

Конспект видеокурса

"Java для начинающих."

https://www.youtube.com/playlist?list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak

Автор видеокурса: **alishev**

<https://www.youtube.com/channel/UCK5d3n3kfkzIArMccS0TTXA>

Составил конспект: **dragosh**

Москва

август 2018

Урок 1: JDK и Hello World.

https://www.youtube.com/watch?v=ziOQ8wkmnSE&index=1&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println ("Hello World!");  
    }  
}
```

ВЫВОД В КОНСОЛЬ:

Hello World!

Урок 2: Переменные. Примитивные типы данных.

https://www.youtube.com/watch?v=lmKl5qB_r70&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=2

```
public class Variables {  
    public static void main(String[] args) {  
        byte myByte;  
        myByte = 16;  
        System.out.println ("myByte - " + myByte);  
  
        short myShort = 211;  
        System.out.println ("myShort - " + myShort);  
  
        int myInt = 557;  
        System.out.println ("myInt - " + myInt);  
  
        long myLong = 234546677L;  
        System.out.println ("myLong - " + myLong);  
  
        float myFloat = 234.0F;  
        System.out.println ("myFloat - " + myFloat);  
  
        double myDoble = 234.0;  
        System.out.println ("myDoble - " + myDoble);  
  
        boolean myBoolean = true;  
        System.out.println ("myBoolean - " + myBoolean);  
  
        char myChar = 'c';  
        System.out.println ("myChar - " + myChar);  
    }  
}
```

ВЫВОД В КОНСОЛЬ:

```
myByte - 16
myShort - 211
myInt - 557
myLong - 234546677
myFloat - 234.0
myDoble - 234.0
myBoolean - true
myChar - c
```

Урок 3: Строки(String) в Java. Ссылочные ТИПЫ ДАННЫХ.

https://www.youtube.com/watch?v=-YK8B4WO7nI&index=3&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak

```
public class Strings {
    public static void main(String[] args) {
        String string = "Hello";
        String space = " ";
        String name = "Bob";
        System.out.println (string + space + name);
        System.out.println ("Hello" + " " + "John");

        int x = 10;
        System.out.println ("My number is " + x);
    }
}
```

ВЫВОД В КОНСОЛЬ:

```
Hello Bob
Hello John
My number is 10
```

Урок 4: Цикл while.

https://www.youtube.com/watch?v=l5PjODTSTaw&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=4

```
public class WhileLoops {
    public static void main(String[] args) {
        boolean t = true;
        System.out.println (t);

        boolean t2 = 2 > 1;
        System.out.println (t2);

        boolean t3 = 5 >= 5;
        System.out.println (t3);

        boolean t4 = 5 == 5;
```

```

        System.out.println (t4);

        int value = 0;
        boolean t5 = value > 5;
        System.out.println (t5);

        while (value < 5){
            System.out.println ("Hello - " + value);
            value++; // value = value + 1;
        }
    }
}

```

ВЫВОД В КОНСОЛЬ:

```

true
true
true
true
false
Hello - 0
Hello - 1
Hello - 2
Hello - 3
Hello - 4

```

Урок 5: Цикл for.

https://www.youtube.com/watch?v=UYbdAmqcNVc&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=5

```

public class ForLoop {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println ("Hello - " + i);
        }

        System.out.println ();

        for (int i = 0; i < 10; i = i + 5) {
            System.out.println ("Bye - " + i);
        }

        System.out.println ();

        for (int i = 10; i > 0; i--) {
            System.out.println ("Hi - " + i);
        }
    }
}

```

ВЫВОД В КОНСОЛЬ:

```

Hello - 0

```

```
Hello - 1
Hello - 2
Hello - 3
Hello - 4
Hello - 5
Hello - 6
Hello - 7
Hello - 8
Hello - 9
```

```
Bye - 0
Bye - 5
```

```
Hi - 10
Hi - 9
Hi - 8
Hi - 7
Hi - 6
Hi - 5
Hi - 4
Hi - 3
Hi - 2
Hi - 1
```

Урок 6: Условный оператор if.

https://www.youtube.com/watch?v=ryR033ld_N0&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=6

```
public class If {
    public static void main(String[] args) {
        System.out.println ("if №1");
        if (5 == 5){
            System.out.println ("5 == 5 - Yes, is true!");
        } else {
            System.out.println ("5 == 5 - No, is false!");
        }

        System.out.println ("if №2");
        if (5 == 0){
            System.out.println ("5 == 0 - Yes, is true!");

        } else if (5 == 3){
            System.out.println ("5 == 3 - Yes, is true!");
        } else {
            System.out.println ("5 == 3 - No, is false!");
        }

        System.out.println ("if №3");
        if (5 == 5){
            System.out.println ("5 == 5 - Yes, is true!");
        } else if (3 == 3){
            System.out.println ("3 == 3 - Yes, is true!");
        }

        // Это условие тоже верно, но оно идет Вторым и поэтому в консоль выводится
        // строка из Первого условия "5 == 5 - Yes, is true!".
        } else {
```

```

        System.out.println ("3 == 3 - No, is false!");
    }
}
}

```

Вывод в консоль:

```

if №1
5 == 5 - Yes, is true!
if №2
5 == 3 - No, is false!
if №3
5 == 5 - Yes, is true!

```

Урок 7: Ввод данных. Класс Scanner

https://www.youtube.com/watch?v=Y2iB_DwdyFM&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=7

```

import java.util.Scanner;

public class Input {
    public static void main(String[] args) {
        Scanner s = new Scanner (System.in);
        // System.in - это стандартный поток входа данных.
        System.out.println ("Введите что-нибудь");
        String string = s.nextLine ();
        System.out.println ("Вы ввели " + string);

        System.out.println ("Введите число");
        int x = s.nextInt ();
        System.out.println ("Вы ввели " + x);
        // Если вместо числа мы введем буквы, программа выдаст ошибку компиляции.
    }
}

```

Вывод в консоль:

```

Введите что-нибудь
Привет! // Ввел с клавиатуры.
Вы ввели Привет!
Введите число
123 // Ввел с клавиатуры.
Вы ввели 123

```

Урок 8: Цикл do...while.

https://www.youtube.com/watch?v=XVlM9sSWrhI&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=8

Реализация цикла While.

```

import java.util.Scanner;

public class DoWhile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner (System.in);
        System.out.println ("Add 5!");
        int value = scanner.nextInt ();
        while (value != 5){
            System.out.println ("No! Do add 5!");
            value = scanner.nextInt ();
        }
        System.out.println ("You have 5!");
    }
}

```

Вывод в консоль:

```

Add 5!
1 // Ввел с клавиатуры.
No! Do add 5!
2 // Ввел с клавиатуры.
No! Do add 5!
5 // Ввел с клавиатуры.
You have 5!

```

Реализация цикла DoWhile.

```

import java.util.Scanner;

public class DoWhile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner (System.in);
        int value;
        do {
            System.out.println ("Add 5!");
            value = scanner.nextInt ();
        } while (value != 5);
        System.out.println ("You have 5!");
    }
}

```

Вывод в консоль:

```

Add 5!
1 // Ввел с клавиатуры.
Add 5!
2 // Ввел с клавиатуры.
Add 5!
5 // Ввел с клавиатуры.
You have 5!

```

Урок 9: Операторы break и continue.

https://www.youtube.com/watch?v=otfClv5XVAY&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=9

Реализация Break.

```
public class Break_Continue {
    public static void main(String[] args) {
        int i = 1;
        while (true){
            if (i == 5){
                break;
            }
            System.out.println ("Петля №" + i);
            i++;
        }
        System.out.println ("Мы вышли из цикла for!");
    }
}
```

Вывод в консоль:

```
Петля №1
Петля №2
Петля №3
Петля №4
Мы вышли из цикла for!
```

Реализация Continue.

```
public class Break_Continue {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if (i % 2 == 0){
                continue;
            }
            // continue не пропускает ниже выполнение кода если выполнено условие
            // в if (i % 2 == 0).
            System.out.println ("Петля №" + i);
            // Выводим в консоль только Петли с нечетными номерами.
        }
        System.out.println ("Мы вышли из цикла for!");
    }
}
```

Вывод в консоль:

```
Петля №1
Петля №3
Петля №5
Петля №7
Петля №9
Мы вышли из цикла for!
```


Урок 10: Оператор switch.

https://www.youtube.com/watch?v=QJHcGWbzk3U&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=10

```
import java.util.Scanner;

import java.util.Scanner;

public class Switch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner (System.in);
        System.out.println ("How old are You?");
        int age = scanner.nextInt ();
        switch (age){
            case 0:
                System.out.println ("You were born.");
                break;
            case 7:
                System.out.println ("You are a schoolboy.");
                break;
            case 18:
                System.out.println ("You are a student.");
                break;
            default:
                System.out.println ("Your case is not here!!!");
        }
    }
}
```

Вывод в консоль:

```
How old are You?
0 // Ввел с клавиатуры.
You were born.
```

Вывод в консоль:

```
How old are You?
7 // Ввел с клавиатуры.
You are a schoolboy.
```

Вывод в консоль:

```
How old are You?
18 // Ввел с клавиатуры.
You are a student.
```

Вывод в консоль:

```
How old are You?
1 // Ввел с клавиатуры.
Your case is not here!!!
```

```

import java.util.Scanner;

public class Switch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner (System.in);
        System.out.println ("How old are You?");
        String age = scanner.nextLine ();
        switch (age){
            case "zero":
                System.out.println ("You were born.");
                break;
            case "seven":
                System.out.println ("You are a schoolboy.");
                break;
            case "eighteen":
                System.out.println ("You are a student.");
                break;
            default:
                System.out.println ("Your case is not here!!!");
        }
    }
}

```

Вывод в консоль:

```

How old are You?
zero // Ввел с клавиатуры.
You were born.

```

Вывод в консоль:

```

How old are You?
seven // Ввел с клавиатуры.
You are a schoolboy.

```

Вывод в консоль:

```

How old are You?
eighteen // Ввел с клавиатуры.
You are a student.

```

Вывод в консоль:

```

How old are You?
18 // Ввел с клавиатуры.
Your case is not here!!!

```

```

import java.util.Scanner;

public class Switch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner (System.in);
        System.out.println ("How old are You?");
        String age = scanner.nextLine ();
    }
}

```

```

        switch (age) {
            case "zero":
                System.out.println ("You were born.");
                // break; закомментировал!
            case "seven":
                System.out.println ("You are a schoolboy.");
                // break; закомментировал!
            case "eighteen":
                System.out.println ("You are a student.");
                break;
            default:
                System.out.println ("Your case is not here!!!");
        }
    }
}

```

Вывод в консоль:

```

How old are You?
zero // Ввел с клавиатуры.
You were born.
You are a schoolboy.
You are a student.

```

Вывод в консоль:

```

How old are You?
zer // Ввел с клавиатуры слово с ошибкой.
Your case is not here!!!

```

Урок 11: Массивы в Java.

https://www.youtube.com/watch?v=li86TEAEhYM&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=11

```

public class Arrays {
    public static void main(String[] args) {
        int num = 3; // Примитивный тип данных.
        char character = 'a'; // Примитивный тип данных.
        String string1 = new String ("Hello"); // Объект Класса String.
        String string2 = "Hello"; // Объект Класса String.

        int number = 10;
        int[] numbers = new int[5];
        // numbers - это Ссылка на Объект целочисленного массива "new int[5]" с пятью
        // "ячейками"
        System.out.println (numbers[0]);
        // по умолчанию неинициализированные элементы целочисленного массива имеют
        // значение "0"
        // элементы массива пронумерованы от 0 до 4
        System.out.println ("Инициализация массива numbers \\"вручную\\". Способ
1.");
        numbers[0] = 10;
        numbers[1] = 20;
        numbers[2] = 30;
    }
}

```

```

        numbers[3] = 40;
        numbers[4] = 50;
// вывод массива в консоль с помощью цикла
// numbers.length - возвращает длину массива (количество элементов в массиве)
        for (int i = 0; i < numbers.length; i++) {
            System.out.println (numbers[i]);
        }

        System.out.println ("Инициализация массива numbers \"вручную\". Способ
2.");
        int[] numbers2 = new int[] {100, 200, 300, 400, 500};
// вывод массива в консоль с помощью цикла
// numbers.length - возвращает длину массива (количество элементов в массиве)
        for (int i = 0; i < numbers2.length; i++) {
            System.out.println (numbers2[i]);
        }

        System.out.println ("Инициализация массива numbers \"вручную\". Способ
3.");
        int[] numbers3 = {1000, 2000, 3000, 4000, 5000};
// вывод массива в консоль с помощью цикла
// numbers.length - возвращает длину массива (количество элементов в массиве)
        for (int i = 0; i < numbers3.length; i++) {
            System.out.println (numbers3[i]);
        }

        System.out.println ("Инициализация массива numbers с помощью цикла
for");
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = i * 10;
        }
// вывод массива в консоль с помощью цикла
        for (int i = 0; i < numbers.length; i++) {
            System.out.println (numbers[i]);
        }
    }
}

```

Вывод в консоль:

```

0
Инициализация массива numbers "вручную". Способ 1.
10
20
30
40
50
Инициализация массива numbers "вручную". Способ 2.
100
200
300
400
500
Инициализация массива numbers "вручную". Способ 3.
1000
2000
3000
4000
5000

```

```
Инициализация массива numbers с помощью цикла for
0
10
20
30
40
```

Урок 12: Цикл for each, Массивы строк.

https://www.youtube.com/watch?v=8AD55r64yNw&index=12&list=PLAma_mKfftOSUkXp26rqdnC0PicnmnDak

```
public class Arrays_of_Strings {
    public static void main(String[] args) {
        String[] strings = new String[3];
        strings[0] = "Hello";
        strings[1] = "Bye";
        strings[2] = "Java";
        System.out.println (strings[0]);
        System.out.println (strings[2]);

        System.out.println ();
        // пустая строка - разделитель

        System.out.println ("Вывод текста в консоль с помощью цикла for");
        for (int i = 0; i < strings.length; i++) {
            System.out.println (strings[i]);
        }

        System.out.println ();
        // пустая строка - разделитель

        System.out.println ("Вывод текста в консоль с помощью цикла foreach");
        // String - тип данных
        // s - переменная цикла
        // strings - имя массива String, который выводим в консоль
        for (String s : strings) {
            System.out.println (s);
        }

        System.out.println ();
        // пустая строка - разделитель

        System.out.println ("Находим сумму всех элементов массива с помощью
        цикла foreach");
        int[] numbers1 = {1, 2, 3};
        int sum = 0;
        for (int x : numbers1) {
            sum = sum + x;
        }
        System.out.println ("Сумма всех элементов массива равна: " + sum);

        System.out.println ();
        // пустая строка - разделитель

        int value = 0; // "0" - это Число
```

```

String s; // "null" - это Пустота
String s1 = null; // "null" - это Пустота
//System.out.println (s); ошибка компеляции!!!
// вывести в консоль можно только s1, так как она
// проинициализированна "null"
System.out.println ("s1 ссылается на " + s1);
    }
}

```

Вывод в консоль:

```

Hello
Java

```

Вывод текста в консоль с помощью цикла for

```

Hello
Bye
Java

```

Вывод текста в консоль с помощью цикла foreach

```

Hello
Bye
Java

```

Находим сумму всех элементов массива с помощью цикла foreach

Сумма всех элементов массива равна: 6

s1 ссылается на null

Урок 13: Многомерные массивы.

https://www.youtube.com/watch?v=17FwEtVsIMQ&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=13

```

public class Multidimensional_arrays {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3};
        System.out.println ("Выводим в консоль элемент одномерного массива
numbers[1] (число 2): " + numbers[1]);

        System.out.println ();

        int[][] matrice = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9, 0}};
        System.out.println ("Выводим в консоль элемент многомерного массива
matrice[0][1](число 2): " + matrice[0][1]);
    }
    /*
    Сначала указываем порядковый номер Одномерного массива в Многомерном массиве
    [0].
    Затем указываем порядковый номер Элемента в Одномерном массиве [1].
    */
    System.out.println ("Выводим в консоль элемент многомерного массива
matrice[1][0](число 4): " + matrice[1][0]);
    // Сначала указываем порядковый номер Одномерного массива в
    Многомерном массиве [1].
    // Затем указываем порядковый номер Элемента в Одномерном массиве

```

```

[0].
    System.out.println ("Выводим в консоль элемент многомерного массива
matrice[2][3] (число 0): " + matrice[2][3]);

    System.out.println ();
    System.out.println ("Выводим в консоль элементы многомерного массива
matrice с помощью for");
    /*
matrice.length - возвращает количество Элементов (Одномерных массивов) во
Многомерном массиве.
matrice[i].length - возвращает количество Элементов (Чисел) в одном из
Одномерных массивов по порядку.
*/
    for (int i = 0; i < matrice.length; i++) {
        for (int j = 0; j < matrice[i].length; j++) {
            System.out.print (matrice[i][j] + " ");
        }
        System.out.println ();
    }
}
}

```

Вывод в консоль:

Выводим в консоль элемент одномерного массива numbers[1] (число 2): 2

Выводим в консоль элемент многомерного массива matrice[0][1] (число 2): 2

Выводим в консоль элемент многомерного массива matrice[1][0] (число 4): 4

Выводим в консоль элемент многомерного массива matrice[2][3] (число 0): 0

Выводим в консоль элементы многомерного массива matrice с помощью for

```

1 2 3
4 5 6
7 8 9 0

```

Урок 14: Классы и объекты.

https://www.youtube.com/watch?v=uPK2FVz6qUs&index=14&list=PLAma_mKfftOSUkXp26rqdnC0PicnmnDak

```

public class ClassesAndObjects {
    public static void main(String[] args) {
        Person person1 = new Person ();
        person1.name = "Roman";
        person1.age = 50;
        System.out.println ("My name is " + person1.name + ", and I am " +
person1.age + " years old.");

        Person person2 = new Person ();
        person2.name = "Vovan";
        person2.age = 20;
        System.out.println ("My name is " + person2.name + ", and I am " +
person2.age + " years old.");
    }
}

```

```

}

class Person{
    // У Класса есть Поля (Данные)
    String name;
    int age;
}

```

Вывод в консоль:

```

My name is Roman, and I am 50 years old.
My name is Vovan, and I am 20 years old.

```

Урок 15: Методы в Java.

https://www.youtube.com/watch?v=ayUby2sB-IU&index=15&list=PLAma_mKfftOSUkXp26rqdnC0PicnmnDak

```

public class ClassesAndObjects {
    public static void main(String[] args) {
        Person person1 = new Person ();
        person1.name = "Roman";
        person1.age = 50;
        person1.speak ();

        Person person2 = new Person ();
        person2.name = "Vovan";
        person2.age = 20;
        person2.speak ();
    }
}

class Person{
    // У Класса есть Поля (Данные)
    String name;
    int age;

    // У Класса есть Методы (Действия)
    void speak(){
        System.out.println ("My name is " + name + ", and I am " + age + "
years old.");
    }
}

```

Вывод в консоль:

```

My name is Roman, and I am 50 years old.
My name is Vovan, and I am 20 years old.

```

```

public class ClassesAndObjects {
    public static void main(String[] args) {
        Person person1 = new Person ();
        person1.name = "Roman";
    }
}

```



```

        person1.age = 50;
        person1.sayHello ();
        person1.speak ();

        Person person2 = new Person ();
        person2.name = "Vovan";
        person2.age = 20;
        person2.speak ();
    }
}

class Person{
    // У Класса есть Поля (Данные)
    String name;
    int age;

    // У Класса есть Методы (Действия)
    void speak(){
        System.out.println ("My name is " + name + ", and I am " + age + "
years old.");
    }

    void sayHello(){
        System.out.print ("Hello! ");
    }
}

```

Вывод в консоль:

Hello! My name is Roman, and I am 50 years old.
 My name is Vovan, and I am 20 years old.

Урок 16: Тип возвращаемого значения метода.

https://www.youtube.com/watch?v=wEewTdZEWPY&index=16&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak

```

public class ClassesAndObjects {
    public static void main(String[] args) {
        Person person1 = new Person ();
        person1.name = "Roman";
        person1.age = 50;
        person1.calculateYearsToRetirement ();

        Person person2 = new Person ();
        person2.name = "Vovan";
        person2.age = 20;
        person2.calculateYearsToRetirement ();
    }
}

class Person{

```

```

        // У Класса есть Поля (Данные)
        String name;
        int age;

        // У Класса есть Методы (Действия)
        /*
        "void" переводится как "вакуум" или "пустота"
        метод "void" не возвращает данные в метод main, и у него отсутствует
        "return...;"
        */

        void calculateYearsToRetirement(){
            int yesrs = 65 - age;
            System.out.println ("You must still work for " + yesrs + " years");
        }
    }
}

```

Вывод в консоль:

```

You must still work for 15 years
You must still work for 45 years

```

```

public class ClassesAndObjects {
    public static void main(String[] args) {
        Person person1 = new Person ();
        person1.name = "Roman";
        person1.age = 50;
        int year1 = person1.calculateYearsToRetirement ();
        System.out.println ("You must still work for " + year1 + " years");

        Person person2 = new Person ();
        person2.name = "Vovan";
        person2.age = 20;
        int year2 = person2.calculateYearsToRetirement ();
        System.out.println ("You must still work for " + year2 + " years");
    }
}

```

```

class Person{
    // У Класса есть Поля (Данные)
    String name;
    int age;

    // У Класса есть Методы (Действия)
    /*
    return не только возвращает значение в метод main, но и сразу прекращает
    (как break) работу метода
    этот метод возвращает только тот тип данных, кокого является сам (int ->
    int)
    */
    int calculateYearsToRetirement(){
        int yesrs = 65 - age;
        return yesrs;
    }
}

```

Вывод в консоль:

You must still work for 15 years
You must still work for 45 years

Урок 17: Параметры метода.

https://www.youtube.com/watch?v=J8ZLRvOO6vk&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak&index=17

```
public class ClassesAndObjects {
    public static void main(String[] args) {
        Person person1 = new Person ();
        person1.setNameAndAge ("Roman", 50);
        person1.speak ();

        Person person2 = new Person ();
        String vovanName = "Vovan";
        int vovanAge = 20;
        person2.setNameAndAge (vovanName, vovanAge);
        person2.age = 20;
        person2.speak ();
    }
}

class Person {
    String name;
    int age;

    void setNameAndAge(String userName, int userAge) {
        name = userName;
        age = userAge;
    }

    void speak() {
        System.out.println ("My name is " + name + ", and I am " + age + "
years old.");
    }
}
```

Вывод в консоль:

My name is Roman, and I am 50 years old.
My name is Vovan, and I am 20 years old.

Урок 18: Инкапсуляция. Сеттеры и геттеры.

https://www.youtube.com/watch?v=zf3lDojNx1A&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak&index=18

```

public class ClassesAndObjects {
    public static void main(String[] args) {
        Person person1 = new Person ();
        person1.setName ("Roman");
        person1.setAge (50);

        person1.speak ();

        System.out.println ("person1.getName () - " + person1.getName ());
        System.out.println ("person1.getAge () - " + person1.getAge ());
    }
}

class Person {
    private String name;
    private int age;

    public void setName(String userName){
        name = userName;
    }
    public String getName(){
        return name;
    }

    public void setAge(int userAge){
        age = userAge;
    }
    public int getAge() {
        return age;
    }

    void speak() {
        System.out.println ("My name is " + name + ", and I am " + age + "
years old.");
    }
}

```

Вывод в консоль:

```

My name is Roman, and I am 50 years old.
person1.getName () - Roman
person1.getAge () - 50

```

Проверка введенных данных в классе Person.

```

public class ClassesAndObjects {
    public static void main(String[] args) {
        Person person1 = new Person ();
        person1.setName ("");
        person1.setAge (-1);

        person1.speak ();

        System.out.println ("person1.getName () - " + person1.getName ());
        System.out.println ("person1.getAge () - " + person1.getAge ());
    }
}

```

```

class Person {
    private String name;
    private int age;

    public void setName(String userName){
        if (userName.isEmpty ()){
            System.out.println ("Please, add your name!");
        } else {
            name = userName;
        }
    }
    public String getName(){
        return name;
    }

    public void setAge(int userAge){
        if (userAge <= 0){
            System.out.println ("Please, add your age!");
        } else {
            age = userAge;
        }
    }
    public int getAge() {
        return age;
    }

    void speak() {
        System.out.println ("My name is " + name + ", and I am " + age + "
years old.");
    }
}

```

Вывод в консоль:

```

Please, add your name!
Please, add your age!
My name is null, and I am 0 years old.
person1.getName () - null
person1.getAge () - 0

```

Урок 19: Ключевое слово this.

https://www.youtube.com/watch?v=sPPaDe_5fcQ&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak&index=19

```

public class Lesson19 {
    public static void main(String[] args) {
        Human human1 = new Human ();
        human1.setName ("Tom");
        human1.setAge (18);
        human1.getInfo ();
    }
}

class Human{
    private String name;

```

```

private int age;

public void setName(String myName){
    name = myName;
}
public String getName(){
    return name;
}

public void setAge(int myAge){
    age = myAge;
}
public int getAge() {
    return age;
}

public void getInfo(){
    System.out.println (name + " - " + age);
}
}

```

Вывод в консоль:

Tom - 18

Используем в коде ключевое слово `this`.
`this` указывает, что `this.name` - это `private String name`;
так же `this` вызывает Объект внутри класса

```

private String name;

public void setName(String name){
    this.name = name;
}

public class Lesson19 {
    public static void main(String[] args) {
        Human human1 = new Human ();
        human1.setName ("Tom");
        human1.setAge (18);
        human1.getInfo ();
    }
}

class Human{
    private String name;
    private int age;

    public void setName(String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }

    public void setAge(int age){
        this.age = age;
    }
}

```

```

    public int getAge() {
        return age;
    }

    public void getInfo(){
        System.out.println (name + " - " + age);
    }
}

```

Вывод в консоль:

Tom - 18

Урок 20: Конструкторы.

https://www.youtube.com/watch?v=Muytoo-x-KM&list=PLAma_mKfftOSUkXp26rgdnCOPicnmnDak&index=20

```

public class Lesson19 {
    public static void main(String[] args) {
        Human human1 = new Human ();
    }
}

class Human{
    // Конструктор по умолчанию.
    /*
    У конструктора нет типа данных.
    Имя конструктора всегда совпадает с именем класса
    и оно всегда пишется с большой буквы.
    */
    public Human() {
        System.out.println ("Hello from the first Constructor!");
    }
}

```

Вывод в консоль:

Hello from the first Constructor!

Перегрузка конструкторов

```

public class Lesson19 {
    public static void main(String[] args) {
        Human human1 = new Human ();
        // Hello from the first Constructor!
        Human human2 = new Human ("Bob");
        // Hello from the second Constructor!
        Human human3 = new Human ("Bob", 45);
        // Hello from the third Constructor!
    }
}

class Human{
    private String name;

```

```

    private int age;

    /*
    Перегрузка конструкторов - это наличие конструкторов с ОДНИМ ИМЕНЕМ, но с
    РАЗНЫМИ ПАРАМЕТРАМИ.
    */
    // Конструктор по умолчанию.
    public Human() {
        System.out.println ("Hello from the first Constructor!");
        this.name = "Xxx";
        this.age = 0;
    }

    public Human(String name) {
        System.out.println ("Hello from the second Constructor!");
        this.name = name;
    }

    public Human(String name, int age) {
        System.out.println ("Hello from the third Constructor!");
        this.name = name;
        this.age = age;
    }
}

```

Вывод в консоль:

```

Hello from the first Constructor!
Hello from the second Constructor!
Hello from the third Constructor!

```

Урок 21: Ключевое слово static.

https://www.youtube.com/watch?v=GZzVfeY7yEM&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=21

```

public class Lesson19 {
    public static void main(String[] args) {
        /*
        Присвоить и переприсвоить имя и возраст мы может ТОЛЬКО Объекту
        human1, но не Классу Human.
        Поэтому переменные и методы Класса Human принадлежат ТОЛЬКО human1.
        */
        Human human1 = new Human ("Bob", 30);
        human1.setName ("Tom");
    }
}

class Human{
    // переменная Объекта human1 "name"
    private String name;
    // переменная Объекта human1 "age"
    private int age;
    /*
    Переменные Объекта инициализируются только при создании Объекта human1
    Класса Human.
    */
    // пример переопределения конструктора по умолчанию
}

```



```

public Human() {
    this("Bob", 12);
}

public Human(String name, int age) {
    System.out.println ("Hello from the third Constructor!");
    this.name = name;
    this.age = age;
}

// метод Объекта human1 "setName"
public void setName(String name) {
    this.name = name;
}

// метод Объекта human1 "setAge"
public void setAge(int age) {
    this.age = age;
}
}

```

Вывод в консоль:

Нет данных в консоли

```

public class Lesson19 {
    public static void main(String[] args) {
        /*
         * К статической переменной discription Класса Human мы можем обратиться
         * не через создание Объекта human1, а обратившись на прямую к Классу Human.
         */
        Human.discription = "Nice";
        /*
         * К статическому методу getDescription Класса Human мы можем обратиться
         * не через создание Объекта human1, а обратившись на прямую к Классу Human.
         */
        Human.getDescription ();
    }
}

class Human {
    // переменная Объекта human1 "name"
    private String name;
    // переменная Объекта human1 "age"
    private int age;
    // СТАТИЧЕСКАЯ переменная "discription" КЛАССА Human
    public static String discription;

    public Human(String name, int age) {
        System.out.println ("Hello from the third Constructor!");
        this.name = name;
        this.age = age;
    }

    // метод Объекта human1 "setName"
    public void setName(String name) {
        this.name = name;
    }

    // метод Объекта human1 "setAge"
    public void setAge(int age) {

```

```

        this.age = age;
    }
    // СТАТИЧЕСКИЙ метод КЛАССА Human
    public static void getDescription(){
        System.out.println (description);
    }
}

```

Вывод в консоль:

Nice

```

public class Lesson19 {
    public static void main(String[] args) {
        /*
         * К статической переменной description Класса Human мы можем обратиться
         * не через создание Объекта human1, а обратившись на прямую к Классу Human.
         */
        Human.description = "Nice";
        /*
         * К статическому методу getDescription Класса Human мы можем обратиться
         * не через создание Объекта human1, а обратившись на прямую к Классу Human.
         */
        Human.getDescription ();
        /*
         * К статическому методу getDescription Класса Human мы можем обратиться
         * И через создание Объекта human1, НЕ обратившись на прямую к Классу Human.
         * Это возможно, но не совсем корректно с точки зрения ООП.
         */
        Human human1 = new Human ("Bob", 40);
        // Переопределяем public static String description; через Объект
        human1.
        human1.description = "Bad :-( ";
        human1.getDescription();

        // Лучше делать ТАК!!!
        Human.description = "Nice! Nice! Nice!";
        Human.getDescription ();
    }
}

class Human{
    // переменная Объекта human1 "name"
    private String name;
    // переменная Объекта human1 "age"
    private int age;
    // СТАТИЧЕСКАЯ переменная "description" КЛАССА Human
    public static String description;

    public Human(String name, int age){
        System.out.println ("Hello from the third Constructor!");
        this.name = name;
        this.age = age;
    }
    // метод Объекта human1 "setName"
    public void setName(String name){
        this.name = name;
    }
}

```

```

// метод Объекта human1 "setAge"
public void setAge(int age) {
    this.age = age;
}
// СТАТИЧЕСКИЙ метод КЛАССА Human
public static void getDescription() {
    System.out.println (discription);
}
}

```

Вывод в консоль:

```

Nice
Hello from the third Constructor!
Bad :-(
Nice! Nice! Nice!

```

```

public class Lesson19 {
    public static void main(String[] args) {
        Human human1 = new Human ("Bob", 40);
        Human human2 = new Human ("Tom", 30);

        Human.discription = "Nice";
        human1.getAllFields ();
        human2.getAllFields ();

        Human.discription = "Bad";
        human1.getAllFields ();
        human2.getAllFields ();
    }
}

class Human{
    private String name;
    private int age;
    /*
    Если статическая переменная создана, но не проинициализирована,
    то она имеет значение по умолчанию.
    int == 0, String == null.
    */
    public static String discription;

    public Human(String name, int age){
        this.name = name;
        this.age = age;
    }

    public void setName(String name){
        this.name = name;
    }

    public void setAge(int age){
        this.age = age;
    }

    public void getAllFields(){
        System.out.println (name + ", " + age + ", " + discription);
    }
}

```

```

    /*
    !!! В СТАТИЧЕСКОМ МЕТОДЕ НЕЛЬЗЯ ИСПОЛЬЗОВАТЬ НЕСТАТИЧЕСКИЕ ПЕРЕМЕННЫЕ
    name И age.
    МОЖНО ИСПОЛЬЗОВАТЬ ТОЛЬКО discription!
    ЭТОТ СТАТИЧЕСКИЙ МЕТОД НЕПРАВИЛЬНЫЙ!
    public static void getAllFields(){
        System.out.println (name + ", " + age + ", " + discription);
    }
    */
}

```

Вывод в консоль:

```

Bob, 40, Nice
Tom, 30, Nice
Bob, 40, Bad
Tom, 30, Bad

```

Примеры "Встроенных Статических методов".

```

public class Lesson19 {
    public static void main(String[] args) {
        // Примеры "Встроенных Статических методов".
        System.out.println (Math.pow (3, 2));
        System.out.println (Math.PI);
    }
}

```

Вывод в консоль:

```

9.0
3.141592653589793

```

Программа подсчета созданных объектов класса Human.

```

public class Lesson19 {
    public static void main(String[] args) {
        Human human1 = new Human ("Bob", 40);
        Human human2 = new Human ("Tom", 30);
        Human human3 = new Human ("Tim", 28);

        System.out.println ("Мы создали " + Human.countPeople + " Объекта.");
    }
}

class Human{
    private String name;
    private int age;
    /*
    Если статическая переменная создана, но не проинициализированна,
    то она имеет значение по умолчанию.
    int == 0.
    */
    public static int countPeople; // Не инициализировать нулем! Просто нет
    смысла!)))
}

```

```

    public Human(String name, int age){
        this.name = name;
        this.age = age;
        countPeople++;
        // При создании нового Объекта human Класса Human,
        // countPeople увеличится на 1 (countPeople++);
    }
}

```

Вывод в консоль:

Мы создали 3 Объекта.

Урок 22: Ключевое слово final.

https://www.youtube.com/watch?v=uIewh5JOKa0&index=22&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak

```

public class Lesson22 {
    public static void main(String[] args) {
        Test test = new Test (10);
        System.out.println ("CONSTANT1 - " + test.CONSTANT1);

        System.out.println ("CONSTANT2 - " + Test.CONSTANT2);

        // создание final переменной в методе main
        final int x = 1000;
        System.out.println ("x - " + x);
    }
}

class Test{
    // переменная final пишется прописными буквами
    public final int CONSTANT1;
    // конструктор для инициализации CONSTANT1,
    // при создании Объекта Test.
    public Test(int CONSTANT1){
        this.CONSTANT1 = CONSTANT1;
    }
    /*
    НЕ ВОЗМОЖНО СОЗДАТЬ МЕТОД, ПЕРЕОПРЕДЕЛЯЮЩИЙ CONSTANT!!!
    public void setCONSTANT(int x){
        this.CONSTANT = x;
    }
    Чтобы экономить память, переменную final лучше сделать static.
    Но инициализировать ее нужно сразу после ее создания.
    */
    public static final int CONSTANT2 = 777;
}

```

Вывод в консоль:

```

CONSTANT1 - 10
CONSTANT2 - 777

```

x - 1000

Урок 23: StringBuilder

https://www.youtube.com/watch?v=Vw2GPL3APD4&index=23&list=PLAma_mKfftOSUkXp26rqdnC0PicnmnDak

```
public class Lesson23 {
    public static void main(String[] args) {
        // создаем Объект Класса String двумя способами
        String s = new String ("Hello");
        String x = "Hello";
        /*
        Метод toUpperCase Класса String не изменяет строку x,
        а на базе строки x создает НОВУЮ строку x с изменениями.
        После изменения переменная x перестает ссылаться на СТАРЫЙ Объект
        String Hello.
        И теперь x ссылается на вновь созданный Объект String HELLO.
        */
        x.toUpperCase ();
        // Строка x не изменилась.
        System.out.println ("\nСтарая Строка\" (Строка x не изменилась): " + x);
        //Чтобы строка x изменилась, ее нужно переопределить.
        x = x.toUpperCase ();
        // Строка x изменилась.
        System.out.println ("\nНовая Строка\" (Строка x изменилась): " + x);
    }
}
```

Вывод в консоль:

```
"Старая Строка" (Строка x не изменилась): Hello
"Новая Строка" (Строка x изменилась): HELLO
```

// Конкатенация строк.

```
public class Lesson23 {
    public static void main(String[] args) {
        String string1 = "Hello";
        String string2 = "My";
        String string3 = "Frieng";
        String stringAll = string1+string2+string3;
        /*
        При конкатенации строк строк каждый "+" создает НОВЫЙ Объект
        Класса String. Следующий "+" создает еще один НОВЫЙ Объект Класса
        String.
        Это УВЕЛИЧИВАЕТ РАСХОД памяти.
        */
        System.out.println (stringAll);
        /*
        StringBuilder создан, чтобы избежать РАСХОДА памяти,
        при объединении строк.
        Если в конструктор new StringBuilder ("Hello");
        не поставлять Hello, то sb будет "пустым" - null.
        */
    }
}
```

```

Добавляем еще строки к sb,
но без создания НОВЫХ Объектов.
*/
StringBuilder sb = new StringBuilder ("hello");
sb.append ("my");
sb.append ("friend");
System.out.println (sb.toString ());
/*
Делаем тоже самое, но с "ЧЕЙНИНГ".
*/
StringBuilder sB = new StringBuilder ("HELLO");
sB.append ("MY").append ("FRIEND");
System.out.println (sB.toString ());
}
}

```

Вывод в консоль:

```

HelloMyFrieng
hellomyfriend
HELLOMYFRIEND

```

Урок 23(продолжение): Форматирование строк в Java

https://www.youtube.com/watch?v=WABydTyTlFs&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=24

// Перевод Строки на следующую строку.

```

public class Lesson23 {
    public static void main(String[] args) {
        // System.out.print - НЕТ перевода Строки на следующую строку.
        System.out.print ("(System.out.print) Строка 1 ");
        System.out.println ("Строка 2 ");

        // Деление Текста Пустой Строкой.
        System.out.println ();

        // System.out.println - перевод Строки на следующую строку.
        System.out.println ("(System.out.println) Строка 1 ");
        System.out.println ("Строка 2 ");
    }
}

```

Вывод в консоль:

```

(System.out.print) Строка 1 Строка 2

(System.out.println) Строка 1
Строка 2

```

// Форматирование Строки с помощью printf.

```
public class Lesson23 {
    public static void main(String[] args) {
        /*
        Вставка значений в Строку.
        %s или %d - обозначает то место где будет вставлена ВСТАВКА,
        и какой тип будет иметь эта ВСТАВКА.
        */
        System.out.printf ("This is a string, %s", "NICE");
        System.out.println ();
        System.out.printf ("This is a string, %d", 10);
        System.out.println ();
        System.out.printf ("This is a %s string, %d", "NICE", 10);
        System.out.println ();
        /*
        Форматирование строки.
        Задаем ширину числа в количестве выводимых знаков.
        */
        /*
        "\n" переводит на следующую строку так же,
        как System.out.println ();
        */
        System.out.printf ("|%10d|\n", 532); // | - чтобы обозначить ширину
        числа в 10 знаков.
        System.out.printf ("|%-10d|\n", 532); // теперь цифры прижаты к Левой
        стороне
        System.out.printf ("|%.2f|\n", 532.0); // вещественное число float с
        двумя знаками после запятой.
        System.out.printf ("|%.2f|\n", 532.005555555); // и округляет до
        второго знака после запятой.
    }
}
```

Вывод в консоль:

```
This is a string, NICE
This is a string, 10
This is a NICE string, 10
|          532|
|532         |
|532,00|
|532,01|
```

Урок 24: Класс Object и метод toString()

https://www.youtube.com/watch?v=KEQ043yT3F4&index=25&list=PLAma_mKfftOSUkXp26rqdnC0PicnmnDak

```
public class Lesson24 {
    public static void main(String[] args) {
        /*
        Чтобы понять, как работает toString.
        Создаем и выводим на экран два объекта
        */
    }
}
```



```

        Классов Human и String.
        */
        String string1 = new String ("Bob, 35");
        System.out.println ("Это string1 - " + string1);

        Human human1 = new Human ("Bob", 35);
        /*
        На экране мы увидим хешкод (Human@4554617c).
        Хешкод - это уникальный номер объекта Human (human1).
        */
        System.out.println ("Это human1 - " + human1);
        // ИЛИ
        System.out.println ("Это human1 - " + human1.toString ());
    }
}

class Human{
    private String name;
    private int age;

    public Human(String name, int age){
        this.name = name;
        this.age = age;
    }
}

```

Вывод в консоль:

```

Это string1 - Bob, 35
Это human1 - Human@4554617c
Это human1 - Human@4554617c

```

Переопределение метода toString ().

```

public class Lesson24 {
    public static void main(String[] args) {
        Human human1 = new Human ("Bob", 35);
        /*
        Чтобы метод toString () выводил на экран данные Объекта human1,
        а не его хешкод
        (toString () хешкод выводит по умолчанию),
        метод toString (), доставшийся классу Human от Object
        нужно переопределить в классе Human.
        */
        System.out.println ("Это human1 - " + human1);
        // ИЛИ
        System.out.println ("Это human1 - " + human1.toString ());
    }
}

class Human{
    private String name;
    private int age;

    public Human(String name, int age){
        this.name = name;
        this.age = age;
    }
}

```

```

    /*
    Переопределение метода toString ()
    */
    public String toString(){
        return name + ", " + age;
    }
}

```

Вывод в консоль:

```

Это human1 - Bob, 35
Это human1 - Bob, 35

```

Урок 25: Наследование.

https://www.youtube.com/watch?v=28NP_V2yc60&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak&index=26

```

public class Lesson25 {
    public static void main(String[] args) {
        Animal animal = new Animal ();
        System.out.print ("animal: ");
        animal.eat ();
        System.out.print ("animal: ");
        animal.sleep ();

        System.out.println ();

        /*
        Класс Dog наследует от класса Animal
        методы eat и sleep
        */
        Dog dog = new Dog ();
        System.out.print ("dog: ");
        dog.eat ();
        System.out.print ("dog: ");
        dog.sleep ();
    }
}

public class Animal {
    public void eat(){
        System.out.println ("I am eating");
    }

    public void sleep(){
        System.out.println ("I am sleeping");
    }
}

// Dog extends Animal
// Dog наследует 2 метода от Animal

```

```
public class Dog extends Animal {  
}
```

Вывод в консоль:

```
animal: I am eating  
animal: I am sleeping
```

```
dog: I am eating  
dog: I am sleeping
```

Дополняем класс Dog методом bark (лаять).
Этот метод не сможет использовать объект класса Animal, только объекты класса Dog и классы наследники класса Dog.

```
public class Lesson25 {  
    public static void main(String[] args) {  
        Animal animal = new Animal ();  
        System.out.print ("animal: ");  
        animal.eat ();  
        System.out.print ("animal: ");  
        animal.sleep ();  
  
        System.out.println ();  
  
        /*  
        Класс Dog наследует от класса Animal  
        методы eat и sleep  
        */  
        Dog dog = new Dog ();  
        System.out.print ("dog: ");  
        dog.eat ();  
        System.out.print ("dog: ");  
        dog.sleep ();  
        /*  
        Новый метод класса Dog - bark.  
        */  
        System.out.print ("dog: ");  
        dog.bark ();  
    }  
}
```

```
public class Animal {  
    public void eat(){  
        System.out.println ("I am eating");  
    }  
  
    public void sleep(){  
        System.out.println ("I am sleeping");  
    }  
}
```

```
/*  
Dog extends Animal  
Dog наследует методы от Animal
```

Дополняем класс *Dog* методом *bark* (лаять).
Этот метод не сможет использовать объект
класса *Animal*, только объекты класса *Dog*
и классы наследники класса *Dog*.
*/

```
public class Dog extends Animal {  
    public void bark() {  
        System.out.println ("I am barking");  
    }  
}
```

Вывод в консоль:

```
animal: I am eating  
animal: I am sleeping  
  
dog: I am eating  
dog: I am sleeping  
dog: I am barking
```

В классе *Dog* переопределяем метод *eat* класса *Animal*.

```
public class Lesson25 {  
    public static void main(String[] args) {  
        Animal animal = new Animal ();  
        System.out.print ("animal: ");  
        animal.eat ();  
        System.out.print ("animal: ");  
        animal.sleep ();  
  
        System.out.println ();  
  
        /*  
        Класс Dog наследует от класса Animal  
        методы eat и sleep  
        */  
        Dog dog = new Dog ();  
        System.out.print ("dog: ");  
        dog.eat ();  
        System.out.print ("dog: ");  
        dog.sleep ();  
        /*  
        Новый метод класса Dog - bark.  
        */  
        System.out.print ("dog: ");  
        dog.bark ();  
    }  
}  
  
public class Animal {  
    public void eat(){  
        System.out.println ("Animal is eating");  
    }  
  
    public void sleep(){  
        System.out.println ("Animal is sleeping");  
    }  
}
```

```

    }
}

/*
Dog extends Animal
Dog наследует методы от Animal
Дополняем класс Dog методом bark (лаять).
Этот метод не сможет использовать объект класса Animal, только объекты класса
Dog и классы наследники класса Dog.
В классе Dog переопределяем метод eat класса Animal
*/

public class Dog extends Animal {
    public void bark() {
        System.out.println ("I am barking");
    }
    /*
    Если сигнатура метода eat в классе Dog,
    совпадает с сигнатурой метода eat класса Animal,
    то метод eat класса Animal становится НЕАКТУАЛЬНЫМ для объекта класса
    Dog!!!
    */
    public void eat(){
        System.out.println ("Dog is eating");
    }
}

```

Вывод в консоль:

```

animal: Animal is eating
animal: Animal is sleeping

```

```

dog: Dog is eating
dog: Animal is sleeping
dog: I am barking

```

Добавляем в класс Animal новое поле name.

```

public class Lesson25 {
    public static void main(String[] args) {
        Animal animal = new Animal ();
        System.out.print ("animal: ");
        animal.eat ();
        System.out.print ("animal: ");
        animal.sleep ();

        System.out.println ();

        /*
        Класс Dog наследует от класса Animal
        методы eat и sleep
        */
        Dog dog = new Dog ();
        System.out.print ("dog: ");
        dog.eat ();
        System.out.print ("dog: ");
    }
}

```

```

        dog.sleep ();
        /*
        Новый метод класса Dog - bark.
        */
        System.out.print ("dog: ");
        dog.bark ();

        System.out.print ("dog: ");
        dog.showName ();
    }
}

// Создаем поле String name; в классе Animal
public class Animal {

    String name = "Some animal";

    public void eat(){
        System.out.println ("Animal is eating");
    }

    public void sleep(){
        System.out.println ("Animal is sleeping");
    }
}

// Создаем поле метод showName(); в классе Dog
public class Dog extends Animal {
    public void bark() {
        System.out.println ("I am barking");
    }
    /*
    Если сигнатура метода eat в классе Dog,
    совпадает с сигнатурой метода eat класса Animal,
    то метод eat класса Animal становится НЕАКТУАЛЬНЫМ для объекта класса
    Dog!!!
    */
    public void eat(){
        System.out.println ("Dog is eating");
    }

    /*
    В классе Dog нет поля String name;
    Оно наследуется из класса - Родителя Animal, ЕСЛИ оно не !!!private!!!
    */

    public void showName(){
        System.out.println (name);
    }
}

```

Вывод в консоль:

```

animal: Animal is eating
animal: Animal is sleeping

```

```
dog: Dog is eating
dog: Animal is sleeping
dog: I am barking
dog: Some animal
```

Урок 26: Интерфейсы.

https://www.youtube.com/watch?v=uCgF5-yCbGA&index=27&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak

Чтобы использовать в одном проекте классы с одинаковыми именами, нужно разнести их по разным папкам (Package) в папке src.

Создаем новый пакет (папку) Interfaces.

Для создания в IJ Constructor, Setter или Getter, зажимаем клавиши Ctrl + N.

```
package Interfaces;
```

```
public class Test {
    public static void main(String[] args) {
        Animal animal1 = new Animal (1);
        Person person1 = new Person ("Bob");

        animal1.sleep ();
        person1.sayHello ();
    }
}
```

```
package Interfaces;
```

```
public class Animal {
    public int id;

    public Animal(int id){
        this.id = id;
    }

    public void sleep(){
        System.out.println ("I am sleeping");
    }
}
```

```
package Interfaces;
```

```
public class Person {
    public String name;

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

    public void sayHello(){
        System.out.println ("Hello");
    }
}
```

```
}  
}
```

Вывод в консоль:

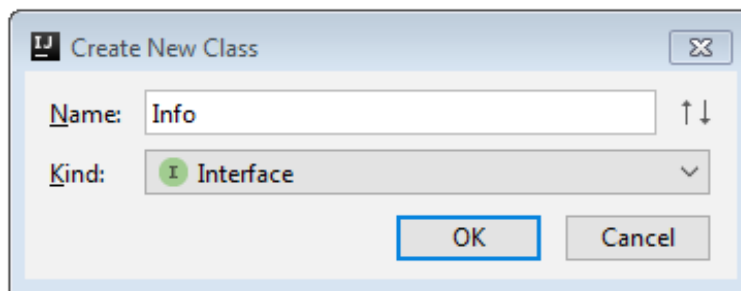
```
I am sleeping  
Hello
```

Создаем Interfaces для классов Animal и Person.

Interfaces -> Animal -> New -> Java Class

Name: Info

Kind: Interface



```
package Interfaces;
```

```
public class Test {  
    public static void main(String[] args) {  
        Animal animal1 = new Animal (1);  
        Person person1 = new Person ("Bob");  
  
        animal1.sleep ();  
        person1.sayHello ();  
  
        animal1.showInfo ();  
        person1.showInfo ();  
    }  
}
```

```
package Interfaces;
```

```
public interface Info {  
    public void showInfo();  
}
```

```
package Interfaces;
```

```
public class Animal implements Info{  
    public int id;  
  
    public Animal(int id){  
        this.id = id;  
    }  
  
    public void sleep(){
```



```

        System.out.println ("I am sleeping");
    }

    public void showInfo() {
        System.out.println ("showInfo(); - Id is " + this.id);
    }
}

package Interfaces;

public class Person implements Info{
    public String name;

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

    public void sayHello(){
        System.out.println ("Hello");
    }

    @Override
    public void showInfo() {
        System.out.println ("showInfo(); - Name is " + this.name);
    }
}

```

Вывод в консоль:

```

I am sleeping
Hello
showInfo(); - Id is 1
showInfo(); - Name is Bob

```

```

package Interfaces;

public class Test {
    public static void main(String[] args) {
        /*
        Если классы Animal и Person реализуют один интерфейс Info,
        то возможна такая реализация.
        */
        Info info1 = new Animal (1);
        Info info2 = new Person ("Bob");

        info1.showInfo ();
        info2.showInfo ();
        /*
        info1 и info2 не имеют доступа к методам классов Animal и Person
        !!! неправильно - info1.sleep ();
        !!! неправильно - info2.sayHello ();
        */
    }
}

```

```
package Interfaces;
```

```
public interface Info {  
    public void showInfo();  
}
```

```
package Interfaces;
```

```
public class Animal implements Info{  
    public int id;  
  
    public Animal(int id){  
        this.id = id;  
    }  
  
    public void sleep(){  
        System.out.println ("I am sleeping");  
    }  
  
    public void showInfo(){  
        System.out.println ("showInfo(); - Id is " + this.id);  
    }  
}
```

```
package Interfaces;
```

```
public class Person implements Info{  
    public String name;  
  
    public Person(String name){  
        this.name = name;  
    }  
  
    public void sayHello(){  
        System.out.println ("Hello");  
    }  
  
    @Override  
    public void showInfo() {  
        System.out.println ("showInfo(); - Name is " + this.name);  
    }  
}
```

Вывод в консоль:

```
showInfo(); - Id is 1  
showInfo(); - Name is Bob
```

```
package Interfaces;
```

```
public class Test {  
    public static void main(String[] args) {  
        /*  
        Если классы Animal и Person реализуют один интерфейс Info,  
        то возможна такая реализация.  
        */  
    }  
}
```

```

        Info info1 = new Animal (1);
        Info info2 = new Person ("Bob");

        /*
        Заменяем
        info1.showInfo ();
        info2.showInfo ();
        на...
        */

        outputInfo (info1);
        outputInfo (info2);

    }
    public static void outputInfo(Info info){
        info.showInfo ();
    }
}

package Interfaces;

public interface Info {
    public void showInfo();
}

package Interfaces;

public class Animal implements Info{
    public int id;

    public Animal(int id){
        this.id = id;
    }

    public void sleep(){
        System.out.println ("I am sleeping");
    }

    public void showInfo(){
        System.out.println ("showInfo(); - Id is " + this.id);
    }
}

package Interfaces;

public class Person implements Info{
    public String name;

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

    public void sayHello(){
        System.out.println ("Hello");
    }

    @Override

```

```

    public void showInfo() {
        System.out.println ("showInfo(); - Name is " + this.name);
    }
}

```

Вывод в консоль:

```

showInfo(); - Id is 1
showInfo(); - Name is Bob

```

```

package Interfaces;

```

```

public class Test {
    public static void main(String[] args) {
        /*
        Заменяем
        Info info1 = new Animal (1);
        Info info2 = new Person ("Bob");
        на...
        */
        Animal animal1 = new Animal (1);
        Person person1 = new Person ("Bob");
        outputInfo (animal1);
        outputInfo (person1);

    }
    public static void outputInfo(Info info){
        info.showInfo ();
    }
}

```

```

package Interfaces;

```

```

public interface Info {
    public void showInfo();
}

```

```

package Interfaces;

```

```

public class Animal implements Info{
    public int id;

    public Animal(int id){
        this.id = id;
    }

    public void sleep(){
        System.out.println ("I am sleeping");
    }

    public void showInfo(){
        System.out.println ("showInfo(); - Id is " + this.id);
    }
}

```

```

package Interfaces;

public class Person implements Info{
    public String name;

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

    public void sayHello(){
        System.out.println ("Hello");
    }

    @Override
    public void showInfo() {
        System.out.println ("showInfo(); - Name is " + this.name);
    }
}

```

Вывод в консоль:

```

showInfo(); - Id is 1
showInfo(); - Name is Bob

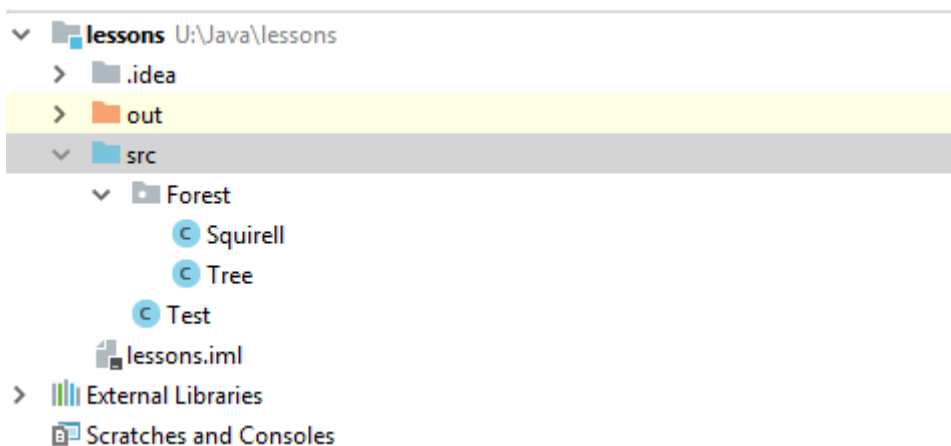
```

Урок 27: Пакеты.



https://www.youtube.com/watch?v=7VZRmUjuSlq&list=PLAma_mKfftOSUkXp26rgdnC0PicnmDak&index=28

Создаем:



- класс Test;
- пакет Forest;
 - класс Squirell;
 - класс Tree;



Организация нашего проекта в папке src на компьютере.

Имя	Дата изменения	Тип	Размер
 Forest	09.08.2018 08:45	Папка с файлами	
 Test.java	09.08.2018 08:44	Файл "JAVA"	1 КБ

В папке Forest.

Имя	Дата изменения	Тип	Размер
 Squirell.java	09.08.2018 08:44	Файл "JAVA"	1 КБ
 Tree.java	09.08.2018 08:45	Файл "JAVA"	1 КБ

Создаем Объект класса Tree в классе Test.

```
public class Test {
    public static void main(String[] args) {
        Tree tree1 = new Tree ();
    }
}
```

!!!Компилятор выдает ошибку о том, что он не видит класс Tree!!!
Импортируем в класс Test класс Tree пакета Forest.

```
import Forest.Tree;
```

```
public class Test {
    public static void main(String[] args) {
        Tree tree1 = new Tree ();
    }
}
```

Ошибка компиляции исчезла.

```
import Forest.Tree;
import Forest.Squirell;
```

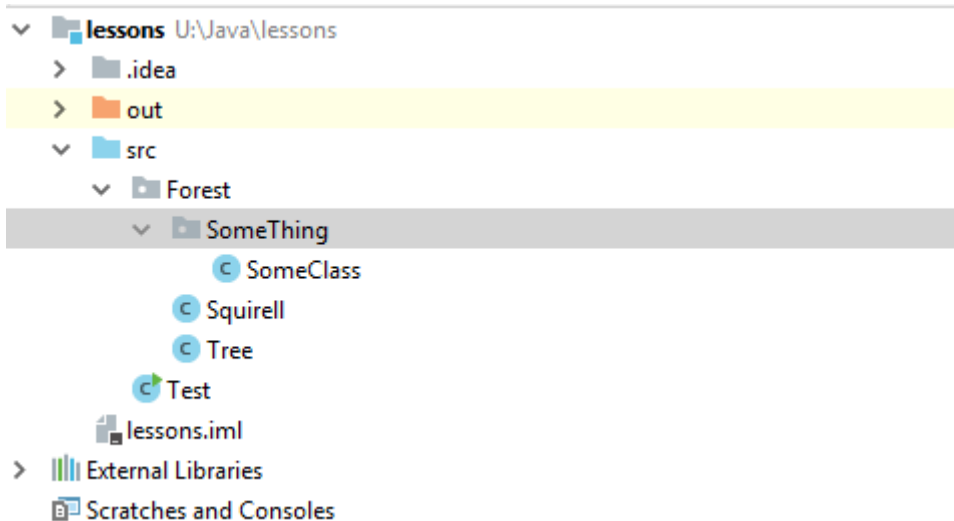
```
public class Test {
    public static void main(String[] args) {
        Tree tree1 = new Tree ();
        Squirell squirell1 = new Squirell ();
    }
}
```

Класс Scanner находится тоже в другом пакете, который тоже нужно импортировать.

```
import java.util.Scanner;
```

```
public class Test {
    public static void main(String[] args) {
        Scanner scanner = new Scanner (System.in);
    }
}
```

Создаем еще один пакет `Something` и класс `SomeClass` в пакете `Forest`.



Создаем Объект класса `SomeClass` в классе `Test`.

Импортируем в класс `Test` класс `SomeClass` пакета `Something` пакета `Forest`.


```
import Forest.Something.SomeClass;

public class Test {
    public static void main(String[] args) {
        SomeClass someClass1 = new SomeClass ();
    }
}
```

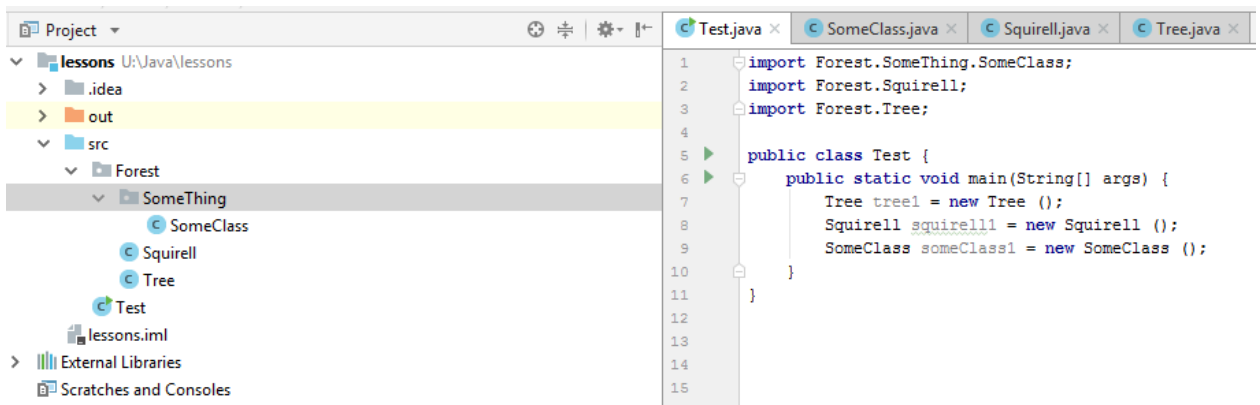
Имя	Дата изменения	Тип	Размер
.idea	09.08.2018 09:10	Папка с файлами	
out	06.08.2018 19:29	Папка с файлами	
src	09.08.2018 09:11	Папка с файлами	
lessons.iml	04.08.2018 17:49	Файл "IML"	1 КБ

Имя	Дата изменения	Тип	Размер
Forest	09.08.2018 09:08	Папка с файлами	
Test.java	09.08.2018 09:11	Файл "JAVA"	1 КБ

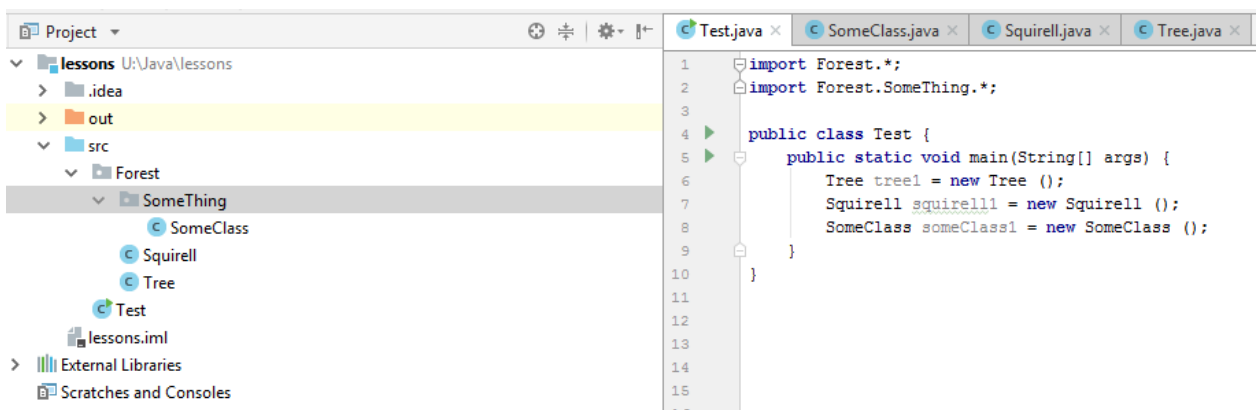
Имя	Дата изменения	Тип	Размер
Something	09.08.2018 09:08	Папка с файлами	
Squirell.java	09.08.2018 08:44	Файл "JAVA"	1 КБ
Tree.java	09.08.2018 08:45	Файл "JAVA"	1 КБ

Имя	Дата изменения	Тип	Размер
 SomeClass.java	09.08.2018 09:08	Файл "JAVA"	1 КБ

Сокращенный импорт классов.
Из...



В...



Урок 28: Модификаторы доступа public, private.

https://www.youtube.com/watch?v=e14xUIUc6y0&index=29&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak

/*

public - доступен в рамках / пределах всего Проекта. В Java файле должен быть хотя бы один public класс.

private - доступен в рамках / пределах всего Класса в котором protected переменная / метод задекларированны.

default - доступен в рамках / пределах всего Пакета. default еще называют модификатором доступа в пакете. Модификатор доступа по умолчанию. default - не пишем перед переменной. Он устанавливается автоматически, если вообще не указать модификатор .

protected – доступен в рамках / пределах всего Пакета. А также в классах – наследниках (extends) материнского класса в котором *protected* переменная / метод задекларированны.

```
*/
public class Test {
    public int id1; // public
    private int id2; // private
    int id3; // default
    protected int id4; // protected

    public static void main(String[] args) {

    }

    public void method1() {}
    private void method2() {}
}
// Вспомогательные классы могут быть без модификатора доступа.
class Xxx{

}
/*
!!! Ошибка!!!
Не может быть два класса с модификатором public.

public class Xxx2{

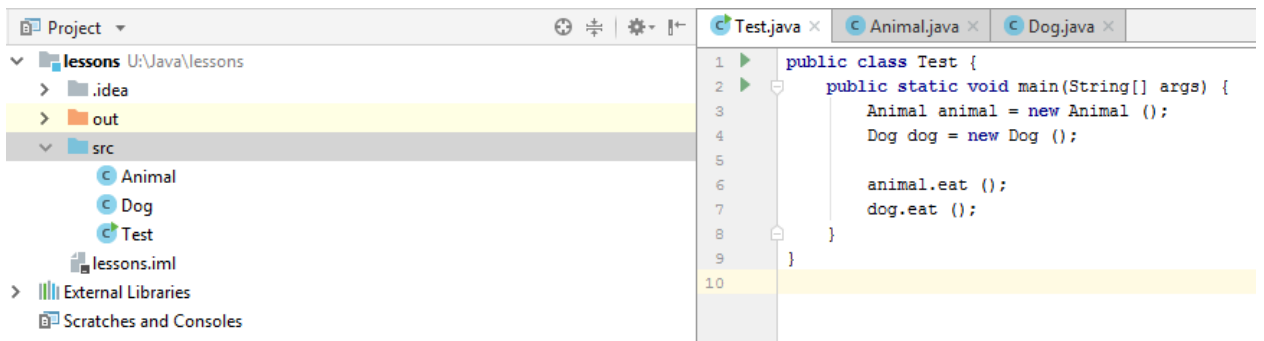
}
*/
```

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	X
no modifier	Y	Y	X	X
private	Y	X	X	X

Урок 29: Полиморфизм.

https://www.youtube.com/watch?v=c8eGIPF-o3Q&index=30&list=PLAma_mKfftOSUkXp26rgdnC0PicmnDak

В языках программирования и теории типов **полиморфизмом** называется способность функции обрабатывать данные разных типов.



```
public class Test {  
    public static void main(String[] args) {  
        Animal animal = new Animal ();  
        Dog dog = new Dog ();  
  
        animal.eat ();  
        dog.eat ();  
    }  
}  
  
public class Animal {  
    public void eat(){  
        System.out.println ("Animal is eating...");  
    }  
}  
  
public class Dog extends Animal{  
}
```

Вывод в консоль:

```
Animal is eating...  
Animal is eating...
```

```
public class Test {  
    public static void main(String[] args) {  
        Animal animal = new Dog ();  
        animal.eat ();  
        /*  
        !!! не можем обратиться в через animal к методу bark  
        как - animal.bark();  
        не все животные лают!!!  
        */  
        Dog dog = new Dog ();  
        dog.eat ();  
        dog.bark ();  
    }  
}  
  
public class Animal {  
    public void eat(){  
        System.out.println ("Animal is eating...");  
    }  
}
```

```
}  
}
```

```
public class Dog extends Animal{  
    public void bark(){  
        System.out.println ("Dog is barking...");  
    }  
}
```

Вывод в консоль:

```
Animal is eating...  
Animal is eating...  
Dog is barking...
```

Позднее связывание.

Переопределяем public void eat(){} в классе Dog

Объект такого типа и реализации

```
Animal animal = new Dog ();
```

выведет на экран переопределенный в потомке метод родителя eat таким образом:

"Dog is eating...".

Это и есть "Позднее связывание".

```
public class Test {  
    public static void main(String[] args) {  
        Animal animal = new Dog ();  
        animal.eat ();  
    }  
}
```

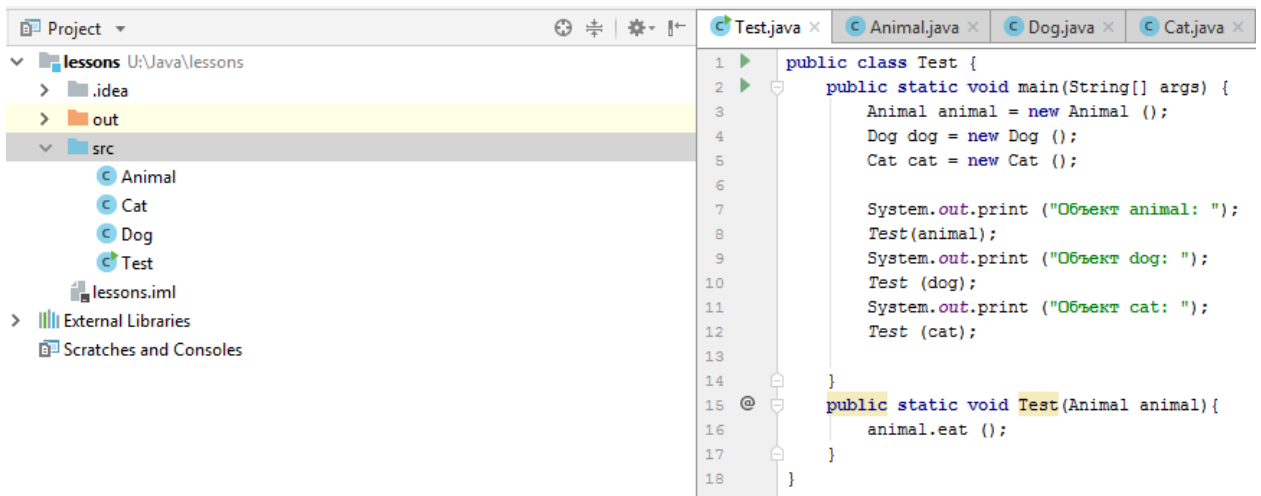
```
public class Animal {  
    public void eat(){  
        System.out.println ("Animal is eating...");  
    }  
}
```

```
public class Dog extends Animal{  
    /*  
    Переопределяем public void eat(){} в классе Dog  
    */  
    @Override  
    public void eat() {  
        System.out.println ("Dog is eating...");  
    }  
  
    public void bark(){  
        System.out.println ("Dog is barking...");  
    }  
}
```

Вывод в консоль:

```
Dog is eating...
```

Возможность передавать в метод разные типы, если у них один и тот же родитель.



```
public class Test {
    public static void main(String[] args) {
        Animal animal = new Animal ();
        Dog dog = new Dog ();
        Cat cat = new Cat ();

        System.out.print ("Объект animal: ");
        Test(animal);
        System.out.print ("Объект dog: ");
        Test (dog);
        System.out.print ("Объект cat: ");
        Test (cat);
    }

    public static void Test (Animal animal){
        animal.eat ();
    }
}
```

```
public class Animal {
    public void eat(){
        System.out.println ("Animal is eating...");
    }
}
```

```
public class Dog extends Animal{
    /*
     * Переопределяем public void eat(){} в классе Dog
     */
    @Override
    public void eat() {
        System.out.println ("Dog is eating...");
    }

    public void bark(){
        System.out.println ("Dog is barking...");
    }
}
```

```

    }
}

public class Cat extends Animal{
}

```

Вывод в консоль:

```

Объект animal: Animal is eating...
Объект dog: Dog is eating...
Объект cat: Animal is eating...

```

Урок 30: Приведение примитивных типов данных.

https://www.youtube.com/watch?v=bHWDG9bfDRg&index=31&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak

```

public class Test {
    public static void main(String[] args) {
        byte myByte;
        myByte = 16;
        System.out.println ("myByte - " + myByte);

        short myShort = 211;
        System.out.println ("myShort - " + myShort);

        int myInt = 557;
        System.out.println ("myInt - " + myInt);

        long myLong = 234546677L;
        System.out.println ("myLong - " + myLong);

        float myFloat = 234.0F;
        System.out.println ("myFloat - " + myFloat);

        double myDoble = 234.0;
        System.out.println ("myDoble - " + myDoble);

        boolean myBoolean = true;
        System.out.println ("myBoolean - " + myBoolean);

        char myChar = 'c';
        System.out.println ("myChar - " + myChar);
    }
}

```

Вывод в консоль:

```

myByte - 16
myShort - 211
myInt - 557

```

```
myLong - 234546677
myFloat - 234.0
myDoble - 234.0
myBoolean - true
myChar - c
```

```
public class Test {
    public static void main(String[] args) {
        /*
         * Все вещественные типы данных по умолчанию double. Поэтому, если мы
         * хотим применить тип float, мы должны указать это компилятору с помощью
         * литеры f или F таким образом:
         * float fl = 123.2f;
         * float fl = 123.2F;
         */
        float fl = 123.2F;
        /*
         * Тот же принцип и у long
         * Все целочисленные типы данных по умолчанию int. Поэтому, если мы хотим
         * применить тип long, мы должны указать это компилятору с помощью литеры
         * l или L таким образом:
         * long lg = 123456789l;
         * long lg = 123456789L;
         */
        long lg = 123L;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        double d;
        float f = 111.1F;

        // Неявное приведение типов.
        d = f;
        System.out.println ("Неявное приведение типов из float в double: " +
d);

        d = 222.0;

        // Явное приведение типов.
        f = (float) d;
        System.out.println ("Явное приведение типов из double в float: " + f);

        long lg;
        int i = 111;

        // Неявное приведение типов.
        lg = i;
        System.out.println ("Неявное приведение типов из int в long: " + lg);

        lg = 222L;

        // Явное приведение типов.
        i = (int) lg;
        System.out.println ("Явное приведение типов из long в int: " + i);
    }
}
```

Вывод в консоль:

Неявное приведение типов из float в double: 111.0999984741211
Явное приведение типов из double в float: 222.0
Неявное приведение типов из int в long: 111
Явное приведение типов из long в int: 222

```
public class Test {
    public static void main(String[] args) {
        double d;
        int i = 111;

        // Неявное приведение типов.
        d = i;
        System.out.println ("Неявное приведение типов из int в double: " + d);

        d = 222.123456;

        // Явное приведение типов. int i потеряет числа (d = 222.123456) после
        точки - .123456.
        i = (int) d;
        System.out.println ("Явное приведение типов из double в int: " + i);

        /*
        Если мы пытаемся в "коробку" переменной вместить большее по размеру
        число, то получим "урезанное" число.
        */
        byte b = (byte) 128; // Диапазон значений byte от -128 до 127
        System.out.println ("Явное приведение типов из int в byte большего
        значения (128): " + b);

        b = (byte) 256; // Диапазон значений byte от -128 до 127
        System.out.println ("Явное приведение типов из int в byte большего
        значения (256): " + b);
    }
}
```

Вывод в консоль:

Неявное приведение типов из int в double: 111.0
Явное приведение типов из double в int: 222
Явное приведение типов из int в byte большего значения (128): -128
Явное приведение типов из int в byte большего значения (256): 0

Урок 31: Классы-обертки примитивных типов данных.

https://www.youtube.com/watch?v=P7b_dzMFG7s&index=32&list=PLAma_mKfftOSUkXp26rgdnCOPicmnDak

Табл. 2.1. Классы-оболочки для базовых типов

Базовый тип	Класс-оболочка
<code>byte</code>	<code>Byte</code>
<code>short</code>	<code>Short</code>
<code>int</code>	<code>Integer</code>
<code>long</code>	<code>Long</code>
<code>float</code>	<code>Float</code>
<code>double</code>	<code>Double</code>
<code>char</code>	<code>Character</code>
<code>boolean</code>	<code>Boolean</code>

1. В Java переменные имеют тип.

Тип переменной определяет, какие данные могут записываться в переменную, какой объем памяти выделяется под переменную и какие операции могут выполняться с переменной.

2. В Java существует несколько базовых, или простых, типов данных.

- Целочисленные типы (`byte`, `short`, `int` и `long`).
- Действительные числовые типы (`float` и `double`).
- Символьный тип `char` (значение – буква или символ).
- Логический тип `boolean` (два значения – `true` и `false`).

3. При объявлении переменной указывается тип ее значения и имя.

Одновременно с объявлением, можно переменной присвоить значение (инициализировать переменную).

Несколько переменных можно объявлять одновременно.

Объявление и инициализация переменной производится в любом месте метода, но до ее первого использования.

4. Константа от переменной отличается тем, что значение константы после инициализации (которая выполняется вместе с объявлением) изменить нельзя.

Константы описываются так же, как переменные, но для них указывается еще ключевое слово `final`.

5. Для базовых типов существуют классы-оболочки:

`Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Character` и `Boolean` соответственно для типов

`byte`, `short`, `int`, `long`, `float`, `double`, `char` и `boolean`.

Класс-оболочка предназначен для реализации того или иного значения в виде объекта.

6. В отличие от примитивных типов данных у классов-оболочек имеется ряд полезных методов.

В этом их главное преимущество перед примитивами!!!

Например, у класса `Integer` есть статический метод `parseInt()`, с помощью которого текст, содержащий представление целого числа, преобразуется в число.

7. Классы-оболочки/обёртки лежат в `import java.lang` и импортируются по умолчанию.

```
public class Test {
    public static void main(String[] args) {
        int x = 123;
        System.out.println ("int x = 1; - " + x);
        Integer x2 = new Integer (123);
        System.out.println ("Integer x2 = new Integer (123); - " + x2);
    }
}
```



```

        Integer.parseInt ("123");
        System.out.println ("Integer.parseInt (\"123\"); - " + Integer.parseInt
("123"));
    }
}

```

Вывод в консоль:

```

int x = 1; - 123
Integer x2 = new Integer (123); - 123
Integer.parseInt ("123"); - 123

```

АвтоУпаковка и автоРаспаковка.

```

public class Test {
    public static void main(String[] args) {
        /*
        АвтоУпаковка выглядит так:
        Integer x2 = new Integer (123);
        Но это избыточная форма упаковки.
        Лучше делать так:
        Integer x2 = 123;
        или
        Integer x2 = x;
        */
        int x = 123;
        Integer x2 = 123;
        Integer x3 = x;
        System.out.println ("АвтоУпаковка int x = 123; - " + x);
        System.out.println ("АвтоУпаковка Integer x2 = 123; - " + x2);
        System.out.println ("АвтоУпаковка Integer x3 = x; - " + x3);

        System.out.println ();

        /*
        АвтоРаспаковка выглядит так:
        */
        int y;
        y = x2;
        System.out.println ("АвтоРаспаковка y = x2; - " + y);
        y = x3;
        System.out.println ("АвтоРаспаковка y = x3; - " + y);
    }
}

```

Вывод в консоль:

```

АвтоУпаковка int x = 123; - 123
АвтоУпаковка Integer x2 = 123; - 123
АвтоУпаковка Integer x3 = x; - 123

```

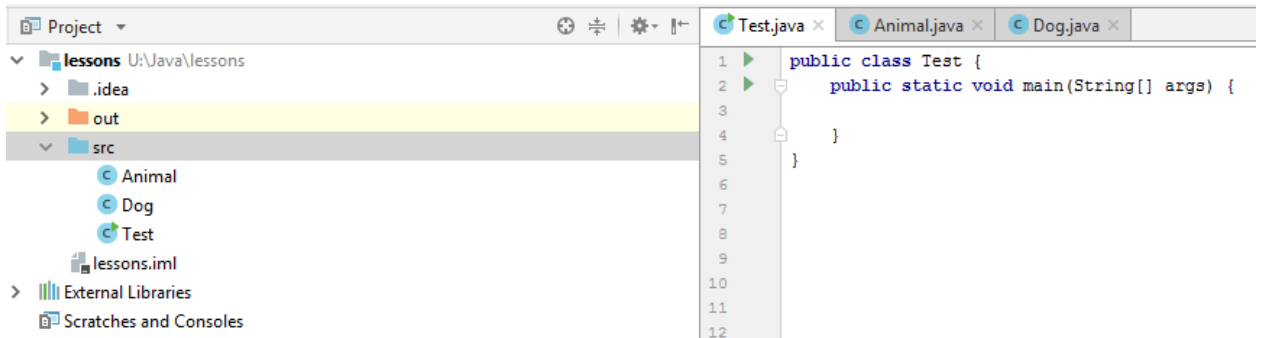
```

АвтоРаспаковка y = x2; - 123
АвтоРаспаковка y = x3; - 123

```

Урок 32: Восходящее и нисходящее преобразование.

https://www.youtube.com/watch?v=88P-SGqIeeE&index=33&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak



Восходящее преобразование.

```
public class Test {
    public static void main(String[] args) {
        /*
        UPcasting - Восходящее преобразование.
        Восходящее преобразование происходит неявно,
        как long = int и double = float.
        Animal animal = new Dog ();
        Animal animal <-- new Dog ();

        Class Animal
        ^ ^ ^ ^ ^
        Class Dog

        Можно записать это еще так:
        Dog dog = new Dog ();
        Animal animal = dog;
        */
        Dog dog = new Dog ();
        Animal animal = dog;
        /*
        !!! animal не может использовать метод bark();
        animal.bark();
        !!! Ошибка
        */

        /*
        DOWNcasting - Нисходящее преобразование.
        Нисходящее преобразование происходит явно, т.е. принудительно
        как int = (int) long и float = (float) double.
        Dog dog2 = (Dog) animal;
        Dog dog2 <-- (Dog) animal;
        */
        Dog dog2 = (Dog) animal;
        /*
        animal, преобразованная в dog может использовать метод bark();

```

```

        */
        System.out.print ("dog2.bark (); - ");
        dog2.bark ();
    }
}

```

Вывод в консоль:

```
dog2.bark (); - Dog is barking...
```

!!!ОШИБКА при нисходящем преобразовании.

```

public class Test {
    public static void main(String[] args) {
        /*
        DOWNcasting - ОПАСЕН!!!
        Если мы пытаемся Объект - Родитель преобразовать в Объект - наследник,
        В этом случае мы не изменяем Объект, а меняем ссылку на него.
        */
        Animal animal = new Animal ();
        Dog dog = (Dog) animal;
        dog.bark ();
        /*
        ЕСЛИ ВЫЗВАТЬ
        dog.bark ();
        Получим ошибку, как если бы мы вызвали bark(); из animal
        animal.bark();

        Это говорит о том, что:
        - Восходящее преобразование - БЕЗОПАСНО!!!
        Собака (Dog) всегда является животным (Animal).
        Собака (Dog) всегда ест (eat());).

        -Нисходящее преобразование - НЕБЕЗОПАСНО!!!
        Животное (Animal) не всегда является Собакой (Dog).
        Животное (Animal) не всегда лает (bark());).
        */
    }
}

```

Вывод в консоль:

```

Exception in thread "main" java.lang.ClassCastException: Animal cannot be cast
to Dog
    at Test.main(Test.java:9)

```

```
Process finished with exit code 1
```

Урок 33: Введение в параметризацию. (Generics).

https://www.youtube.com/watch?v=iQIR2Zgb93k&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak&index=34

До, и после появления дженериков.

```
import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        System.out.println ("////////// Java 5 //////////");
        List animals = new ArrayList ();
        animals.add ("can"); // add. - добавить значение Object "can" в
        animals. Index элемента в массиве - 0.
        animals.add ("dog"); // add. - добавить значение Object "dog" в
        animals. Index элемента в массиве - 1.
        animals.add ("frog"); // add. - добавить значение Object "frog" в
        animals. Index элемента в массиве - 2.
        /*
        get - получаем элемент "dog" из массива по index.
        Используем DOWNcasting - Нисходящее преобразование.
        Нисходящее преобразование происходит явно, т.е. принудительно
        как int = (int) long и float = (float) double.
        */

        String animal = (String) animals.get (1);
        System.out.println (animal);

        System.out.println ("////////// С появлением дженериков //////////");

        List<String> animals2 = new ArrayList<String> ();
        animals2.add ("can"); // add. - добавить значение <String> "can" в
        animals. Index элемента в массиве - 0.
        animals2.add ("dog"); // add. - добавить значение <String> "dog" в
        animals. Index элемента в массиве - 1.
        animals2.add ("frog"); // add. - добавить значение <String> "frog" в
        animals. Index элемента в массиве - 2.
        /*
        get - получаем элемент "dog" из массива по index.
        !!! НЕ !!! используем DOWNcasting - Нисходящее преобразование.
        Так как есть дженерик <String>.
        */

        String animal2 = animals2.get(1);
        System.out.println (animal2);

    }
}
```

Вывод в консоль:

```
////////// Java 5 //////////
dog
////////// С появлением дженериков //////////
dog
```

Создаем Свой Новый класс ourAnimal

```

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        System.out.println ("////////// Java 5 //////////");
        /*
        Создаем Свой Новый класс Animal и элемент ourAnimal.
        Кладем его как Объект в массив Object.
        Но преобразовать его в String и вывести его на печать, мы не сможем.
        */
        List animals = new ArrayList ();
        Animal ourAnimal = new Animal ();
        animals.add ("can"); // add. - добавить значение Object "can" в

        animals. Index элемента в массиве - 0.
        animals.add ("dog"); // add. - добавить значение Object "dog" в

        animals. Index элемента в массиве - 1.
        animals.add ("frog"); // add. - добавить значение Object "frog" в

        animals. Index элемента в массиве - 2.
        animals.add (ourAnimal);

        System.out.println ("////////// С появлением дженериков //////////");

        List<String> animals2 = new ArrayList<String> ();
        /*
        Создали Свой Новый класс Animal и элемент ourAnimal.
        !!! НО !!! положить его как Объект в массив ArrayList<String>
        мы НЕ СМОЖЕМ.
        */

        animals2.add ("can"); // add. - добавить значение <String> "can" в

        animals. Index элемента в массиве - 0.
        animals2.add ("dog"); // add. - добавить значение <String> "dog" в

        animals. Index элемента в массиве - 1.
        animals2.add ("frog"); // add. - добавить значение <String> "frog" в

        animals. Index элемента в массиве - 2.
        animals2.add (ourAnimal); // !!!ОШИБКА!!!
    }
}

class Animal{
}

```

Вывод в консоль:

```

Information:java: Some messages have been simplified; recompile with
-Xdiags:verbose to get full output
Information:java: Errors occurred while compiling module 'lessons'
Information:javac 1.8.0_171 was used to compile java sources
Information:13.08.2018 10:09 - Compilation completed with 1 error and 0
warnings in 1s 571ms
U:\Java\lessons\src\Test.java

```

Error:(31, 17) java: no suitable method found for add(Animal)
method java.util.Collection.add(java.lang.String) is not applicable
(argument mismatch; Animal cannot be converted to java.lang.String)
method java.util.List.add(java.lang.String) is not applicable
(argument mismatch; Animal cannot be converted to java.lang.String)
Information:java: U:\Java\lessons\src\Test.java uses unchecked or unsafe
operations.
Information:java: Recompile with -Xlint:unchecked for details.

```
import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        /////////// Java 5 ///////////

        List animals = new ArrayList ();
        animals.add ("can"); // add. - добавить значение Object "can" в
        animals. Index элемента в массиве - 0.
        animals.add ("dog"); // add. - добавить значение Object "dog" в
        animals. Index элемента в массиве - 1.
        animals.add ("frog"); // add. - добавить значение Object "frog" в
        animals. Index элемента в массиве - 2.

        /////////// Java 5 (С появлением дженериков "<String>") ///////////

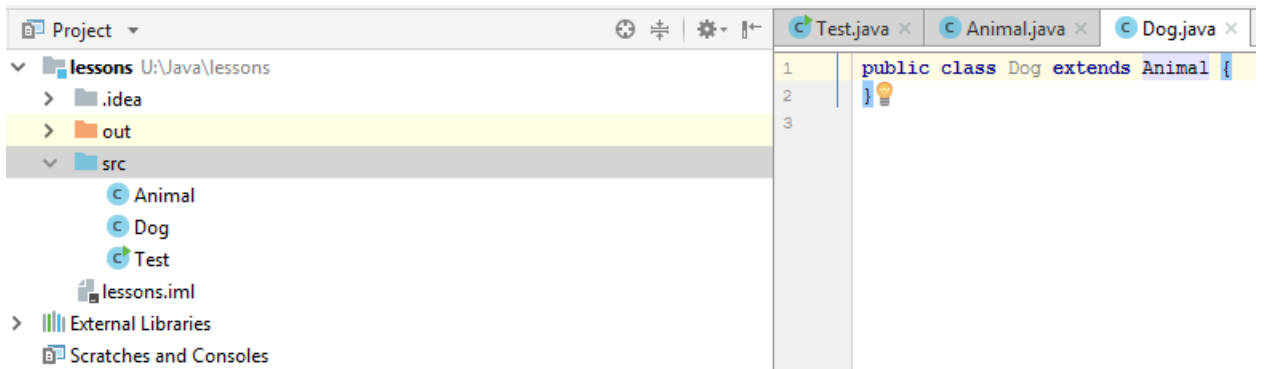
        List<String> animals2 = new ArrayList<String> ();
        animals2.add ("can"); // add. - добавить значение <String> "can" в
        animals. Index элемента в массиве - 0.
        animals2.add ("dog"); // add. - добавить значение <String> "dog" в
        animals. Index элемента в массиве - 1.
        animals2.add ("frog"); // add. - добавить значение <String> "frog" в
        animals. Index элемента в массиве - 2.

        /////////// Java 7 ///////////

        List<String> animals3 = new ArrayList<> (); //new ArrayList<> ();
        МОЖНО ПИСАТЬ БЕЗ String.
        animals3.add ("can"); // add. - добавить значение <String> "can" в
        animals. Index элемента в массиве - 0.
        animals3.add ("dog"); // add. - добавить значение <String> "dog" в
        animals. Index элемента в массиве - 1.
        animals3.add ("frog"); // add. - добавить значение <String> "frog" в
        animals. Index элемента в массиве - 2.
    }
}
```

Wildcards (Generics).

https://www.youtube.com/watch?v=Er_cj823ZTM&index=35&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak



```
import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;
```

```
public class Test {  
    public static void main(String[] args) {  
        List<Animal> listOfAnimal = new ArrayList<> ();  
        listOfAnimal.add (new Animal (1));  
        listOfAnimal.add (new Animal (2));  
  
        test (listOfAnimal);  
    }  
    private static void test(List<Animal> list){  
        for (Animal animal : list) {  
            System.out.println ("new Animal ( id: " + animal + ")");  
        }  
    }  
}
```

```
public class Animal {  
    private int id;  
  
    public Animal(int id){  
        this.id = id;  
    }  
  
    /*  
    Создав конструктор  
    public Animal(int id){  
        this.id = id;  
    }  
    мы удалили конструктор по-умолчанию  
    и поэтому класс Dog выдаст ошибку.  
    Надо создать дефолтный конструктор по-умолчанию.  
    public Animal(){  
    }  
    */  
  
    public Animal(){  
  
    }  
  
    public String toString(){  
  
        return String.valueOf (id);  
    }  
}
```

```
    }  
}
```

```
public class Dog extends Animal {  
}
```

Вывод в консоль:

```
new Animal ( id: 1)  
new Animal ( id: 2)
```

```
import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;
```

```
public class Test {  
    public static void main(String[] args) {  
        List<Animal> listOfAnimal = new ArrayList<> ();  
        listOfAnimal.add (new Animal (1));  
        listOfAnimal.add (new Animal (2));  
  
        List<Dog> listOfDogs = new ArrayList<> ();  
        listOfAnimal.add (new Dog ());  
        listOfAnimal.add (new Dog ());  
  
        test (listOfAnimal);  
        //      test(listOfDogs); // Почему-то и без него выводятся третья и  
        // четвертая строки на экран.  
  
        /*  
        Чтобы передать Объект "new Dog" () в метод test (listOfDogs); с  
        параметром "List<Animal> list"  
        private static void test(List<Animal> list){  
            for (Animal animal : list) {  
                System.out.println ("new Animal ( id: " + animal + ")");  
            }  
        }  
        мы должны изменить параметр метода <Animal> на <? extends Animal>.  
        */  
  
    }  
    private static void test(List<? extends Animal> list){  
        for (Animal animal : list) {  
            animal.eat ();  
        }  
    }  
}
```

/*
Если мы пишем <? super Animal> , то мы разрешаем подавать на вход метода лист, содержащий объекты класса Animal и всех суперклассов (родительских классов) класса Animal. В нашем случае, у класса Animal только один родительский класс - класс Object. То есть, в этом листе могут храниться объекты класса Animal и объекты класса Object. Почему же тогда мы не можем вызывать методы класса Animal внутри метода? А по той причине, что мы разрешаем хранить в этом листе объекты класса Object, у которых нет методов, определенных в классе Animal. Поэтому всегда, когда в wildcard мы пишем super, мы будем иметь доступ только к методам самого общего родителя, класса Object.


```

*/
}

public class Animal {
    private int id;

    public Animal(int id){
        this.id = id;
    }

    /*
    Создав конструктор
    public Animal(int id){
        this.id = id;
    }
    мы удалили конструктор по-умолчанию
    и поэтому класс Dog выдаст ошибку.
    Надо создать дефолтный констрктор по-умолчанию.
    public Animal(){
    }
    */

    public Animal(){

    }

    public void eat(){
        System.out.println ("Animal is eating... ");
    }

    public String toString(){

        return String.valueOf (id);
    }
}

public class Dog extends Animal {
}

```

Вывод в консоль:

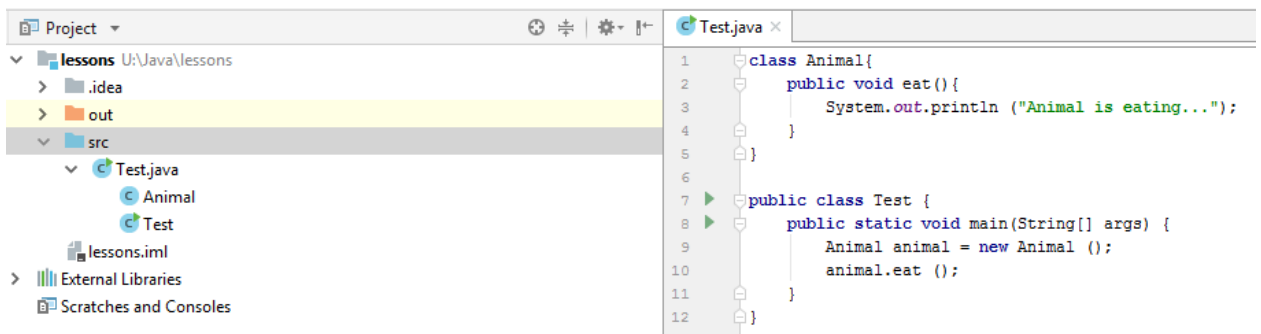
```

Animal is eating...
Animal is eating...
Animal is eating...
Animal is eating...

```

Урок 35: Анонимные классы.

https://www.youtube.com/watch?v=ndnubpPzkNE&index=36&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak



```
class Animal{
    public void eat(){
        System.out.println ("Animal is eating...");
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Animal animal = new Animal ();
        animal.eat ();
    }
}
```

Вывод в консоль:

Animal is eating...

Переопределяем метод eat, через создание класса OtherAnimal.

```
class Animal{
    public void eat(){
        System.out.println ("Animal is eating...");
    }
}

/*
Переопределяем метод eat,
через создание класса OtherAnimal.
*/

class OtherAnimal {
    public void eat(){
        System.out.println ("Other animal is eating...");
    }
}

public class Test {
    public static void main(String[] args) {
        Animal animal = new Animal ();
        animal.eat ();

        OtherAnimal otherAnimal = new OtherAnimal ();
        otherAnimal.eat ();
    }
}
```

Вывод в консоль:

```
Animal is eating...
Other animal is eating...
```

Тот же код, но с анонимным классом.

```
class Animal{
    public void eat(){
        System.out.println ("Animal is eating...");
    }
}

public class Test {
    public static void main(String[] args) {
        Animal animal = new Animal ();
        animal.eat ();

        /*
        Создаем Объект АНОНИМНОГО класса НАСЛЕДНИКА от класса Animal.
        !!! Не объект класса Animal, а его НАСЛЕДНИК
        (по типу class Xxx extends Animal{})..
        */

        Animal animal2 = new Animal () {
            @Override
            public void eat() {
                System.out.println ("Other animal is eating...");
            }
        };
        animal2.eat ();
    }
}
```

Вывод в консоль:

```
Animal is eating...
Other animal is eating...
```

Вариант использования без АНОНИМНОГО класса.

```
interface AbleToEat {
    public void eat();
}

class Animal implements AbleToEat{
    @Override
    public void eat(){
        System.out.println ("Do eat!!!");
    }
}

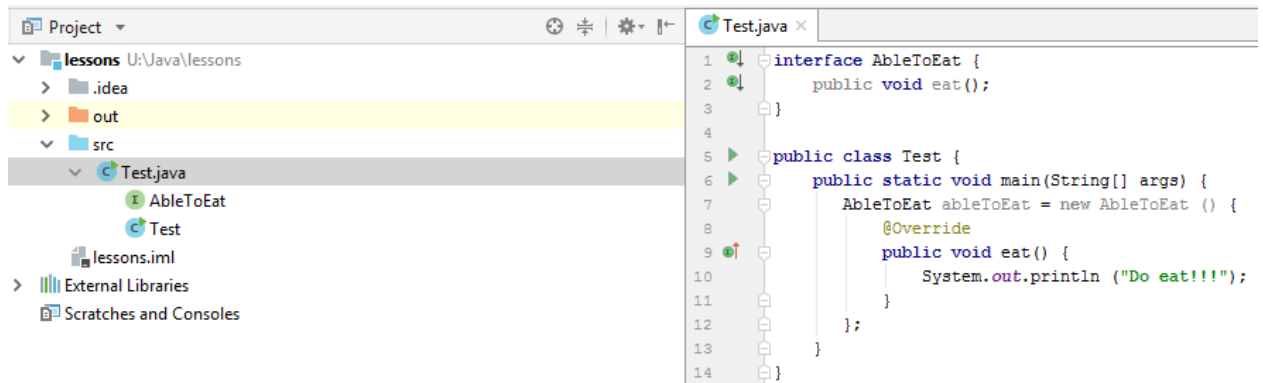
public class Test {
    public static void main(String[] args) {
        AbleToEat ableToEat = new Animal ();
        ableToEat.eat ();
    }
}
```

```
}  
}
```

Вывод в консоль:

Do eat!!!

Вариант использования с АНОНИМНЫМ классом.



```
interface AbleToEat {  
    public void eat();  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        AbleToEat ableToEat = new AbleToEat () {  
            @Override  
            public void eat() {  
                System.out.println ("Do eat!!!");  
            }  
        };  
    }  
}
```

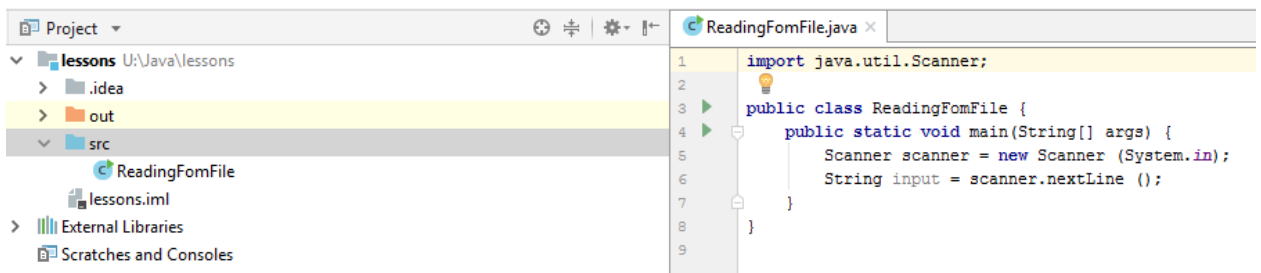
Вывод в консоль:

Do eat!!!

Урок 36: Чтение из файла.

https://www.youtube.com/watch?v=j3I-jLGW8yU&index=37&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak

Считывание данных с клавиатуры.



```
import java.util.Scanner;
```

```
public class ReadingFomFile {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner (System.in);  
        String input = scanner.nextLine ();  
        System.out.println ("Вывел на экран: " + input);  
    }  
}
```

Вывод в консоль:

ASDF

Вывел на экран: ASDF

Создаем файл "test" с каким-нибудь Строковым (String) "текстом"

```
" qwer  
 asdf  
 zxcv"
```

на Рабочем столе.

Перемещаем файл "test" в наш код.

Т.е. создаем абстракцию для работы с файлом.

Меняем слэши в пути к файлу в Windows: "\" -> "/"

C:\Users\Dragosh\Desktop\test

Чтобы не делать это вручную,

```
String path = "C:/Users/Dragosh/Desktop/test.txt";
```

делаем универсальную

для разных ОС систему разделителей в Пути к файлу "/" или "\".

Статическая переменная "separator" у каждой ОС своя.

НО, чтобы вообще не использовать запись Пути к файлу,

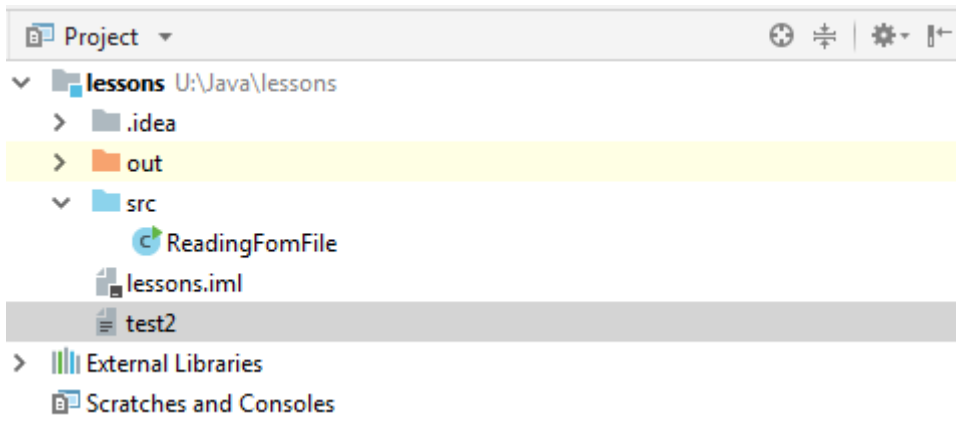
нужно создать файл "text2" в Корневой папке Проекта с программой, читающей файл.

Меняем

```
String path = "C:/Users/Dragosh/Desktop/test2.txt";
```

на

```
String path = "test2.txt";
```



```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadingFomFile {
    public static void main(String[] args) throws FileNotFoundException {
        String separator = File.separator;
        String path = "C:" + separator + "Users" + separator + "Dragosh" +
            separator + "Desktop" + separator + "test.txt";
        /*
         В Windows имя файла писать "ОБЯЗАТЕЛЬНО" с расширением "test.txt".
        */
        File file = new File (path);

        /*
         Компилятор выдаст ошибку, чтобы
         мы записали "throws FileNotFoundException" в
         public static void main(String[] args) throws FileNotFoundException
        {}

        */
        Scanner scanner = new Scanner (file);

        /*
         Считываем данные из файла.
         "hasNextLine" для считывания строк.
        */
        while (scanner.hasNextLine ()) {
            System.out.println (scanner.nextLine ());
        }

        /*
         !!! ОБЯЗАТЕЛЬНО ЗАКРЫТЬ ПОТОК scanner !!!
        */
        scanner.close ();
    }
}
```

Вывод в консоль:

```
qwer
asdf
zxcv
```

Создаем файл "test" с каким-нибудь числовым (int) "текстом" "1 2 3" на Рабочем столе.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Scanner;

public class ReadingFomFile {
    public static void main(String[] args) throws FileNotFoundException {
        String separator = File.separator;
        String path = "C:" + separator + "Users" + separator + "Dragosh" +
separator + "Desktop" + separator + "test.txt";
        File file = new File (path);

        Scanner scanner = new Scanner (file);
        // Считаем строку чисел (String).
        String line = scanner.nextLine ();

        /*
        Создаем массив String[],
        и помещаем в него разделенную на "слова" Строку.
        метод split (" "); делит Строку на слова.
        Его аргумент - знак для деления (Пробел).
        Если используем другой знак препинания (. или " или иной)
        обязательно экранируем его знаком - \ .
        */

        String[] numbers = line.split (" ");
        System.out.println (Arrays.toString (numbers));

        // Выводим числа на экран.
        for (String s : numbers) {
            System.out.println (Integer.parseInt (s));
        }

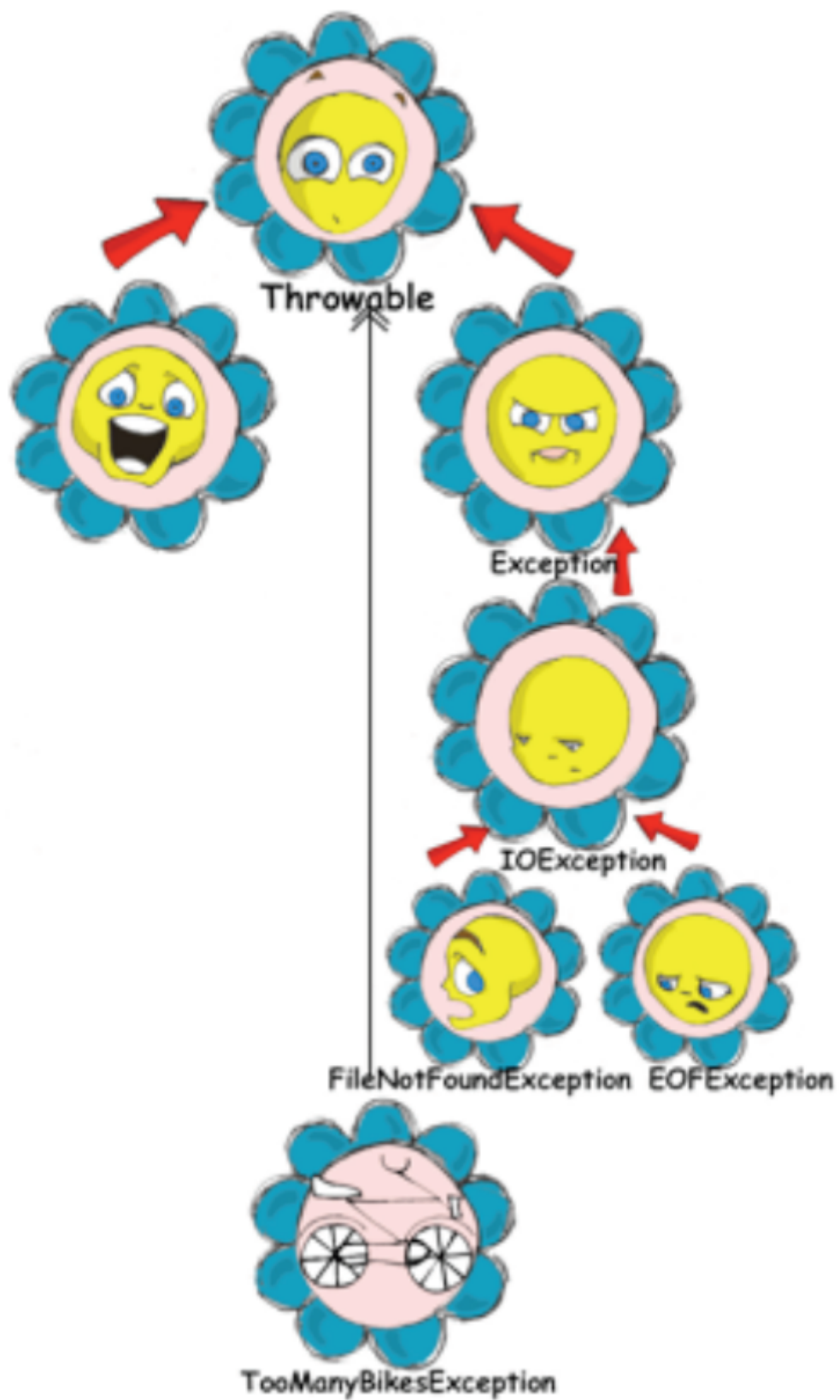
        /*
        !!! ОБЯЗАТЕЛЬНО ЗАКРЫТЬ ПОТОК scanner !!!
        */
        scanner.close ();
    }
}
```

Вывод в консоль:

```
[1, 2, 3]
1
2
3
```

Урок 37: Исключения (часть 1). Обработка исключений.

https://www.youtube.com/watch?v=DElNhj7lYCK&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak&index=38



Обработка ошибки с помощью
`throws FileNotFoundException,`
если файла **НЕТ**.

```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Exceptions1 {
    public static void main(String[] args) throws FileNotFoundException {
        /*
        Вводим Путь к несуществующему файлу.
        Исключение "throws FileNotFoundException"
        не даст жёстко закрыться программе.
        */
        File file = new File ("dfgsd");
        Scanner scanner = new Scanner (file);
    }
}

```

Вывод в консоль:

```

Exception in thread "main" java.io.FileNotFoundException: dfgsd (Не удается
найти указанный файл)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at java.util.Scanner.<init>(Scanner.java:611)
    at Exceptions1.main(Exceptions1.java:13)

```

Обработка ошибки с помощью

try/catch,
если файла НЕТ.

```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Exceptions1 {
    public static void main(String[] args){
        /*
        Вводим Путь к несуществующему файлу.
        Исключение "throws FileNotFoundException"
        не даст жёстко закрыться программе.
        */
        File file = new File ("dfgsd");
        try {
            Scanner scanner = new Scanner (file);
            System.out.println ("Эту фразу Вы не увидите, если файла НЕТ!!!");
        } catch (FileNotFoundException e) {
            /*
            e.printStackTrace (); - По-умолчанию.
            Вывод в консоль:
            java.io.FileNotFoundException: dfgsd (Не удается найти указанный
            файл)
            at java.io.FileInputStream.open0(Native Method)
            at java.io.FileInputStream.open(FileInputStream.java:195)
            at java.io.FileInputStream.<init>(FileInputStream.java:138)
            at java.util.Scanner.<init>(Scanner.java:611)
            at Exceptions1.main(Exceptions1.java:14)
            */

```

```

        System.out.println ("Файл не найден!!!");
    }
    System.out.println ("Код после блока try/catch...");
}
}

```

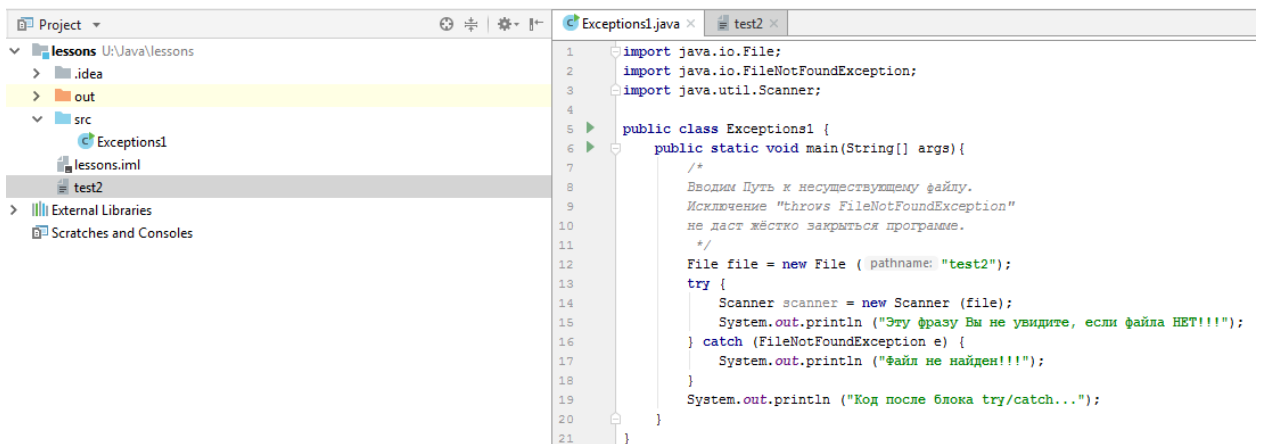
Вывод в консоль:

```

Файл не найден!!!
Код после блока try/catch...

```

Обработка ошибки с помощью
try/catch,
если файл ЕСТЬ.



```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Exceptions1 {
    public static void main(String[] args){
        /*
        Вводим Путь к несуществующему файлу.
        Исключение "throws FileNotFoundException"
        не даст жёстко закрыться программе.
        */
        File file = new File ("test2");
        try {
            Scanner scanner = new Scanner (file);
            System.out.println ("Эту фразу Вы не увидите, если файла
            НЕТ!!!");
        } catch (FileNotFoundException e) {
            System.out.println ("Файл не найден!!!");
        }
        System.out.println ("Код после блока try/catch...");
    }
}

```

Вывод в консоль:

```

Эту фразу Вы не увидите, если файла НЕТ!!!

```

Код после блока try/catch...

Обработка ошибки с помощью

ОДНОГО try/catch,

если файл ЕСТЬ, в методе readFile

```
import java.io.File;
```

```
import java.io.FileNotFoundException;
```

```
import java.util.Scanner;
```

```
public class Exceptions1 {  
    public static void main(String[] args){  
        readFile ();  
        System.out.println ("Код после блока try/catch...");  
    }  
    public static void readFile(){  
        File file = new File ("test2");  
        try {  
            Scanner scanner = new Scanner (file);  
            System.out.println ("Эту фразу Вы не увидите, если файла  
                НЕТ!!!");  
        } catch (FileNotFoundException e) {  
            System.out.println ("Файл не найден!!!");  
        }  
    }  
}
```

Вывод в консоль:

Эту фразу Вы не увидите, если файла НЕТ!!!

Код после блока try/catch...

Обработка ошибки с помощью

ДВУХ "throws FileNotFoundException",

если файл ЕСТЬ, в методе readFile

```
import java.io.File;
```

```
import java.io.FileNotFoundException;
```

```
import java.util.Scanner;
```

```
public class Exceptions1 {  
    public static void main(String[] args) throws FileNotFoundException {  
        readFile ();  
    }  
    public static void readFile() throws FileNotFoundException {  
        File file = new File ("test2");  
        Scanner scanner = new Scanner (file);  
    }  
}
```

Вывод в консоль:

Process finished with exit code 0

Замечание: **"throws FileNotFoundException"** и блок **try/catch** можно использовать совместно.

Урок 38: Исключения (часть 2). Выбрасывание исключений.

https://www.youtube.com/watch?v=jL7-VdBeh9s&index=39&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak

Проверка введенных данных
с помощью **"throws IOException"**.
Вводим только ноль.
Если нет - ошибка.

```
import java.io.IOException;
import java.util.Scanner;

public class Exceptions2{
    public static void main(String[] args) throws IOException{
        Scanner scanner = new Scanner (System.in);
        while (true){
            int x = Integer.parseInt (scanner.nextLine ());

            if (x != 0){
                throw new IOException();
            }
        }
    }
}
```

Вывод в консоль:

```
0
0
0
3
Exception in thread "main" java.io.IOException
    at Exceptions2.main(Exceptions2.java:14)
```

Проверка введенных данных
с помощью блока "**"**
Вводим только ноль.
Если нет - ошибка.

```
import java.io.IOException;
import java.util.Scanner;

public class Exceptions2{
    public static void main(String[] args){
        Scanner scanner = new Scanner (System.in);

        while (true){
```

```

        int x = Integer.parseInt (scanner.nextLine ());

        if (x != 0) {
            try {
                throw new IOException ();
            }
            catch (IOException e) {
                System.out.println ("Вы ввели НЕ ноль!!!");
            }
        }
    }
}

```

Вывод в консоль:

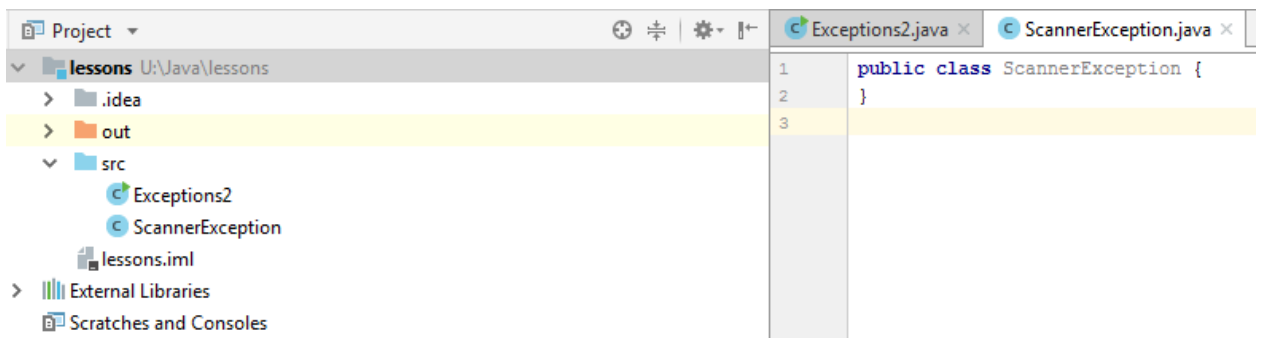
```

0
0
0
3
Вы ввели НЕ ноль!!!

```

Создаем СВОЁ ("ScannerException") исключение ("ToMenyBikeException" - Смотреть рисунок 37 урока).

Создаем новый класс ScannerException.



ВАРИАНТ 1.

```

import java.util.Scanner;

public class Exceptions2{
    public static void main(String[] args) throws ScannerException{
        Scanner scanner = new Scanner (System.in);

        while (true){
            int x = Integer.parseInt (scanner.nextLine ());

            if (x != 0) {
                try {
                    throw new ScannerException ();
                } catch (ScannerException e) {
                    System.out.println ("Вы ввели НЕ ноль!!!");
                }
            }
        }
    }
}

```

```

    }
}

```

```

public class ScannerException extends Exception{
}

```

Вывод в консоль:

```

0
0
0
3
Вы ввели НЕ ноль!!!

```

ВАРИАНТ 2.

```

import java.util.Scanner;

public class Exceptions2{
    public static void main(String[] args) throws ScannerException{
        Scanner scanner = new Scanner (System.in);

        while (true){
            int x = Integer.parseInt (scanner.nextLine ());

            if (x != 0) {
                throw new ScannerException ("Вы ввели НЕ ноль!!!");
            }
        }
    }
}

```

```

public class ScannerException extends Exception{
    public ScannerException(String description){
        super(description);
    }
}

```

Вывод в консоль:

```

0
0
0
3
Exception in thread "main" ScannerException: Вы ввели НЕ ноль!!!
    at Exceptions2.main(Exceptions2.java:12)

```

Урок 39: Исключения (часть 3). Checked и Unchecked исключения.

https://www.youtube.com/watch?v=P7dByAlrz5c&index=40&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak

Все исключения в Java делятся на:

- Checked Exception ("**Compile time exception**" - Проверяемое Исключение);
- Unchecked Exception ("**Runtime exception**" - Исключение, возникающее во время выполнения программы).

Все исключения имплементируются от Class Exception.

Все они "Checked Exception", кроме одного - это "**Runtime exception**".

[AclNotFoundException](#), [ActivationException](#), [AlreadyBoundException](#), [ApplicationException](#), [AWTException](#), [BackingStoreException](#), [BadAttributeValueExpException](#), [BadBinaryOpValueExpException](#), [BadLocationException](#), [BadStringOperationException](#), [BrokenBarrierException](#), [CertificateException](#), [CloneNotSupportedException](#), [DataFormatException](#), [DatatypeConfigurationException](#), [DestroyFailedException](#), [ExecutionException](#), [ExpandVetoException](#), [FontFormatException](#), [GeneralSecurityException](#), [GSSException](#), [IllegalClassFormatException](#), [InterruptedException](#), [IntrospectionException](#), [InvalidApplicationException](#), [InvalidMidiDataException](#), [InvalidPreferencesFormatException](#), [InvalidTargetObjectTypeException](#), [IOException](#), [JAXBException](#), [JMEException](#), [KeySelectorException](#), [LastOwnerException](#), [LineUnavailableException](#), [MarshalException](#), [MidiUnavailableException](#), [MimeTypeParseException](#), [MimeTypeParseException](#), [NamingException](#), [NoninvertibleTransformException](#), [NotBoundException](#), [NotOwnerException](#), [ParseException](#), [ParserConfigurationException](#), [PrinterException](#), [PrintException](#), [PrivilegedActionException](#), [PropertyVetoException](#), [ReflectiveOperationException](#), [RefreshFailedException](#), [RemarshalException](#), [RuntimeException](#), [SAXException](#), [ScriptException](#), [ServerNotActiveException](#), [SOAPException](#), [SQLException](#), [TimeoutException](#), [TooManyListenersException](#), [TransformerException](#), [TransformException](#), [UnmodifiableClassException](#), [UnsupportedAudioFileException](#), [UnsupportedCallbackException](#), [UnsupportedFlavorException](#), [UnsupportedLookAndFeelException](#), [URIReferenceException](#), [URISyntaxException](#), [UserException](#), [XAException](#), [XMLParseException](#), [XMLSignatureException](#), [XMLStreamException](#), [XPathException](#).

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Exception3 {
    public static void main(String[] args) {
        File file = new File ("test");
        /*
        Checked Exception ("Compile time exception" - Проверяемое Исключение)
        !!! МЫ ПРЕДПОЛАГАЕМ, что файл может быть не найден. По этому на
        всякий случай обрабатываем ВОЗМОЖНОЕ отсутствие файла.
        */
    }
}
```



```

    try {
        Scanner scanner = new Scanner (file);
    } catch (FileNotFoundException e) {
        System.out.println ("Мы обработали Исключение!!!");
    }

    /*
    Unchecked Exception ("Runtime exception" - Исключение, возникающее во
    время выполнения программы)
    !!! Мы ЗНАЕМ, что делить число на ноль - ЗАПРЕЩЕНО. ЭТО ОШИБКА,
    которую нужно УДАЛИТЬ, а не обработать как Исключение.
    Ниже примеры "популярных" "Runtime exception".
    */

    System.out.println ("Exception: int a = 1 / 0;");
    int a = 1 / 0;

    System.out.println ("Exception: String name = null; name.length ();");
    String name = null;
    name.length ();

    System.out.println ("Exception: int[] arr = new int[2];
    System.out.println (arr[2]);");
    int[] arr = new int[2];
    System.out.println (arr[2]);

    }
}

```

Вывод в консоль:

```

Мы обработали Исключение!!!
Exception: int a = 1 / 0;
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Exception3.main(Exception3.java:25)

Exception: String name = null; name.length ();
Exception in thread "main" java.lang.NullPointerException
    at Exception3.main(Exception3.java:29)

Exception: int[] arr = new int[2]; System.out.println (arr[2]);
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
    at Exception3.main(Exception3.java:33)

```

На случай, если нужно все же ОБРАБОТАТЬ ОШИБКУ.

```

public class Exception3 {
    public static void main(String[] args) {
        /*
        На случай, если нужно все же ОБРАБОТАТЬ ОШИБКУ с "Runtime exception"
        */

        int[] arr = new int[2];
        try {
            System.out.println (arr[2]);
        }
        catch (RuntimeException e){

```

```

        System.out.println ("ОШИБКА!!! Вы вышли за пределы массива
arr!!!");
    }
}
}

```

Вывод в консоль:

ОШИБКА!!! Вы вышли за пределы массива arr!!!

Урок 40: Исключения (часть 4).

https://www.youtube.com/watch?v=9gw81XDJoKs&index=41&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak

Для обработки ошибок разных типов можно одновременно использовать сразу несколько catch-блоков с Исключениями
!!! Если Исключения Родитель и Наследник, то сначала в Блоке идет Наследник, а затем Родитель. В противном случае Родитель перехватит Ошибку Наследника.
При этом роль Наследника в блоке утратится.

```

import java.io.IOException;

public class Exception4 {
    public static void main(String[] args) {
        try {
            run();
        } catch (IllegalAccessException e) {
            e.printStackTrace ();
        } catch (IOException e) {
            e.printStackTrace ();
        }
    }
    public static void run() throws IOException, IllegalAccessException {

    }
}

```

Вывод в консоль:

Process finished with exit code 0

Multy-Catch

```

import java.io.IOException;

public class Exception4 {
    public static void main(String[] args) {
        try {
            run();
        } catch (IllegalAccessException | IOException e) {
            e.printStackTrace ();
        }
    }
}

```

```

    }
}

public static void run() throws IOException, IllegalAccessException {

}

}

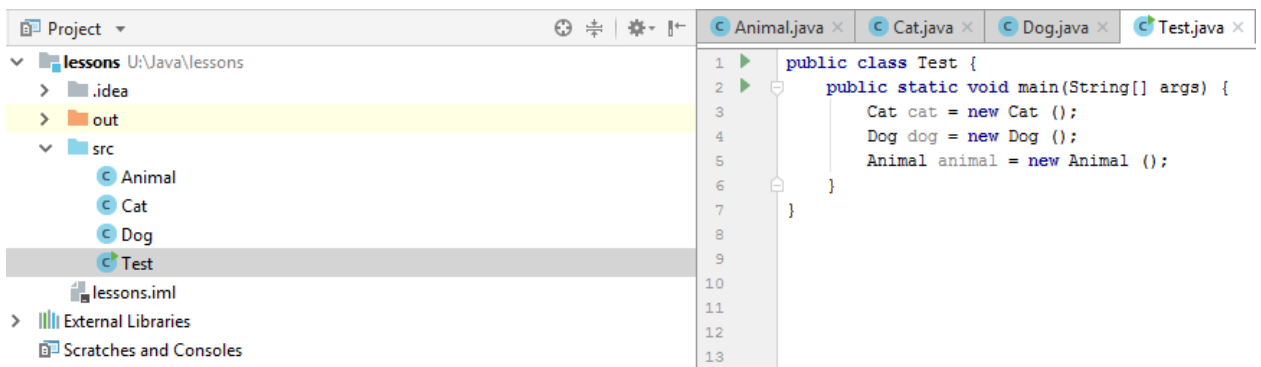
```

Вывод в консоль:

Process finished with exit code 0

Урок 41: Абстрактные классы.

https://www.youtube.com/watch?v=kY07wFP2JiA&list=PLAma_mKffTOSUkXp26rgdnC0PicmnDak&index=42



Создадим объекты классов:

- Animal animal = new Animal ();
- Cat cat = new Cat ();
- Dog dog = new Dog ();

```

public class Test {
    public static void main(String[] args) {
        Animal animal = new Animal ();
        Cat cat = new Cat ();
        Dog dog = new Dog ();
    }
}

```

```

public class Animal {
    public void eat(){
        System.out.println ("Animal is eating...");
    }
}

```

```

public class Cat extends Animal{
    public void makeSound(){
        System.out.println ("Cat said Meow...");
    }
}

```

```

    }
}

public class Dog extends Animal{
    public void makeSound(){
        System.out.println ("Dog said Woff...");
    }
}

```

Вывод в консоль:

Process finished with exit code 0

!!!НО...

Мы можем увидеть Кошку (cat) и Собаку (dog) .

А как увидеть Животное (animal)?

Животное (animal) - мы не можем создавать и в программе, потому что - это АБСТРАКЦИЯ - ОБЩЕЕ ОПИСАНИЕ гипотетического Животного (Собака, Кошка, Лягушка). И это животное не может быть в одном Теле (Объекте).

По этому МЕНЯЕМ:

```
public class Animal { -> public abstract class Animal {
```

После этого мы не сможем создать Объект Абстрактного Класа Animal!!!

```

public class Test {
    public static void main(String[] args) {
        Cat cat = new Cat ();
        Dog dog = new Dog ();
    }
}

public abstract class Animal {
    public void eat(){
        System.out.println ("Animal is eating...");
    }
}

```

```

public class Cat extends Animal{
    public void makeSound(){
        System.out.println ("Cat said Meow...");
    }
}

```

```

public class Dog extends Animal{
    public void makeSound(){
        System.out.println ("Dog said Woff...");
    }
}

```

Вывод в консоль:

Process finished with exit code 0

Создаем АБСТРАКТНЫЕ Методы.

Только в абстрактных классах мы создаем методы БЕЗ ТЕЛА!!!

И в классах наследниках мы должны имплементировать эти АБСТРАКТНЫЕ методы.

И внести в эти методы действия свойственные объекту этого класса (Кошка- **"Cat said Meow..."** , а Собака- **"Dog said Woff..."**)

```
public class Test {
    public static void main(String[] args) {
        Cat cat = new Cat ();
        Dog dog = new Dog ();

        cat.makeSound ();
        dog.eat ();
        dog.makeSound ();
        cat.eat ();
    }
}

public abstract class Animal {
    public void eat(){
        System.out.println ("Animal is eating...");
    }
    public abstract void makeSound();
}

public class Cat extends Animal{
    @Override
    public void makeSound() {
        System.out.println ("Cat said Meow...");
    }
}

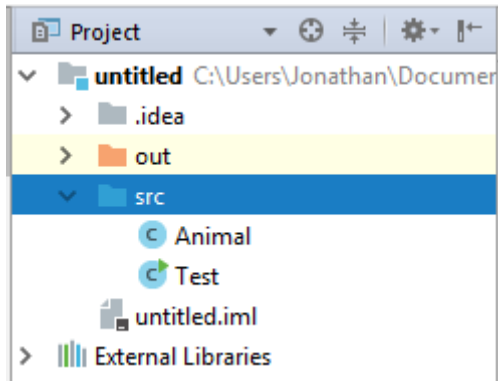
public class Dog extends Animal{
    @Override
    public void makeSound() {
        System.out.println ("Dog said Woff...");
    }
}
```

Вывод в консоль:

```
Cat said Meow...
Animal is eating...
Dog said Woff...
Animal is eating...
```

Урок 42: Метод equals() и String Pool.

https://www.youtube.com/watch?v=m7vFGL-N9eY&index=43&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak



Метод equals() ДЛЯ СРАВНЕНИЯ ОБЪКТОВ.

```
public class Test {
    public static void main(String[] args) {
        /*
        Примитивы сравниваются, как числа в арифметике.
        Но вместо "=", используем "==".
        */
        int x = 1;
        int y = 1;
        System.out.print("int x == int y : ");
        System.out.println(x == y );

        /*
        Сравниваем объекты класса Animal.
        Объекты НЕ сравниваются, как числа в арифметике.
        */
        Animal animal1 = new Animal();
        Animal animal2 = new Animal();
        System.out.print("Animal animal1 == Animal animal2 : ");
        System.out.println(animal1 == animal2 );
        /*
        В результате получим
        Animal animal1 == Animal animal2 : false.
        Потому что мы сравнили аббревиатуру ссылок animal1 и animal2,
        а не сами Объекты animal1 и animal2 (их информационное содержание).
        */
    }
}

public class Animal {
}
```

Вывод в консоль:

```
int x == int y : true
```

```
Animal animal1 == Animal animal2 : false
```

Создаем конструктор для класса Animal.
Объектам присваиваем РАВНЫЕ ЗНАЧЕНИЯ «1» И «1».

```
public class Test {
    public static void main(String[] args) {
        /*
        Примитивы сравниваются, как числа в арифметике.
        Но вместо "=", используем "==".
        */
        int x = 1;
        int y = 1;
        System.out.print("int x == int y : ");
        System.out.println(x == y );

        /*
        Сравниваем объекты класса Animal.
        Объекты НЕ сравниваются, как числа в арифметике.
        */
        Animal animal1 = new Animal(1);
        Animal animal2 = new Animal(1);
        System.out.print("Animal animal1 == new Animal(1) == Animal animal2 =
new Animal(1) : ");
        System.out.println(animal1 == animal2 );
        /*
        В результате получим
        Animal animal1 = new Animal(1) == Animal animal2 = new Animal(1) :
false.
        Потому что мы сравнили аббревиатуру ссылок animal1 и animal2,
        а не сами Объекты animal1 и animal2 (их информационное содержание).
        */
    }
}
```

```
public class Animal {
    private int id;

    Animal(int v){
        id = v;
    }
}
```

Вывод в консоль:

```
int x == int y : true
Animal animal1 = new Animal(1) == Animal animal2 = new Animal(1) : false
```

Используем для сравнения метод equals.()

```
public class Test {
    public static void main(String[] args) {
        /*
        Используем для сравнения метод equals.()
        */
    }
}
```

```

        Animal animal1 = new Animal(1);
        Animal animal2 = new Animal(1);
        System.out.print("animal1.equals(animal2) : ");
        System.out.println(animal1.equals(animal2) );
        /*
        В результате получим
        animal1.equals(animal2) : false.
        Потому что мы ТАК ЖЕ сравнили аббревиатуру ссылок animal1 и animal2,
        а не сами Объекты animal1 и animal2 (их информационное содержание).
        */
    }
}

```

```

public class Animal {
    private int id;

    Animal(int v){
        id = v;
    }
}

```

Вывод в консоль:

```

animal1.equals(animal2) : false

```

Сравниваем Объекты animal1 и animal2 СТРУКТУРНО (их информационное содержание). Для этого переопределяем метод equals.() в классе Animal таким образом.

```

public boolean equals(Object obj) {
    // Применяем ДаурКастинг.
    Animal otherAnimal = (Animal) obj;
    return id == otherAnimal.id;
}

```

```

public class Test {
    public static void main(String[] args) {
        /*
        Используем для сравнения ПЕРЕОПРЕДЕЛЕННЫЙ метод equals.()
        */
        Animal animal1 = new Animal(1);
        Animal animal2 = new Animal(1);
        System.out.print("animal1.equals(animal2) : ");
        System.out.println(animal1.equals(animal2) );
        /*
        В результате получим animal1."ПЕРЕОПРЕДЕЛЕННЫЙ"equals(animal2) : true.
        Потому что мы ТАК ЖЕ сравнили СТРУКТУРНО сами Объекты animal1 и animal2 (их
        информационное содержание).
        */
    }
}

```

```

public class Animal {
    private int id;

```



```

Animal(int v){
    id = v;
}

public boolean equals(Object obj) {
    // Применяем ДаурКастинг.
    Animal otherAnimal = (Animal) obj;
    return id == otherAnimal.id;
}
}

```

Вывод в консоль:

```
animal1.equals(animal2) : true
```

Метод equals() ДЛЯ СРАВНЕНИЯ Строк (String).
 !!!Напомню, что String тоже Объект.

```

public class Test {
    public static void main(String[] args) {
        /*
        Метод equals() ДЛЯ СРАВНЕНИЯ Строк (String).
        !!!String тоже Объект.
        String string2 = new String("Hello");
        */

        String string2 = new String("Hello");
        String string3 = new String("Hello");

        System.out.print("String string2 = new String(\"Hello\") == String
string3 = new String(\"Hello\") : ");
        System.out.println(string2 == string3 );

        System.out.print("string2.equals(string3) : ");
        System.out.println(string2.equals(string3) );

        System.out.println();

        /*
        Из-за наличия в Java String Pool
        !!!ЗАПРЕЩЕНО СРАВНИВАТЬ String КАК ПРИМИТИВЫ!!!
        !!! string5 == string6 !!!
        РАЗНЫЕ ОБЪЕКТЫ string5 И string6 ИМЕЮТ ОДИНАКОВЫЙ ТЕКСТ "Hello"
        (String Pool).
        ПОЭТОМУ ОНИ ССЫЛАЮТСЯ НА ОДИН И ТОТ ЖЕ ТЕКСТ "Hello".
        И ИМЕЮТ ОДИНАКОВЫЕ ССЫЛКИ(ОДИНАКОВУЮ АББРЕВИАТУРУ ССЫЛОК).
        */
        String string5 = "Hello";
        String string6 = "Hello";

        System.out.print("String string5 = \"Hello\" == String string6 =
\"Hello\" : ");
        System.out.println(string5 == string6 );

        System.out.print("string5.equals(string6) : ");
        System.out.println(string5.equals(string6) );
    }
}

```

```

    }
}

```

Вывод в консоль:

```

String string2 = new String("Hello") == String string3 = new String("Hello") :
false
string2.equals(string3) : true

```

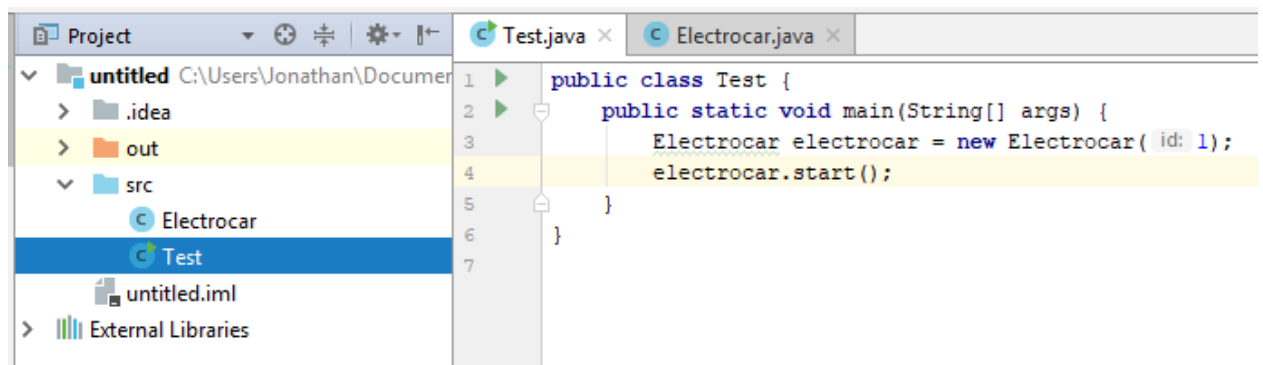
```

String string5 = "Hello" == String string6 = "Hello" : true
string5.equals(string6) : true

```

Урок 43: Вложенные классы.

https://www.youtube.com/watch?v=Rh6n-3TOJD4&list=PLAma_mKffTOSUkXp26rgdnC0PicnmDak&index=44



Существует ТРИ вида ВЛОЖЕННЫХ классов.

1. **ВЛОЖЕННЫЙ неСТАТИЧЕСКИЙ** private КЛАСС Motor.
Он имеет доступ к полям КЛАССА Electrocar.
2. **ВЛОЖЕННЫЙ СТАТИЧЕСКИЙ** public КЛАСС Battery.
Он НЕ имеет доступ к полям КЛАССА Electrocar.
Он имеет доступ только к статическим полям КЛАССА Electrocar.
3. **ВЛОЖЕННЫЙ КЛАСС в МЕТОД** start.
Он имеет доступ к НЕстатическим полям КЛАССА Electrocar.
Он имеет доступ ТОЛЬКО к КОНСТАНТНЫМ (final) полям МЕТОДА start.

```

public class Test {
    public static void main(String[] args) {
        Electrocar electrocar = new Electrocar(1);
        electrocar.start();

        Electrocar.Battery battery = new Electrocar.Battery();
    }
}

```

```

public class Electrocar {
    private int id;
    /*

```

```

ВЛОЖЕННЫЙ неСТАТИЧЕСКИЙ private КЛАСС Motor.
Он имеет доступ к полям КЛАССА Electrocar.
*/
private class Motor{
    public void startMotor() {
        System.out.println("Motor " + id + " is starting...");
    }
}

/*
ВЛОЖЕННЫЙ СТАТИЧЕСКИЙ public КЛАСС Battery.
Он НЕ имеет доступ к полям КЛАССА Electrocar.
Он имеет доступ ТОЛЬКО к статическим полям КЛАССА Electrocar.
*/
public static class Battery {
    public void charge(){
        System.out.println("Battery is charging...");
    }
}

public Electrocar (int id){
    this.id = id;
}

public void start(){
    Motor motor = new Motor();
    motor.startMotor();

    final int x = 1;

    /*
    ВЛОЖЕННЫЙ КЛАСС в МЕТОДА start.
    Он имеет доступ к нестатическим полям КЛАССА Electrocar.
    Он имеет доступ ТОЛЬКО к КОНСТАНТНЫМ (final) полям МЕТОДА start.
    */
    class SomeClass{
        public void someClass(){
            System.out.println(id);
            System.out.println(x);
        }
    }
    System.out.println("Electrocar " + id + " is started...");
}
}

```

Вывод в консоль:

```

Motor 1 is starting...
Electrocar 1 is started...

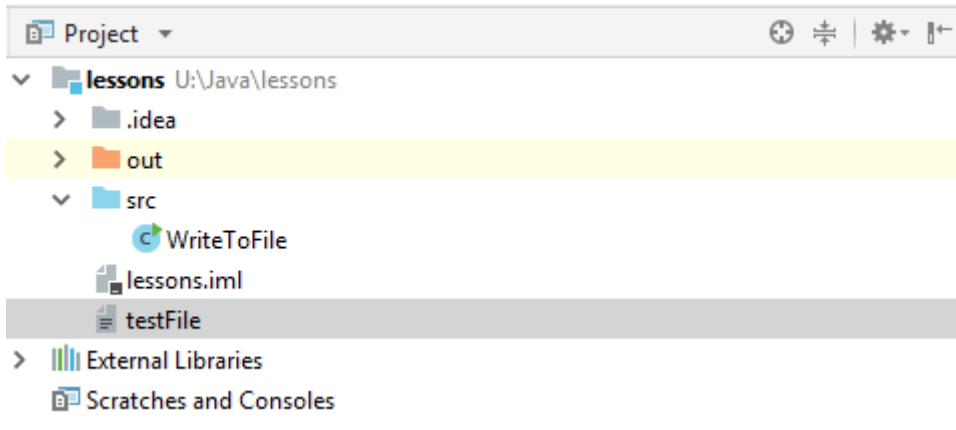
```

Урок 44: Запись в файл.

https://www.youtube.com/watch?v=nmxeAO7CYVg&index=45&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak

Запись только текстовой информации (**НЕ БАЙТОВОЙ!!! - НЕ Сериализация:** Объекты классов, музыка, и пр.).

Создаем в корне проекта текстовый файл "testFile".



```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

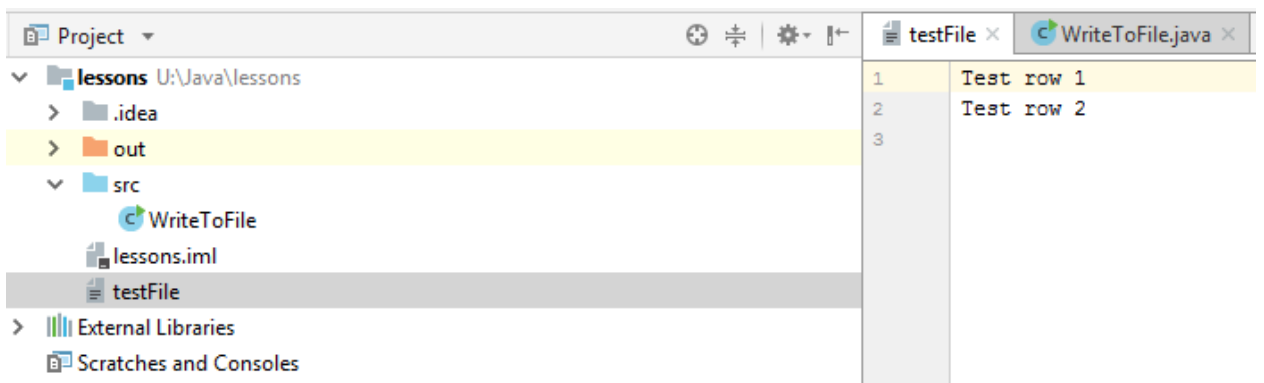
public class WriteToFile {
    public static void main(String[] args) throws FileNotFoundException {
        File file = new File ("testFile");
        PrintWriter pw = new PrintWriter (file);

        /*
        Для записи Текста в файл используем похожий на
        метод "System.out.println (System.in);".
        В аргументах метода указываем Место (Система - Консоль), куда мы
        хотим вывести инфотмацию.

        Метод "pw.println (file);".
        В аргументах метода указываем Текст, который мы хотим поместить в
        Файл (testFile.txt).
        */
        pw.println ("Test row 1");
        pw.println ("Test row 2");

        /*
        !!! ОБЯЗАТЕЛЬНО ЗАКРЫТЬ ПОТОК!!!
        */
        pw.close ();
    }
}
```

Вывод в Файл:



Test row 1
Test row 2

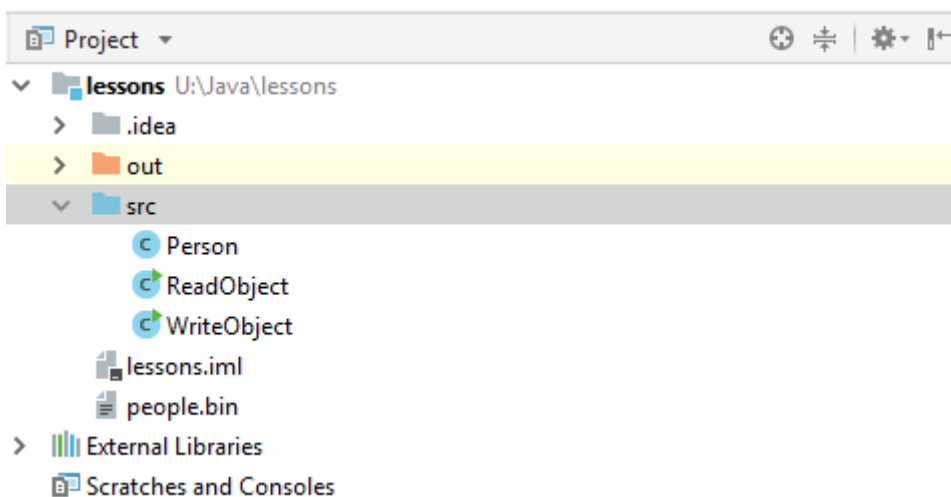
Урок 45: Сериализация (часть 1).

https://www.youtube.com/watch?v=dBcgizwOWLq&list=PLAma_mKffTOSUkXp26rgdnC0PicmnDak&index=46

Создаем Классы

- WriteObject
- ReadObject
- Person
- Файл **"people.bin"**.

В Классах WriteObject и ReadObject будет поочерёдно запускать метод main, в зависимости от того, что мы будем делать: записывать в файл или считывать с него.



ЗАПИСЬ В ФАЙЛ.

```
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.io.ObjectOutputStream;  
  
public class WriteObject {
```

```

public static void main(String[] args) {
    Person person1 = new Person (1, "Bob");
    Person person2 = new Person (2, "Mike");

    /*
    Расширение файла "people.bin" можно не указывать.
    Расширение ".bin" используется для записи бинарных данных.
    */
    try {
        // FileOutputStream - для записи ЛЮБЫХ бинарных данных.
        FileOutputStream fos = new FileOutputStream ("people.bin");

        // ObjectOutputStream - для записи ТОЛЬКО Объектов, состоящих
        из байтов.
        ObjectOutputStream oos = new ObjectOutputStream (fos);

        // Записываем Объекты в файл.
        oos.writeObject (person1);
        oos.writeObject (person2);

        // !!! ЗАКРЫВАЕМ ПОТОК !!!
        oos.close ();

    } catch (IOException e) {
        e.printStackTrace ();
    }
}

```

```

import java.io.Serializable;

```

```

/*
Чтобы разрешить Java сериализацию Объектов Класа Person,
impleментировать Serializable в class Person (public class Person implements
Serializable)
*/
public class Person implements Serializable {
    private int id;
    private String name;

    public Person(int id, String name){
        this.id = id;
        this.name = name;
    }

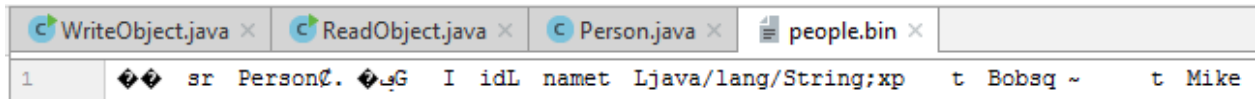
    public int getId() {
        return id;
    }

    public String getName(){
        return name;
    }

    public String toString(){
        return id + " : " + name;
    }
}

```

Вывод в Файл:



ЧТЕНИЕ ИЗ ФАЙЛА.

```
import java.io.*;

public class ReadObject {
    public static void main(String[] args) {
        try {
            // FileInputStream - для чтения ЛЮБЫХ бинарных данных.
            FileInputStream fis = new FileInputStream ("people.bin");
            // ObjectInputStream - для чтения ТОЛЬКО Объектов, состоящих из
            // байтов.
            ObjectInputStream ois = new ObjectInputStream (fis);

            // Считываем Объекты из файла.
            // Делаем Даун кастинг (Person).
            Person person1 = (Person) ois.readObject ();
            Person person2 = (Person) ois.readObject ();

            // Выводим Объекты Класса Person в консоль
            // с помощью переопределенного метода toString в Классе Person.
            System.out.println (person1);
            System.out.println (person2);

            // !!! ЗАКРЫВАЕМ ПОТОК !!!
            ois.close ();

        } catch (IOException e) {
            e.printStackTrace ();
        } catch (ClassNotFoundException e) {
            // Это исключение на случай,
            // если при чтении другим пользователем ,
            // у него не окажется Класса Person в проекте.
            e.printStackTrace ();
        }
    }
}

import java.io.Serializable;

/*
Чтобы разрешить Java сериализацию Объектов Класса Person,
impleментировать Serializable в class Person (public class Person implements
Serializable)
*/
public class Person implements Serializable {
    private int id;
    private String name;

    public Person(int id, String name){
        this.id = id;
        this.name = name;
    }
}
```

```

    public int getId() {
        return id;
    }
    public String getName(){
        return name;
    }

    public String toString(){
        return id + " : " + name;
    }
}

```

ВЫВОД В КОНСОЛЬ:

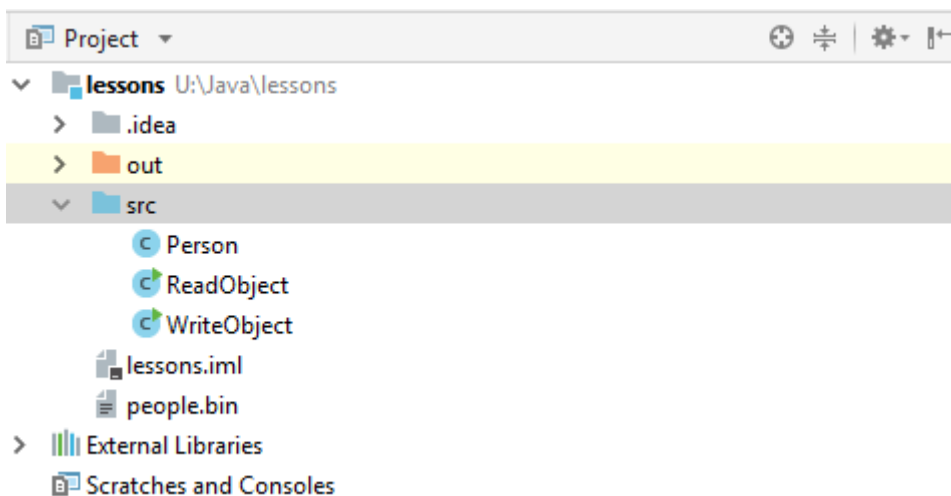
```

1 : Bob
2 : Mike

```

Урок 46: Сериализация (часть 2). Сериализация массивов.

https://www.youtube.com/watch?v=7OZ6oZ5l0U&index=47&list=PLAma_mKfftOSUkXp26rgdnC0PicnmnDak



ПЕРВЫЙ способ.

ЗАПИСЬ В ФАЙЛ.

```

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class WriteObject {
    public static void main(String[] args) {
        // Создаем массив.
    }
}

```



```

    Person[] people = new Person[3];
    people[0] = new Person (1, "Bob");
    people[1] = new Person (2, "Mike");
    people[2] = new Person (3, "Tom");

    try {
        // FileOutputStream - для записи ЛЮБЫХ бинарных данных.
        FileOutputStream fos = new FileOutputStream ("people.bin");

        // ObjectOutputStream - для записи ТОЛЬКО Объектов, состоящих
        // из байтов.
        ObjectOutputStream oos = new ObjectOutputStream (fos);

        // Записываем Объекты в файл.
        // 1 Способ.
        oos.writeInt(people.length);

        for (Person p : people) {
            oos.writeObject(p);
        }

        // !!! ЗАКРЫВАЕМ ПОТОК !!!
        oos.close ();

    } catch (IOException e) {
        e.printStackTrace ();
    }
}

import java.io.Serializable;

/*
Чтобы разрешить Java сериализацию Объектов Класа Person,
impleментировать Serializable в class Person (public class Person implements
Serializable)
*/
public class Person implements Serializable {
    private int id;
    private String name;

    public Person(int id, String name){
        this.id = id;
        this.name = name;
    }

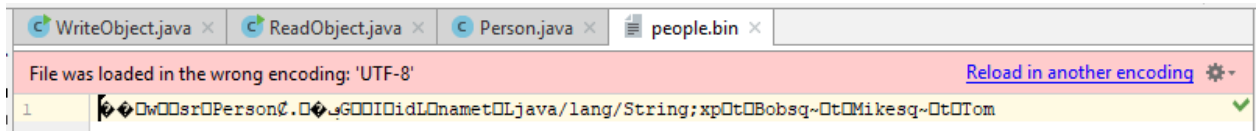
    public int getId() {
        return id;
    }

    public String getName(){
        return name;
    }

    public String toString(){
        return id + " : " + name;
    }
}

```

Вывод в Файл:



ЧТЕНИЕ ИЗ ФАЙЛА.

```
import java.io.*;
import java.util.Arrays;

public class ReadObject {
    public static void main(String[] args) {
        try {
            // FileInputStream - для чтения ЛЮБЫХ бинарных данных.
            FileInputStream fis = new FileInputStream ("people.bin");
            // ObjectInputStream - для чтения ТОЛЬКО Объектов, состоящих из
            // байтов.
            ObjectInputStream ois = new ObjectInputStream (fis);

            // Считываем Объекты из файла.
            int personCount = ois.readInt();
            Person[] people = new Person[personCount];

            for (int i = 0; i < personCount; i++) {
                people[i] = (Person) ois.readObject();
            }
            // Выводим Объекты Класса Person в консоль
            // с помощью метода Arrays.toString.
            System.out.println(Arrays.toString(people));

            // !!! ЗАКРЫВАЕМ ПОТОК !!!
            ois.close ();

        } catch (IOException e) {
            e.printStackTrace ();
        } catch (ClassNotFoundException e) {
            // Это исключение на случай,
            // если при чтении другим пользователем ,
            // у него не окажется Класса Person в проекте.
            e.printStackTrace ();
        }
    }
}

import java.io.Serializable;

/*
Чтобы разрешить Java сериализацию Объектов Класса Person,
impleментировать Serializable в class Person (public class Person implements
Serializable)
*/
public class Person implements Serializable {
    private int id;
```

```

private String name;

public Person(int id, String name){
    this.id = id;
    this.name = name;
}

public int getId() {
    return id;
}

public String getName(){
    return name;
}

public String toString(){
    return id + " : " + name;
}
}

```

Вывод в консоль:

```
[1 : Bob, 2 : Mike, 3 : Tom]
```

ВТОРОЙ способ.

ЗАПИСЬ В ФАЙЛ.

```

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class WriteObject {
    public static void main(String[] args) {
        // Создаем массив.
        Person[] people = new Person[3];
        people[0] = new Person (1, "Bob");
        people[1] = new Person (2, "Mike");
        people[2] = new Person (3, "Tom");

        try {
            // FileOutputStream - для записи ЛЮБЫХ бинарных данных.
            FileOutputStream fos = new FileOutputStream ("people.bin");

            // ObjectOutputStream - для записи ТОЛЬКО Объектов, состоящих
            // из байтов.
            ObjectOutputStream oos = new ObjectOutputStream (fos);

            // Записываем Объекты в файл.
            // 2 Способ.
            oos.writeObject(people);
            // !!! ЗАКРЫВАЕМ ПОТОК !!!
            oos.close ();

        } catch (IOException e) {
            e.printStackTrace ();
        }
    }
}

```

```

import java.io.Serializable;

/*
Чтобы разрешить Java сериализацию Объектов Класса Person,
impleментировать Serializable в class Person (public class Person implements
Serializable)
*/
public class Person implements Serializable {
    private int id;
    private String name;

    public Person(int id, String name){
        this.id = id;
        this.name = name;
    }

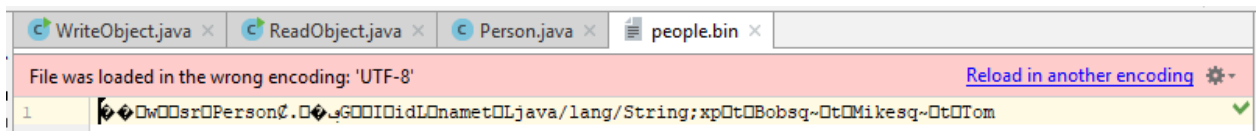
    public int getId() {
        return id;
    }

    public String getName(){
        return name;
    }

    public String toString(){
        return id + " : " + name;
    }
}

```

Вывод в Файл:



ЧТЕНИЕ ИЗ ФАЙЛА.

```

import java.io.*;
import java.util.Arrays;

public class ReadObject {
    public static void main(String[] args) {
        try {
            // FileInputStream - для чтения ЛЮБЫХ бинарных данных.
            FileInputStream fis = new FileInputStream ("people.bin");
            // ObjectInputStream - для чтения ТОЛЬКО Объектов, состоящих из
            // байтов.
            ObjectInputStream ois = new ObjectInputStream (fis);

            // Считываем Объекты из файла.
            Person[] people = (Person[]) ois.readObject();
            // Выводим Объекты Класса Person в консоль
            // с помощью метода Arrays.toString.
            System.out.println(Arrays.toString(people));
        }
    }
}

```

```

        // !!! ЗАКРЫВАЕМ ПОТОК !!!
        ois.close ();

    } catch (IOException e) {
        e.printStackTrace ();
    } catch (ClassNotFoundException e) {
        // Это исключение на случай,
        // если при чтении другим пользователем ,
        // у него не окажется Класса Person в проекте.
        e.printStackTrace ();
    }
}

}

import java.io.Serializable;

/*
Чтобы разрешить Java сериализацию Объектов Класса Person,
impleментировать Serializable в class Person (public class Person implements
Serializable)
*/
public class Person implements Serializable {
    private int id;
    private String name;

    public Person(int id, String name){
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName(){
        return name;
    }

    public String toString(){
        return id + " : " + name;
    }
}

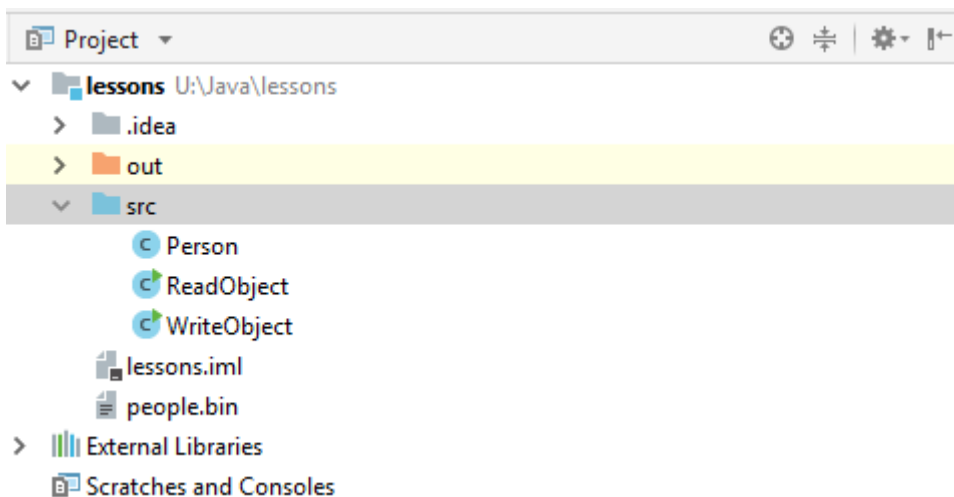
```

Вывод в консоль:

```
[1 : Bob, 2 : Mike, 3 : Tom]
```

Урок 47: Сериализация (часть 3). Transient, serialVersionUID.

https://www.youtube.com/watch?v=nr4_JRKCGBU&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak&index=48



Transient используется тогда, когда мы не хотим сериализовывать какое-либо поле.

```
import java.io.Serializable;

public class Person implements Serializable {
    private int id;
    /*
     * Transient используется тогда,
     * когда мы не хотим сериализовывать какое-либо поле.
     */
    private transient String name;

    public Person(int id, String name){
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName(){
        return name;
    }

    public String toString(){
        return id + " : " + name;
    }
}
```

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class WriteObject {
    public static void main(String[] args) {
        Person person1 = new Person (1, "Mike");
        try {
            // FileOutputStream - для записи ЛЮБЫХ бинарных данных.
```

```

        FileOutputStream fos = new FileOutputStream ("people.bin");

        // ObjectOutputStream - для записи ТОЛЬКО Объектов, состоящих
        // из байтов.
        ObjectOutputStream oos = new ObjectOutputStream (fos);

        // Записываем Объект в файл.
        oos.writeObject(person1);
        // !!! ЗАКРЫВАЕМ ПОТОК !!!
        oos.close ();

    } catch (IOException e) {
        e.printStackTrace ();
    }
}

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class WriteObject {
    public static void main(String[] args) {
        Person person1 = new Person (1, "Mike");
        try {
            // FileOutputStream - для записи ЛЮБЫХ бинарных данных.
            FileOutputStream fos = new FileOutputStream ("people.bin");

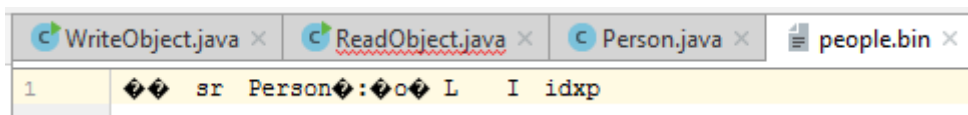
            // ObjectOutputStream - для записи ТОЛЬКО Объектов, состоящих
            // из байтов.
            ObjectOutputStream oos = new ObjectOutputStream (fos);

            // Записываем Объект в файл.
            oos.writeObject(person1);
            // !!! ЗАКРЫВАЕМ ПОТОК !!!
            oos.close ();

        } catch (IOException e) {
            e.printStackTrace ();
        }
    }
}

```

Вывод в Файл:

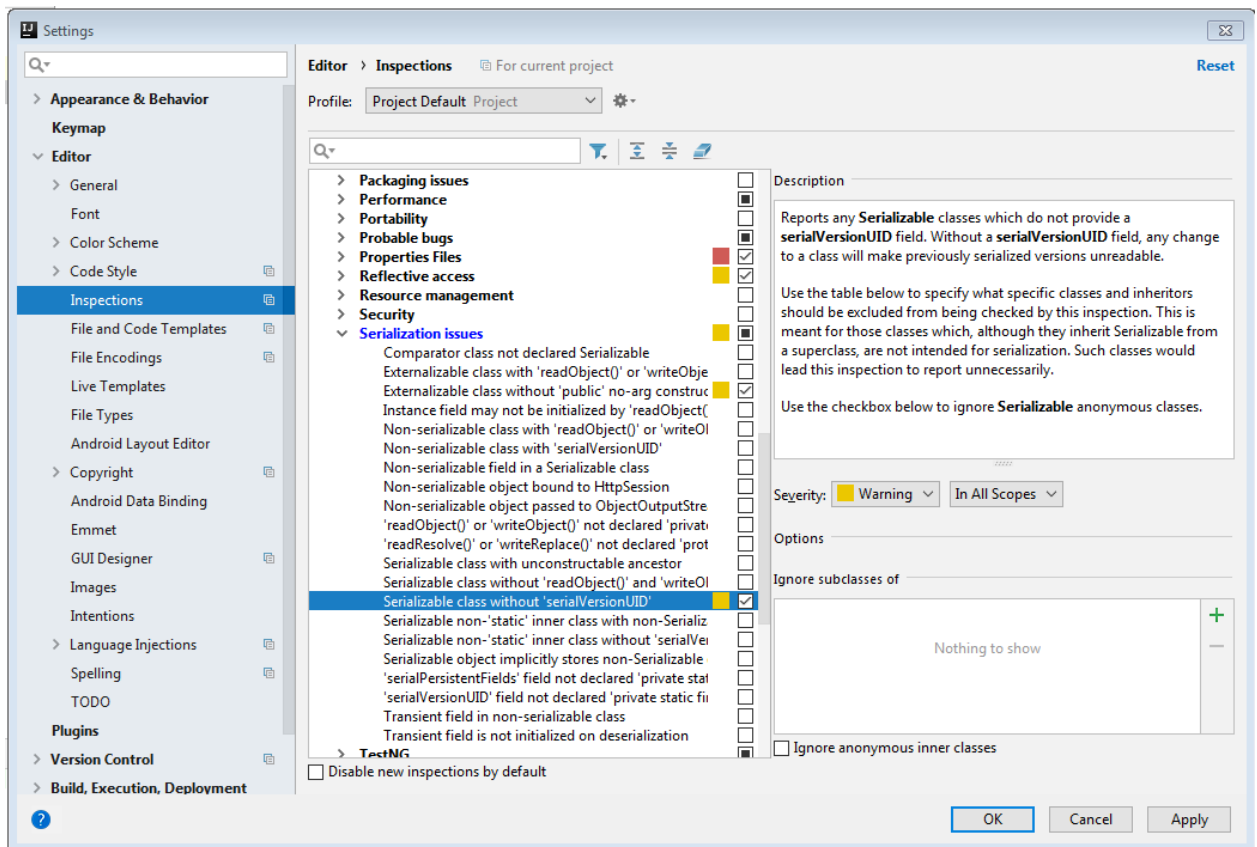


Вывод в консоль:

1 : null

Включаем предупреждение serialVersionUID в "JI".

File -> Settings -> Editor -> Inspection -> Java -> Serialization issues -> Serializable class without 'serialVersionUID' -> [V]



```
import java.io.Serializable;

public class Person implements Serializable {
    /*
     * serialVersionUID - это маркер, который говорит нам о том,
     * является ли записанный в файл Объект тем же Объектом в программе
     * (Был ли он изменен)?
     */
    private static final long serialVersionUID = -8626318026843608878L;
    private int id;
    private String name;

    public Person(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return id + " : " + name;
    }
}
```


АВТОМАТИЧЕСКОЕ ЗАКРЫТИЕ ПОТОКА.

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class WriteObject {
    public static void main(String[] args) {
        Person person1 = new Person (1, "Mike");
        try {
            FileOutputStream fos = new FileOutputStream ("people.bin");
            ObjectOutputStream oos = new ObjectOutputStream (fos);
            oos.writeObject(person1);

            // !!! ЗАКРЫВАЕМ ПОТОК "ВРУЧНУЮ"!!!
            oos.close ();

        } catch (IOException e) {
            e.printStackTrace ();
        }
    }
}
```

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class WriteObject {
    public static void main(String[] args) {
        Person person1 = new Person (1, "Mike");

        // !!! ЗАКРЫВАЕМ ПОТОК "АВТОМАТИЧЕСКИ"!!!
        try(ObjectOutputStream oos = new ObjectOutputStream (new
FileOutputStream ("people.bin"))){

            oos.writeObject(person1);

        } catch (IOException e) {
            e.printStackTrace ();
        }
    }
}
```

```
import java.io.*;

public class ReadObject {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream ("people.bin");
            ObjectInputStream ois = new ObjectInputStream (fis);
            Person person1 = (Person) ois.readObject ();
            System.out.println(person1);
        }
    }
}
```

```

        // !!! ЗАКРЫВАЕМ ПОТОК "ВРУЧНУЮ"!!!
        ois.close ();

    } catch (IOException e) {
        e.printStackTrace ();
    } catch (ClassNotFoundException e) {
        e.printStackTrace ();
    }
}

import java.io.*;

public class ReadObject {
    public static void main(String[] args) {

        // !!! ЗАКРЫВАЕМ ПОТОК "АВТОМАТИЧЕСКИ"!!!
        try (ObjectInputStream ois = new ObjectInputStream (new FileInputStream
("people.bin"))){

            Person person1 = (Person) ois.readObject ();
            System.out.println(person1);
        } catch (IOException e) {
            e.printStackTrace ();
        } catch (ClassNotFoundException e) {
            e.printStackTrace ();
        }

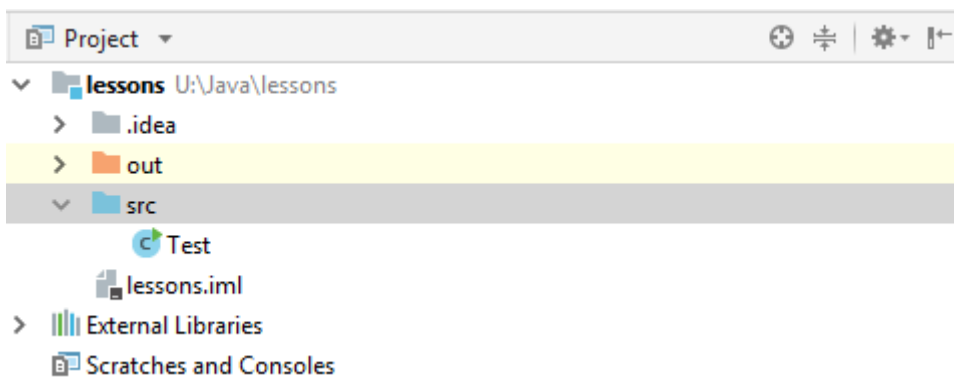
    }
}

```

Урок 48: Enum (Перечисления).

https://www.youtube.com/watch?v=GOzNp1YAm5w&index=49&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak

Как использовались Перечисления на появления Enum.



```

public class Test {
    // Создаем Список ЖИВОТНЫХ.
    private static final int DOG = 0;
    private static final int CAT = 1;
}

```

```

private static final int FROG = 2;

public static void main(String[] args) {
    int animal = DOG;

    switch (animal){
        case DOG:
            System.out.println ("It's a dog!");
            break;
        case CAT:
            System.out.println ("It's a cat!");
            break;
        case FROG:
            System.out.println ("It's a frog!");
            break;
        default:
            System.out.println ("It's an animal!");
    }
}
}

```

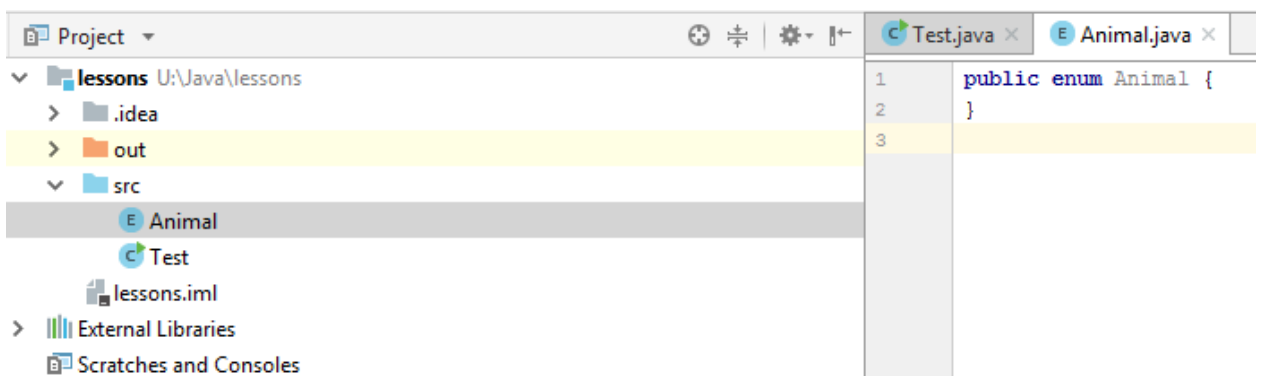
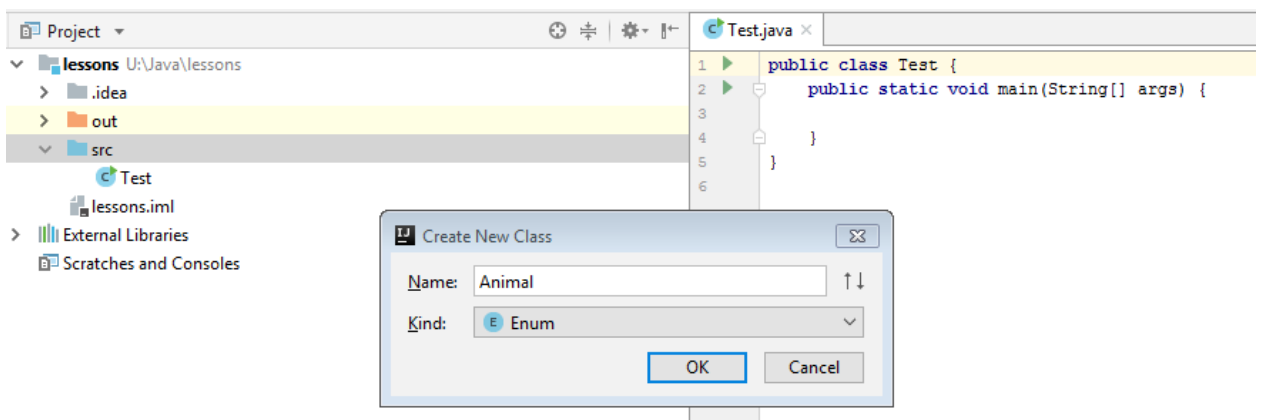
Вывод в консоль:

It's a dog!

Как используется появления с Enum.

Создаем Enum.

Обычно дается имя в Единственном числе.



```

public enum Animal {
    DOG, CAT, FROG
}

public class Test {
    public static void main(String[] args) {
        Animal animal = Animal.CAT;

        switch (animal){
            case DOG:
                System.out.println ("It's a dog!");
                break;
            case CAT:
                System.out.println ("It's a cat!");
                break;
            case FROG:
                System.out.println ("It's a frog!");
                break;
            default:
                System.out.println ("It's an animal!");
        }
    }
}

```

Вывод в консоль:

It's a cat!

Делаем переводчик.

```

public enum Animal {
    DOG("Собака"), CAT("Кошка"), FROG("Лягушка"); // <- Ставим ";" !!!
    /*
    Создаем конструктор для перевода
    названий животных на русский язык.
    !!! В Enum конструктор "private" по умолчанию!!!
    "private" можно не указывать.
    */
    private String translation;
    Animal(String translation){
        this.translation = translation;
    }

    public String getTranslation(){
        return translation;
    }

    public String toString(){
        return "Перевод на русский язык: " + translation;
    }
}

public class Test {
    public static void main(String[] args) {

```

```

        Animal animalD = Animal.DOG;
        System.out.println (animalD);

        Animal animalC = Animal.CAT;
        System.out.println (animalC);
    }
}

```

Вывод в консоль:

Перевод на русский язык: Собака
 Перевод на русский язык: Кошка

Полезные методы.

```

public class Test {
    public static void main(String[] args) {
        Animal animalD = Animal.DOG;
        System.out.println (animalD.name ());
    }
}

```

Вывод в консоль:

DOG

```

public class Test {
    public static void main(String[] args) {
        Animal frog = Animal.valueOf ("FROG");
        System.out.println (frog);
    }
}

```

Вывод в консоль:

Перевод на русский язык: Лягушка

```

public class Test {
    public static void main(String[] args) {
        /*
        Элементы Enum имеют свой индекс
        от "0" и далее.
        DOG("Собака"), CAT("Кошка"), FROG("Лягушка")
        */
        Animal dog = Animal.DOG;
        System.out.println (dog.ordinal ());
    }
}

```

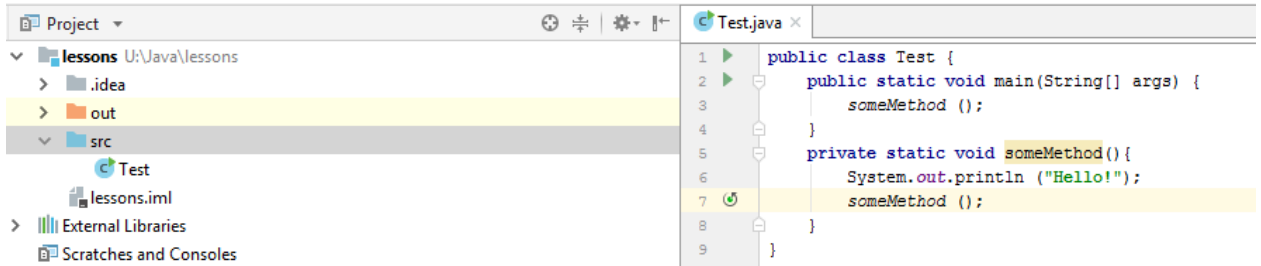
Вывод в консоль:

0

Урок 49: Рекурсия.

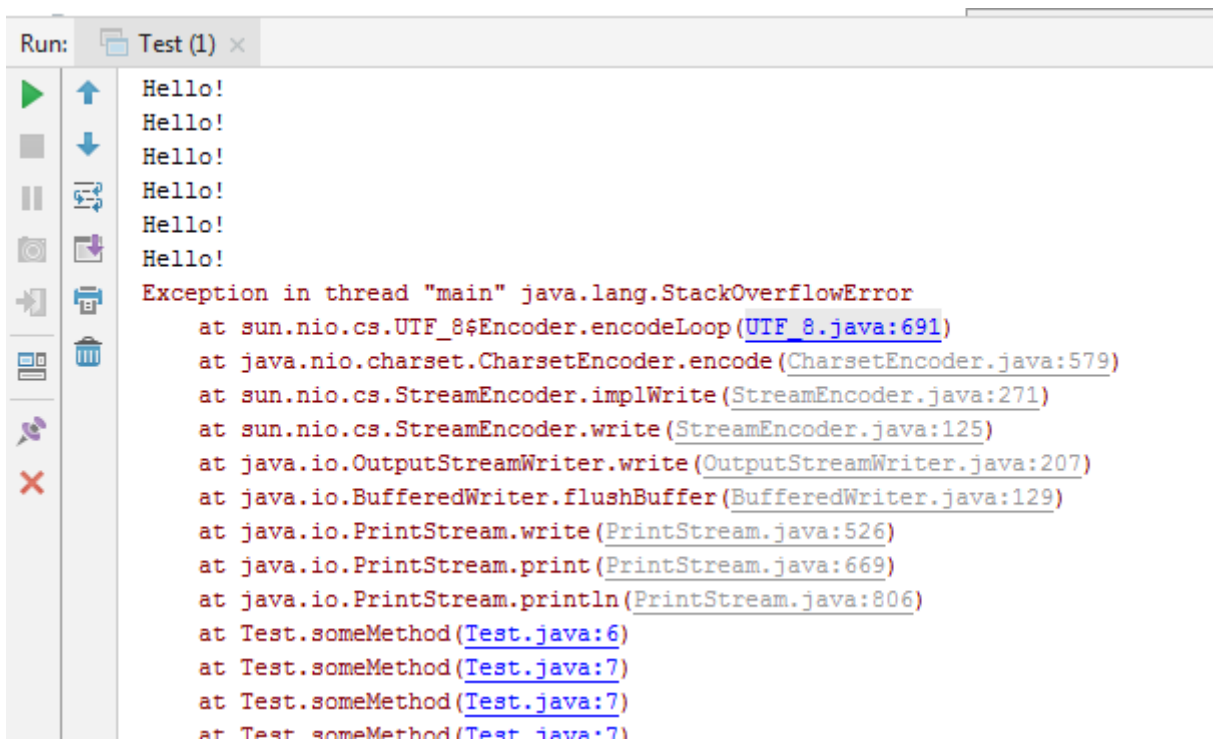
https://www.youtube.com/watch?v=9Hs7DuIJ3LE&list=PLAma_mKffTOSUkXp26rgdnC0PicnmDak&index=50

Рекурсия - это вызов метода в теле этого-же метода.



```
public class Test {  
    public static void main(String[] args) {  
        someMethod ();  
    }  
    private static void someMethod(){  
        System.out.println ("Hello!");  
        someMethod ();  
    }  
}
```

Вывод в консоль:



Предотвращаем попытку Рекурсии выйти за пределы стека памяти.

```
public class Test {
    public static void main(String[] args) {
        someMethod ();
    }

    static int i = 0;
    private static void someMethod() {
        System.out.println ("Hello!");
        i++;
        if (i == 3) {
            return;
        }
        someMethod ();
    }
}
```

Вывод в консоль:

```
Hello!
Hello!
Hello!
```

Поиск факториала **с** Рекурсией.

```
public class Test {
    public static void main(String[] args) {
        System.out.println (fac (4));
    }

    private static int fac(int n){
        if (n == 1) {
            return 1;
        }
        return n * fac (n -1);
    }
}
```

Вывод в консоль:

```
24
```

Поиск факториала **без** Рекурсии, с циклом for.

```
public class Test {
    public static void main(String[] args) {
        System.out.println (fac (4));
    }
    public static int fac(int x){
        int y = 1;
        for (int i = 1; i <= x ; i++) {
```

```
        y *= i;
    }
    return y;
}
```

Вывод в консоль:

24