|  |  |  |
| --- | --- | --- |
| Дата | Содержание выполняемой работы | Подпись руководите ля |
| 18.05.2022 | **Практическая работа №19**  **Тема:** Шаблоны проектирования.  **Наименование работы:** Использование шаблонов при разработке программного продукта.  **Цель:** Ознакомиться с понятием шаблоны проектирования; изучить их классификации, рассмотреть примеры использования шаблонов проектирования в конкретной ситуации.  **Задание 1.**  Изучил теоретический материал, представленный в документе «Шаблоны проектирования.pdf» и законспектировал в дневнике-отчете определения, классификацию паттернов, каждому виду дал характеристику.  Архитектурные паттерны - множество предварительно определенных подсистем со спецификацией их ответственности, правил и базовых принципов установления отношений между ними.  Архитектурные паттерны предназначены для спецификации фундаментальных схем структуризации программных систем.  Паттерны проектирования - специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними.  Паттерны проектирования описывают общую структуру взаимодействия элементов программной системы, которые реализуют исходную проблему проектирования в конкретном контексте.  Паттерны анализа - специальные схемы для представления общей организации процесса моделирования.  Паттерны анализа относятся к одной или нескольким предметным областям и описываются в терминах предметной области.  Паттерны тестирования - специальные схемы для представления общей организации процесса тестирования программных систем.  Паттерны реализации - совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода.  Эта категория паттернов делится на следующие подкатегории: паттерны организации программного кода, паттерны оптимизации программного кода, паттерны устойчивости кода, паттерны разработки графического интерфейса пользователя и др.  **Задание 2.**  Изучил статью «Практика применения паттернов проектирования.pdf».  **Задание 3.**  Нашел информацию в интернете и законспектировал в дневнике-отчете описание 3 шаблонов (паттернов) проектирования.   * Паттерн «Прототип»   Прототип — это порождающий паттерн проектирования, который позволяет копировать объекты, не вдаваясь в подробности их реализации.  Паттерн Прототип поручает создание копий самим копируемым объектам. Он вводит общий интерфейс для всех объектов, поддерживающих клонирование. Это позволяет копировать объекты, не привязываясь к их конкретным классам. Обычно такой интерфейс имеет всего один метод clone. Объект, который копируют, называется прототипом (откуда и название паттерна). Когда объекты программы содержат сотни полей и тысячи возможных конфигураций, прототипы могут служить своеобразной альтернативой созданию подклассов.  В этом случае все возможные прототипы заготавливаются и настраиваются на этапе инициализации программы. Потом, когда программе нужен новый объект, она создаёт копию из приготовленного прототипа.  Структура паттерна:  1. Интерфейс прототипов описывает операции клонирования. В большинстве случаев — это единственный метод clone.  2. Конкретный прототип реализует операцию клонирования самого себя. Помимо банального копирования значений всех полей, здесь могут быть спрятаны различные сложности, о которых не нужно знать клиенту. Например, клонирование связанных объектов, распутывание рекурсивных зависимостей и прочее.  3. Клиент создаёт копию объекта, обращаясь к нему через общий интерфейс прототипов.   * Паттерн «Снимок»   Снимок — это поведенческий паттерн проектирования, который позволяет сохранять и восстанавливать прошлые состояния объектов, не раскрывая подробностей их реализации.  Паттерн Снимок поручает создание копии состояния объекта самому объекту, который этим состоянием владеет. Вместо того, чтобы делать снимок «извне», наш редактор сам сделает копию своих полей, ведь ему доступны все поля, даже приватные.  Паттерн предлагает держать копию состояния в специальном объекте-снимке с ограниченным интерфейсом, позволяющим, например, узнать дату изготовления или название снимка. Но, с другой стороны, снимок должен быть открыт для своего создателя, позволяя прочесть и восстановить его внутреннее состояние.  Такая схема позволяет создателям производить снимки и отдавать их для хранения другим объектам, называемым опекунами. Опекунам будет доступен только ограниченный интерфейс снимка, поэтому они никак не смогут повлиять на «внутренности» самого снимка. В нужный момент опекун может попросить создателя восстановить своё состояние, передав ему соответствующий снимок.  Структура паттерна:  1. Создатель может производить снимки своего состояния, а также воспроизводить прошлое состояние, если подать в него готовый снимок.  2. Снимок — это простой объект данных, содержащий состояние создателя. Надёжнее всего сделать объекты снимков неизменяемыми, передавая в них состояние только через конструктор.  3. Опекун должен знать, когда делать снимок создателя и когда его нужно восстанавливать.  4. Опекун может хранить историю прошлых состояний создателя в виде стека из снимков. Когда понадобится отменить выполненную операцию, он возьмёт «верхний» снимок из стека и передаст его создателю для восстановления.  5. В данной реализации снимок — это внутренний класс по отношению к классу создателя. Именно поэтому он имеет полный доступ к полям и методам создателя, даже приватным.   * Паттерн «Одиночка»   Одиночка — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.  Все реализации одиночки сводятся к тому, чтобы скрыть конструктор по умолчанию и создать публичный статический метод, который и будет контролировать жизненный цикл объекта-одиночки.  Если у вас есть доступ к классу одиночки, значит, будет доступ и к этому статическому методу. Из какой точки кода вы бы его ни вызвали, он всегда будет отдавать один и тот же объект.  Структура паттерна:  1. Одиночка определяет статический метод getInstance, который возвращает единственный экземпляр своего класса.  Конструктор одиночки должен быть скрыт от клиентов. Вызов метода getInstance должен стать единственным способом получить объект этого класса. |  |