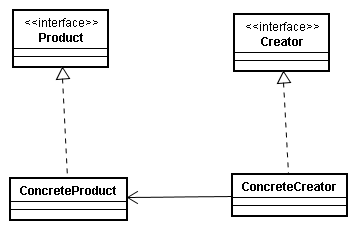
**ARCHITECTURE AND DESIGN**

**LABORATORY WORK 9**

**EXERCISE 01 – CHOOSE THE BEST PATTERN**

1. Creational pattern name: **Factory Method**

* Problem description: Мне необходимо создавать разные типы объектов, такие как "Job Seeker", "Employer", "Admin", без жесткой привязки к конкретным классам. Жесткие связи затрудняют гибкость и расширяемость проекта.
* Problem solution: Использую Factory Method, который позволяет создать общий фабричный метод для создания объектов, не указывая конкретный класс. Например, при регистрации нового пользователя фабричный метод возвращает объект нужного типа в зависимости от выбранной роли.
* Advantage of this pattern for your problem: Это упрощает добавление новых типов пользователей в будущем и делает систему более гибкой, поскольку не нужно менять основной код для добавления новых классов.
* Disadvantage of this pattern for your problem: При необходимости добавления новых типов объектов может возникнуть сложность в поддержке кода. Также использование фабричного метода может затруднить понимание кода для новых разработчиков.
* Graphical representation (class diagram):



1. Creational pattern name: **Abstract Factory**

* Problem description: В моем проекте нужно создавать семейства связанных объектов, таких как аккаунты для разных ролей (Job Seeker, Employer, Admin), при этом сохраняя согласованность между ними.
* Problem solution: Применяю Abstract Factory, который позволяет создать фабрику для создания семейств связанных объектов. Это дает возможность создавать аккаунты с согласованными интерфейсами, что обеспечивает целостность и унификацию.
* Advantage of this pattern for your problem: Abstract Factory обеспечивает согласованность между связанными объектами и позволяет легко добавлять новые семейства. Он также способствует поддержанию хорошей структуры кода.
* Disadvantage of this pattern for your problem: Увеличивается сложность при добавлении новых семейств объектов, и поддержка кода может стать сложнее. Кроме того, Abstract Factory требует большего числа классов, что может увеличить сложность проекта.
* Graphical representation (class diagram):

Изображение выглядит как текст, диаграмма, линия, План

Автоматически созданное описание

1. Structural patterns name: **Adapter**

* Problem description: Мой проект включает разные модули и системы, которые могут быть несовместимы из-за различий в интерфейсах. Например, если я хочу интегрировать модуль обработки заявлений с базой данных, интерфейсы могут не совпадать.
* Problem solution: Применяю шаблон Adapter, который создает промежуточный класс для согласования несовместимых интерфейсов. Это позволяет интегрировать разные системы, обеспечивая гибкость и унификацию.
* Advantage of this pattern for your problem: Adapter упрощает интеграцию различных модулей и систем, делая их совместимыми. Это также позволяет использовать существующие классы без необходимости их изменения.
* Disadvantage of this pattern for your problem: Adapter может усложнить структуру проекта, увеличив число классов и связей. Также это может замедлить производительность, поскольку адаптеры добавляют дополнительный уровень обработки.
* Graphical representation (class diagram):

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

1. Structural patterns name: **Decorator**

* Problem description: В моем проекте возникает необходимость добавлять новые функции к существующим классам, не изменяя основной код. Например, можно добавить дополнительные привилегии к аккаунтам пользователей.
* Problem solution: Применяю шаблон Decorator, который позволяет оборачивать базовый класс, добавляя новые функции. Это обеспечивает динамичное расширение без изменения базового кода.
* Advantage of this pattern for your problem: Decorator позволяет добавлять функционал к существующим классам, сохраняя основной код неизменным. Это обеспечивает гибкость и упрощает расширение системы.
* Disadvantage of this pattern for your problem: Decorator может усложнить структуру кода, особенно при большом числе оберток. Кроме того, он может затруднить понимание кода для новых разработчиков из-за дополнительного уровня оборачивания.
* Graphical representation (class diagram):

Изображение выглядит как текст, диаграмма, линия, Параллельный

Автоматически созданное описание

1. Behavioral patterns name: **Observer**

* Problem description: В моем проекте требуется уведомлять об изменениях в системе, например, когда изменяется статус вакансии или приходит новое сообщение.
* Problem solution: Использую шаблон Observer, который позволяет создавать список наблюдателей и уведомлять их об изменениях. Это обеспечивает реактивность системы и уменьшает жесткие зависимости.
* Advantage of this pattern for your problem: Observer позволяет уведомлять о событиях и изменениях без жестких связей между классами, что делает систему более гибкой. Это также упрощает добавление новых наблюдателей или событий.
* Disadvantage of this pattern for your problem: Observer может увеличить сложность структуры, а при большом количестве наблюдателей может возникнуть проблема с производительностью. Кроме того, управление наблюдателями может потребовать дополнительного внимания.
* Graphical representation (class diagram):

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

1. Behavioral patterns name: **State**

* Problem description: В проекте требуется изменение поведения в зависимости от состояния, например, при переходе заявлений между разными этапами обработки.
* Problem solution: Применяю шаблон State, который позволяет изменять поведение системы в зависимости от текущего состояния. Это обеспечивает гибкость и упрощает переходы между состояниями.
* Advantage of this pattern for your problem: State позволяет управлять поведением системы, делая переходы между состояниями более управляемыми. Это также облегчает добавление новых состояний и переходов.
* Disadvantage of this pattern for your problem: State может усложнить код при большом числе состояний или переходов между ними. Кроме того, поддержка кода может стать сложнее из-за увеличения числа состояний.
* Graphical representation (class diagram):

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание