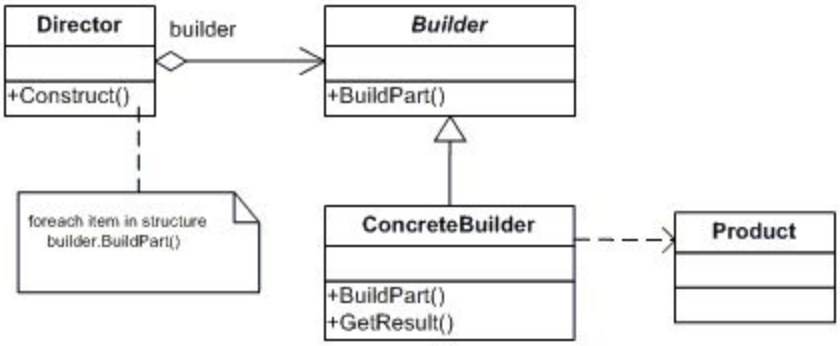
Порождающие шаблоны

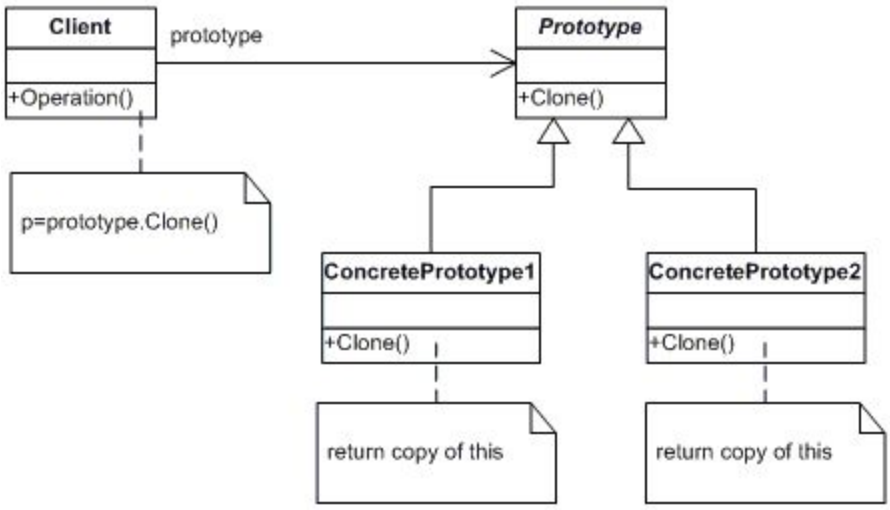
Abstract Factory (абстрактная фабрика) (93)
Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов. В каждом семействе реализуются одни и те же продукты, но по-своему.

Builder (строитель) (103)
Отделяет конструирование сложного объекта от его представления, позволяя использовать один и тот же процесс конструирования для создания различных представлений.
Применение: объект Maze, Pizza
Отличия: абстрактная фабрика сразу выдаёт готовый объект, а строитель собирает объект постепенно и потом возвращает его



Factory Method (фабричный метод) (111)
Определяет интерфейс для создания объектов, при этом выбранный класс инстанцируется подклассами. Каждый реализатор фабричного метода создаёт свой тип продукта

Prototype (прототип) (121)
Описывает виды создаваемых объектов с помощью прототипа и создает новые объекты путем его копирования.
Использование: избежать дополнительных усилий по созданию объекта стандартным путем. Может оказаться удобнее установить соответствующее число прототипов и клонировать их, а не инстанцировать каждый раз класс вручную в подходящем состоянии. избежать наследования создателя объекта (object creator) в клиентском приложении, как это делает паттерн abstract factory.

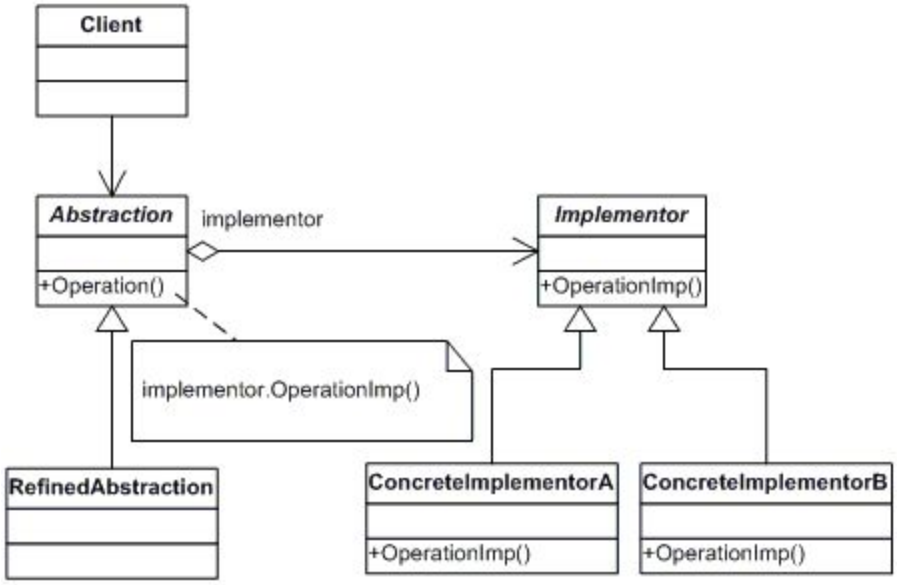


Singleton (одиночка) (130)
Гарантирует, что некоторый класс может иметь только один экземпляр, и предоставляет глобальную точку доступа к нему.

Структурные шаблоны

Adapter (адаптер) (141)
Преобразует интерфейс класса в некоторый другой интерфейс, ожидаемый клиентами. Обеспечивает совместную работу классов, которая была бы невозможна без данного паттерна из-за несовместимости интерфейсов.

Bridge (мост) (152)
Отделяет абстракцию от реализации, благодаря чему появляется возможность независимо изменять то и другое.
Пример: кроссплатформенный класс Window



Composite (компоновщик) (162)
Группирует объекты в древовидные структуры для представления иерархий типа «часть-целое». Позволяет клиентам работать с единичными объектами так же, как с группами объектов.
Применение: структура лист-ветка

Decorator (декоратор) (173)
Динамически возлагает на объект новые функции. Декораторы применяются для расширения имеющейся функциональности и являются гибкой альтернативой порождению подклассов.

Facade (фасад) (183)
Предоставляет унифицированный интерфейс к множеству интерфейсов в некоторой подсистеме. Определяет интерфейс более высокого уровня, облегчающий работу с подсистемой.

Flyweight (приспособленец) (191)
Использует разделение для эффективной поддержки большого числа мелких объектов.

Proxy (заместитель) (203)
Подменяет другой объект для контроля доступа к нему.

Поведенческие шаблоны

Chain of Responsibility (цепочка обязанностей) (217)

Можно избежать жесткой зависимости отправителя запроса от его получателя, при этом запросом начинает обрабатываться один из нескольких объектов. Объекты-получатели связываются в цепочку, и запрос передается по цепочке, пока какой-то объект его не обработает.

Command (команда) (226)
Инкапсулирует запрос в виде объекта, позволяя тем самым параметризовывать клиентов типом запроса, устанавливать очередность запросов, протоколировать их и поддерживать отмену выполнения операций.
Пример: команды меню

Interpreter (интерпретатор) (236)
Для заданного языка определяет представление его грамматики, а также интерпретатор предложений языка, использующий это представление.

Iterator (итератор) (173)
Дает возможность последовательно обойти все элементы составного объекта, не раскрывая его внутреннего представления.

Mediator (посредник) (263)
Определяет объект, в котором инкапсулировано знание о том, как взаимодействуют объекты из некоторого множества. Способствует уменьшению числа связей между объектами, позволяя им работать без явных ссылок друг на друга. Это, в свою очередь, дает возможность независимо изменять схему взаимодействия.

Memento (хранитель) (272)
Позволяет, не нарушая инкапсуляции, получить и сохранить во внешней памяти внутреннее состояние объекта, чтобы позже объект можно было восстановить точно в таком же состоянии.

Observer (наблюдатель) (281)
Определяет между объектами зависимость типа один-ко-многим, так что при изменении состоянии одного объекта все зависящие от него получают извещение и автоматически обновляются.
Пример: обработка событий и сигналов

State (состояние)(291)
Позволяет объекту варьировать свое поведение при изменении внутреннего состояния. При этом создается впечатление, что поменялся класс объекта.

Strategy (стратегия) (300)
Определяет семейство алгоритмов, инкапсулируя их все и позволяя под-

ставлять один вместо другого. Можно менять алгоритм независимо от клиента, который им пользуется.

Template Method (шаблонный метод) (309)
Определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.

Visitor (посетитель) (314)
Представляет операцию, которую надо выполнить над элементами объекта. Позволяет определить новую операцию, не меняя классы элементов, к которым он применяется.

Правила проектирования:

1. Программируйте в соответствии с интерфейсом, а не с реализацией.
2. Предпочитайте композицию наследованию.

Композиция и наследование
Композиция - объединение объектов с чётко определёнными интерфейсами. Чёрный ящик, поскольку детали внутреннего устройства объектов остаются скрытыми.

Агрегирование подразумевает, что один объект владеет другим или несёт за него ответственность
Осведомлённость - это когда объекту известно о другом объекте, когда объект использует другой объект, но не несёт за него ответственности