

Абстракции и интерфейсы



Юрий
Пеньков



Юрий Пеньков

Java Software Engineer в InnoSTage





План занятия

1. [Принцип абстракции](#)
2. [Абстрактные классы](#)
3. [Интерфейсы](#)
4. [Множественное наследование через интерфейсы](#)



Принцип абстракции



Абстракция

Абстра́кция (лат. abstractio — отвлечение) — теоретическое обобщение...

Абстрактный класс как и обычный:

- содержит поля и методы

Но:

- нельзя создать экземпляр
- может содержать методы без реализации (абстрактные)

Figure

Вспомним класс `Figure` с прошлой лекции:

```
class Figure {  
    private Color fillColor;  
    public Color(fillColor) {  
        this.fillColor = fillColor;  
    }  
    public Color getFillColor() {  
        return fillColor;  
    }  
    public double getPerimeter() {  
        throw new NotImplementedException();  
    }  
}
```

- он обобщает свойства разных фигур;
- нет необходимости его создавать.

Сделаем его абстрактным!

abstract class Figure

Используем абстрактный класс и методы:

```
abstract class Figure {  
    private Color fillColor;  
    public Color(Color fillColor) {  
        this.fillColor = fillColor;  
    }  
    public Color getFillColor() {  
        return fillColor;  
    }  
    public abstract double getPerimeter();  
    public abstract double getArea();  
}
```

Абстрактные методы:

- не содержат реализации;
- **обязательно** должны быть переопределены в потомках;
- могут быть только внутри абстрактного класса.



Преимущества абстрактного класса

В этой ситуации абстрактный класс решает сразу несколько проблем прошлой реализации:

- не может быть создан, значит не будет объектов с нереализованными методами
- нет необходимости бросать исключения
- обязывает потомков переопределить методы
- гарантирует наличие переопределенного метода в классе-потомке

Интерфейсы

Интерфейс описывает поведение (набор методов) объекта, реализующего его (в отличие от класса, который может еще и хранить состояние).

Например, мы хотим добавить нашим фигурам возможность изменять масштаб:

```
public interface Scalable {  
    public void scale(int factor);  
}
```

Теперь любой класс, реализующий `Scalable`, обязан иметь метод `scale`.

Реализация интерфейса

Подготовим точку для использования в масштабируемых фигурах:

```
class Point implements Scalable{
    private int x, y;

    public void scale(int factor){
        x = x * factor;
        y = y * factor;
    }
}
```

Реализация интерфейса

Теперь попробуем сделать масштабируемый треугольник:

```
class Triangle extends Figure implements Scalable {  
    private Point a, b, c; // вершины треугольника  
    @Override  
    public void scale(int factor){  
        a = a.scale(factor);  
        b = b.scale(factor);  
        c = c.scale(factor);  
    }  
}
```


Будем считать, что сдвинув все три точки на указанный фактор мы корректно масштабируем треугольник относительно начала координат. :)

Использование интерфейсов

Реализация интерфейса классом говорит о том, что объекты этого класса можно использовать определенным образом. Это позволяет работать одинаково с объектами разных классов. Например:

```
public void figureResizer(Scalable figure) {  
    int factor = ...  
    //сложная логика, определяющая, какой фактор  
    использовать для этой фигуры  
    figure.scale(factor);  
}
```

Автор такого метода может быть уверен, что все объекты, которые попадут в метод, корректно реализуют `scale(...)`. Авторы этих объектов должны заботиться о корректности реализации сами.



Различия абстрактного класса и интерфейса

Абстрактные классы:

- могут хранить состояние (поля) и описывать поведение (методы);
- описывают сходные по свойствам классы;
- наследовать несколько абстрактных классов нельзя.

Интерфейсы:

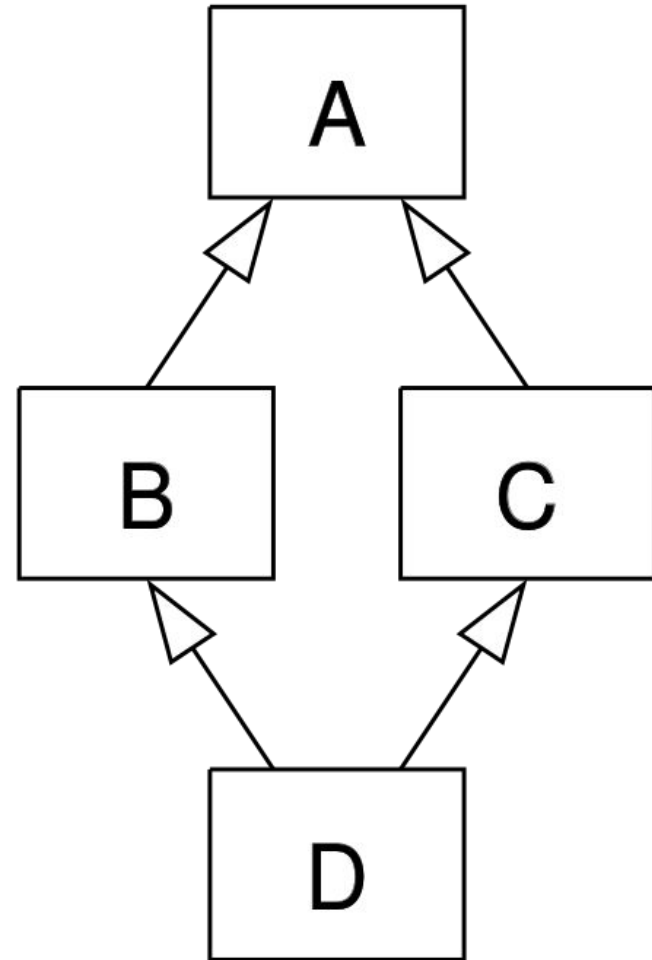
- описывают только поведение (методы);
- могут быть реализованы на абсолютно разных классах;
- класс может реализовывать несколько интерфейсов.

Множественное наследование

Java класс может наследоваться только от одного класса. Это сделано, чтобы избежать проблемы ромбовидного наследования:

Когда классы B и C наследуют класс A, и класс D наследуется от них обоих, возникает проблема неопределенности — методы в B и C могут быть переопределены по-разному.

Считается, что если в вашей программе требуется множественное наследование — нужно пересмотреть архитектуру и, возможно, использовать интерфейсы.





Чему мы научились

- делать абстрактные классы и методы;
- делать иерархию с абстрактными классами;
- делать интерфейсы и реализовывать их;
- узнали, почему множественное наследование — это не очень хорошо.



Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Юрий Пеньков