**Тема: «Паттерны проектирования»**

**Часть 2**

**Реализация паттернов Фабричный метод, Декоратор и Команда**

В практической части вам предстоит реализовать три паттерна:

1. **Фабричный метод** – создание объектов без жесткой привязки к их конкретным классам.
2. **Декоратор** – динамическое расширение функционала объектов.
3. **Команда** – превращение запросов в объекты для управления действиями.

**Задание 1: Реализация паттерна "Фабричный метод"**

**Описание:**  
Фабричный метод применяется для создания объектов без жесткой привязки к их конкретным классам. Это позволяет делегировать создание объектов подклассам, обеспечивая гибкость и расширяемость кода.

**Общие компоненты:**

* **Абстрактный класс или интерфейс продукта** – определяет общий интерфейс создаваемых объектов.
* **Конкретные классы продукта** – реализуют интерфейс продукта.
* **Абстрактный класс фабрики** – определяет метод создания объектов.
* **Конкретные классы фабрики** – реализуют создание конкретных объектов.

**Список задач**

**1. Система отчетов (PDF, Excel, Word)**

**Классы и их взаимодействие:**

* **IReport** – интерфейс отчета с методом Generate().
* **PdfReport, ExcelReport, WordReport** – классы, реализующие IReport.
* **ReportFactory** – абстрактный класс с методом CreateReport().
* **PdfReportFactory, ExcelReportFactory, WordReportFactory** – конкретные фабрики, создающие отчеты.

**2. Виды банковских карт (кредитная, дебетовая, виртуальная)**

**Классы и их взаимодействие:**

* **IBankCard** – интерфейс банковской карты с методом Use().
* **CreditCard, DebitCard, VirtualCard** – классы карт.
* **BankCardFactory** – абстрактная фабрика с CreateCard().
* **CreditCardFactory, DebitCardFactory, VirtualCardFactory** – конкретные фабрики.

**3. Персонажи игры (маг, воин, лучник)**

**Классы и их взаимодействие:**

* **ICharacter** – интерфейс с Attack().
* **Mage, Warrior, Archer** – классы персонажей.
* **CharacterFactory** – абстрактная фабрика.
* **MageFactory, WarriorFactory, ArcherFactory** – конкретные фабрики.

**4. Вид транспорта (машина, автобус, велосипед)**

**Классы и их взаимодействие:**

* **IVehicle** – интерфейс с Move().
* **Car, Bus, Bicycle** – классы транспорта.
* **VehicleFactory** – абстрактная фабрика.
* **CarFactory, BusFactory, BicycleFactory** – конкретные фабрики.

**5. Виды пользователей (админ, модератор, обычный пользователь)**

**Классы и их взаимодействие:**

* **IUser** – интерфейс с GetPermissions().
* **AdminUser, ModeratorUser, RegularUser** – классы пользователей.
* **UserFactory** – абстрактная фабрика.
* **AdminFactory, ModeratorFactory, RegularFactory** – конкретные фабрики.

**6. Оплата (наличные, карта, криптовалюта)**

**Классы и их взаимодействие:**

* **IPayment** – интерфейс с Pay().
* **CashPayment, CardPayment, CryptoPayment** – классы оплаты.
* **PaymentFactory** – абстрактная фабрика.
* **CashFactory, CardFactory, CryptoFactory** – конкретные фабрики.

**7. Мультимедийные файлы (аудио, видео, изображение)**

**Классы и их взаимодействие:**

* **IMediaFile** – интерфейс с Play().
* **AudioFile, VideoFile, ImageFile** – классы файлов.
* **MediaFactory** – абстрактная фабрика.
* **AudioFactory, VideoFactory, ImageFactory** – конкретные фабрики.

**8. Система уведомлений (email, SMS, push)**

**Классы и их взаимодействие:**

* **INotification** – интерфейс с Send().
* **EmailNotification, SMSNotification, PushNotification** – классы уведомлений.
* **NotificationFactory** – абстрактная фабрика.
* **EmailFactory, SMSFactory, PushFactory** – конкретные фабрики.

**9. Виды животных (собака, кошка, птица)**

**Классы и их взаимодействие:**

* **IAnimal** – интерфейс с MakeSound().
* **Dog, Cat, Bird** – классы животных.
* **AnimalFactory** – абстрактная фабрика.
* **DogFactory, CatFactory, BirdFactory** – конкретные фабрики.

**10. Виды брони (легкая, средняя, тяжелая)**

**Классы и их взаимодействие:**

* **IArmor** – интерфейс с GetDefense().
* **LightArmor, MediumArmor, HeavyArmor** – классы брони.
* **ArmorFactory** – абстрактная фабрика.
* **LightArmorFactory, MediumArmorFactory, HeavyArmorFactory** – конкретные фабрики.

**11. Виды подписок (бесплатная, премиум, VIP)**

**Классы и их взаимодействие:**

* **ISubscription** – интерфейс с GetBenefits().
* **FreeSubscription, PremiumSubscription, VIPSubscription** – классы подписок.
* **SubscriptionFactory** – абстрактная фабрика.
* **FreeFactory, PremiumFactory, VIPFactory** – конкретные фабрики.

**12. Электроника (ноутбук, планшет, смартфон)**

**Классы и их взаимодействие:**

* **IElectronicDevice** – интерфейс с TurnOn().
* **Laptop, Tablet, Smartphone** – классы устройств.
* **ElectronicDeviceFactory** – абстрактная фабрика.
* **LaptopFactory, TabletFactory, SmartphoneFactory** – конкретные фабрики.

**13. Оружие (меч, лук, арбалет)**

**Классы и их взаимодействие:**

* **IWeapon** – интерфейс с Attack().
* **Sword, Bow, Crossbow** – классы оружия.
* **WeaponFactory** – абстрактная фабрика.
* **SwordFactory, BowFactory, CrossbowFactory** – конкретные фабрики.

**14. Система бронирования билетов (авиа, поезд, автобус)**

**Классы и их взаимодействие:**

* **ITicket** – интерфейс с Book().
* **PlaneTicket, TrainTicket, BusTicket** – классы билетов.
* **TicketFactory** – абстрактная фабрика.
* **PlaneFactory, TrainFactory, BusFactory** – конкретные фабрики.

**15. Вид топлива (бензин, дизель, электричество)**

**Классы и их взаимодействие:**

* **IFuelType** – интерфейс с GetEnergySource().
* **Gasoline, Diesel, Electric** – классы топлива.
* **FuelFactory** – абстрактная фабрика.
* **GasolineFactory, DieselFactory, ElectricFactory** – конкретные фабрики.

**Задание 2: Реализация паттерна "Декоратор"**

**Общее описание:**  
Паттерн «Декоратор» позволяет динамически добавлять объекту новую функциональность без изменения его исходного кода. Для реализации этого паттерна используется базовый интерфейс (или абстрактный класс) компонента, базовая реализация компонента, а также один или несколько классов-декораторов, которые оборачивают базовый компонент и расширяют его поведение. Каждый декоратор реализует тот же интерфейс, что и базовый класс, и содержит ссылку на объект-компонент. Таким образом, можно «наслоить» несколько декораторов, комбинируя функциональность.

**Задача 1: Пицца с дополнительными ингредиентами**

* **Компонент:** IPizza
  + Метод: string GetDescription() – возвращает описание пиццы.
  + Метод: double GetCost() – возвращает стоимость пиццы.
* **Конкретный компонент:** BasicPizza – базовая пицца без дополнительных ингредиентов.
* **Декораторы:**
  + CheeseDecorator – добавляет сыр, изменяя описание (добавляет "с сыром") и увеличивает стоимость.
  + PepperoniDecorator – добавляет пепперони.
  + VeggieDecorator – добавляет овощи.
* **Взаимодействие:** Клиент создает базовую пиццу, затем оборачивает её в один или несколько декораторов для получения нужного сочетания ингредиентов.

**Задача 2: Кофе с добавками**

* **Компонент:** ICoffee
  + string GetDescription(), double GetCost()
* **Конкретный компонент:** BasicCoffee – черный кофе.
* **Декораторы:**
  + MilkDecorator – добавляет молоко, изменяя описание и стоимость.
  + SugarDecorator – добавляет сахар.
  + SyrupDecorator – добавляет сироп (например, ванильный).
* **Взаимодействие:** Клиент создает базовый кофе и добавляет выбранные декораторы для получения напитка с нужными добавками.

**Задача 3: Оружие с улучшениями**

* **Компонент:** IWeapon
  + Метод: string GetDescription()
  + Метод: int GetDamage()
* **Конкретный компонент:** BasicWeapon – базовое оружие.
* **Декораторы:**
  + FireEnhancementDecorator – добавляет огненный эффект, увеличивая урон и описание ("с огненной атакой").
  + IceEnhancementDecorator – добавляет ледяной эффект.
  + CriticalHitDecorator – увеличивает шанс критического удара.
* **Взаимодействие:** Клиент выбирает базовое оружие, после чего применяет один или несколько декораторов для повышения эффективности оружия.

**Задача 4: Текстовый редактор с применением стилей**

* **Компонент:** IText
  + Метод: string GetContent() – возвращает текстовое содержимое.
* **Конкретный компонент:** PlainText – обычный текст.
* **Декораторы:**
  + BoldDecorator – добавляет стиль «жирный».
  + ItalicDecorator – добавляет стиль «курсив».
  + UnderlineDecorator – добавляет подчеркивание.
* **Взаимодействие:** Текст оборачивается в один или несколько декораторов, изменяющих его представление.

**Задача 5: Автомобиль с дополнительными опциями**

* **Компонент:** ICar
  + Метод: string GetFeatures() – возвращает описание автомобиля.
  + Метод: double GetPrice() – возвращает цену.
* **Конкретный компонент:** BasicCar – базовый автомобиль.
* **Декораторы:**
  + SunroofDecorator – добавляет люк, увеличивая цену и описание.
  + NavigationDecorator – добавляет навигационную систему.
  + LeatherSeatsDecorator – добавляет кожаные сиденья.
* **Взаимодействие:** Клиент начинает с базового автомобиля и добавляет опции-декораторы для индивидуальной комплектации.

**Задача 6: Фотоприложение с фильтрами**

* **Компонент:** IImage
  + Метод: string GetDescription() – возвращает описание изображения.
  + Метод: Image Process() – возвращает обработанное изображение (для примера можно считать, что возвращает сам объект, символизируя применение фильтра).
* **Конкретный компонент:** BaseImage – исходное изображение.
* **Декораторы:**
  + BlackWhiteFilterDecorator – применяет черно-белый фильтр.
  + BlurFilterDecorator – применяет размытие.
  + SharpenFilterDecorator – улучшает резкость.
* **Взаимодействие:** Изображение оборачивается в один или несколько декораторов, и каждый из них модифицирует результат обработки.

**Задача 7: Робот-помощник с дополнительными функциями**

* **Компонент:** IRobot
  + Метод: string GetStatus() – возвращает статус робота.
* **Конкретный компонент:** BasicRobot – базовый робот без дополнительных функций.
* **Декораторы:**
  + VoiceControlDecorator – добавляет голосовое управление.
  + NavigationDecorator – улучшает навигацию.
  + SensorDecorator – добавляет дополнительные датчики.
* **Взаимодействие:** Базовый робот оборачивается в декораторы, которые расширяют его возможности.

**Задача 8: Отель с дополнительными услугами**

* **Компонент:** IRoomService
  + Метод: string GetServiceDetails() – возвращает описание услуг.
  + Метод: double GetCost() – возвращает стоимость услуг.
* **Конкретный компонент:** BasicRoomService – базовый сервис (например, уборка).
* **Декораторы:**
  + BreakfastDecorator – добавляет завтрак в услугу.
  + SpaDecorator – добавляет спа-процедуры.
  + AirportPickupDecorator – добавляет трансфер из аэропорта.
* **Взаимодействие:** Клиент выбирает базовый сервис, а затем добавляет дополнительные услуги через декораторы.

**Задача 9: Игровой персонаж с дополнительными навыками**

* **Компонент:** IGameCharacter
  + Метод: string GetAbilities() – возвращает перечень навыков.
* **Конкретный компонент:** BaseCharacter – базовый персонаж.
* **Декораторы:**
  + StealthDecorator – добавляет способность к скрытности.
  + PowerAttackDecorator – добавляет мощную атаку.
  + HealingDecorator – добавляет возможность лечения.
* **Взаимодействие:** Игрок выбирает базового персонажа и расширяет его возможности декораторами.

**Задача 10: Музыкальный плеер с улучшением звука**

* **Компонент:** IMusicPlayer
  + Метод: string GetSoundQuality() – возвращает описание качества звука.
* **Конкретный компонент:** BasicMusicPlayer – базовый плеер.
* **Декораторы:**
  + BassBoostDecorator – улучшает басы.
  + NoiseReductionDecorator – уменьшает шумы.
  + EqualizerDecorator – предоставляет настройку эквалайзера.
* **Взаимодействие:** Плеер оборачивается в один или несколько декораторов, улучшая звучание.

**Задача 11: Социальная сеть – реакции пользователей**

* **Компонент:** IPost
  + Метод: string GetContent() – возвращает содержимое поста.
* **Конкретный компонент:** BasicPost – базовый пост без реакций.
* **Декораторы:**
  + LikeDecorator – добавляет функциональность лайка.
  + DislikeDecorator – добавляет дизлайк.
  + ShareDecorator – добавляет возможность поделиться.
* **Взаимодействие:** Пост оборачивается в декораторы, расширяя функционал взаимодействия с пользователями.

**Задача 12: Документ с форматированием**

* **Компонент:** IDocument
  + Метод: string GetFormattedText() – возвращает отформатированный текст документа.
* **Конкретный компонент:** PlainDocument – документ без форматирования.
* **Декораторы:**
  + HeaderDecorator – добавляет заголовок в документ.
  + FooterDecorator – добавляет подвал.
  + PageNumberDecorator – добавляет номера страниц.
* **Взаимодействие:** Документ оборачивается в декораторы, формируя окончательное форматирование.

**Задача 13: Веб-запрос – добавление HTTP-заголовков**

* **Компонент:** IHttpRequest
  + Метод: string GetHeaders() – возвращает список заголовков.
* **Конкретный компонент:** BasicHttpRequest – базовый запрос без заголовков.
* **Декораторы:**
  + AuthHeaderDecorator – добавляет заголовок авторизации.
  + ContentTypeDecorator – добавляет заголовок типа содержимого.
  + CustomHeaderDecorator – добавляет произвольный заголовок.
* **Взаимодействие:** Запрос оборачивается в декораторы, которые добавляют необходимые заголовки перед отправкой.

**Задача 14: Система логирования – расширение информации о событиях**

* **Компонент:** ILogger
  + Метод: string Log(string message) – возвращает лог-сообщение.
* **Конкретный компонент:** BasicLogger – логгер, выводящий простое сообщение.
* **Декораторы:**
  + TimestampDecorator – добавляет метку времени к сообщению.
  + SeverityDecorator – добавляет уровень важности (INFO, WARN, ERROR).
  + UserDecorator – добавляет имя пользователя, инициировавшего событие.
* **Взаимодействие:** Сообщение логгируется базовым логгером, а декораторы добавляют дополнительную информацию.

**Задача 15: Защита данных – шифрование и компрессия**

* **Компонент:** IDataProcessor
  + Метод: string ProcessData(string data) – возвращает обработанные данные.
* **Конкретный компонент:** BasicDataProcessor – возвращает исходные данные без изменений.
* **Декораторы:**
  + EncryptionDecorator – шифрует данные, добавляя алгоритм шифрования к результату.
  + CompressionDecorator – сжимает данные, уменьшая размер сообщения.
  + (Возможна цепочка: сначала шифрование, затем компрессия).
* **Взаимодействие:** Исходные данные оборачиваются в декораторы, которые последовательно модифицируют их: сначала данные шифруются, а затем сжимаются.

**Задание 3: Реализация паттерна "Команда"**

**Общее описание:**  
Паттерн "Команда" (Command) инкапсулирует запрос в виде объекта, позволяя передавать запросы как параметры, ставить их в очередь, отменять и логировать. В каждом примере будут задействованы следующие компоненты:

* **Интерфейс команды (ICommand)** с методом Execute().
* **Конкретная команда (ConcreteCommand)**, которая реализует ICommand и инкапсулирует запрос.
* **Получатель (Receiver)**, который выполняет конкретное действие.
* **Инициатор (Invoker)**, который вызывает команду.
* **Клиент (Client)**, который связывает команды с получателями.

**Задача 1: Управление освещением в умном доме**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретная команда:** LightOnCommand и LightOffCommand, реализующие Execute(), вызывающие методы TurnOn() и TurnOff() у объекта-получателя.
* **Получатель:** Light с методами TurnOn() и TurnOff().
* **Инициатор:** RemoteControl вызывает команду для включения/выключения света.

**Задача 2: Управление роботизированным пылесосом**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** StartCleaningCommand, StopCleaningCommand, ReturnToBaseCommand.
* **Получатель:** RobotVacuum с методами StartCleaning(), StopCleaning(), ReturnToBase().
* **Инициатор:** RobotController инициирует выполнение команд пылесоса.

**Задача 3: Редактор текста – отмена и повтор**

* **Интерфейс:** ICommand с методом Execute() (и, возможно, Undo() для расширенного варианта).
* **Конкретные команды:** CopyCommand, PasteCommand, UndoCommand, RedoCommand.
* **Получатель:** TextEditor с методами Copy(), Paste(), Undo(), Redo().
* **Инициатор:** EditorInvoker вызывает команды для обработки текста.

**Задача 4: Управление музыкальным плеером**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** PlayCommand, PauseCommand, StopCommand, реализующие вызов методов плеера.
* **Получатель:** MusicPlayer с методами Play(), Pause(), Stop().
* **Инициатор:** PlayerRemote инициирует команды для управления воспроизведением.

**Задача 5: Банковская система – перевод средств**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретная команда:** TransferMoneyCommand, инкапсулирующая информацию о сумме и счетах отправителя/получателя.
* **Получатель:** BankAccountService с методом Transfer(decimal amount, string fromAccount, string toAccount).
* **Инициатор:** BankingTerminal вызывает команду для перевода денег.

**Задача 6: Ресторан – оформление заказа**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** PlaceOrderCommand и CancelOrderCommand, вызывающие методы заказа.
* **Получатель:** OrderSystem с методами PlaceOrder() и CancelOrder().
* **Инициатор:** OrderInvoker инициирует выполнение заказа или его отмену.

**Задача 7: Пульт дистанционного управления телевизором**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** TVPowerOnCommand, TVPowerOffCommand, VolumeUpCommand, VolumeDownCommand.
* **Получатель:** Television с методами PowerOn(), PowerOff(), IncreaseVolume(), DecreaseVolume().
* **Инициатор:** TVRemote управляет телевизором через команды.

**Задача 8: Система безопасности – управление сигнализацией**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** ActivateAlarmCommand и DeactivateAlarmCommand.
* **Получатель:** AlarmSystem с методами Activate() и Deactivate().
* **Инициатор:** SecurityPanel вызывает команды для управления сигнализацией.

**Задача 9: Печать документов**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** PrintDocumentCommand, CancelPrintCommand.
* **Получатель:** Printer с методами Print() и Cancel().
* **Инициатор:** PrintManager запускает команды печати или отмены.

**Задача 10: Онлайн-магазин – управление корзиной**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** AddToCartCommand и RemoveFromCartCommand.
* **Получатель:** ShoppingCart с методами AddItem() и RemoveItem().
* **Инициатор:** CartController инициирует команды для управления покупками.

**Задача 11: Чат-приложение – отправка сообщений**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретная команда:** SendMessageCommand, которая инкапсулирует текст сообщения и адресата.
* **Получатель:** ChatService с методом SendMessage(string message).
* **Инициатор:** ChatClient инициирует отправку сообщения.

**Задача 12: Файловые операции – копирование и перемещение**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** CopyFileCommand и MoveFileCommand.
* **Получатель:** FileManager с методами CopyFile(string source, string destination) и MoveFile(string source, string destination).
* **Инициатор:** FileOperationInvoker управляет файловыми операциями.

**Задача 13: Планировщик задач – запуск и отмена задач**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** StartTaskCommand и CancelTaskCommand.
* **Получатель:** TaskScheduler с методами StartTask() и CancelTask().
* **Инициатор:** SchedulerController инициирует запуск или отмену задач.

**Задача 14: Игровой персонаж – выполнение действий**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** JumpCommand, AttackCommand, DefendCommand.
* **Получатель:** GameCharacter с методами Jump(), Attack(), Defend().
* **Инициатор:** GameController управляет действиями персонажа.

**Задача 15: Система умного орошения – управление поливом**

* **Интерфейс:** ICommand с методом Execute().
* **Конкретные команды:** StartIrrigationCommand и StopIrrigationCommand.
* **Получатель:** IrrigationSystem с методами StartIrrigation() и StopIrrigation().
* **Инициатор:** IrrigationController инициирует полив или его остановку.

**Общая схема реализации паттерна "Команда":**

1. **Интерфейс команды (ICommand)** определяет метод Execute(), который будет вызываться инициатором.
2. **Конкретные команды** реализуют интерфейс, храня ссылку на объект-получатель и параметры операции.
3. **Получатель (Receiver)** выполняет реальные операции (например, включение света, отправка сообщения, печать документа).
4. **Инициатор (Invoker)** вызывает метод Execute() у команды, не зная деталей реализации.