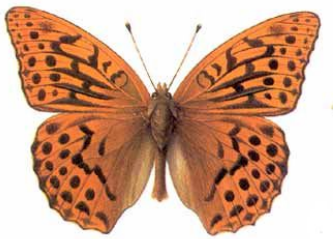
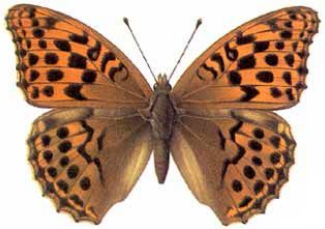
Полиморфизм в JAVA

Еще раз про ООП

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:

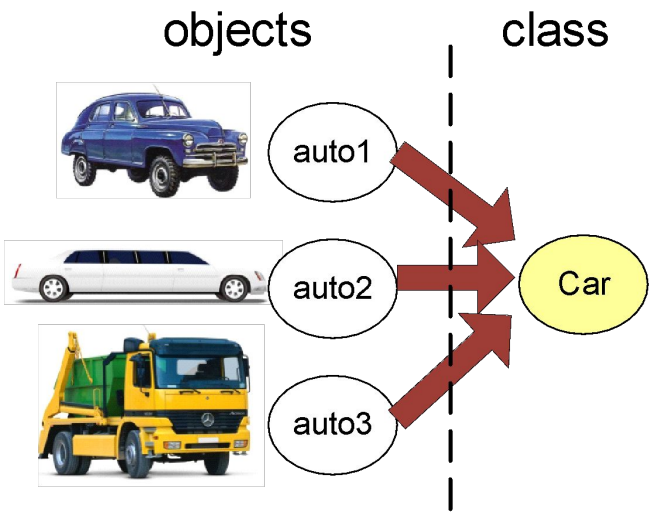
- ✔ ***Перегрузка (overload)*** метода
- ✔ ***Переопределение (override)*** метода

Еще раз про класс

- ***Класс*** (***class***) описывает признаки состояния и поведение множества схожих объектов
- Класс – это пользовательский ***тип данных***

```
class Car {
    String name;
    int speed;
    int fuel;

    void setName(String newName) {…}
    void speedUp(int delta) {…}
    void fillFuel(int delta) {…}
    void printCurrentState() {…}
}
```



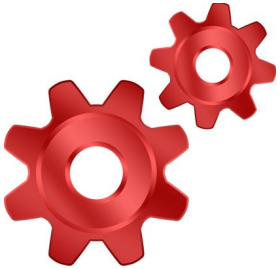

```
Car auto1 = new Car();
Car auto2 = new Car();
Car auto3 = new Car();
```

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

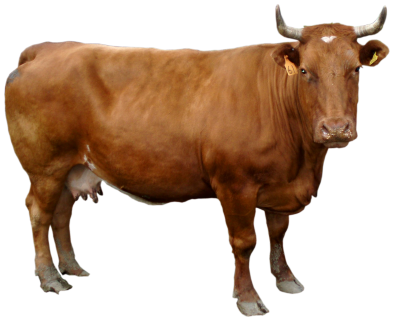


Абстрактный метод

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

Пример

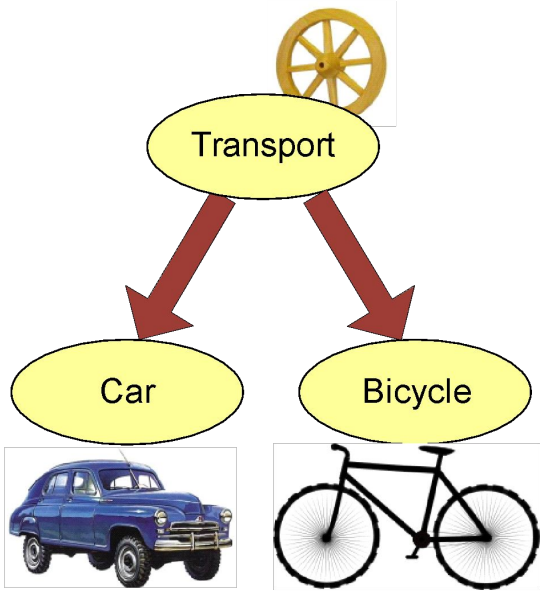
```java
public abstract class Animal {
    private String type;
    abstract void getSound();
    Animal(String aType) {
        type = aType;
    }
    String getType() {
        return type;
    }
}
```



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

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

ЗАДАЧА 1



Создать абстрактный класс Transport, и два класса наследника.

Абстрактный метод в Transport – **beep()**

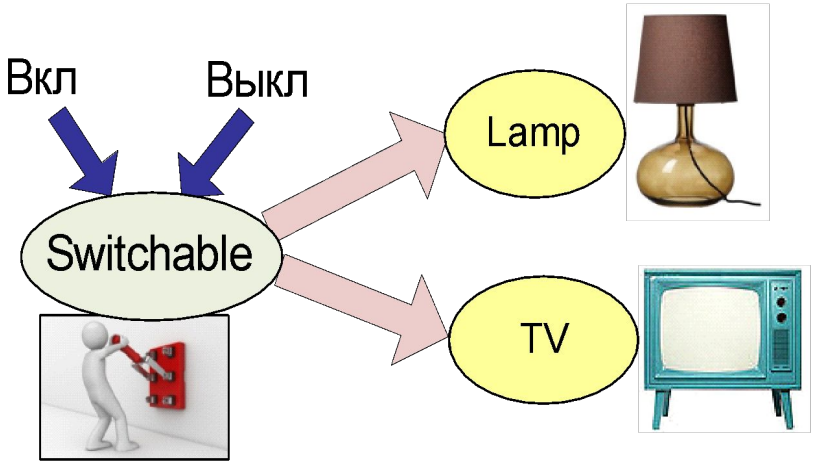
Интерфейс

Определяет возможное поведение объектов (описывает некоторое семейство типов и содержит лишь декларации операций)

- Интерфейс представляет собой совокупность методов без реализации
- При объявлении класса можно указать, какие интерфейсы он будет поддерживать

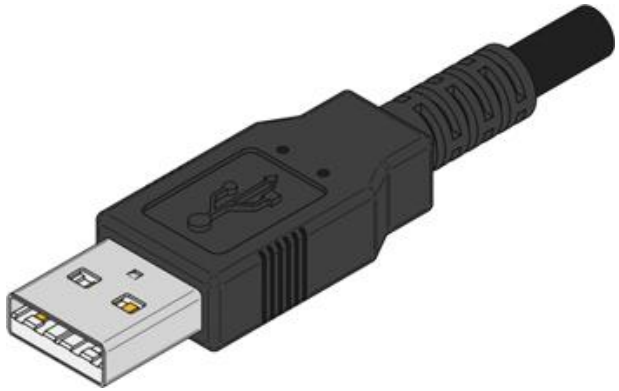
```
interface Switchable {
    void switchOn();
    void switchOff();
}

class Lamp
    implements Switchable {
    …
}
```



Объявление интерфейсов

- ✔ **Бывают:**
  - ✔ публичные (***public***)
  - ✔ непубличные – доступны внутри пакета
- ✔ **Могут содержать:**
  - ✔ абстрактные методы (методы без реализации)
  - ✔ статические константы
  - ✔ (Java SE 8) статические методы
  - ✔ (Java SE 8) *методы по умолчанию* (*default methods*) с реализацией



- ✔ **Все элементы являются публичными (*public*)**
  - ✔ все поля интерфейса являются *static* и *final*
- ✔ **Название**
  - – Имя интерфейса состоит из одного или нескольких идущих подряд слов
  - – Первая буква каждого слова заглавная, остальные буквы – в нижнем регистре
  - – Имя интерфейса обычно заканчивается на ‘*able*’

Имплементация интерфейсов

При объявлении класса можно указать, какие интерфейсы он будет поддерживать

Класс, реализующий интерфейс:

- ✔ может иметь свои собственные методы (не объявленные в интерфейсе)
- ✔ может иметь свои собственные поля
- ✔ должен реализовать все методы интерфейса, или объявляется как ***абстрактный*** (***abstract***)

```
public class NewClass
    implements Interface1, Interface2, Interface3  {
     ...
}
```

Пример

```java
public interface Computable {
    double compute();
}
```

```java
public class Summator implements Computable {
    private double x = 0;
    private double y = 0;
    public Summator(double nx, double ny) {
        x = nx;
        y = ny;
    }
    @Override
    public double compute() {
        return x+y;
    }
}
```

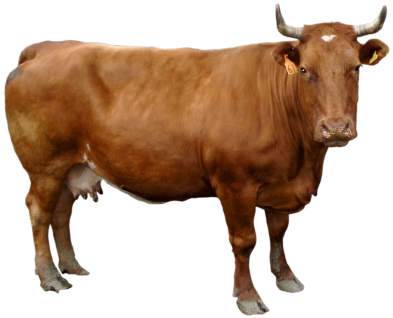
```java
public class Devider implements Computable {
    private double x = 0;
    private double y = 0;
    Devider(double nx, double ny) {
        x = nx;
        y = ny;
    }
    @Override
    public double compute() {
        return x/y;
    }
}
```

ЗАДАЧА 2

Создать интерфейс Animal Transport, и два класса Fish и Cow, которые будут имплементировать его.

Поля интерфейса: количество ног.

Методы интерфейса: say(), canSwim(), canRun(), CanFly().



Наследование интерфейсов

```
public interface Switchable {
      void switchOn();
      void switchOff();
}


public interface MediaPlayer extends Switchable {
      void play();
      void pause();
      void stop();
}

public class AudioPlayer implements MediaPlayer{
      @Override
      public void switchOn() {...}
      @Override
      public void switchOff(){...}
      @Override
      public void play(){...}
      @Override
      public void pause(){...}
      @Override
      public void stop(){...}
}
```

Abstract class vs Interface

Абстрактные классы

- описывают поведение **для иерархии классов**
- могут реализовывать алгоритмы
- могут содержать скрытые и защищенные элементы
- класс может наследоваться только от одного абстрактного класса

Интерфейсы

- описывают **поведение для группы классов**, реализующих данный интерфейс
- не могут реализовывать алгоритмы;
- содержат только публичные элементы
- класс может реализовывать несколько интерфейсов

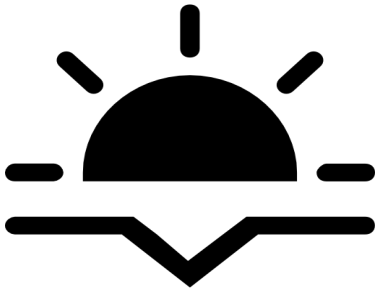
Связывание

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
```