****

**ПРЕДМЕТ:**

**“Проектирование программного обеспечение”**

**САМОСТОЯТЕЛЬНАЯ РАБОТА**

**Тема:** Шаблоны дизайна

**Выполнил: студент Абдурахмонов Самандар группы 655-20**

**«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.**

**Принял(-а): преподаватель. М.Содикова**

**Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.**

## Аннотация: Шаблоны дизайна— это проверенные и готовые к использованию решения часто возникающих в повседневном программировании задач.

**Ключевые слова:** Шаблон; дизайн; паттерн; идиомы; архитектурные шаблоны; алгоритмы; Кристофер Александр; ADT; Кент Бэк; Эрих Гамма; Fundamental, Порождающие(Creational), Structural, Поведенческие, Частные и Прочие типы шаблонов; и другие..

## План:

Введение(Определения)

* 1 История
* 2 Польза
* 3 Критика
* 4 Типы шаблонов проектирования
  + 4.1 Основные
  + 4.2 Частные
    - 4.2.1 Шаблоны параллельного программирования (Concurrency)
    - 4.2.2 MVC
    - 4.2.3 Enterprise
    - 4.2.4 Прочие
  + 4.3 Другие типы шаблонов

5 Заключение

Примечания  
Литература

## Введение

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

**Шаблон** – не законченный образец, не код; это пример решения задачи, который можно использовать в различных ситуациях.

ОО шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

**Идиомы** – «низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования. Они не универсальные.

**Архитектурные шаблоны** – «наивысший» уровень, охватывают архитектуру всей программной системы.

**Алгоритмы** – шаблоны вычисления, так как решают вычислительные задачи.

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-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются **идиомами**. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

На наивысшем уровне существуют **архитектурные шаблоны**, они охватывают собой архитектуру всей программной системы.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

## 1. История

В 1970-е годы архитектор Кристофер Александр составил набор шаблонов проектирования. В области архитектуры эта идея не получила такого развития, как позже в области программной разработки.

В 1987 году Кент Бэк (Kent Beck) и Вард Каннигем (Ward Cunningham) взяли идеи Александра и разработали шаблоны применительно к разработке программного обеспечения для разработки графических оболочек на языке Smalltalk.

В 1988 году Эрих Гамма (Erich Gamma) начал писать докторскую диссертацию при цюрихском университете об общей переносимости этой методики на разработку программ.

В 1989—1991 годах Джеймс Коплин (James Coplien) трудился над разработкой идиом для программирования на C++ и опубликовал в 1991 году книгу Advanced C++ Idioms.

В этом же году Эрих Гамма заканчивает свою докторскую диссертацию и переезжает в США, где в сотрудничестве с Ричардом Хелмом (Richard Helm), Ральфом Джонсоном (Ralph Johnson) и Джоном Влиссидсом (John Vlissides) публикует книгу Design Patterns — Elements of Reusable Object-Oriented Software. В этой книге описаны 23 шаблона проектирования. Также команда авторов этой книги известна общественности под названием «Банда четырёх» (англ. *Gang of Four*, часто сокращается до *GoF*). Именно эта книга стала причиной роста популярности шаблонов проектирования.

## 2. Польза

Главная польза каждого отдельного шаблона состоит в том, что он описывает решение целого класса абстрактных проблем. Также тот факт, что каждый шаблон имеет свое имя, облегчает дискуссию об абстрактных структурах данных (ADT) между разработчиками, так как они могут ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация терминологии, названий модулей и элементов проекта.

Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова.

## 3. Критика

Иногда шаблоны консервируют громоздкую и малоэффективную систему понятий, разработанную узкой группой. Когда количество шаблонов возрастает, превышая критическую сложность, исполнители начинают игнорировать шаблоны и всю систему, с ними связанную. Нередко шаблонами заменяется отсутствие или недостаточность документации в сложной программной среде.

Есть мнение, что слепое применение шаблонов из справочника, без осмысления причин и предпосылок выделения каждого отдельного шаблона, замедляет профессиональный рост программиста, так как подменяет творческую работу механической подстановкой шаблонов. Люди, придерживающиеся данного мнения, считают, что знакомиться со списками шаблонов необходимо тогда, когда программист «дорос» до них в профессиональном плане — и не раньше. Хороший критерий нужной степени профессионализма — выделение шаблонов самостоятельно, на основании собственного опыта. При этом, разумеется, знакомство с теорией, связанной с шаблонами, полезно на любом уровне профессионализма и направляет развитие программиста в правильную сторону. Сомнению подвергается только использование шаблонов «по справочнику».

Шаблоны могут пропагандировать плохие стили разработки приложений, и зачастую слепо применяются. Для преодоления этих недостатков используется рефакторинг.

## 4. Типы шаблонов проектирования

### 4.1. Основные

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Оригинальное название** | **Описание** | **Описан в *Design Patterns*** |
| **Основные шаблоны (Fundamental)** | | | |
| Шаблон делегирования | Delegation pattern | Объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту | Н/Д |
| Шаблон функционального дизайна | Functional design | Гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы | Н/Д |
| Неизменяемый объект | Immutable | Объект, который не может быть изменён после своего создания | Н/Д |
| Интерфейс | Interface | Общий метод для структурирования компьютерных программ для того, чтобы их было проще понять | Н/Д |
| Marker interface | Marker interface |  | Н/Д |
| Property Container | Property Container |  | Н/Д |
| Event Channel | Event Channel |  | Н/Д |
| **Порождающие шаблоны (Creational)** — шаблоны проектирования, которые абстрагируют процесс инстанцирования. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять инстанцируемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту. | | | |
| Абстрактная фабрика | Abstract factory | Класс, который представляет собой интерфейс для создания компонентов системы. | Да |
| Строитель | Builder | Класс, который представляет собой интерфейс для создания сложного объекта | Да |
| Фабричный метод | Factory method | Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанциировать | Да |
| Отложенная инициализация | Lazy initialization | Объект, инициализируемый во время первого обращения к нему | Нет |
| Пул одиночек | Multiton | Гарантирует, что класс имеет поименованные экземпляры объекта и обеспечивает глобальную точку доступа к ним | Нет |
| Объектный пул | Object pool | Класс, который представляет собой интерфейс для работы с набором инициализированных и готовых к использованию объектов | Нет |
| Прототип | Prototype | Определяет интерфейс создания объекта через клонирование другого объекта вместо создания через конструктор | Да |
| Получение ресурса есть инициализация | Resource acquisition is initialization (RAII) | Получение некоторого ресурса совмещается с инициализацией, а освобождение — с уничтожением объекта | Нет |
| Одиночка | Singleton | Класс, который может иметь только один экземпляр. | Да |
| **Структурные шаблоны (Structural)** определяют различные сложные структуры, которые изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу. | | | |
| Адаптер | Adapter / Wrapper | Объект, обеспечивающий взаимодействие двух других объектов, один из которых использует, а другой предоставляет несовместимый с первым интерфейс | Да |
| Мост | Bridge | Структура, позволяющая изменять интерфейс обращения и интерфейс реализации класса независимо | Да |
| Компоновщик | Composite | Объект, который объединяет в себе объекты, подобные ему самому | Да |
| Декоратор или Wrapper/Обёртка | Decorator | Класс, расширяющий функциональность другого класса, без использования наследования | Да |
| Фасад | Facade | Объект, который абстрагирует работу с несколькими классами, объединяя их в единое целое | Да |
| Единая точка входа | Front Controller | Обеспечивает унифицированный интерфейс для интерфейсов в подсистеме. Front Controller определяет высокоуровневый интерфейс, упрощающий использование подсистемы | Нет |
| Приспособленец | Flyweight | Это объект, представляющий себя как уникальный экземпляр в разных местах программы, но по факту не являющийся таковым | Да |
| Заместитель | Proxy | Объект, который является посредником между двумя другими объектами, и который реализовывает/ограничивает доступ к объекту, к которому обращаются через него | Да |
| **Поведенческие шаблоны (Behavioral)** определяют взаимодействие между объектами, увеличивая таким образом его гибкость. | | | |
| Цепочка ответственности | Chain of responsibility | Предназначен для организации в системе уровней ответственности | Да |
| Команда, Action, Transaction | Command | Представляет действие. Объект команды заключает в себе само действие и его параметры | Да |
| Интерпретатор | Interpreter | Решает часто встречающуюся, но подверженную изменениям, задачу | Да |
| Итератор, Cursor | Iterator | Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из объектов, входящий в состав агрегации | Да |
| Посредник | Mediator | Обеспечивает взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга | Да |
| Хранитель, Token | Memento | Позволяет не нарушая инкапсуляцию зафиксировать и сохранить внутреннее состояния объекта так, чтобы позднее восстановить его в этом состоянии | Да |
|  | Null object | Предотвращает нулевые указатели, предоставляя объект «по умолчанию» | Нет |
| Наблюдатель, Dependents, Publish-Subscribe, Listener | Observer или Publish/subscribe | Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии | Да |
| Слуга | Servant | Используется для обеспечения общей функциональности группе классов | Нет |
|  | Specification | Служит для связывания биснес-логики | Нет |
| Состояние, Objects for States | State | Используется в тех случаях, когда во время выполнения программы объект должен менять свое поведение в зависимости от своего состояния | Да |
| Стратегия | Strategy | Предназначен для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости | Да |
| Шаблонный метод | Template method | Определяет основу алгоритма и позволяет наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом. | Да |
| Посетитель | Visitor | Описывает операцию, которая выполняется над объектами других классов. При изменении класса Visitor нет необходимости изменять обслуживаемые классы. | Да |
| Simple Policy |  |  | Нет |
| Event listener |  |  | Нет |
| Single-serving visitor | Single-serving visitor | Оптимизирует реализацию шаблона посетитель, который инициализируется, единожды используется, и затем удаляется | Нет |
| Hierarchical visitor | Hierarchical visitor | Предоставляет способ обхода всех вершин иерархической структуры данных (напр. древовидной) | Нет |

### 4.2. Частные

#### 4.2.1. Шаблоны параллельного программирования (Concurrency)

Используются для более эффективного написания многопоточных программ, и предоставляет готовые решения проблем синхронизации.

|  |  |  |
| --- | --- | --- |
| **Название** | **Оригинальное название** | **Описание** |
| Active Object | Active object | Служит для отделения потока выполнения метода от потока, в котором он был вызван. Использует шаблоны асинхронный вызов методов и планировщик |
| Balking | Balking | Служит для выполнения действия над объектом только тогда, когда тот находится в корректном состоянии |
|  | Binding Properties | Комбинирует несколько наблюдателей для обеспечения синхронизации свойств в различных объектах[1] |
|  | Messaging design pattern (MDP) | Позволяет компонентам и приложениям обмениваться информацией (сообщениями) |
| Блокировка с двойной проверкой | Double-checked locking | Предназначен для уменьшения накладных расходов, связанных с получением блокировки |
|  | Event-based asynchronous | Addresses problems with the Asynchronous pattern that occur in multithreaded programs.[2] |
| Guarded suspension | Guarded suspension | Используется для блокировки выполнения действия над объектом только тогда, когда тот находится в корректном состоянии |
| Half-Sync/Half-Async |  |  |
| Leaders/followers |  |  |
|  | Lock | Один поток блокирует ресурс для предотвращения доступа или изменения его другими потоками[3] |
| Monitor Object | Monitor object | Объект, предназначенный для безопасного использования более чем одним потоком |
| Reactor | Reactor | Предназначен для синхронной передачи запросов сервису от одного или нескольких источников |
| Read write lock | Read-write lock | Позволяет нескольким потокам одновременно считывать информацию из общего хранилища, но позволяя только одному потоку в момент времени изменять её |
| Планировщик | Scheduler | Обеспечивает механизм реализации политики планирования, но при этом не зависящих ни от одной конкретной политики |
| Thread pool | Thread pool | Предоставляет пул потоков для обработки заданий, представленных обычно в виде очереди |
| Thread-Specific Storage | Thread-specific storage | Служит для предоставления различных глобальных переменных для разных потоков |
| Однопоточное выполнение | Single Thread Execution | Препятствует конкурентному вызову метода, тем самым запрещая параллельное выполнение этого метода |
| Кооперативный паттерн | Cooperative pattern |  |

#### 4.2.2. MVC

* Model-View-Controller (MVC) Модель-представление-контроллер
* Model-View-Presenter
* Model-View-View Model
* Presentation-Abstraction-Control

#### 4.2.3. Enterprise

* Business Delegate
* Composite Entity/Составная Сущность
* Composite View
* DAO (Data Access Object) Объект Доступа к Данным
* Dispatcher View
* Front Controller
* Intercepting Filter
* Registry
* Service Activator
* Service Locator/Локатор Службы
* Service to Worker
* Session Facade/Фасад Сессии
* Transfer Object Assembler
* Transfer Object/Объект Перемещения
* Value List Handler/Обработчик Списка Значений
* View Helper
* Unit of Work

#### 4.2.4. Прочие

* Repository/Хранилище

### 4.3. Другие типы шаблонов

Также на сегодняшний день существует ряд других шаблонов:

* Carrier Rider Mapper описывают предоставление доступа к хранимой информации
* Аналитические шаблоны описывают основной подход для составления требований для программного обеспечения (requirement analysis) до начала самого процесса программной разработки
* Коммуникационные шаблоны описывают процесс общения между отдельными участниками/сотрудниками организации
* Организационные шаблоны описывают организационную иерархию предприятия/фирмы
* Анти-паттерны (Anti-Design-Patterns) описывают, как не следует поступать при разработке программ, показывая характерные ошибки в дизайне и в реализации

1. **Заключение**

* Имеется ещё много шаблонов проектирования, которые полезно знать. Если есть интерес прочитать о других шаблонах проектирования, то страница Википедии [Design Patterns](https://en.wikipedia.org/wiki/Software_design_pattern) (Шаблоны проектирования).  
    
  И последнее: используя эти шаблоны проектирования, всегда проверяйте, что вы пытаетесь решить правильно поставленную задачу. Как я упоминал ранее, эти шаблоны проектирования необходимо использовать очень осторожно: они — при использовании в ненадлежащем контексте — могут ухудшить ситуацию, но при правильном использовании они просто жизненно необходимы.

## Примечания

1. Binding Properties - c2.com/cgi/wiki?BindingProperties
2. Event-based Asynchronous Pattern // Professional C# 2008. — Wiley, 2008. — P. 570–571. — ISBN 0470191376
3. Lock Pattern - c2.com/cgi/wiki?LockPattern

## Литература

* *Мартин Фаулер* Шаблоны корпоративных приложений = Patterns of Enterprise Application Architecture (Addison-Wesley Signature Series). — М.: «Вильямс», 2009. — С. 544. — ISBN 0-321-12742-0
* *Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес* Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software. — СПб: «Питер», 2007. — С. 366. — ISBN 978-5-469-01136-1 (также ISBN 5-272-00355-1)
* *Марк Гранд* Шаблоны проектирования в JAVA. Каталог популярных шаблонов проектирования, проиллюстрированных при помощи UML = Patterns in Java, Volume 1. A Catalog of Reusable Design Patterns Illustrated with UML. — М.: «Новое знание», 2004. — С. 560. — ISBN 5-94735-047-5
* *Крэг Ларман* Применение UML 2.0 и шаблонов проектирования = Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development. — М.: «Вильямс», 2006. — С. 736. — ISBN 0-13-148906-2
* *Джошуа Кериевски* Рефакторинг с использованием шаблонов (паттернов проектирования) = Refactoring to Patterns (Addison-Wesley Signature Series). — М.: «Вильямс», 2006. — С. 400. — ISBN 0-321-21335-1
* *Скотт В. Эмблер, Прамодкумар Дж. Садаладж* Рефакторинг баз данных: эволюционное проектирование = Refactoring Databases: Evolutionary Database Design (Addison-Wesley Signature Series). — М.: «Вильямс», 2007. — С. 368. — ISBN 0-321-29353-3