помощью динамической загрузки;
- для того чтобы избежать построения иерархий классов или фабрик, параллельных иерархии классов продуктов;
- экземпляры класса могут находиться в одном из нескольких различных состояний. Может оказаться удобнее установить соответствующее число прототипов и клонировать их, а не инстанцировать каждый раз класс вручную в подходящем состоянии.



5. **Адаптер** (*Adapter*) — структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс

   Плюсы

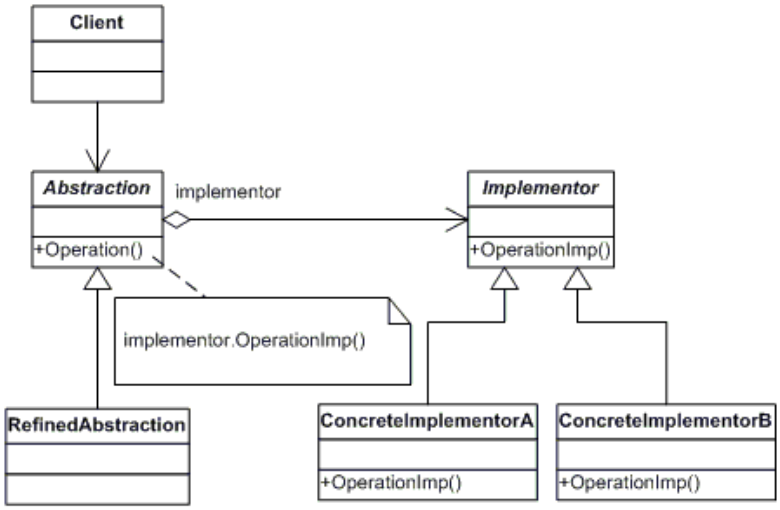
   - инкапсуляция реализации внешних классов (компонентов, библиотек), система становится независимой от интерфейса внешних классов;
   - переход на использование других внешних классов не требует переделки самой системы, достаточно реализовать один класс



6. **Мост**(*Bridge*) — шаблон проектирования, используемый в проектировании программного обеспечения чтобы «разделять абстракцию и реализацию так, чтобы они могли изменяться независимо». Шаблон bridge (от англ. — мост) использует инкапсуляцию, агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами.

   Цель

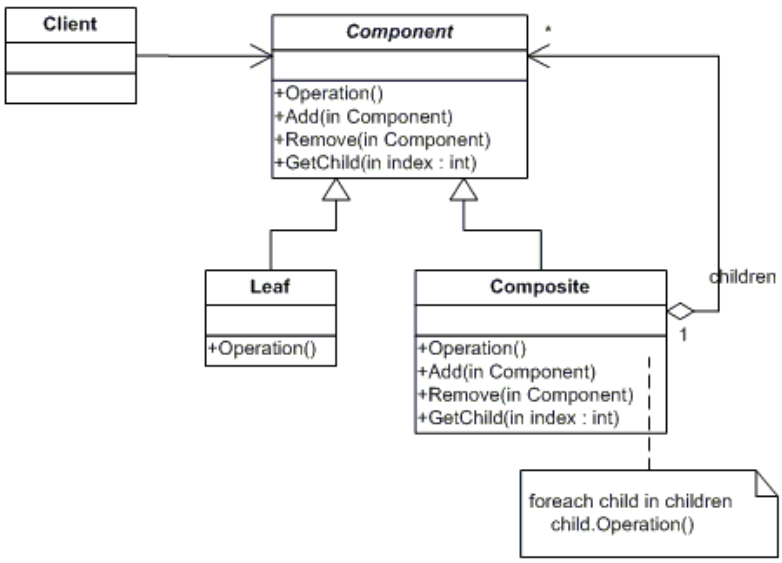
   При частом изменении класса, преимущества объектно-ориентированного подхода становятся очень полезными, позволяя делать изменения в программе, обладая минимальными сведениями о реализации программы. Шаблон bridge является полезным там, где не только сам класс часто меняется, но и то, что класс делает.



7. **Компоновщик** (англ. *Composite pattern*) — шаблон проектирования, объединяет объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково.

   Цель

   Паттерн определяет иерархию классов, которые состоят из примитивных и сложных объектов, упрощает архитектуру клиента, делает процесс добавления новых видов объекта более простым.



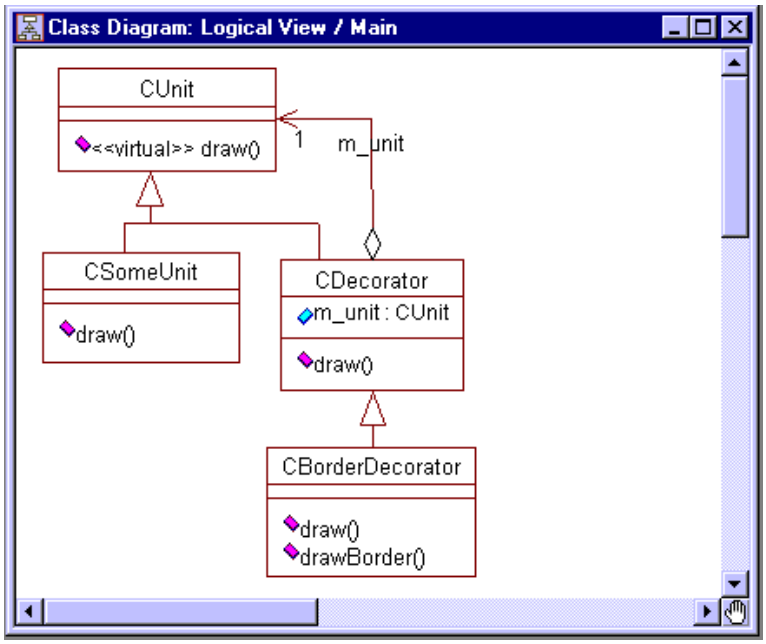
8. **Декоратор**(*Decorator*) — структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Шаблон Декоратор предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности.

   Цель

   Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанциировать. Фабричный метод позволяет классу делегировать создание подклассам.

   Плюсы

- нет необходимости создавать подклассы для расширения функциональности объекта;
- возможность динамически подключать новую функциональность до или после основной функциональности объекта ConcreteComponent.



9. **Одиночка** (англ. *Singleton*) — порождающий шаблон проектирования.

Цель

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.

Плюсы

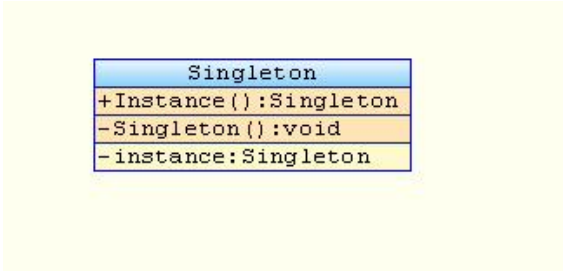
- контролируемый доступ к единственному экземпляру;
- уменьшение числа имён;
- допускает уточнение операций и представления;
- допускает переменное число экземпляров;
- большая гибкость, чем у операций класса.

Минусы

- Глобальные объекты могут быть вредны для объектного программирования, в некоторых случаях приводя к созданию немасштабируемого проекта.

Применимость

- должен быть ровно один экземпляр некоторого класса, легко доступный всем клиентам;
- единственный экземпляр должен расширяться путем порождения подклассов, и клиентам нужно иметь возможность работать с расширенным экземпляром без модификации своего кода класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и планируется локализовать знание о том, какой класс принимает эти обязанности на себя.



Задание

Преобразовать разработанные в предыдущей лабораторной работе UML-диаграммы информационной системы таким образом, чтобы в диаграммах классов присутствовали следующие 6 паттернов проектирования:
- для порождающих: Фабрика, Строитель
- для структурных Фасад, Компоновщик
- для паттернов поведения Шаблонный метод, Посетитель.

Самостоятельно продумать какие еще паттерны целесообразны в Вашей информационной системе, внести необходимые изменения в набор UML-диаграмм.

На хорошую оценку разработать программный каркас приложения (допускается применение средств автоматического формирования каркаса, типа IBM Rational Rose).

На отличную оценку реализовать часть функциональности.

Для сдачи лабораторной работы подготовить модифицированные UML-диаграммы, список элементов, формирующих вышеуказанные паттерны, а также описание каркаса приложения (не менее одной страницы А4).