Урок 6

Продвинутое ООП

Углубленное изучение вопросов объектно-ориентированного программирования: наследование, полиморфизм

Оглавление

Основы наследования и полиморфизм

Одним из основополагающих принципов объектно-ориентированного программирования является наследование, используя которое, можно создать класс(суперкласс), который определяет какие-то общие характеристики. Затем этот общий класс может наследоваться другими, более специализированными классами(подклассами), каждый из которых будет добавлять свои особые характеристики. Подкласс наследует все члены, определенные в суперклассе, добавляя к ним собственные. Для реализации наследования используется ключевое слово extends в следующей форме:

```
class имя_подкласса extends имя_суперкласса
```

Пример создания класса Cat, который является подклассом класса Animal:

```
public class Animal {
    protected String name;

    public Animal() {
    }

    public Animal(String name) {
        this.name = name;
    }

    public void animalInfo() {
        System.out.println("Animal: " + name);
    }
}

public class Cat extends Animal {
    protected String color;

    public Cat(String name, String color) {
        this.name = name;
        this.color = color;
    }

    public void catInfo() {
        System.out.println("Cat: " + name + " " + color);
    }
}

public class MainClass {
    public static void main(String[] args) {
        Animal animal = new Animal("Animal");
        Cat cat = new Cat("Barsik", "White");
        animal.animalInfo();
        cat.animalInfo();
        cat.catInfo();
    }
}
```

Результат:

```
Animal: animal
Animal: Barsik
Cat: Barsik White
```

Подкласс Cat включает в себя все члены своего суперкласса Animal. Именно поэтому объект cat имеет доступ к методу animalInfo(), и в методе catInfo() возможна непосредственная ссылка на переменную name, как если бы они была частью класса Cat.
Несмотря на то что класс Animal является суперклассом для класса Cat, он в то же время остается полностью независимым и самостоятельным классом. То, что один класс является суперклассом для другого класса, совсем не исключает возможность его самостоятельного использования. Более того, один подкласс может быть суперклассом другого подкласса. Для каждого создаваемого подкласса можно указать только один суперкласс, в Java не поддерживается множественное наследование.

Ключевое слово super

У ключевого слова super имеются две общие формы. Первая форма служит для вызова конструктора суперкласса, вторая - для обращения к члену суперкласса, скрываемому членом подкласса. Из подкласса можно вызывать конструктор, определенный в его суперклассе, используя следующую форму ключевого слова super:

```
super(список_аргументов)
```

где список_аргументов определяет аргументы, требующиеся конструктору суперкласса. Вызов метода super() **всегда должен быть** первым оператором, выполняемым в конструкторе подкласса. Если в конструкторе подкласса явно не использовать super(), то автоматически первой строкой будет вызываться конструктор по умолчанию из суперкласса. Такая конструкция позволяет заполнять даже поля суперкласса с модификатором доступа private. Например:

```
public class Animal {
    private int a;
    protected int z;
    public Animal(int a) {
        this.a = a;
    }
}

public class Cat extends Animal {
    private int b;
    protected int z;
    public Cat(int a, int b) {
        super(a); // первым делом вызываем конструктор Animal
        this.b = b;
    }
    public void test() {
        z = 10; // Обращение к полю z класса Cat
        super.z = 20; // Обращение к полю z класса Animal
    }
}

public class SuperCat extends Cat {
    private int c;
    public SuperCat(int a, int b, int c) {
        super(a, b); // первым делом вызываем конструктор Cat
        this.c = c;
    }
}
```

Вторая форма применения ключевого слова super действует подобно ключевому слову this, за исключением того, что ссылка всегда делается на суперкласс. Вторая форма наиболее пригодна в тех случаях, когда имена членов подкласса скрывают члены суперкласса с такими же именами, в

примере выше поле z класса Cat, скрывает поле z суперкласса, поэтому для доступа к полю суперкласса используется запись super.z, то же справедливо и для методов.
Порядок вызова конструкторов. При вызове конструктора SuperCat будут по цепочке вызваны конструкторы родительских классов, начиная с самого первого класса.

```
SuperCat sc = new SuperCat();
// Animal() => Cat() => SuperCat()
```

Конструкторы вызываются в порядке наследования, поскольку суперклассу ничего неизвестно о своих подклассах, и поэтому любая инициализация должна быть выполнена в нем совершенно независимо от любой инициализации, выполняемой подклассом. Следовательно, она должна выполняться в первую очередь.

Переопределение методов

Если у супер- и подкласса совпадают имена и сигнатуры типов методов, то говорят, что метод из подкласса переопределяет метод из суперкласса. Когда переопределенный метод вызывается из своего подкласса, он всегда ссылается на свой вариант, определенный в подклассе. А вариант метода, определенный в суперклассе, будет скрыт. Рассмотрим следующий пример:

```
public class Animal {
    void voice() {
        System.out.println("Животное издало звук");
    }
}

public class Cat extends Animal {
    @Override
    void voice() {
        System.out.println("Кот мяукнул");
    }
}

public class MainClass {
    public static void main(String[] args) {
        Animal a = new Animal();
        Cat c = new Cat();
        a.voice();
        c.voice();
    }
}

Результат:
Животное издало звук
Кот мяукнул
```

Когда метод voice() вызывает объект типа Cat, выбирается вариант этого метода, определенный в классе Cat. Если при переопределении метода необходим функционал из этого метода суперкласса, можно использовать конструкцию *super.method()*. Например:

```
public class Cat extends Animal {
    @Override
    public void voice() {
        super.voice(); // вызываем метод voice() суперкласса
        System.out.println("Кот мяукнул");
    }
}
```

Переопределение методов выполняется только в том случае, если имена и сигнатуры типов обоих методов одинаковы. В противном случае оба метода считаются перегружаемыми. Переопределенные методы позволяют поддерживать в Java полиморфизм во время выполнения, он позволяет определить в общем классе методы, которые станут общими для всех производных от него классов, а в подклассах - конкретные реализации некоторых или всех этих методов.

Абстрактные классы и методы

Иногда суперкласс требуется определить таким образом, чтобы объявить в нем структуру заданной абстракции, не предоставляя полную реализацию каждого метода. Например, определение метода voice() в классе Animal служит лишь в качестве шаблона, поскольку все животные издают разные звуки, а значит нет возможности прописать хоть какую-то реализацию этого метода в классе Animal. Для этой цели служит абстрактный метод (с модификатором типа abstract). Иногда они называются методами под ответственностью подкласса, поскольку в суперклассе для них никакой реализации не предусмотрено, и они обязательно должны быть переопределены в подклассе.

```
abstract void voice();
```

Как видите, в этой форме тело метода отсутствует. Класс, содержащий хоть один абстрактный метод, должен быть объявлен как абстрактный. Нельзя создавать объекты абстрактного класса, поскольку он определен не полностью. Кроме того, нельзя объявлять абстрактные конструкторы или абстрактные статические методы. Любой подкласс, производный от абстрактного класса, обязан реализовать все абстрактные методы из своего суперкласса. При этом абстрактный класс вполне может содержать конкретные реализации методов. Пример:

```
public abstract class Animal {
    public abstract void voice();
    public void jump() {
        System.out.println("Животное подпрыгнуло");
    }
}

public class Cat extends Animal {
    @Override
    public void voice() {
        System.out.println("Кот мяукнул");
    }
}
```

Несмотря на то что абстрактные классы не позволяют получать экземпляры объектов, их все же можно применять для создания ссылок на объекты подклассов.

```
Animal a = new Cat();
```

Ключевое слово final в сочетании с наследованием

Существует несколько способов использования ключевого слова final:
Первый способ: создание именованной константы.

```
final int MONTHS_COUNT = 12; // final в объявлении поля или переменной
```

Второй способ: предотвращение переопределения методов.

```
public final void run() { // final в объявлении метода
}
```

Третий способ: запрет наследования от текущего класса.

```
public final class A { // final в объявлении класса
}
public class B extends A { // Ошибка, класс A не может иметь подклассы
}
```

Домашнее задание

1. Создать классы Собака и Кот с наследованием от класса Животное.
2. Животные могут выполнять действия: бежать, плыть, перепрыгивать препятствие. В качестве параметра каждому методу передается величина, означающая или длину препятствия (для бега и плавания), или высоту (для прыжков).
3. У каждого животного есть ограничения на действия (бег: кот 200 м., собака 500 м.; прыжок: кот 2 м., собака 0.5 м.; плавание: кот не умеет плавать, собака 10 м.).
4. При попытке животного выполнить одно из этих действий, оно должно сообщить результат в консоль. (Например, dog1.run(150); -> результат: run: true)
5. * Добавить животным разброс в ограничениях. То есть у одной собаки ограничение на бег может быть 400 м., у другой 600 м.

Дополнительные материалы

1. Кей С. Хорстманн, Гари Корнелл Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. - М.: Вильямс, 2014. - 864 с.
2. Брюс Эккель Философия Java // 4-е изд.: Пер. с англ. – СПб.: Питер, 2016. – 1168 с.
3. Г. Шилдт Java 8. Полное руководство // 9-е изд.: Пер. с англ. - М.: Вильямс, 2015. - 1376 с.
4. Г. Шилдт Java 8: Руководство для начинающих. // 6-е изд.: Пер. с англ. - М.: Вильямс, 2015. - 720 с.