	Паттерны								
	Проверенное и готовое к использованию логическое решение, которе может быть реализовано по-разному в разных языках программирования.								
processanonaumas 2	Плюсы: - снижение сложности разработни за счёт готовых абстракций - облегчение коммуникации между разработчиками - мичусы: - слепое сладование некоторому шаблоку может привести к усложнению программы жалание попробовать некоторому шаблон в деле без особых на то оснований.								
характеристики шаблонов.	Мия - все шаблоны имеют уникальное имя, стуккащее для их идентификации; Назначение данного шаблоне Задача, которую шаблон повыволет решить; Способ решиния, предлагаемый в цаблоне для решения задачи в том контексте, где этот шаблон был найден; Участника - сущности, причинающие участие в решения задачи, в том контексте, где этот шаблон был найден; Участника - сущности, причинающие участие в решения задачи, в шаблоне; Разлизация - сисловаемый цаблона какр результат действий, выполняемых в шаблоне; Разлизация - сисловаемый варамет реализации цаблона.								
Назовите три основные группы паттернов.	Порождающие - отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов без внесения в программу лишвих заявилистем. Структурные - отвечают за построение удобных в поддержже нерархий классов Поведенческие - заботятся об эффективной коммунисации между объектами. Основные - основные строительные блоки, используемые для построения других шаблонов. Например, интерфейс.								
Расскажите про паттерн Одиночка (Singleton).	Порождающий паттери проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа. Конструктор помечается как private, а для создания нового объекта Singleton использует специальный метод getInstance(). Он либо создаёт объект, либо отдаёт существующий объект, сели он уже был создан. private static Singleton instance; public static Singleton getInstance() { if (instance = null) { instance = new Singleton(); } return instance; } } ** можно не создавать мномество объектов для ресурсоемих задач, а пользоваться одним **- нарушает принцип еднекой стветственности. так как его могут использовать множество объектов Почему считается антипаттариом? - нелья тестировать с помощью тюск, не можно использовать роменМоск. - Нарушает принцип еднекой стветственности. - Нарушает принственности. - Нар								
	3) Порождающий паттерии, который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений одного объекта. Паттери предлагает вынести конструкрование объекта за пределые его собственного класса, поручие это дело отдельным объектам, называемым строителями. Подносе конструкрования объекта разбить на отдельные шаги (например, построить Стены, вставить Двери). Чтобы создать объект, мам кужно посчередно вызывать меторы строителя. Причём не кужно запускать все шаги, а только те, что кужны для прокведства объекта определённой конфитурации. Можно пойти дальше и выделять вызовы меторы строитель вызовы меторы строитель вызовы меторы строитель вызовы меторы строительства, а строитель — выполнять их. +: Позволяет использовать один и тот же код для создания различных объектов. Изолирует сложный код сборки объектов от его основной бизнес-погики: Устоиненет код программы из-за введения дополнительных классов.	The state of the s	(MA(1)						
	Порождающий шаблон проектирования, в котором предоставляет интерфейс для создания объектов в родительском классе, но позволяет подклассам изменять тип создаваемых объектов, подклассы имплементируют общий интерфейс с методом для создания объектов. Переопределенный метод и каждом наследнике возвращает чункный вариант объекта. Объекты всё равно будут создаваться при помощи пем, но делать это будет фабричный метод. Таким образом можно переопределить фабричный метод в подклассь чтобы миженить тип создаваемого продукта. Чтобы эта система заработала, все возвращаемые объекты должны иметь общий интерфейс. Подклассы смогут производить объекты различных классов, спедующих одному и тому же интерфейсу. За масел том и тому же интерфейсу. За мажет призводства объектов в одно место, упрощая поддержку кода. Реализует принцип открытости/закрытости. За мажет призводства объектов в одно место, упрощая поддержку кода. Реализует принцип открытости/закрытости. За мажет причаести к созданию больших паралительных мерариий классов, так как для каждого класса продукта наро создать свой подиласс создателя. Пример: у нас есть интерфейс (разработние* и его реализацию нумно создать производителя для этого создаем еще одни интерфейс который будет имень асето одне фабричный метод.	mate class Propos (fatty("o-lp; cfatty)(pecfatty.");	CppDeveloper og ataDeveloper OppDeveloper OppDeveloper writeCode()	● Developer Factory → createDeveloper() actory (f) Developer • considered transfer of the Code() • AnniDevelope • witteCode() • witteCode()				

Расскажите про паттерн Абстрактная фабрика (Abstract Factory).	Порождающий паттерн проектирования, Абстрактная фабрика задаёт интерфейс создания всех доступных типов продуктов, а каждая конкретная реализация фабрики порождает продукты одной из вариаций. Абстрактна фабрики порождает продукты одной из вариаций. Абстрактна фабрики порождает продуктов, составляющих семейства. Так, все вариации кресеп получат общий интерфейс Креспо, все двезны реализуют интерфейс Диван и так далее. Далее вы создателе выстрактную дабрижу — общий интерфейс двезны т так далее. Далее вы создателе абстрактную дабрижу — общий интерфейс двезны т так далее. Далее на создателенные интерфейсами, которые оздать Креспо, создать Диван и создать Стотик). Эти сперации должны возвращать абстрактные типы продуктов, представленные интерфейсами, которые мы выделили двезе — Креспа, Диваны и Стотики. - : гарантированию будет создаваться тип одного семейства - : Усложнеят код програжмы из-за введения множества дополнительных классов. Порождающий паттеры проектирования, который позволяет копировать объекты, не вдаваясь в подробности их реализации. Паттерь поручает создание когий самим колируемым объекты. Он вводит общий интерфейс с методом сіоле для всех объектов, поддерживающих ключирование. Реализация этого метода в разных классам счень сижка. Метод создайт новый объект текущего класса и колирует в него значения всех полей собственного объекта, мняющие с сылки на другие объекты. - : Позволяет ключировать объекты, не привязываясь к их конкретным классам. - : Спожно ключировать составные объекты, имвощие ссылки на другие объекты.	public developer gathering implements frajectTeamFactory(
	Структурный паттеры проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе. Это объект-пересцие, который трансформирует интерфей кип данные одного объекта в такой выд., чтобы он стал понятен другому объекту. При этом адаптер оборачивает один из объектов, так что другой объект даже не знает о наличии первого. -: Отделяет и скрывает от клиента подробности преобразования различных интерфейсов. -: Условияет и одражения из-за введения дополнительных класосв.	public ovid near(); public void near(); bystem.out.println("Saving Java object"); bystem.out.println("New object"); }
Расскажите про паттерн Декоратор (Decorator).	Структурный паттери проектирования, который позволяет добавлять объектам новую функциональность, оборачивам их в полезные обебртия» (Надстройка когда уже есть готовый функционал(класс) и мых отим вызывать этот же функционал но с добавлением своей реализации) Ligineoù объект омещается в другой объект-обертку, который запускает базовее поведение оберунгого объекта, а затем добавляет к результату тото свое. Оба объекта имеют общей интерфейе, поэтому для пользователи нет имелами размиць, с заком объектом работать — чистым или оберунтым. Вы можете использовать нежолько размых обфетк одновременно— результат будет иметь объединённое поведение всех оберток сразу. Адаптер не менят состояния объекта, а декоратор может менять. - Еслышая гибкость, чем у наследования. - Труднее конфитурировать многоратно обернутые объекты.	<pre>public static void main(String[] args) { FileStream FileStream = new FileBufferedReader(new FileStreamReader()); fileStream.read(); } interface FileStream { void read(); } class FileStreamReader implements FileStream { Soveride read(); public void read(); } yitem.out.println("read file"); } } **Override public void read() { printle void read() {</pre>
Расскажите про	Структурный паттери проектирования, который позволяет подставлять вместо реальных объектов спекумальные объекты-заменители, которые перехатывают вызова к оргинительном у объекту, позволяя сделать что-то до или после передачи вызова оргинител. Заместитель предлагает создать новый клас-дублёр, имеющий тот же витерфейс, что и оригинальный служебный объект. При получении запроса от клиента объект-заместитель сам бы создавал экземпляр служебного объекта, выполняя промежуточную логику, которая выполнялась бы до (или после) вызовое этих же меторов в настоящем объекте. -: Повволяет контролировать сервисный объект незаметно для клиента: Умеличивает время отклика от сервисы	nublic class JavaDeveloper implements Collection { private String name;
	Поведенческий паттери проектирования, который даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутрениего представления. Идеа состит в том, чтобы вынести поведение обхода коллекции из самой коллекции в отдельный класс. Детали: Создается итератор и интерфейс, который возвращает итератор. В классе, в котором надо будет вызывать итератор, имплементируем интерфейс, возвращающий итератор, а сам итератор делаем там нестатическим вложенным классом, так как он нигде использоваться спольци не бушет public forestime at the forestime start (); public object next(); public interface collection {	<pre>private String[] skills; public Javabeveloper(String name, String[] skills) this.name = name; this.skills = skills; this.skills = skills; } goverride public Iterator getIterator() { return new SkillIterator(); } source description in the skills i</pre>
Расскажите про паттерн Шаблонный	Поведенческий паттери проектирования, это метод, внутренние блоки которого могут переопределяться подклассами для избежания поеторного когирования (который пошлетов определяет апторитм и позоляет наследениямия переопоределять некоторые шаги апторитмя, но коменяя его структуру в целом). Паттери предлагает разбить апторитмя на последовательность шагов, описать эти шали в отдельных методах и вызывать их в одном шаблонном методе друг за другом. Для описания шагов используется абстрактный класс. Общие шали можно будет описать прямо в абстрактном класс. Это поволит подклассам переопределять некоторые шали апторитма, оставляя без изменений его структуры и статывым шарти этогорые пра этого в образования в переопределять некоторые шали апторитма, оставляя без изменений его структуры и статывым шарти этогорые пра этого в образования в подкластности.	public abstract class websitements (public observation) (System.out.println("Medical"); showdmageContent(); system.out.println("Footer"); system.out.println("Footer"); public abstract void showMageContent();
Расскажите про паттерн Цепочка обязанностей (Chain	Поведенческий паттери проектирования, позволяет передавать запрос по цепочке потенциальных обработчиков, пока один из них не обработает запрос. Каждый обработчик решает, может ли он обработать запрос сам и стоит ли передавть запрос дальше по цепи Область применения цепочки обказенностей — всеевозможные обработчики событий, последовательные проверки доступа и просавором в сером в размурков на том, чтобы превратить каждую проверку в отдельный класс с единственным методом выполнения. Данные запроса над которым происходит проверко, будут предаваться в метод каж артументы. Каждый из методов будет иметь ссылку на следующий метод-обработчик, что образует цепь. Таким образом, при получении запроса обработчик сможет не только сам что-то с ним сделать, но и передать обработку спедующему объекту в цепочке. Может и не передавать, если проверка в одном из методов не прошла, например.	<pre>abstract class MessagePrinter { MessagePrinter maxMessagePrinter; void setNoxMessagePrinter messagePrinter; void setNoxMessagePrinter messagePrinter; void print(String message); if (mestNessagePrinter i = mult) { mexMessagePrinter i = mult) { messagePrinter i = mult) {</pre>

