Системы Реального Времени

Отчёт по лабораторной работе №4  
«Just Patterns»  
подготовил: Сагитов Александр, 3МО 2гр

**Паттерн проектирования**: Компоновщик

Паттерн «Компоновщик» (Composite) относится к структурным паттернам проектирования, основная цель которых — организовать классы и объекты в более удобные для управления и расширения структуры. Компоновщик решает проблему представления иерархии объектов в виде дерева, позволяя работать с отдельными объектами и их составными структурами одинаково. Это особенно полезно, когда требуется сгруппировать объекты, чтобы их поведение и взаимодействие были одинаковыми для простых элементов и сложных компонентов, состоящих из этих элементов.

**Основная идея** компоновщика заключается в том, чтобы объединить объекты в древовидную структуру «часть-целое». Это позволяет клиентам работать с отдельными объектами или их группами без необходимости различать их. Для этого паттерн использует общий интерфейс для всех элементов структуры, что позволяет взаимодействовать с ними одинаковым образом.

**Применение паттерна**

*Основные ситуации, в которых уместно использовать компоновщик*:

* Когда существует иерархия объектов, и необходимо обрабатывать их одинаково, вне зависимости от того, являются ли они простыми или составными.
* Когда требуется динамически формировать сложные структуры объектов, которые должны быть представлены иерархически.
* Когда нужно иметь возможность добавлять новые типы объектов в систему, не нарушая её целостности и не изменяя код, который взаимодействует с существующими объектами.

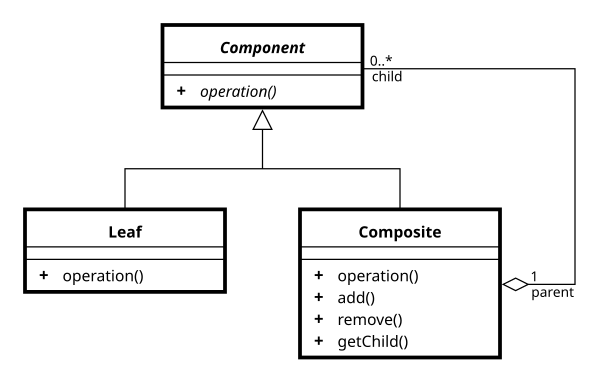
**Тип паттерна**: структурный

**Компоновщик** — это структурный паттерн, так как он отвечает за организацию и взаимодействие между классами и объектами. Его **главная цель** — создание гибких структур, которые можно масштабировать и изменять без модификации кода, использующего эти структуры.

**Структура и UML диаграмма**

*Компоновщик часто реализуется с помощью следующих компонентов*:

* Компонент (Component) — это интерфейс или абстрактный класс, который определяет поведение как для простых объектов, так и для составных.
* Лист (Leaf) — это простые объекты, которые не могут содержать другие объекты.
* Композит (Composite) — это составной объект, который может содержать другие объекты и реализует поведение, связанное с хранением и управлением этими объектами.
* Клиент (Client) — это класс, который взаимодействует с компонентами через общий интерфейс, не зная, являются ли они простыми или составными.

**UML диаграмма для паттерна Компоновщик:**

Источник: <https://commons.wikimedia.org/wiki/File:Composite_UML_class_diagram_(fixed).svg?uselang=ru>

Диаграмма показывает, что компоненты могут быть как простыми (листовыми), так и составными. Важно, что клиент взаимодействует с компонентами через единый интерфейс, что упрощает работу с объектами.

**Примеры использования**

*Видеоигры*

В мире видеоигр Компоновщик находит широкое применение. Например, представим игру, в которой есть инвентарь. Он может включать в себя разные предметы, такие как оружие, броня или зелья. Каждый предмет сам по себе является объектом, который можно использовать или выбросить. Однако, некоторые предметы могут быть объединены в наборы — например, комплект брони, который включает в себя шлем, доспехи, ботинки и щит. Вместо того чтобы обращаться с каждым предметом по отдельности, паттерн Компоновщик позволяет рассматривать комплект как единый объект, с которым можно работать так же, как с отдельным предметом.

Если игрок хочет надеть комплект брони, он делает это одним действием, не задумываясь о каждом предмете. В этом случае паттерн Компоновщик позволяет объединить простые предметы в составные, упрощая взаимодействие игрока с игрой.

*Графические редакторы*

Другой распространённый пример Компоновщика — это графические редакторы, такие как Photoshop или Figma. В таких программах объекты на холсте могут быть представлены как отдельные элементы — линии, фигуры, изображения, — но могут быть также объединены в группы. Например, логотип компании может состоять из текста и изображения, которые сгруппированы вместе. Паттерн Компоновщик позволяет работать с группой как с единым целым, а не с отдельными элементами.

Таким образом, графический дизайнер может передвигать логотип по холсту как один объект, а не беспокоиться о каждом отдельном его элементе. В то же время, если нужно внести изменения в логотип, элементы можно “разгруппировать” и редактировать по отдельности. Компоновщик предоставляет гибкость в работе с объектами как с группами, так и с отдельными частями.

*Файловая система*

Файловые системы операционных систем также могут служить примером паттерна Компоновщик. В файловой системе есть файлы и папки. Файл является листовым элементом, который не может содержать других объектов. Папка же может содержать как файлы, так и другие папки. Благодаря паттерну Компоновщик можно обрабатывать файлы и папки единообразно — например, операции копирования, перемещения или удаления применимы как к файлам, так и к папкам.

Когда пользователь перемещает папку, операционная система автоматически перемещает все содержимое этой папки, независимо от того, сколько вложенных папок и файлов там находится. Это иллюстрирует, как Компоновщик упрощает работу с вложенными структурами.

**Преимущества и недостатки компоновщика**

*Преимущества*:

* Упрощение кода клиента. Клиенту не нужно различать простые и составные объекты — все они представляют один и тот же интерфейс.
* Гибкость в добавлении новых компонентов. Легко добавлять новые типы компонентов, не меняя код, который уже работает с системой.
* Легкость в работе с древовидными структурами. Компоновщик особенно полезен, когда объекты организованы в сложные иерархии.

*Недостатки*:

* Могут быть созданы избыточные структуры. Если компоненты слишком мелкие, может появиться слишком много объектов, что усложнит систему.
* Может потребоваться сложная реализация. Если требуется сложная логика для взаимодействия между компонентами, это может усложнить разработку.

**Заключение**

Паттерн Компоновщик предоставляет мощный инструмент для работы с иерархическими структурами объектов, позволяя унифицировать обращение с ними и легко управлять как простыми элементами, так и сложными составными структурами. Он широко применяется в разработке видеоигр, графических редакторов, операционных систем и многих других областях, где требуется работа с древовидными структурами.