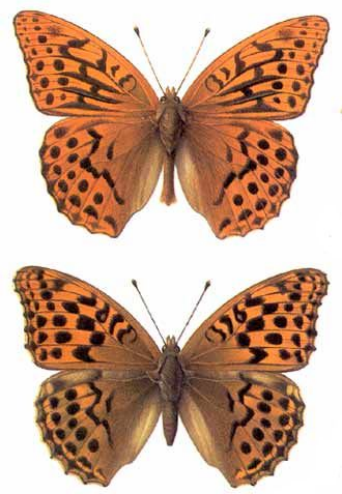
Основы программирования на языке Java

Занятие 7. Наследование и полиморфизм

Полиморфизм

[греч. ***poly*** — много и ***morphe*** — вид, форма, образ]



Полиморфизм (polymorphism)

- ✔ имеется несколько реализаций алгоритма
- ✔ выбор реализации осуществляется в зависимости от типа объекта и типа параметров

Механизмы реализации:

- ✔ ***Перегрузка (overload)*** метода
- ✔ ***Переопределение (override)*** метода

Перегрузка

Методы выполняют схожую функцию над разными типами данных.

```
public class Frog {
    public void eat(String type) {
        // Работа со строкой type
    }
    public void eat(int count) {
        // Работа с переменной типа int
    }
}
```

Переопределение

Методы предка и наследника могут быть одноименными, но выполнять разную функцию.

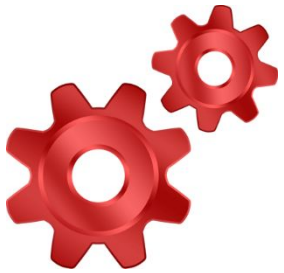
```
class Animal {
    protected void move() {
        // передвигается
    }
}
class Frog extends Animal {
    @Override
    protected void move() {
        // прыгает
    }
}

Frog greenFrog = new Frog();
// обращение к методу класса Frog
greenFrog.move();

Animal dog = new Animal();
// обращение к методу класса Animal
dog.move();
```

Абстрактный класс

- ✔ определяет общее поведение для порожденных им классов
- ✔ предполагает наличие дочерних классов
- ✔ объявляется со спецификатором ***abstract***
- ✔ не может иметь объектов
- ✔ может содержать или не содержать ***абстрактные методы***

Класс должен быть объявлен как абстрактный если:

1. класс содержит абстрактные методы
2. класс наследуется от абстрактного класса, но не реализует абстрактные методы
3. класс имплементирует интерфейс, но не реализует все методы интерфейса

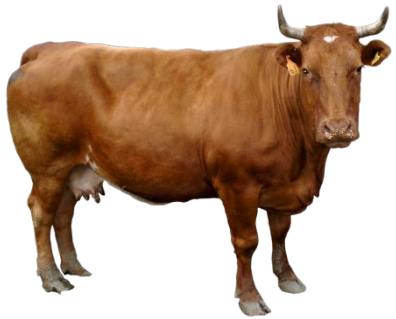


Абстрактный метод

- не имеет реализации
- объявляется со спецификатором ***abstract***
- переопределяется в дочерних классах

Пример

```
public abstract class Animal {
    private String type;
    abstract void getSound();
    Animal(String aType) {
        type = aType;
    }
    String getType() {
        return type;
    }
}
```



```
public class Cow extends Animal {
    Cow(String aType) {
        super(aType);
    }
    @Override
    void getSound(){
        System.out.println("Mu-mu");
    }
}
```

```
public class Cat extends Animal {
    Cat(String aType) {
        super(aType);
    }
    @Override
    void getSound() {
        System.out.println("myau-myau");
    }
}
```

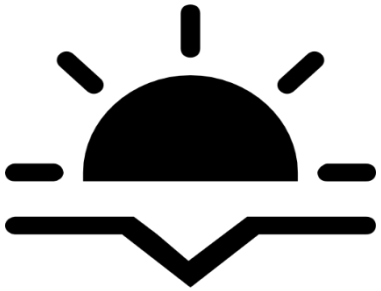
Связывание

процесс определения, какой именно метод надо вызывать

РАННЕЕ

выполняемое на этапе компиляции

Компилятор разбирается с ПЕРЕГРУЗКОЙ.



ПОЗДНЕЕ

выполняемое во время исполнения

Позднее связывание служит для того, чтобы разобраться с ПЕРЕОПРЕДЕЛЕНИМ

Пример 1

Вызов статического метода. Это метод класса, а не экземпляра, переопределить его **НЕЛЬЗЯ!**

```
public static class Parent{
    public void test(){
        System.out.println("parent::test");
    }

    public static void staticCall(){
        System.out.println("static call parent");
    }
}
```

```
public static class Child extends Parent{
    public void test(){
        System.out.println("child::test");
    }

    public static void staticCall(){
        System.out.println("static call child");
    }
}
```

Пример 1

```
public static void main(String[] args){
    Parent p = new Child();
    p.staticCall();
    p.test();
    Child c = new Child();
    c.staticCall();
    c.test();
}
```

Результат:

```
static call parent
child::test
static call child
child::test
```

Пример 2

Два типа связывания

```
public static class Parent{
    public void test(){
        System.out.println("parent::test");
    }
}

public static class Child extends Parent{
    public void test(){
        System.out.println("child::test");
    }
}

public static class Tester{
    public void test(Parent obj){
        System.out.println("Testing parent...");
        obj.test();
    }
    public void test(Child obj){
        System.out.println("Testing child...");
        obj.test();
    }
}
```

Пример 2

```
public static void main(String[] args){
    Parent obj = new Child();
    Tester t = new Tester();
    t.test(obj);
}
```

```
Testing parent...
child::test
```