*Занятие № 19*

*Номер учебной группы:* П-16

*Фамилия, инициалы учащегося:* Язубец Е.В.

*Дата выполнения работы:* 29.11.2022

*Тема работы:* «Использование шаблонов при разработке программного продукта»

*Результат выполнения работы*

**Задание 1**

Изучил теоретический материал по теме «Шаблоны проектирования».

**Задание 2**

Разработал шаблоны проектирования и реализации для будущего использования при разработке ПО.

**Цепочка обязанностей** — это поведенческий паттерн проектирования, который позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи.

// Интерфейс обработчиков.

**interface** **ComponentWithContextualHelp** **is**

**method** showHelp()

// Базовый класс простых компонентов.

**abstract class** **Component** **implements** ComponentWithContextualHelp **is**

**field** tooltipText: string

// Контейнер, содержащий компонент, служит в качестве

// следующего звена цепочки.

**protected** **field** container: Container

// Базовое поведение компонента заключается в том, чтобы

// показать всплывающую подсказку, если для неё задан текст.

// В обратном случае — перенаправить запрос своему

// контейнеру, если тот существует.

**method** showHelp() **is**

**if** (tooltipText != **null**)

// Показать подсказку.

**else**

container.showHelp()

// Контейнеры могут включать в себя как простые компоненты, так

// и другие контейнеры. Здесь формируются связи цепочки. Класс

// контейнера унаследует метод showHelp от своего родителя —

// базового компонента.

**abstract class** **Container** **extends** Component **is**

**protected** **field** children: array of Component

**method** add(child) **is**

children.add(child)

child.container = **this**

// Большинство примитивных компонентов устроит базовое поведение

// показа помощи через подсказку, которое они унаследуют из

// класса Component.

**class** **Button** **extends** Component **is**

// ...

// Но сложные компоненты могут переопределять метод показа

// помощи по-своему. Но и в этом случае они всегда могут

// вернуться к базовой реализации, вызвав метод родителя.

**class** **Panel** **extends** Container **is**

**field** modalHelpText: string

**method** showHelp() **is**

**if** (modalHelpText != **null**)

// Показать модальное окно с помощью.

**else**

**super**.showHelp()

// ...то же, что и выше...

**class** **Dialog** **extends** Container **is**

**field** wikiPageURL: string

**method** showHelp() **is**

**if** (wikiPageURL != **null**)

// Открыть страницу Wiki в браузере.

**else**

**super**.showHelp()

// Клиентский код.

**class** **Application** **is**

// Каждое приложение конфигурирует цепочку по-своему.

**method** createUI() **is**

dialog = **new** Dialog("Budget Reports")

dialog.wikiPageURL = "http://..."

panel = **new** Panel(0, 0, 400, 800)

panel.modalHelpText = "This panel does..."

ok = **new** Button(250, 760, 50, 20, "OK")

ok.tooltipText = "This is an OK button that..."

cancel = **new** Button(320, 760, 50, 20, "Cancel")

// ...

panel.add(ok)

panel.add(cancel)

dialog.add(panel)

// Представьте, что здесь произойдёт.

**method** onF1KeyPress() **is**

component = **this**.getComponentAtMouseCoords()

component.showHelp()

Данный шаблон частично реализуется в моей программе.

**Состояние** — это поведенческий паттерн проектирования, который позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.

// Общий интерфейс всех состояний.

**abstract class** **State** **is**

**protected** **field** player: AudioPlayer

// Контекст передаёт себя в конструктор состояния, чтобы

// состояние могло обращаться к его данным и методам в

// будущем, если потребуется.

**constructor** State(player) **is**

**this**.player = player

**abstract** **method** clickLock()

**abstract** **method** clickPlay()

**abstract** **method** clickNext()

**abstract** **method** clickPrevious()

// Конкретные состояния реализуют методы абстрактного состояния

// по-своему.

**class** **LockedState** **extends** State **is**

// При разблокировке проигрователя с заблокированными

// клавишами он может принять одно из двух состояний.

**method** clickLock() **is**

**if** (player.playing)

player.changeState(**new** PlayingState(player))

**else**

player.changeState(**new** ReadyState(player))

**method** clickPlay() **is**

// Ничего не делать.

**method** clickNext() **is**

// Ничего не делать.

**method** clickPrevious() **is**

// Ничего не делать.

// Конкретные состояния сами могут переводить контекст в другое

// состояние.

**class** **ReadyState** **extends** State **is**

**method** clickLock() **is**

player.changeState(**new** LockedState(player))

**method** clickPlay() **is**

player.startPlayback()

player.changeState(**new** PlayingState(player))

**method** clickNext() **is**

player.nextSong()

**method** clickPrevious() **is**

player.previousSong()

**class** **PlayingState** **extends** State **is**

**method** clickLock() **is**

player.changeState(**new** LockedState(player))

**method** clickPlay() **is**

player.stopPlayback()

player.changeState(**new** ReadyState(player))

**method** clickNext() **is**

**if** (event.doubleclick)

player.nextSong()

**else**

player.fastForward(5)

**method** clickPrevious() **is**

**if** (event.doubleclick)

player.previous()

**else**

player.rewind(5)

// Проигрыватель выступает в роли контекста.

**class** **AudioPlayer** **is**

**field** state: State

**field** UI, volume, playlist, currentSong

**constructor** AudioPlayer() **is**

**this**.state = **new** ReadyState(**this**)

// Контекст заставляет состояние реагировать на

// пользовательский ввод вместо себя. Реакция может быть

// разной, в зависимости от того, какое состояние сейчас

// активно.

UI = **new** UserInterface()

UI.lockButton.onClick(**this**.clickLock)

UI.playButton.onClick(**this**.clickPlay)

UI.nextButton.onClick(**this**.clickNext)

UI.prevButton.onClick(**this**.clickPrevious)

// Другие объекты тоже должны иметь возможность заменять

// состояние проигрывателя.

**method** changeState(state: State) **is**

**this**.state = state

// Методы UI будут делегировать работу активному состоянию.

**method** clickLock() **is**

state.clickLock()

**method** clickPlay() **is**

state.clickPlay()

**method** clickNext() **is**

state.clickNext()

**method** clickPrevious() **is**

state.clickPrevious()

// Сервисные методы контекста, вызываемые состояниями.

**method** startPlayback() **is**

// ...

**method** stopPlayback() **is**

// ...

**method** nextSong() **is**

// ...

**method** previousSong() **is**

// ...

**method** fastForward(time) **is**

// ...

**method** rewind(time) **is**

// ...

Данный шаблон в моей программе не реализуется.

**Задание 3**

Ответила на контрольные вопросы.

1. Что вы понимаете под шаблонами проектирования?

**Ответ:** Шаблоны проектирования — это допускающие многократное использование оптимизированные решения проблем программирования, с которыми мы сталкиваемся каждый день. Шаблон проектирования — это не класс или библиотека, которые мы можем просто вставить в нашу систему. Он — много больше. Это — некоторый шаблон, который должен быть реализован в надлежащей ситуации. Он не зависит от языка.

Хороший шаблон проектирования должен быть таким, чтобы его можно было использовать с большинством языков (если не со всеми) в зависимости от характеристик языка. Чрезвычайно важно то, что любой шаблон проектирования необходимо использовать очень осторожно — если он применён в ненадлежащем месте, то его действие может быть разрушительным и породить много проблем для вас. Однако применённый в нужном месте в нужное время он может стать вашим спасителем

1. Перечислите плюсы и минусы применения шаблонов.

**Ответ:** В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению своё имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования.

Хотя легкое изменение кода под известный шаблон может упростить понимание кода, по мнению Стива Макконнелла, с применением шаблонов могут быть связаны две сложности. Во-первых, слепое следование некоторому выбранному шаблону может привести к усложнению программы. Во-вторых, у разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

Многие шаблоны проектирования в объектно-ориентированном проектировании можно рассматривать как идиоматическое воспроизведение элементов функциональных языков. Питер Норвиг утверждает, что 16 из 23 шаблонов, описанных в книге «Банды четырёх», в динамически-типизируемых языках реализуются существенно проще, чем в C++, либо оказываются незаметны.

3. Перечислите типы шаблонов проектирования и реализации.

**Ответ:** Основные типы шаблонов проектирования

Порождающие шаблоны - применяются для генерации объектов. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Объект содержит экземпляр заранее описанного класса.

Структурные шаблоны - изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу.

Поведенческие шаблоны - определяют взаимодействие между объектами, увеличивая его гибкость. Дочерние классы наследуют абстрактный класс, таким образом, конкретный дочерний класс поддерживает необходимые методы, объявленные в родительском классе.

Архитектурные шаблоны - охватывают собой архитектуру всей программной системы.

Шаблоны баз данных - помогаю извлекать данные из баз данных.

Шаблоны параллельного программирования - применяются для написания многопоточных программ.

Анти-паттерны - описывают характерные ошибки при проектировании, в дизайне и в реализации

4. Дайте характеристику шаблону проектирования «Делегирование»

**Ответ:** В разработке программного обеспечения, шаблон делегирования - это объектно-ориентированный шаблон проектирования, который позволяет композиции объекта достичь того же повторного использования кода, что и наследование.

При делегировании объект обрабатывает запрос, делегируя второму d объект (делегат). Делегат - это вспомогательный объект, но с исходным контекстом. При поддержке делегирования на уровне языка это делается неявно, когда selfв делегате ссылается на исходный (отправляющий) объект, а не на делегате (принимающий объект). В шаблоне делегата это вместо этого достигается путем явной передачи исходного объекта делегату в качестве аргумента метода.

5. Дайте характеристику шаблону проектирования «Функциональный дизайн».

**Ответ:** Функциональный дизайн (англ. Functional design) — основной шаблон проектирования, использующийся для упрощения проектирования ПО. Функциональный дизайн гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы. Функционально разработанные модули имеют низкое зацепление.

6. Дайте характеристику шаблону проектирования «Интерфейс».

**Ответ:** Интерфейс (англ. interface) — основной шаблон проектирования, являющийся общим методом для структурирования компьютерных программ для того, чтобы их было проще понять. В общем, интерфейс — это класс, который обеспечивает программисту простой или более программно-специфический способ доступа к другим классам.

Интерфейс может содержать набор объектов и обеспечивать простую, высокоуровневую функциональность для программиста (например, Шаблон Фасад); он может обеспечивать более чистый или более специфический способ использования сложных классов («класс-обёртка»); он может использоваться в качестве «клея» между двумя различными API (Шаблон Адаптер); и для многих других целей.

7. Дайте характеристику шаблону проектирования «Абстрактная фабрика».

**Ответ:** 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.

8. Дайте характеристику шаблону проектирования «Строитель».

**Ответ:** **Строитель (англ. Builder)** — порождающий шаблон проектирования предоставляет способ создания составного объекта.

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

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

9. Дайте характеристику шаблону проектирования «Фабричный метод».

**Ответ:** ***Шаблон «Фабричный метод»*** *является порождающим шаблоном проектирования, который делает именно то, что означает это слово: этот класс действует как фабрика экземпляров объектов.*

Основной целью этого шаблона является вложение порождающей процедуры, которая может свести различные классы в одну функцию. Если фабричному методу обеспечить надлежащий контекст, то он будет в состоянии вернуть правильный объект.

Наилучшей ситуацией для использования шаблона «Фабричный метод» является наличие нескольких различных вариантов одного объекта. Допустим, имеется класс «кнопка»; у этого класса есть различные варианты — например, ImageButton (кнопка изображения), InputButton (кнопка ввода) и FlashButton (флэш-кнопка).

10. Дайте характеристику шаблону проектирования «Прототип».

**Ответ:** Прототип, (англ. Prototype) — порождающий шаблон проектирования. Задаёт виды создаваемых объектов с помощью экземпляра-прототипа и создаёт новые объекты путём копирования этого прототипа. Он позволяет уйти от реализации и позволяет следовать принципу «программирование через интерфейсы». В качестве возвращающего типа указывается интерфейс/абстрактный класс на вершине иерархии, а классы-наследники могут подставить туда наследника, реализующего этот тип.

Проще говоря, это паттерн создания объекта через клонирование другого объекта вместо создания через конструктор.

11. Дайте характеристику шаблону проектирования «Наблюдатель».

**Ответ:** Наблюдатель (англ. Observer) — поведенческий шаблон проектирования. Также известен как «подчинённые» (англ. Dependents). Реализует у класса механизм, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов и тем самым наблюдать за ними.

Классы, на события которых другие классы подписываются, называются субъектами (Subjects), а подписывающиеся классы называются наблюдателями (англ. Observers).

Определяет зависимость типа один ко многим между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.