Абстракция

Выход из тени

- ~~Инкапсуляция~~
- ~~Наследование~~
- ~~Полиморфизм~~
- Абстракция <- вы тут

Абстракция — это принцип ООП, который помогает скрыть сложные детали реализации и предоставить простой интерфейс для работы с объектами.

Когда вы используете телевизор, вам не нужно знать, как он устроен внутри, чтобы смотреть шоу — вам достаточно знать, как пользоваться пультом/кнопками.

Этот принцип позволяет одному и тому же методу выполнять различные функции в зависимости от объекта, к которому он применяется.

Контрольная точка

- Понятно ли что такое абстракция?

Если все ясно, ставим плюсы, иначе - задаем вопросы.

Абстрактные классы и методы

```java
package l26;

// абстрактный класс
abstract class Animal {
    abstract void makeSound(); // Абстрактный метод

    void breathe() {
        System.out.println("Animal is breathing");
    }
}

class Dog extends Animal {
    void swim() {System.out.println("Swim");}

    @Override
    void makeSound() {
        System.out.println("Woof");
    }
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("Meow");
    }
}

abstract class AnimalInSky extends Animal {}
```

Абстрактный класс — это класс, который не может быть инстанциирован напрямую.

Это означает, что вы не можете создать объект абстрактного класса прямым вызовом конструктора.

Абстрактные классы часто содержат абстрактные методы — методы без тела, которые должны быть реализованы в подклассах.

Контрольная точка

- Понятно ли что такое абстрактный класс?
- Понятно ли что такое абстрактный метод?

Если все ясно, ставим плюсы, иначе - задаем вопросы.

Иерархии наследования с абстрактными классами

```java
package l26;

// абстрактный класс
abstract class Animal {
    abstract void makeSound(); // Абстрактный метод

    void breathe() {
        System.out.println("Animal is breathing");
    }
}

class Dog extends Animal {
    void swim() {System.out.println("Swim");}

    @Override
    void makeSound() {
        System.out.println("Woof");
    }
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("Meow");
    }
}

abstract class AnimalInSky extends Animal {}
```

Абстрактные классы лежат в основе иерархий наследования, позволяя создавать общие базовые классы, которые определяют шаблон для производных классов.

Подклассы абстрактных классов должны реализовать все абстрактные методы базового класса, обеспечивая таким образом соблюдение определенного контракта.

Иерархии наследования с абстрактными классами

```java
package l26;

// абстрактный класс
abstract class Animal {
    abstract void makeSound(); // Абстрактный метод

    void breathe() {
        System.out.println("Animal is breathing");
    }
}

class Dog extends Animal {
    void swim() {System.out.println("Swim");}

    @Override
    void makeSound() {
        System.out.println("Woof");
    }
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("Meow");
    }
}

abstract class AnimalInSky extends Animal {}
```

В этом примере классы Dog и Cat наследуются от абстрактного класса Animal и предоставляют свои реализации для абстрактного метода makeSound.

Таким образом, несмотря на то что мы не можем создать объект типа Animal напрямую, мы можем использовать его для обобщения и работы с объектами его подклассов.

Контрольная точка

- Понятно ли что такое иерархия наследования абстрактных классов?

Если все ясно, ставим плюсы, иначе - задаем вопросы.

Плюсы использования

```java
package l26;

// абстрактный класс
abstract class Animal {
    abstract void makeSound(); // Абстрактный метод

    void breathe() {
        System.out.println("Animal is breathing");
    }
}

class Dog extends Animal {
    void swim() {System.out.println("Swim");}

    @Override
    void makeSound() {
        System.out.println("Woof");
    }
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("Meow");
    }
}

abstract class AnimalInSky extends Animal {}
```

Упрощение сложности: абстракция в программировании позволяет скрыть детали реализации и сосредоточиться на ключевых аспектах системы. Это помогает упростить понимание и поддержку кода.

Модульность: возможность разбить систему на модули или классы, которые могут работать независимо друг от друга. Это способствует повторному использованию кода и улучшает масштабируемость проекта.

Повышение безопасности: абстракция позволяет скрыть некоторые детали реализации, что делает код более безопасным и защищенным. Внешние компоненты не имеют прямого доступа к внутренним деталям объекта или системы.

Вывод

```java
package l26;

// абстрактный класс
abstract class Animal {
    abstract void makeSound(); // Абстрактный метод

    void breathe() {
        System.out.println("Animal is breathing");
    }
}

class Dog extends Animal {
    void swim() {System.out.println("Swim");}

    @Override
    void makeSound() {
        System.out.println("Woof");
    }
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("Meow");
    }
}

abstract class AnimalInSky extends Animal {}
```

Абстрактные классы и методы являются мощными инструментами абстракции в ООП, позволяя создавать гибкие и масштабируемые иерархии классов.

Они помогают скрыть детали реализации и предоставить четкий интерфейс для работы с различными типами объектов, облегчая таким образом разработку и поддержку программного обеспечения.

Домашнее задание

Статьи: первая, вторая

Домашнее задание:

доделать домашку 24 и

Минимум: создайте иерархию классов для представления различных 5 геометрических фигур, используя абстрактные классы и методы.

Дополнительно: написать класс ShapeArrayList по аналогии с CarArrayList.java в разделе кода 21 занятия, заполнить его пятью уникальными объектами для кажого своего класса геометрических фигур (для этого нужно использовать циклы).

Дополнительное дополнение: написать модульные тесты с покрытием не менее 70% по методам

The end