**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

по учебной практике

Тема: **Шаблоны проектирования в языке Java**

Студент гр. 2307 **Стукен В.А.**

Санкт-Петербург 2024

Оглавление

[Поведенческий шаблон проектирования Observer (наблюдатель) 3](#_Toc169454969)

[**Назначение** 3](#_Toc169454970)

[**Область применения** 3](#_Toc169454971)

[Порождающий шаблон проектирования Factory (фабрика) 9](#_Toc169454972)

[**Назначение** 9](#_Toc169454973)

[**Область применения** 9](#_Toc169454974)

[Структурный шаблон проектирования Decorator (декоратор) 14](#_Toc169454975)

[**Назначение** 14](#_Toc169454976)

[**Область применения** 14](#_Toc169454977)

[Вывод 20](#_Toc169454978)

# Поведенческий шаблон проектирования Observer (наблюдатель)

Реализует у класса механизм, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов и тем самым наблюдать за ними.

Классы, на события которых другие классы подписываются, называются субъектами (Subjects), а подписывающиеся классы называются наблюдателями.

## **Назначение**

Определяет зависимость типа один ко многим между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.

## **Область применения**

Шаблон «наблюдатель» применяется в тех случаях, когда система обладает следующими свойствами:

* существует как минимум один объект, рассылающий сообщения;
* имеется не менее одного получателя сообщений, причём их количество и состав могут изменяться во время работы приложения;
* позволяет избежать сильного зацепления взаимодействующих классов.

Данный шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают получатели с предоставленной им информацией.

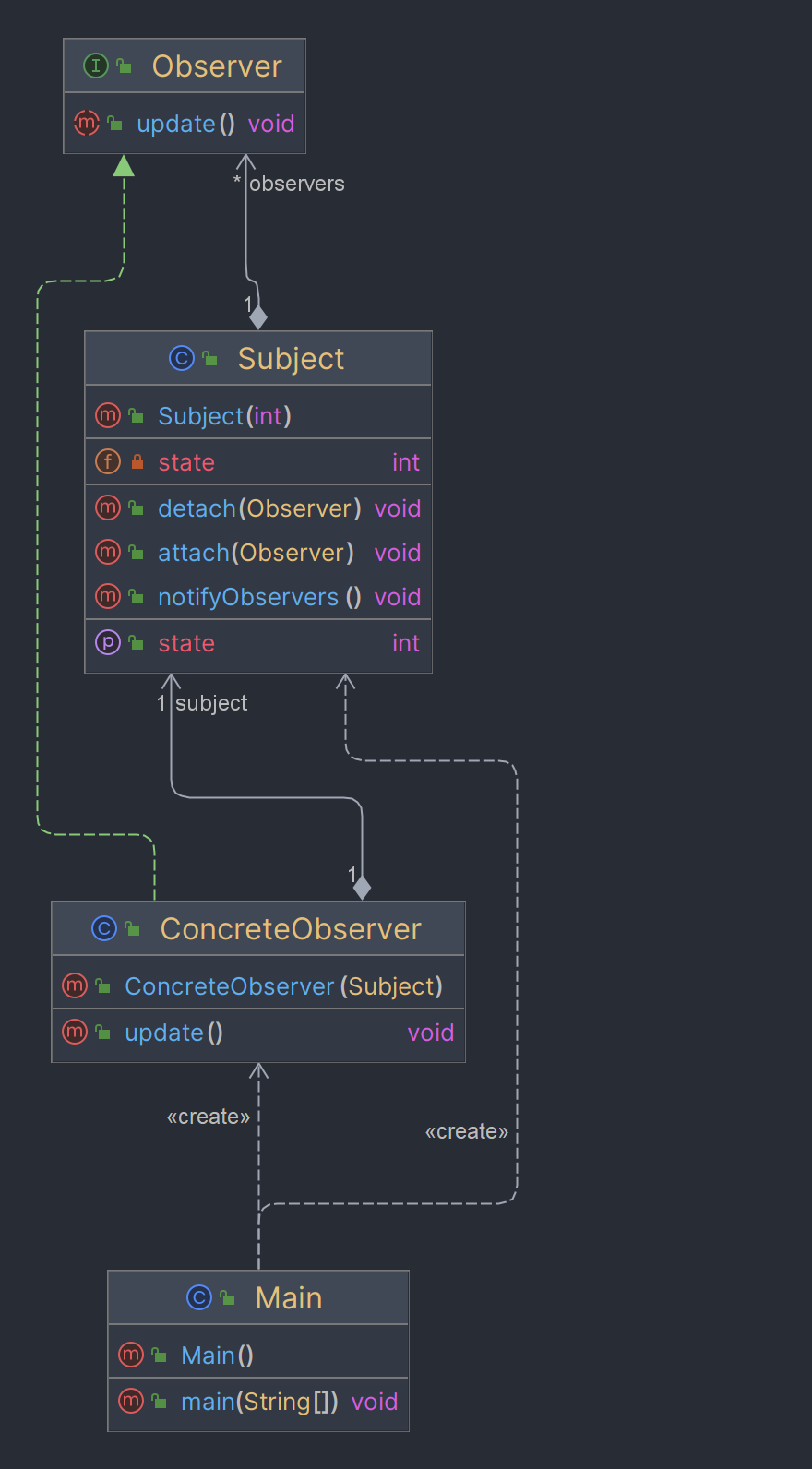


Рисунок 1 Диаграмма классов

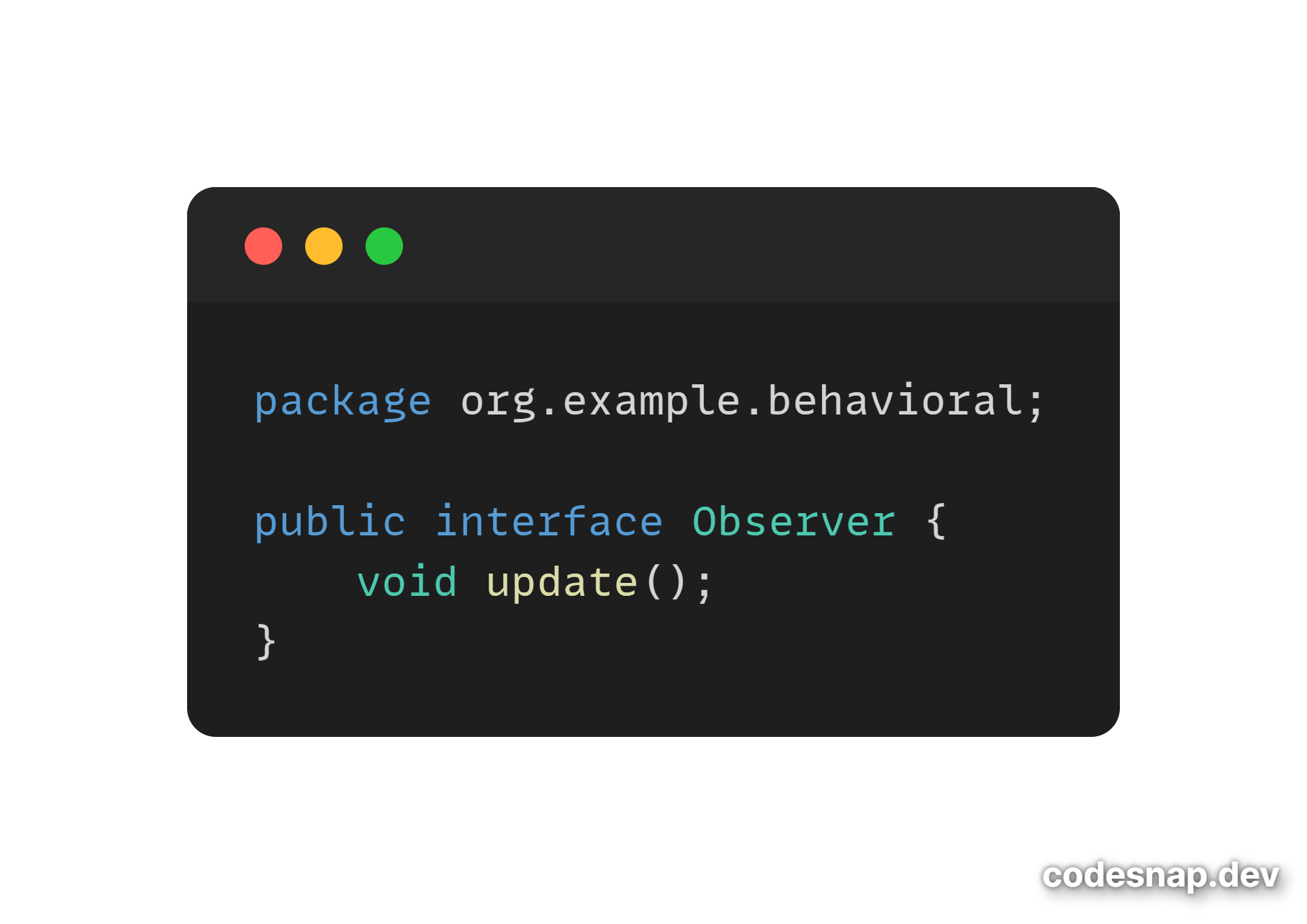


Рисунок 2 Интерфейс Observer



Рисунок 3 класс ConcreteObserver, реализующий интерфейс Observer



Рисунок 4 класс Subject



Рисунок 5 класс Main, показывающий использование шаблона

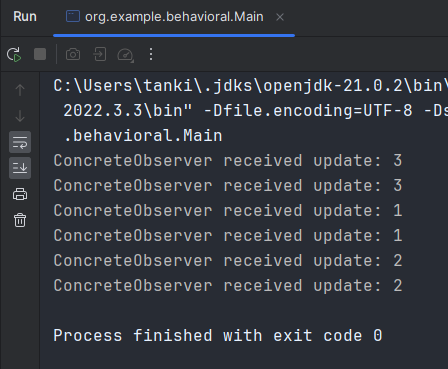


Рисунок 6 результат работы программы

# Порождающий шаблон проектирования Factory (фабрика)

Фабрика — это шаблон проектирования, который помогает решить проблему создания различных объектов в зависимости от некоторых условий.

## **Назначение**

Основная цель шаблона "Фабрика" заключается в том, чтобы отделить процесс создания объектов от их использования. Это позволяет:

1. Избежать жесткой привязки к конкретным классам: Создание объектов осуществляется через интерфейс или абстрактный класс, что позволяет использовать различные реализации без изменения кода клиента.
2. Упростить добавление новых классов: Новые типы объектов могут быть добавлены без модификации существующего кода.
3. Контроль над созданием объектов: Легче управлять созданием объектов, например, внедрять логирование, кэширование или другие аспекты.

## **Область применения**

Шаблон "Фабрика" полезен в следующих ситуациях:

1. Когда заранее неизвестны типы и зависимости объектов, с которыми будет работать программа: Это позволяет создавать объекты по мере необходимости, обеспечивая гибкость и расширяемость.
2. Когда система должна быть независимой от процесса создания, композиции и представления продуктов: Это важно для соблюдения принципа открытости/закрытости (Open/Closed Principle).
3. Для управления жизненным циклом объектов: Если создание объектов включает сложную логику (например, настройки, подключение к базам данных и т. д.), "Фабрика" упрощает это.
4. Когда требуется инкапсуляция логики создания объектов: Например, если нужно скрыть детали создания объекта от клиента.

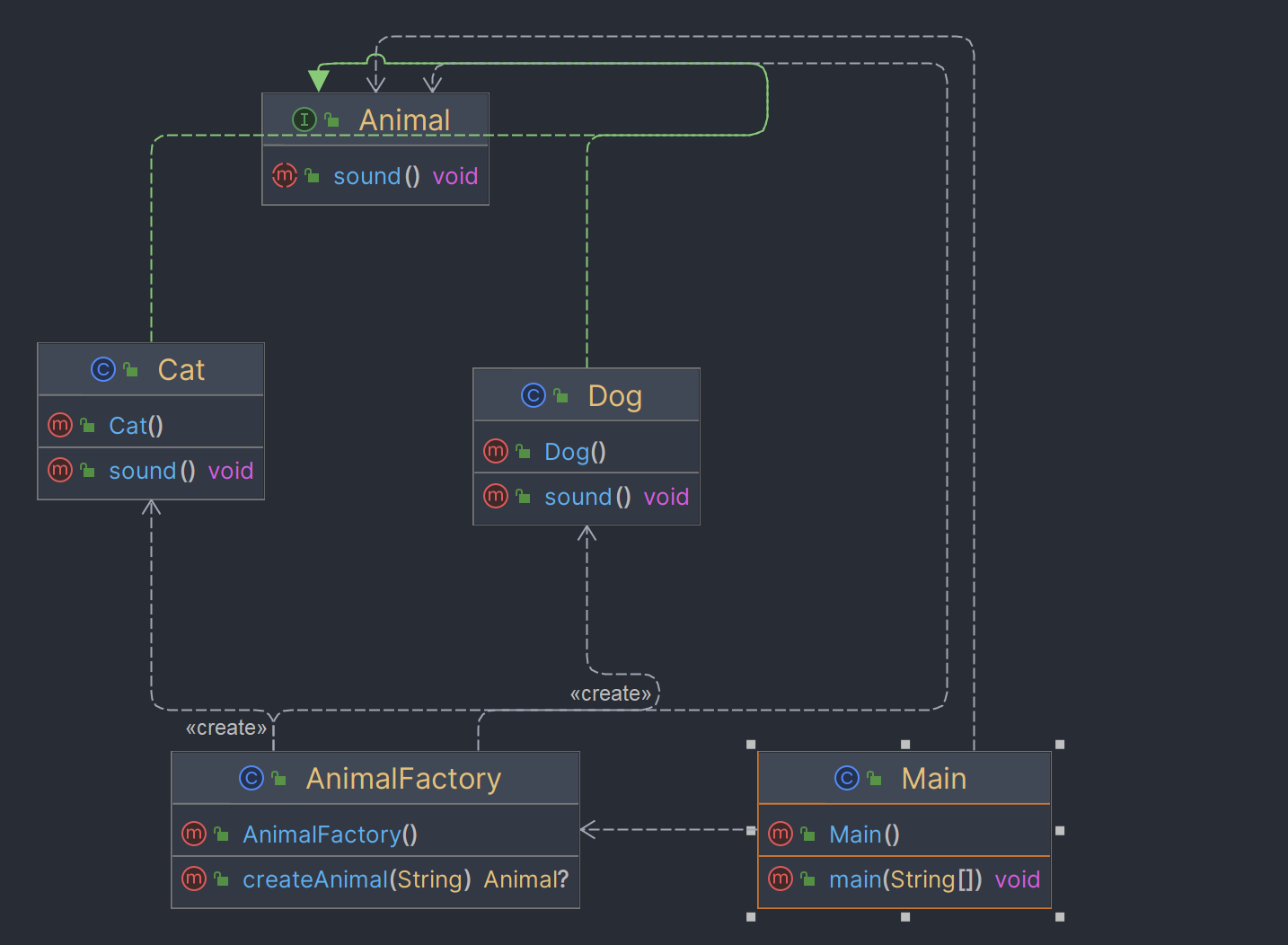


Рисунок 7 диаграмма классов



Рисунок 8 класс AnimalFactory

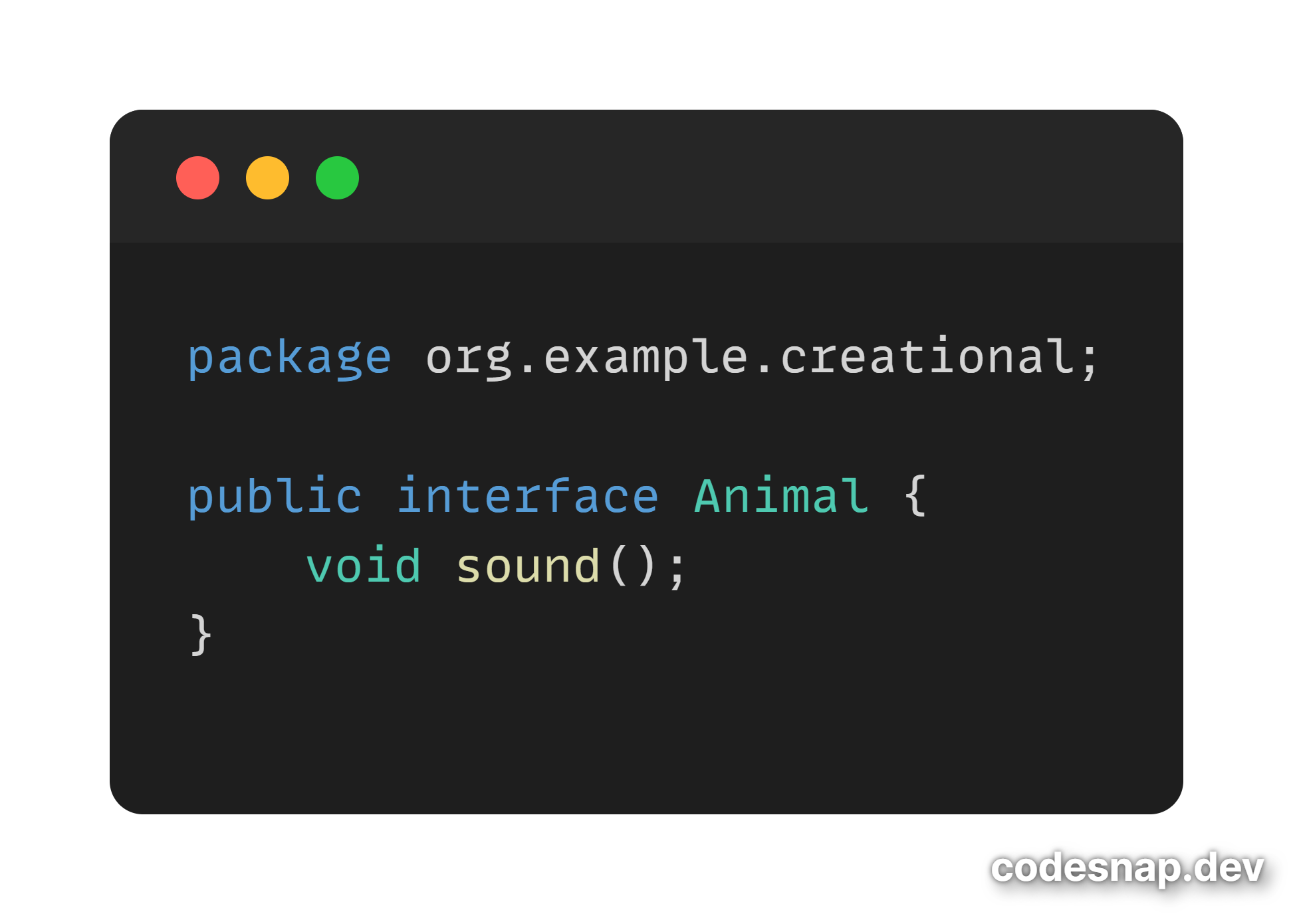


Рисунок 9 базовый класс Animal

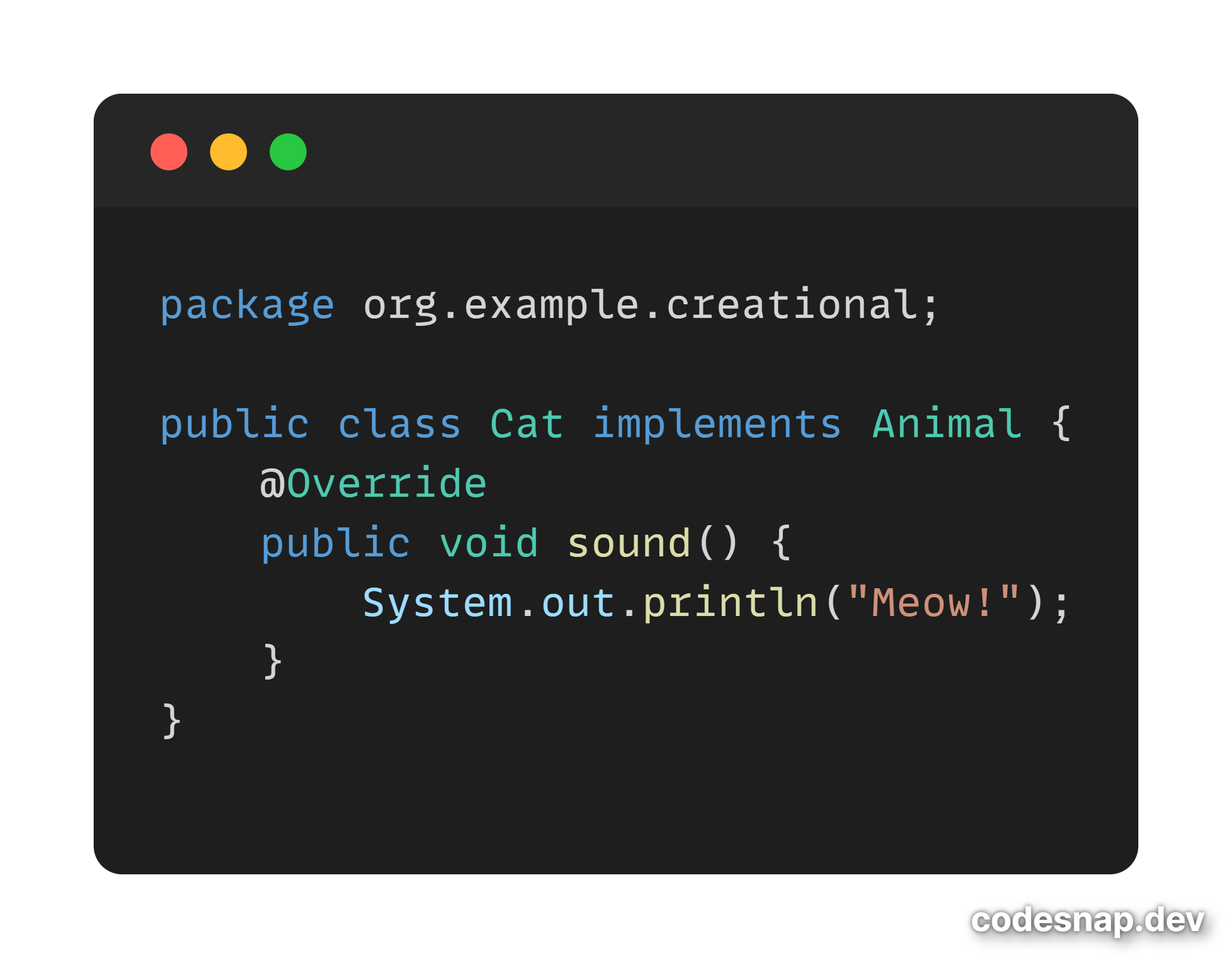


Рисунок 10 класс Cat

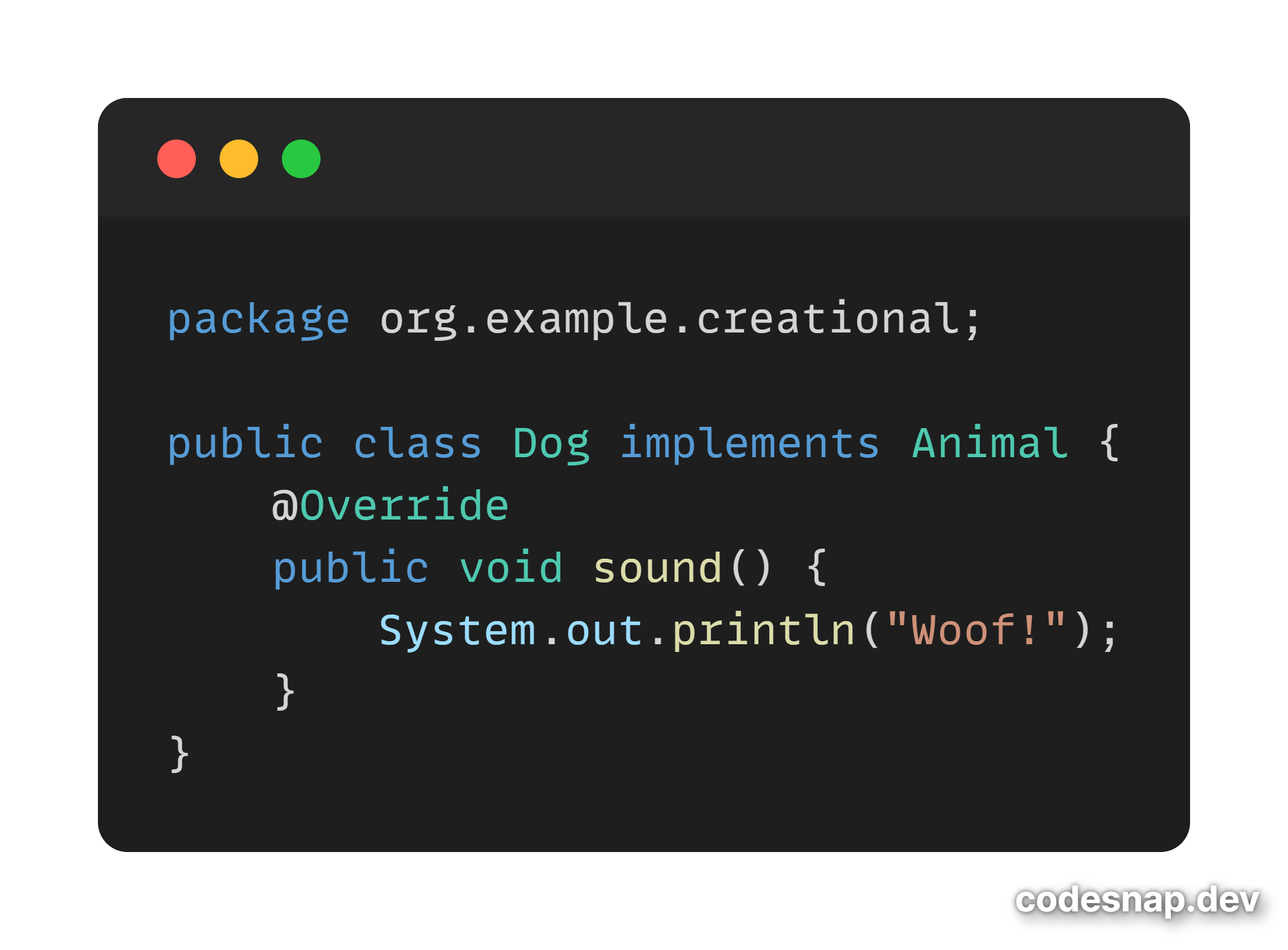


Рисунок 11 класс Dog

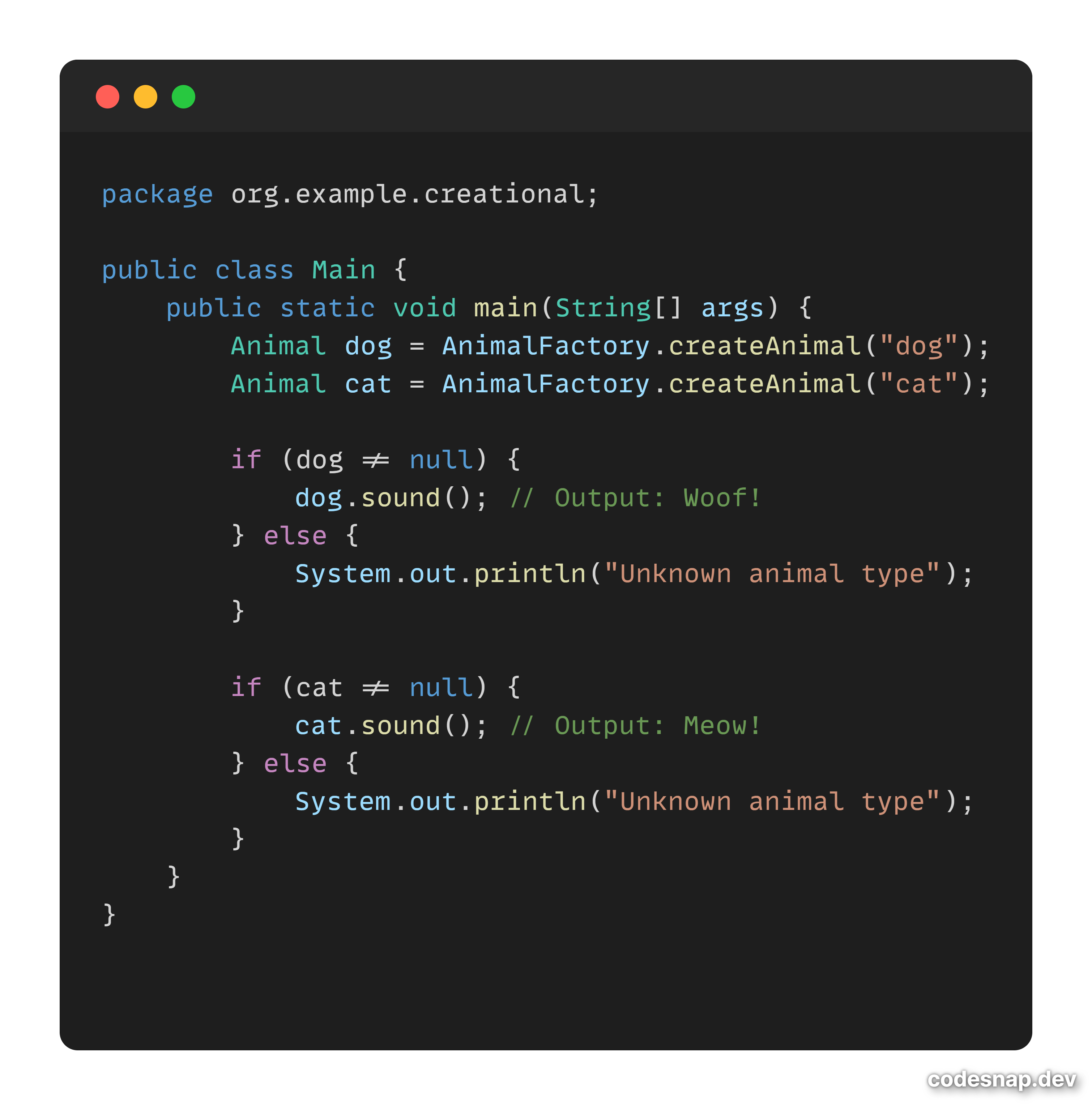


Рисунок 12 класс Main, иллюстрирующий использование шаблона

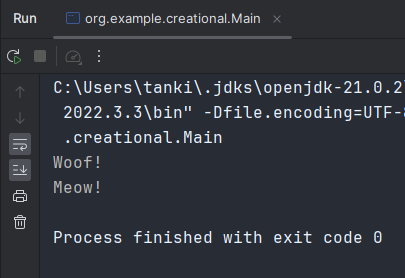


Рисунок 13 результат работы программы

# Структурный шаблон проектирования Decorator (декоратор)

Шаблон проектирования "Декоратор" (Decorator) — это структурный паттерн, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в классы-обертки (декораторы). Декораторы предоставляют гибкую альтернативу наследованию для расширения функциональности.

## **Назначение**

Основная цель шаблона "Декоратор" заключается в том, чтобы добавлять объектам новую функциональность без изменения их кода. Это позволяет:

1. **Избежать монолитных классов**: Вместо создания одного класса с многочисленными функциями, можно создать несколько маленьких классов-декораторов.
2. **Расширять поведение объектов**: Декораторы позволяют добавлять поведение объектам на лету, без необходимости создания подклассов.
3. **Упрощать тестирование и повторное использование кода**: Поведения, оформленные в виде декораторов, могут быть легко протестированы и использованы повторно.

## **Область применения**

Шаблон "Декоратор" полезен в следующих ситуациях:

1. **Когда нужно добавлять поведение объектам на лету**: Например, добавление различных визуальных эффектов к пользовательскому интерфейсу.
2. **Когда невозможно использовать наследование**: Либо из-за ограничения языка, либо из-за необходимости комбинировать несколько поведений в различных сочетаниях.
3. **Когда необходимо разделить обязанности между классами**: Это помогает соблюдать принцип единственной ответственности, делая код более читабельным и поддерживаемым.

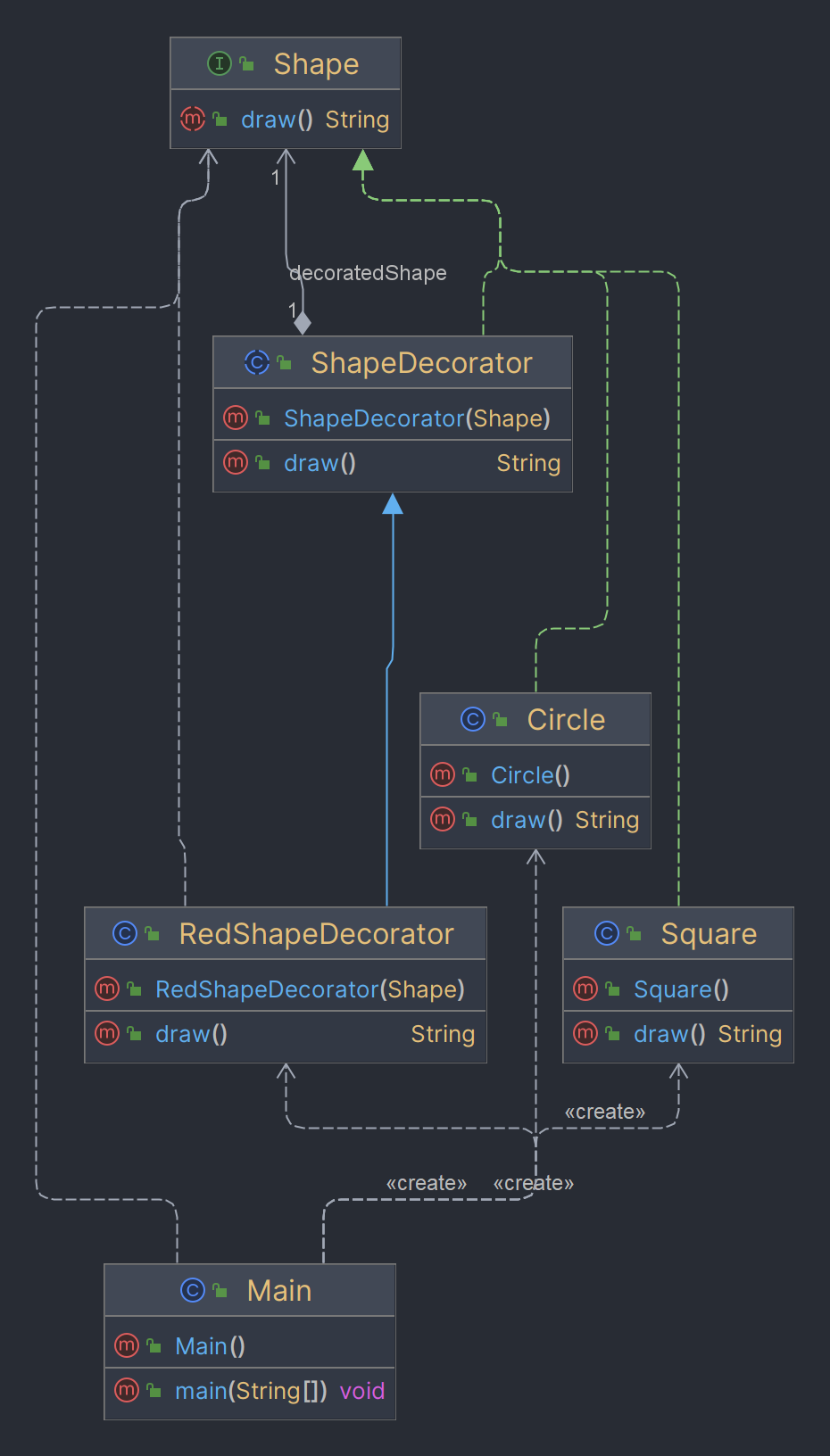


Рисунок 14 диаграмма классов

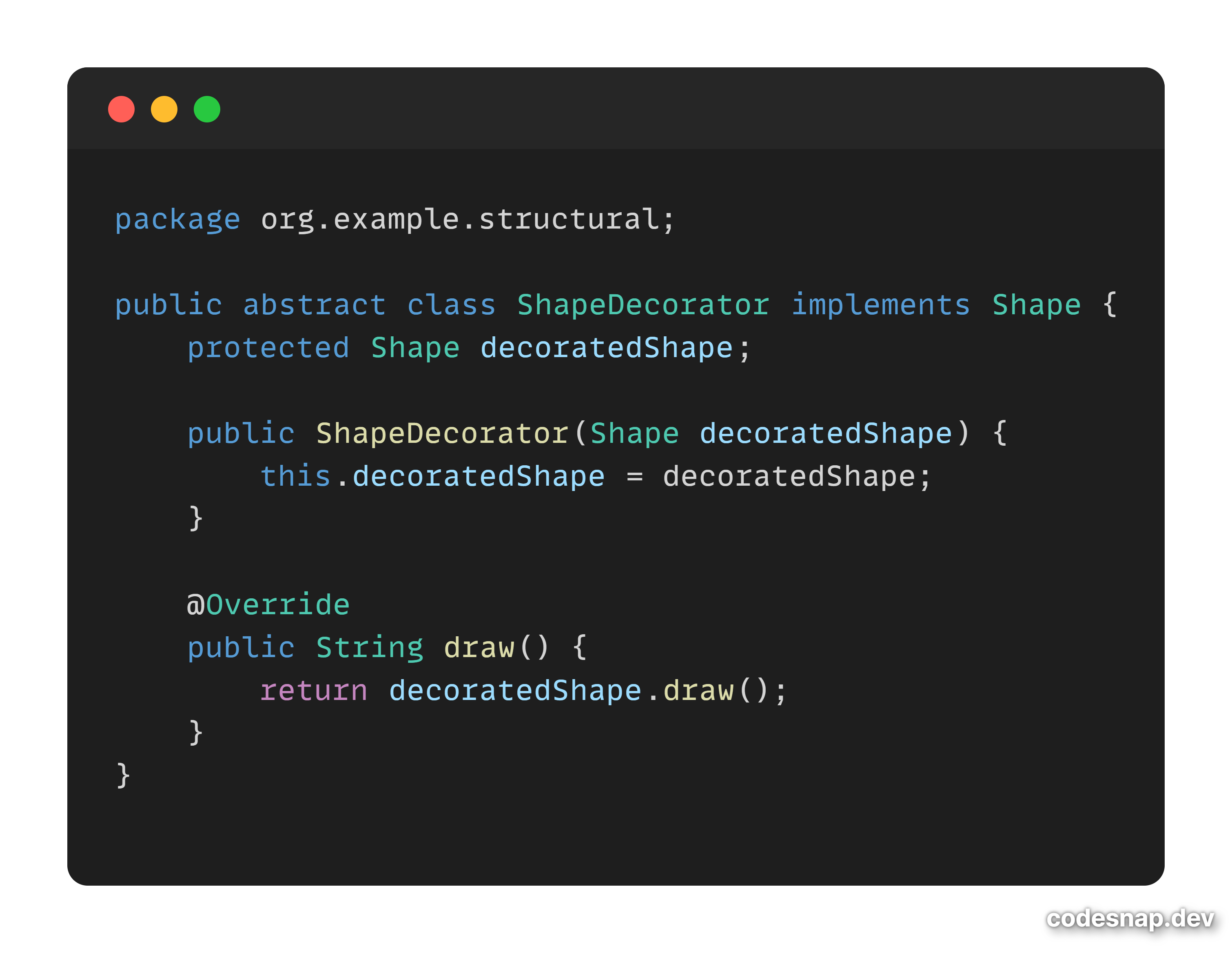


Рисунок 15 класс ShapeDecorator

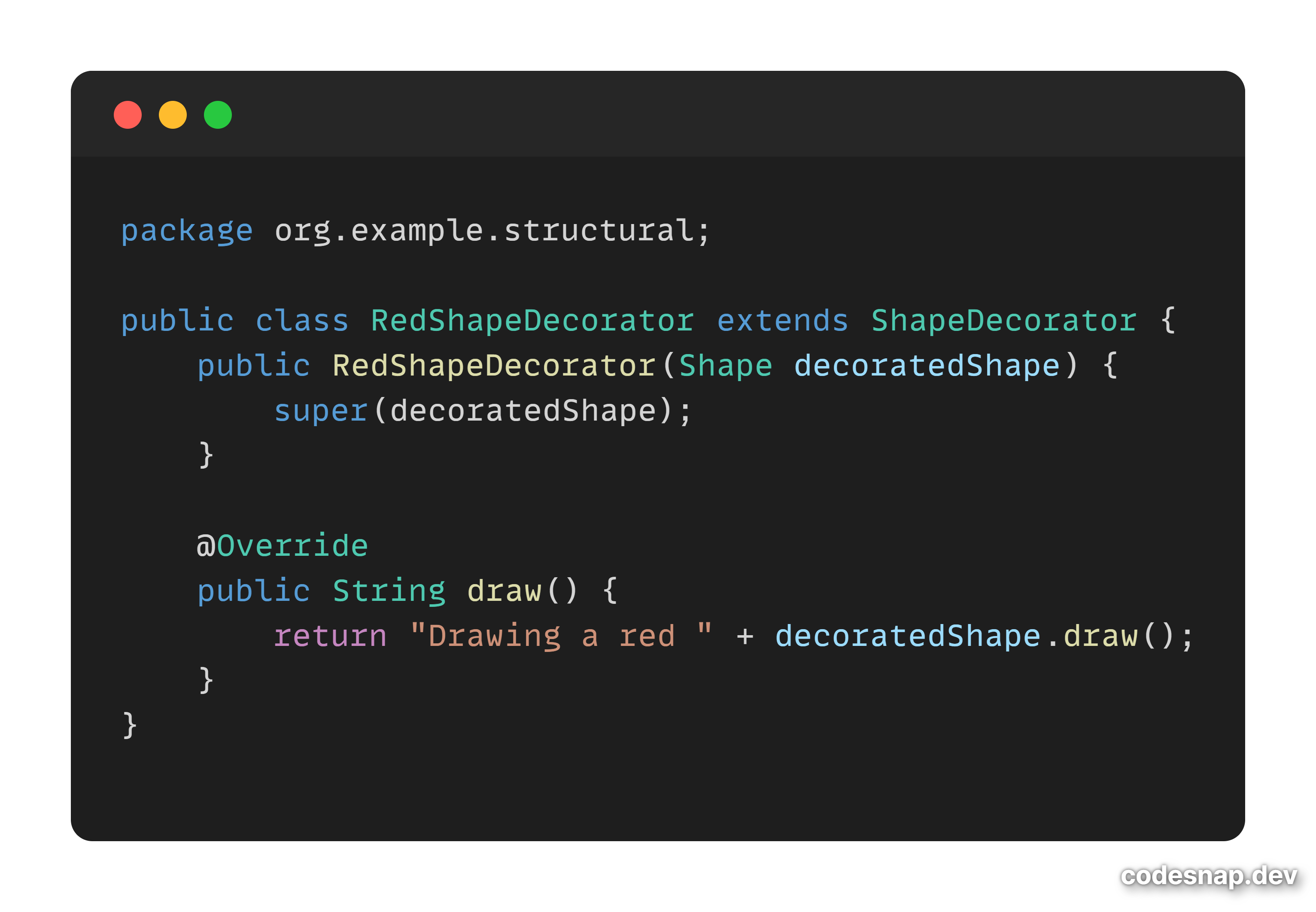


Рисунок 16 класс RedShapeDecorator, расширяющий базовый класс ShapeDecorator

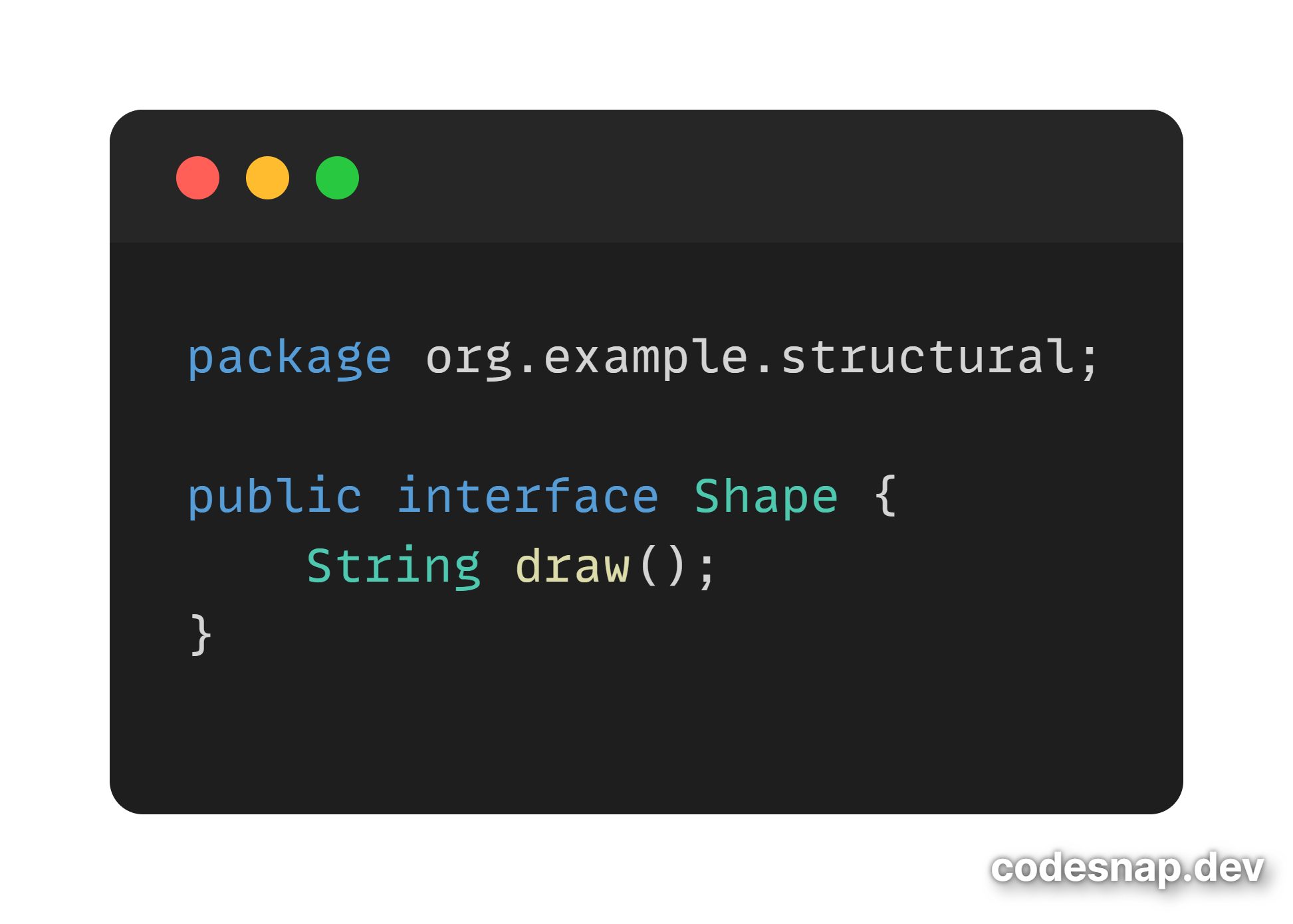


Рисунок 17 базовый класс Shape

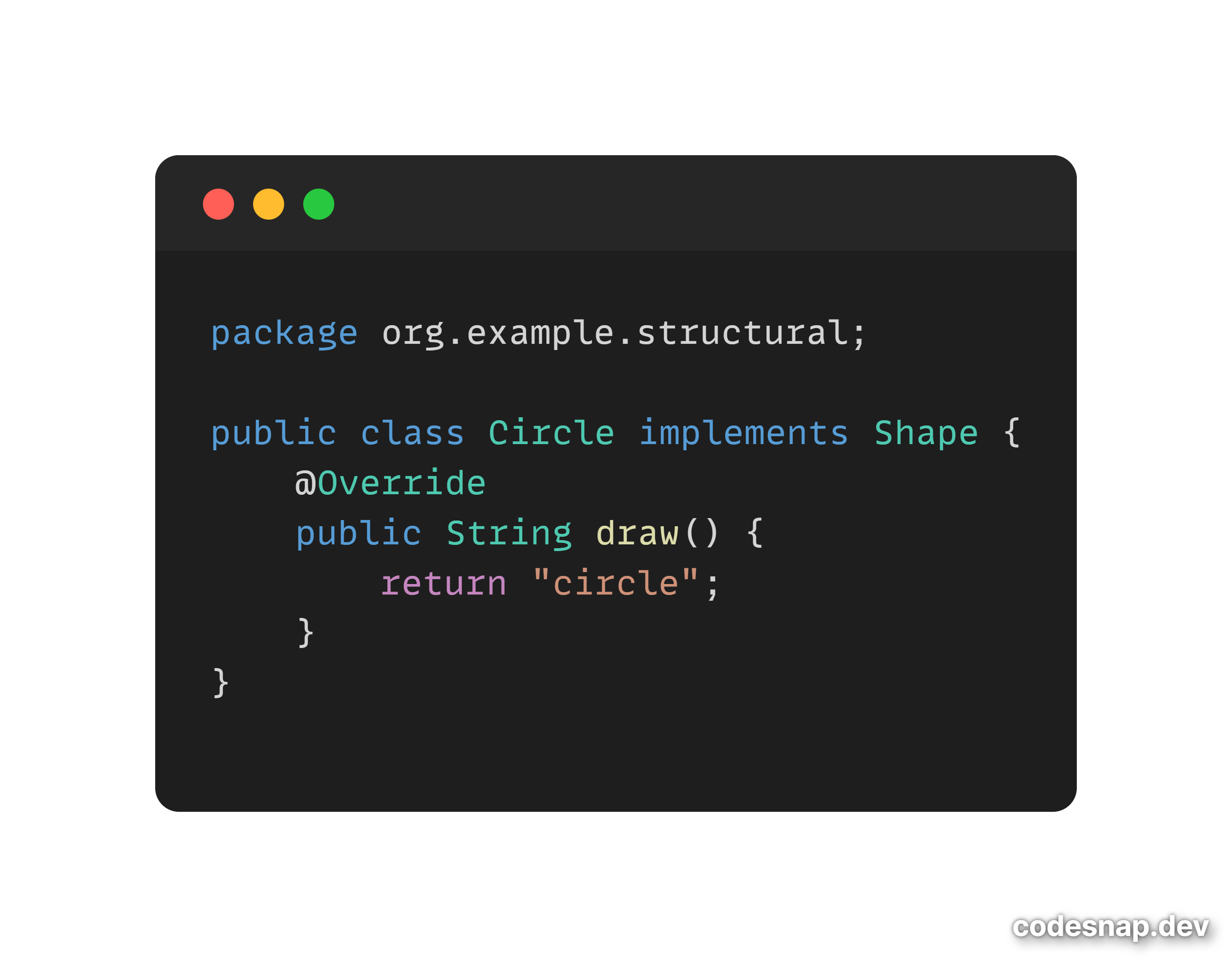


Рисунок 18 класс Circle

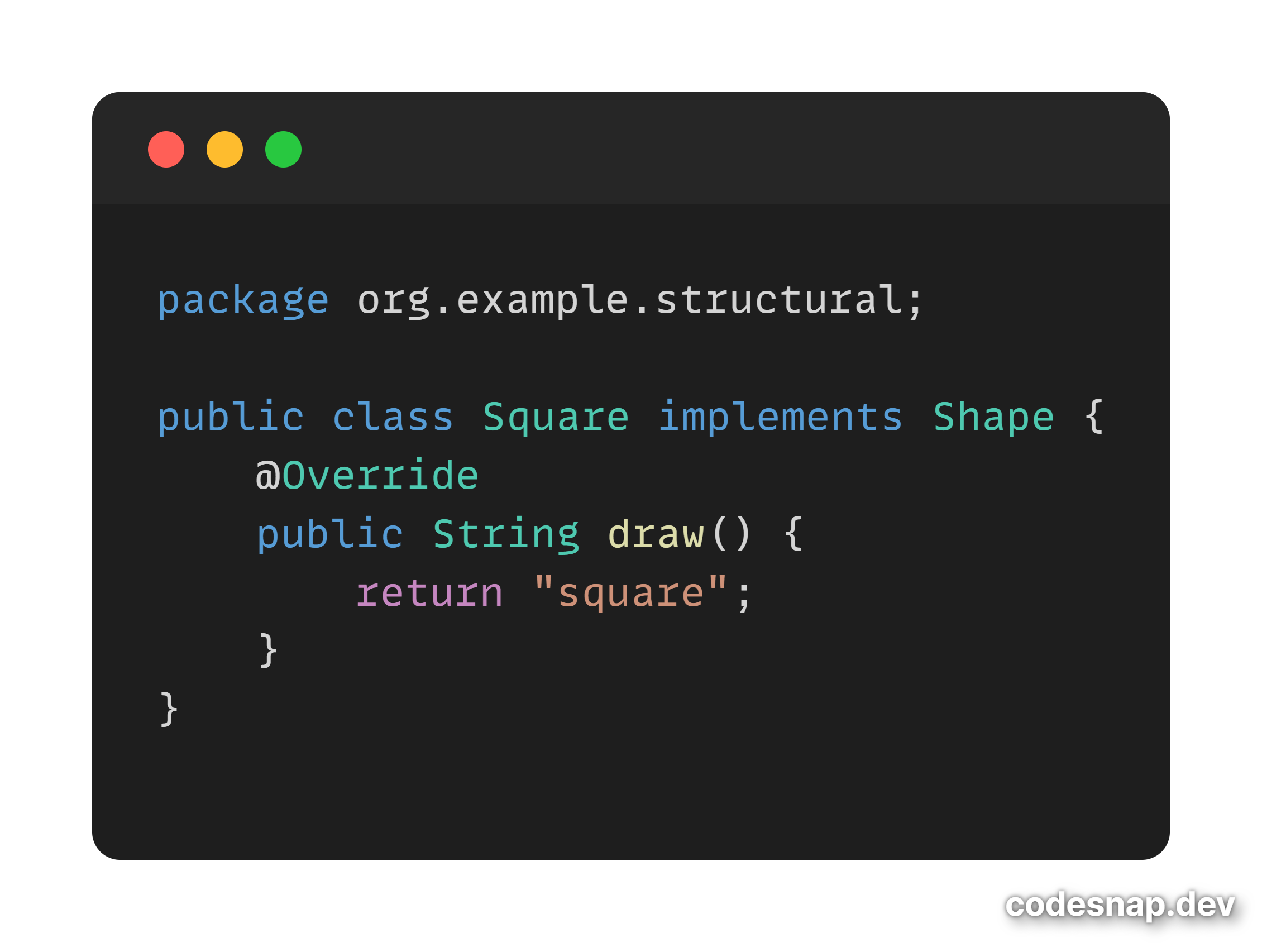


Рисунок 19 класс Square

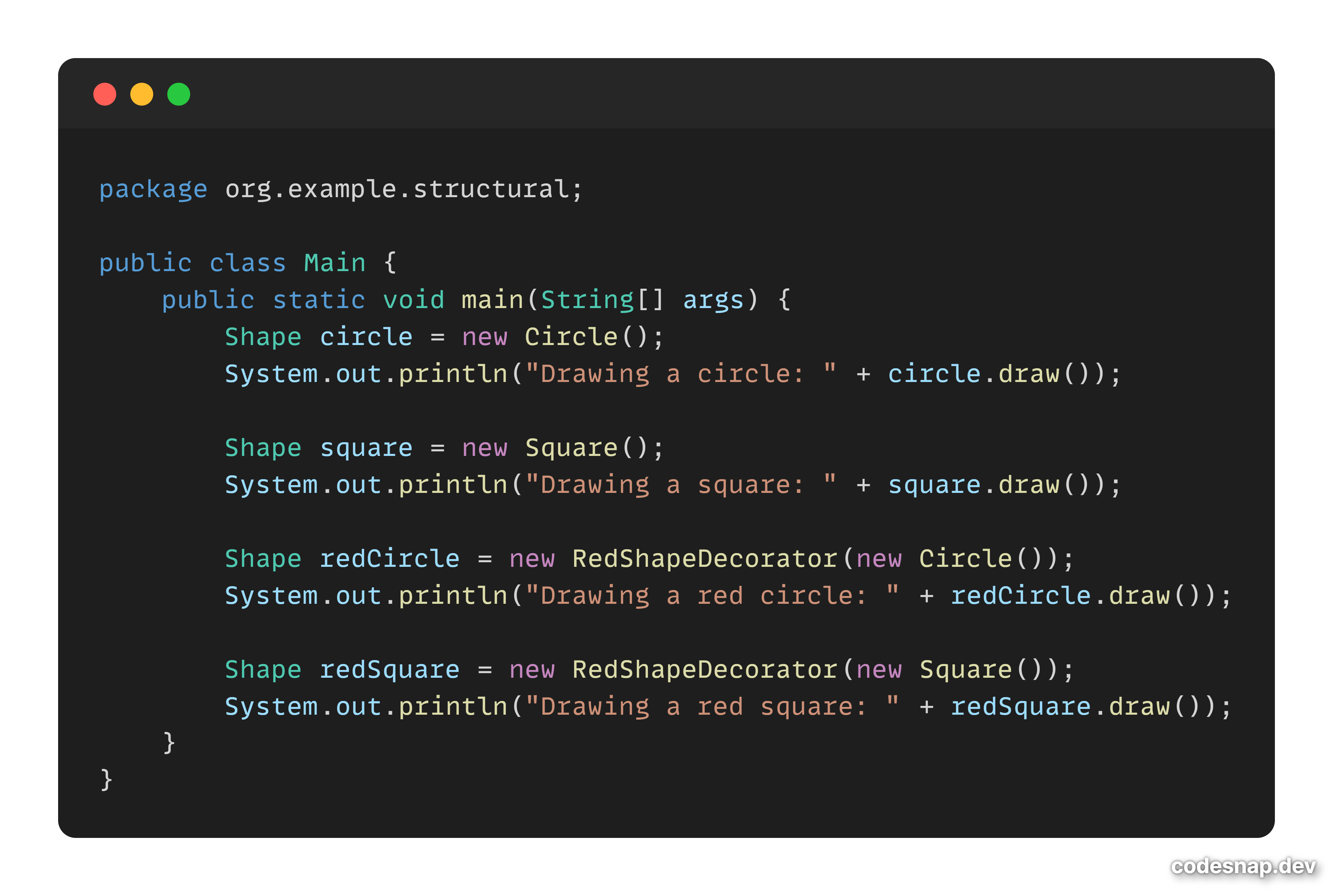


Рисунок 20 класс Main, иллюстрирующий использование шаблона

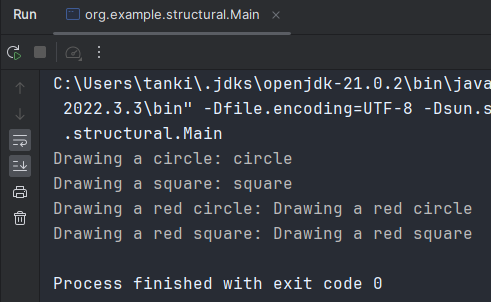


Рисунок 21 результат работы программы

# Вывод

В ходе практической работы познакомился с тремя различными шаблонами проектирования – Observer, Factory, Decorator. Изучил, для чего нужен каждый из них и когда следует использовать их.