## 1. Паттерн "Фабричный метод" (Factory Method)

### Описание

Паттерн "Фабричный метод" используется для создания объектов без указания конкретного класса объекта, который будет создан. Вместо этого, создание объектов делегируется подклассам или функциям, которые реализуют фабричный метод.

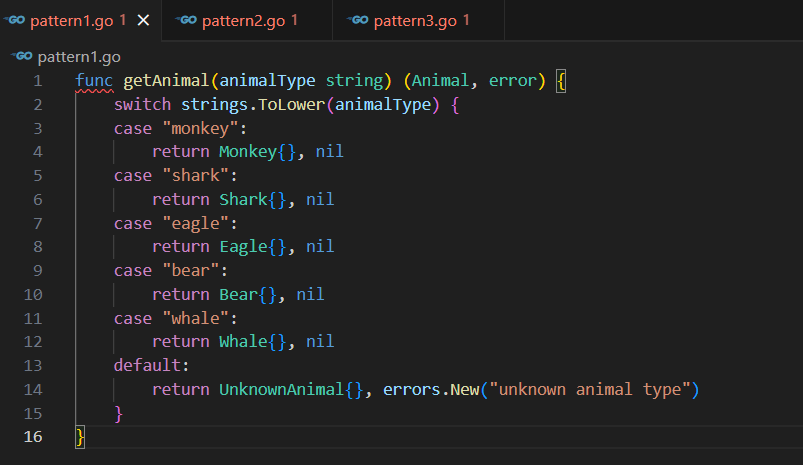
**Применение в проекте**

В проекте "Зоопарк" паттерн "Фабричный метод" реализован в функции getAnimal(animalType string) (Animal, error). Эта функция принимает тип животного в виде строки и возвращает соответствующий объект типа Animal.}

**Обоснование выбора**

Использование фабричного метода позволяет:

* **Избежать жесткой привязки** к конкретным классам животных.
* **Упростить добавление новых типов животных** в будущем. Для этого достаточно добавить новый случай в операторе switch.
* **Улучшить читаемость кода**, так как создание объектов инкапсулировано в одном месте.



**2. Паттерн "Стратегия" (Strategy)**

**Описание**

Паттерн "Стратегия" позволяет определить семейство алгоритмов, инкапсулировать каждый из них и сделать их взаимозаменяемыми. Это позволяет изменять алгоритмы независимо от клиентов, которые их используют.

**Применение в проекте**

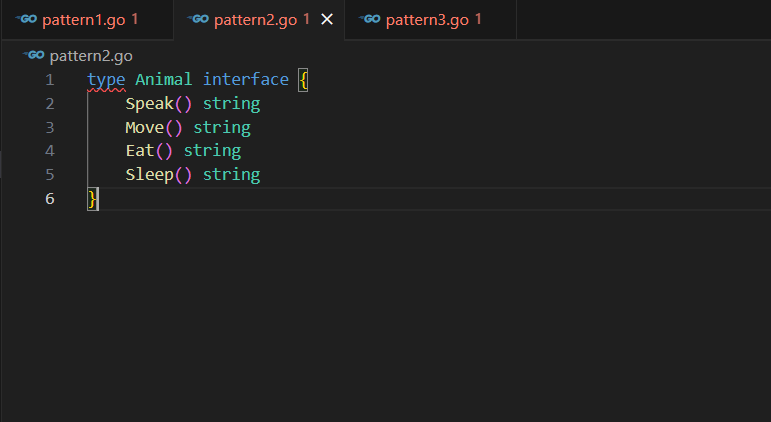
В проекте "Зоопарк" паттерн "Стратегия" реализован через интерфейс Animal, который определяет общие методы для всех животных:

Каждый тип животного (например, Monkey, Shark, Eagle и т.д.) реализует этот интерфейс, предоставляя свои собственные реализации методов.

**Обоснование выбора**

Использование паттерна "Стратегия" позволяет:

* **Инкапсулировать поведение** каждого животного в отдельных классах.
* **Легко добавлять новые типы животных** без изменения существующего кода.
* **Изменять поведение животных** в зависимости от контекста, не изменяя их классы.



**3. Паттерн "Декоратор" (Decorator)**

**Описание**

Паттерн "Декоратор" позволяет динамически добавлять объектам новую функциональность, оборачивая их в объекты-обертки.

**Применение в проекте**

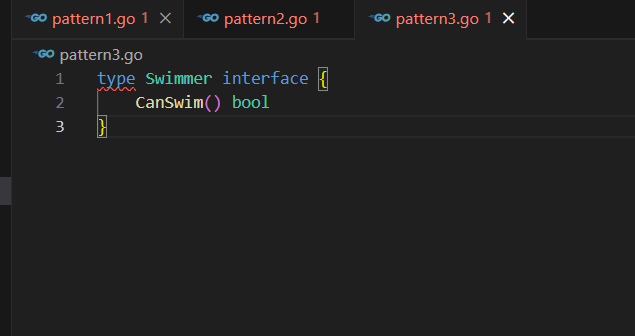
В проекте "Зоопарк" паттерн "Декоратор" не реализован явно, но его концепция присутствует в использовании интерфейса Swimmer:

Некоторые животные (например, Shark и Bear) реализуют этот интерфейс, добавляя себе возможность плавать.

**Обоснование выбора**

Хотя в данном проекте паттерн "Декоратор" не используется в полной мере, его концепция полезна для:

* **Добавления дополнительной функциональности** к существующим объектам без изменения их классов.
* **Создания гибкой структуры**, позволяющей комбинировать различные возможности.



**4. Паттерн "Посетитель" (Visitor)**

**Описание**

Паттерн "Посетитель" позволяет добавлять новые операции к объектам без изменения их классов. Это достигается путем создания нового класса-посетителя, который реализует нужные операции.

**Применение в проекте**

В проекте "Зоопарк" паттерн "Посетитель" не реализован явно, но его концепция может быть применена для добавления новых операций к животным, например, для проверки здоровья, кормления и т.д.

**Обоснование выбора**

Паттерн "Посетитель" может быть полезен для:

* **Добавления новых операций** к животным без изменения их классов.
* **Упрощения кода**, так как все операции могут быть инкапсулированы в отдельных классах-посетителях.

**Заключение**

В проекте "Зоопарк" были успешно применены паттерны проектирования, такие как "Фабричный метод", "Стратегия" и "Декоратор". Эти паттерны помогли создать гибкую и расширяемую архитектуру, которая легко поддерживается и масштабируется. В будущем, при необходимости добавления новых функций или типов животных, можно будет использовать паттерн "Посетитель" для дальнейшего улучшения проекта.