# ЛАБОРАТОРНАЯ РАБОТА №6

# ИНТЕРФЕЙСЫ. АБСТРАКТНЫЕ КЛАССЫ И МЕТОДЫ

Цель: изучить механизм интерфейсов, узнать особенности абстрактных классов и методов

# Абстрактные методы

Абстрактные методы – это методы, которые не имеют тела и объявляются с ключевым словом abstract. Конструкторы и статические методы не могут быть абстрактными. Тело абстрактного метода определено в подклассах.

Если унаследовать класс, в котором определён хоть один абстрактный метод, и в потомках переопределить его, задав там описание данного метода, то для объектов классов потомков метод можно будет вызывать (и работать будут описанные в классах-потомках реализации).

# Абстрактные классы

Чтобы исключить возможность использования абстрактного метода, в Java введено следующее требование: класс, имеющий хоть один абстрактный метод, обязан быть абстрактным классом.

Абстракция оказывается полезной, когда некоторое поведение характерно для большинства или всех объектов данного класса, но некоторые аспекты имеют смысл лишь для ограниченного круга объектов, не составляющих суперкласса. В Java такие классы объявляются с ключевым словом abstract, и каждый метод, не реализованный в классе, также объявляется abstract.

Абстрактный класс – это класс, который имеет хотя бы один абстрактный метод и объявляется с ключевым словом abstract. Абстрактные методы данного класса должны быть обязательно реализованы наследниками. Абстрактный класс не может использоваться непосредственно для порождения объектов. Для этого необходимо, используя этот класс как базовый, породить другой класс, в котором нужно определить все абстрактные методы. Тогда можно будет создавать объекты.

Абстрактные классы и методы наиболее полезны в применении, когда базовый класс является лишь шаблоном для наследников, и порождение объектов от него нецелесообразно.

Например, вряд ли можно завести себе домашнее животного неопределенного вида, то есть у вас дома вполне могли бы жить Cat, Dog или даже Snake, но вряд ли вы бы смогли завести животное Pet, являющееся непонятно кем. Соответсвенно, в рамках реальной задачи вряд ли потребуется создавать объекты класса Pet, а значит его можно сделать абстрактным.

Рассмотрим пример с участием абстрактного класса и абстрактного метода:

Пример 5.1 Использование абстрактных классов и методов

abstract class Pet {  
    String name;  
    int age;  
    boolean hungry;  
    abstract void voice(); //абстрактный метод, который должен быть реализован в наследниках класса Pet  
    void food() {  
        hungry = false;  
    }  
}

class Snake extends Pet {  
    double length; // дополнительное поле length в наследнике  
    void voice() {  
        System.out.println("Шшш-ш-ш"); //собственная реализация абстрактного метода в классе-наследнике Snake  
    }  
}

class Dog extends Pet {  
    void voice() {  
        System.out.println("Гав-гав"); //собственная реализация абстрактного метода в классе-наследнике Dog  
    }  
}

class PatrolDog extends Dog {  
    void voice() {  
        System.out.println("Ррр-р-р");//переопределение метода

    }  
}

public class Main {  
    public static void main(String[] args) {  
        //Pet zorka = new Pet();

// ошибка, т. к. НЕЛЬЗЯ СОЗДАВАТЬ ОБЪЕКТЫ АБСТРАКТНОГО КЛАССА!!!  
        Snake lara= new Snake();  
        lara.voice();  
    }  
}

# Интерфейсы

Интерфейс – это ссылочный тип, который содержит лишь абстрактные методы (public, abstract) и константные поля (static, final). Класс, реализующий интерфейс, должен реализовывать все его абстрактные методы, иначе данный класс объявляют как abstract. Интерфейсы также могут наследовать друг друга. При объявлении интерфейса может вовсе не указываться модификатор доступа, а если указывается, то только public. Нельзя создавать объект интерфейса, а можно - лишь интерфейсную ссылку.

Один интерфейс может быть наследником другого интерфейса. Классы могут реализовывать интерфейсы (т. е. получать от интерфейса список методов и описывать реализацию каждого из них), притом, что один класс может реализовывать сразу несколько интерфейсов.

Перед описанием интерфейса указывается ключевое слово interface. Когда класс реализует интерфейс, то после его имени указывается ключевое слово implements и далее через запятую перечисляются имена тех интерфейсов, методы которых будут полностью описаны в классе.

Объявление интерфейса:

[public] interface InterfaceName [extends Interface1]{

абстрактные методы, константные поля

}

Объявление класса, реализующего интерфейсы:

[Доступ] Class ClassName implements I1, I2, ….In{

реализация абстрактных методов интерфейсов I1, I2, ….In , собственные поля и методы

}

В Java нет механизма множественного наследования как в С++, однако его заменяет механизм интерфейсов, т. к. один класс сразу может реализовать несколько интерфейсов.

Рассмотрим пример, где реализовано два интерфейса с методами доступными для грузового и для легкового транспорта. Класс Pickup (пикап) должен обладать как возможностью перевозки грузов, так и пассажиров, поэтому он реализует сразу оба интерфейса:

Пример 5.2 Реализация классом одного или сразу нескольких интерфейсов

interface PassangersAuto {  
    void transportPassangers();  
}  
interface CargoAuto {  
    void transportCargo();  
}  
class Truck implements CargoAuto {  
    final static int a = 1;  
    public void transportCargo() {  
        System.out.println("Везу груз");  
    }  
}  
class Sedan implements PassangersAuto {  
    public void transportPassangers() {  
        System.out.println("Везу пассажиров");  
    }  
}  
class Pickup implements CargoAuto, PassangersAuto {  
    public void transportCargo() {  
        System.out.println("Везу груз");  
    }  
    public void transportPassangers() {  
        System.out.println("Везу пассажиров");  
    }  
}

Часто всю открытую часть класса (т. е. общедоступные методы) предопределяю как раз в интерфейсе. Тогда, взглянув на один лишь интерфейс, можно понять, какие методы должны использоваться для взаимодействия с объектами данного класса. То есть интерфейсы вполне соответствуют принципам инкапсуляции. Как, впрочем, и принципу полиморфизма. Ведь в нескольких классах метод некоторого интерфейса может быть реализован по-разному, хотя и с одним и тем же именем.

Однако механизм интерфейсов имеет и свои недостатки. Если класс реализует несколько интерфейсов, в которых есть одноимённые методы, то в нём может быть задана лишь одна общая для всех реализация этих методов (и это уже ограничивает полиморфизм при множественном наследовании через интерфейсы в Java).

Пример 5.3 Недостатки интерфейсов

interface Interface1 {  
    int someField = 100;  
    String someMethod();  
}  
interface Interface2 {  
    int someField = 200;  
    String someMethod();  
}  
class SomeClass implements Interface1, Interface2 {  
    public String someMethod() {  
        return "It Works";  
    }  
}  
public class Main {  
    public static void main(String[] args) {  
        SomeClass a = new SomeClass();  
        System.out.println( a.someMethod() ); // It works  
        System.out.println( a.someField ); // ошибка  
        System.out.println( ( (Interface1) a).someField ); // 100  
        System.out.println( Interface1.someField ); // 100  
    }  
}

Пример с домашними животными также можно представить при помощи интерфейсов, немного усложнив структуру.

Пример 5.4 Использование интерфейсов, абстрактных классов и методов

package pr;

public interface Pet {

boolean *hungry*=false; //т. к в интерфейсе могут использоваться только константные поля, поэтому другие поля были исключены

abstract void voice(); //абстрактный метод, который должен быть реализован в наследниках класса Pet

}

public interface Object {

void print (); // интерфейс содержит лишь один абстрактный метод

}

public class Snake implements Pet, Object {

String name;

int age;

int length; // поля

public void voice() {

System.*out*.println("Шшш-ш-ш"); //собственная реализация абстрактного метода в классе Snake

}

public void print() {

System.*out*.println("Змея");

System.*out*.println("Имя: "+getName()+" возраст: "+getAge()+" длина: "+getLength());

System.*out*.println(" голодная?: "+*hungry*); // не используется get-метод, т. к. Поле *hungry* константное

}

public String getName(){

return name;

}

public int getAge(){

return age;

}

public int getLength(){

return length;

}

public Snake(){ //конструктор класса Snake

this.name="Lara";

this.age=13;

this.length=2;

}

}

public abstract class Dog implements Pet, Object { // класс абстрактный, т. к. только один абстрактный метод реализован

String name;

int age;

public abstract void voice(); // нереализованный абстрактный метод

public void print(){

System.*out*.println("Cобака");

System.*out*.println("Имя: "+getName()+" возраст: "+getAge());

System.*out*.println(" голодная?: "+*hungry*);

} // реализованный абстрактный метод

public String getName(){

return name;

}

public int getAge(){

return age;

}

public Dog(String nn){

this.name=nn;

this.age=10;

} // констуктор класса Dog

}

public class SimpleDog extends Dog implements Pet, Object{

public SimpleDog(String nn) {

super(nn);

}// вызываем конструктор супер-класса

public void voice() {

System.*out*.println("Ррр-р-р"); //реализация абстрактного метода

}

}

public class PatrolDog extends Dog implements Pet, Object{

public PatrolDog(String nn) {

super(nn);

// вызываем конструктор супер-класса

}

public void voice() {

System.*out*.println("Гав-гав"); //реализация абстрактного метода

}

}// абстрактный метод print(); не переопределяется в классах PatrolDog и SimpleDog, он реализован в базовом классе Dog и просто наследуется данными методами

public class PetDemo {

public static void main(String[] args) {

Snake s=new Snake();

s.print();

s.voice();

SimpleDog sd=new SimpleDog("Bobik");

sd.print();

sd.voice();

PatrolDog pd=new PatrolDog("Pirat");

pd.print();

pd.voice();

}

}

Результат выполнения программы:

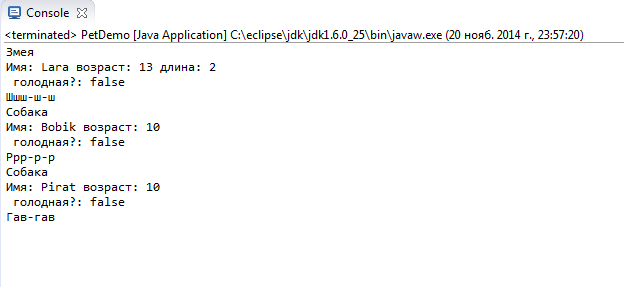


Рис. 5.1. Результат выполнения программы

# Пример выполнения индивидуального задания

Родительский класс требуется заменить одноимённым интерфейсом. Метод, ранее определённый в супер-классе, записать в интерфейс в качестве абстрактного. Добавить интерфейс Object, в котором объявить абстрактный метод print(). Далее создать 3 класса, реализующих оба интерфейса. Один из этих классов нужно представить как абстрактный (не реализовать абстрактные методы). Нереализованные методы реализуются в двух классах-наследниках абстрактного класса.

Пример 4.5 Пример выполнения индивидуального задания

Public interface Object { // интерфейс, содержащий один абстрактный метод

Void printInfo();

}

Public interface GeomFig { // интерфейс, содержащий один абстрактный метод

Public float FindSquare ();

}

Public class Circle implements Object, GeomFig {// класс, реализующий 2 интерфейса

Private double rad;

public Circle(double rad) {

this.rad=rad;

}// конструктор класса Circle

Private double GetRad() {

Return rad;

}// гет-метод для радиуса

Public float FindSquare() {

return (float) (3.14\*rad\*rad);

}// реализация абстрактного метода для нахождения площади круга

Public void printInfo() {

System.*out*.println("Радиус - "+GetRad()+" Площадь - "+FindSquare());

}// реализация абстрактного метода вывода информации об объекте класса для круга

}

Public class Rectangle implements Object, GeomFig {// класс, реализующий 2 интерфейса

private double a, b;

public Rectangle(int a, int b){

this.a=a;

this.b=b;

}// конструктор класса

Public double GetA() {

return a;

}// гет-метод для возврата значения одного из параметров прямоугольника

Public double GetB() {

return b;

}// гет-метод для возврата значения одного из параметров прямоугольника

Public float FindSquare() {

return (float) (a\*b);

}//реализация абстрактного метода нахождения площади фигуры для прямоугольника

Public void printInfo() {

System.*out*.println("Сторона1 - "+GetA()+" Сторона2 - "+GetB()+" Площадь - "+FindSquare());

}

}//реализация абстрактного метода вывода информации об объекте класса

Public abstract class Triangle implements Object, GeomFig {// класс объявлен как абстрактный, так как в нём не реализованы абстрактные методы реализуемых интерфейсов. ОНИ ДОЛЖНЫ БЫТЬ РЕАЛИЗОВАНЫ В КЛАССАХ-НАСЛЕДНИКАХ!!!

Public double x, y, z;

String type;

public Triangle (double x, double y, double z,String type) {

this.x=x;

this.y=y;

this.z=z;

this.type=type;

}// конструктор супер-класса

Public double GetX() {

Return x;

}//гет-метод

Public double GetY() {

Return y;

}//гет-метод

Public double GetZ() {

Return z;

}//гет-метод

Public abstract String GetType();

//абстрактный гет-метод

}

Public class RightAngledTriangle extends Triangle {// класс-наследник класса треугольник, который реализует 2 интерфейса

Public RightAngledTriangle (double a, double b, double c, String type) {

super(a, b, c, type);//вызов конструктора супер-класса

}//конструктор класса

Public float FindSquare() {

return (float) ((x\*y)/2);

}//реализация абстрактного метода поиска площади для данного вида треугольника

Public void printInfo() {

System.*out*.println("Катет1 - "+GetX()+" Катет2 -"+GetY()+" Гипотенуза - "+GetZ()+" Тип - " +GetType()+" Площадь - "+FindSquare());

}// реализация метода вывода информации для прямоугольного треугольника

public String GetType(){

return type;

}// реализация абстрактного метода, возвращается тип треугольника

}

Public class UniversalTriangle extends Triangle{

//класс-наследник треугольника, реализующий абстрактные методы, нереализованные в супер-классе

UniversalTriangle(double x, double y, double z, String type) {

super(x, y, z, type);//вызов конструктора супер-класса

}//конструктор класса

Public float FindSquare() {

return (float) Math.*sqrt*(((x+y+z)/2)\*(((x+y+z)/2)-x)\*(((x+y+z)/2)-y)\*(((x+y+z)/2)-z));

}// реализация абстрактного метода

Public void printInfo() {

System.*out*.println("Сторона1 - "+GetX()+" Сторона2 -"+GetY()+" Сторона3 - "+GetZ()+" Тип - " +GetType()+" Площадь - "+FindSquare());

}//реализация абстрактного метода

public String GetType(){

return type;

}// реализация абстрактного гет-метода

}

Public class EquilateralTriangle extends Triangle{//класс-наследник класса треугольник

Public EquilateralTriangle (double x, double y, double z, String type) {

super(x, y, z, type);//вызов конструктора супер-класса

}//конструктор класса

Public float FindSquare() {

return (float) (x\*x\*(Math.*sqrt*(3))/4);

}//реализация абстрактного метода нахождения площади для равностороннего треугольника

Public void printInfo() {

System.*out*.println("Сторона1 - "+GetX()+" Сторона2 -"+GetY()+" Сторона3 - "+GetZ()+" Тип - " +GetType()+" Площадь - "+FindSquare());

}// реализация абстрактного метода для вывода информации о равностороннем треугольнике

public String GetType(){

returntype;

}// реализация абстрактного гет-метода

}

import java.util.Scanner;

public class InterfaceDemo {

public static void main(String[] args) {

Circle circ1= new Circle(2);

circ1.printInfo();

Rectangle rect1=new Rectangle(3,4);

rect1.printInfo(); //создание объектов круг и прямоугольник, вызов их методов

System.*out*.println("Для создания треугольника:");

Scanner scan = new Scanner(System.*in*);

Double i = 0;

System.*out*.println("Введитесторону1/катет1:");

if(scan.hasNextDouble()) {

i = scan.nextDouble();

}

Double j = 0;

System.*out*.println("Введите сторону2/катет2:");

if(scan.hasNextDouble()) {

j = scan.nextDouble();

}

Double k = 0;

System.*out*.println("Введите сторону3/гипотенузу:");

if(scan.hasNextDouble()) {

k = scan.nextDouble();

}

if(i>=j+k||j>=i+k||k>=i+j) {

System.*out*.println("Такого треугольника не существует");

}// проверка треугольника на существование, вывод сообщения

else {

if (i==j&&i==k) {

System.*out*.println("Вы создали равносторонний треугольник");

EquilateralTriangle trian1= newEquilateralTriangle(i, j, k, "равносторонний");

trian1.printInfo();

}

else{

if(k\*k==i\*i+j\*j) {

System.*out*.println("Вы создали прямоугольный треугольник");

RightAngledTriangle trian1= newRightAngledTriangle(i,j,k,"прямоугольный");

trian1.printInfo();

}

else {

System.*out*.println("Вы создали треугольник");

if(i==j||i==k||j==k){

UniversalTriangle trian1= newUniversalTriangle(i,j,k,"равнобедренный");

trian1.printInfo();

}

else {

UniversalTriangle trian1= newUniversalTriangle(i,j,k,"разносторонний");

trian1.printInfo();

}

}

}//создание треугольника правильного вида, согласно введённым пользователем сторонам, вызов его методов

}

}

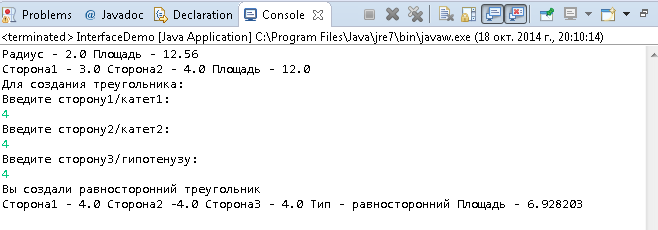


Рис. 5.2 Результат выполнения программы

# Задания для самостоятельного выполнения

Лабораторная работа № 4 выполняется на основе лабораторной работы №3. Родительский класс требуется заменить одноимённым интерфейсом. Метод, ранее определённый в супер-классе, записать в интерфейс в качестве абстрактного. Добавить интерфейс Object, в котором объявить абстрактный метод print(). Далее создать 3 класса, реализующих оба интерфейса. Один из этих классов нужно представить как абстрактный (не реализовать абстрактные методы). Нереализованные методы реализуются в двух классах-наследниках абстрактного класса.

Задание выполняется в соответствие со схемой, вместо букв латинского алфавита студент подставляет данные, согласно своему варианту индивидуального задания.

Интерфейс

А

Абстрактный класс

B

Реализующий класс

E

Реализующий класс G

Интерфейс, описывающий какой-то Объект

interface Object{

void print();}

Производный класс

C

Производный класс D

Рис. 5.3 Схема для выполнения индивидуального задания

**Вариант 1** A – Cлужащий, B - Рабочий, C - Токарь, D - Слесарь, E – Начальник завода, G – Директор завода.

**Вариант 2** A – Аудитория , B - Лаборатория, C – Химическая лаборатория, D - Компьютерная лаборатория, E – Учебный класс , G – Лекционная аудитория.

**Вариант 3** A - Книга, B –Художественная книга , C – Сборник стихов, D - Роман , E – Научная книга , G - Справочник.

**Вариант 4** A - Одежда, B - Куртка, C - Пуховик, D - Ветровка, E - Кофта, G - Брюки.

**Вариант 5** A - Посуда, B - Тарелка, C – Салатница, D - Блюдце, E - Кастрюля , G -Сковорода .

**Вариант 6** A - Человек , B - Родитель , C - Мать , D - Отец, E - Ребёнок, G - Подросток.

**Вариант 7** A - Автомобиль, B – Легковой автомобиль, C - Минивен, D - Пикап, E – Грузовой автомобиль, G - Автобус.

**Вариант 8** A - Обувь, B - Босоножки, C - Вьетнамки, D - Танкетки, E - Сапоги, G - Кроссовки.

**Вариант 9** A - Учащийся, B - Студент, C - Очник, D - Заочник , E - Школьник , G - Аспирант.

**Вариант 10** A - Мебель, B - Стул, C - Табуретка, D – Барный стул, E - шкаф, G - стол.

**Вариант 11** A - Персонаж, B -Монстр, C – Медленный монстр (босс), D – Быстрый монстр, E - Воин , G - Маг .

**Вариант 12** A - Фильм, B - Мультфильм, C – Кукольный мультфильм, D – Рисованный мультфильм, E - Мюзикл, G – Исторический фильм.

**Вариант 13** A – Молочный продукт, B - Йогурт, C – Питьевой йогурт, D – Йогурт в стаканчике, E - Творог, G - Кефир.

**Вариант 14** A - Игра, B - Стратегия, C – Карточная стратегия, D – Пошаговая стратегия, E – Шутер, G - Квест.

**Вариант 15** A - Сайт, B - Образовательный, C – Сайт по химии, D –Сайт по метаматематике, E - Информационный, G – Социальная сеть.

# Контрольные вопросы

1. Дайте определение абстрактного метода и абстрактного класса?
2. Можно ли создать объект абстрактного класса?
3. Запишите шаблон объявления интерфейса
4. В чём отличие интерфейса от абстрактного класса, какими общими свойствами они обладают?
5. Приведите пример интерфейса и реализующих его классов
6. Что в Java заменяет механизм множественного наследования?
7. Какие недостатки имеет механизм интерфейсов?