Теория: сокрытие и переопределение

12 минут 0/4 проблемы решены Пропустить эту тему Начать практиковать

§1. Переопределение методов экземпляра

Java предоставляет возможность объявить метод в подклассе с тем же именем, что и метод в суперклассе. Это известно как **переопределение метода** . Преимущество переопределения состоит в том, что подкласс может дать свою собственную конкретную реализацию метода суперкласса.

Переопределение методов в подклассах позволяет классу наследовать от суперкласса, поведение которого является **«достаточно близким»,** а затем изменять это поведение по мере необходимости подкласса.

Методы экземпляра могут быть переопределены, если они наследуются подклассом. Переопределенный метод должен иметь то же имя, параметры (количество и тип параметров) и тип возвращаемого значения (или подкласс типа), что и переопределенный метод.

Пример. Вот пример переопределения.

```
class Mammal {

    public String sayHello() {
        return "ohlllalalalalalaoaoaoa";
    }
}

class Cat extends Mammal {

    @Override
    public String sayHello() {
        return "meow";
    }
}

class Human extends Mammal {

    @Override
    public String sayHello() {
        return "hello";
    }
}
```

Иерархия включает в себя три класса: `Mammal` , `Cat` и `Human` . У класса `Mammal` есть метод `sayHello` . Каждый подкласс переопределяет этот метод. `@Override` Аннотации указывает на то, что метод переопределяется. Эта аннотация необязательна, но полезна.

Давайте создадим экземпляры и вызовем метод.

```
Mammal mammal = new Mammal();
System.out.println(mammal.sayHello()); // it prints "ohlllalalalalalaoaoaoa"

Cat cat = new Cat();
System.out.println(cat.sayHello()); // it prints "meow"

Human human = new Human();
System.out.println(human.sayHello()); // it prints "hello"
```

Как видите, у каждого подкласса есть своя реализация метода `sayHello` .

Содержание:

Вы можете вызвать метод базового класса в переопределенном методе, используя ключевое слово `super`.

§2. Правила для переопределения методов

Существует несколько правил для методов подклассов, которые должны переопределять методы суперкласса:

- метод должен иметь то же имя, что и в суперклассе;
- аргументы должны быть точно такими же, как в методе суперкласса;
- возвращаемый тип должен быть того же типа или подтипа возвращаемого типа, объявленного в методе суперкласса;
- уровень доступа должен быть таким же или более открытым, чем уровень доступа переопределенного метода;
- закрытый метод не может быть переопределен, потому что он не наследуется подклассами;
- если суперкласс и его подкласс находятся в одном пакете, то закрытые для пакета методы могут быть переопределены;
- статические методы не могут быть переопределены.

Для проверки этих правил есть специальная аннотация `@Override`. Это позволяет вам знать, будет ли метод на самом деле **переопределен** или нет. Если по какой-то причине компилятор решит, что метод не может быть переопределен, он выдаст ошибку. Но, помните, аннотация не обязательна, она только для удобства.

§3. Запрещающий переопределение

Если вы хотите запретить переопределение метода, объявите его с ключевым словом `final`.

```
public final void method() {
    // do something
}
```

Теперь, если вы попытаетесь переопределить этот метод в подклассе, произойдет ошибка времени компиляции.

§4. Переопределение и перегрузка методов вместе

Напомним, что **перегрузка** - это функция, которая позволяет классу иметь более одного метода с одним и тем же именем, если их аргументы разные.

Мы также можем переопределить и перегрузить метод экземпляра в подклассе одновременно. Перегруженные методы не переопределяют методы экземпляров суперкласса. Это новые методы, уникальные для подкласса.

Следующий пример демонстрирует это.

```
class SuperClass {

    public void invokeInstanceMethod() {
        System.out.println("SuperClass: invokeInstanceMethod");
    }
}

class SubClass extends SuperClass {

    @Override
    public void invokeInstanceMethod() {
        System.out.println("SubClass: invokeInstanceMethod is overridden");
    }

    // @Override -- method doesn't override anything
    public void invokeInstanceMethod(String s) {
        System.out.println("SubClass: overloaded invokeInstanceMethod");
    }
}
```

Следующий код создает экземпляр и вызывает оба метода:

```
SubClass clazz = new SubClass();

clazz.invokeInstanceMethod();    // SubClass: invokeInstanceMethod() is over
ridden
clazz.invokeInstanceMethod("s"); // SubClass: overloaded invokeInstanceMetho
d(String)
```

Помните, что переопределение и перегрузка - это разные механизмы, но вы можете смешивать их вместе в одной иерархии классов.

§5. Скрытие статических методов

Статические методы не могут быть переопределены. Если у подкласса есть статический метод с той же сигнатурой (именем и параметрами), что и у статического метода в суперклассе, то метод в подклассе скрывает метод в суперклассе. Это полностью отличается от переопределения методов.

Вы получите ошибку во время компиляции, если у подкласса есть статический метод с той же сигнатурой, что и у метода экземпляра в суперклассе или наоборот. Но если методы имеют одинаковое имя, но разные параметры, проблем не должно быть.

100 пользователям понравилась эта теория. **1** не понравилось **А что насчет тебя?**

😍 🙂 😐 ☹ 😡

Начать практиковать

Показать обсуждение (5)