

2017

# JAVA STANDARD EDITION

Volume 4

ЖУАНЫШЕВ ИЛЪЯС  
BITLAB TEAM

## Оглавление

<b>Что такое язык программирования JAVA?</b> .....	<b>4</b>
<b>Введение, первая программа. Переменные, примитивные типы данных.</b> .....	<b>4</b>
Установка и запуск первой программы.....	4
<b>Переменные, примитивные типы данных.</b> .....	<b>7</b>
Целочисленные типы .....	7
Типы с плавающей точкой .....	8
Логический тип .....	8
Ссылочные типы .....	8
Строки.....	9
<b>Типы переменных, локальные переменные, статические переменные</b> .....	<b>10</b>
Локальные переменные .....	10
Статические переменные .....	10
<b>Базовые операторы, арифметические, логические.</b> .....	<b>12</b>
Логические операторы .....	12
Операторы сравнения.....	12
<b>Класс Scanner, ввод и вывод данных.</b> .....	<b>13</b>
<b>Условия if, if-else, switch-case.</b> .....	<b>17</b>
if - else .....	17
else-if.....	18
switch .....	19
<b>Циклы: while, do-while, for.</b> .....	<b>24</b>
while.....	24
do-while .....	24
for.....	25
<b>Массивы, одномерные массивы</b> .....	<b>34</b>
<b>Строки, класс String</b> .....	<b>48</b>
Методы класса String .....	48
<b>Методы и аргументы.</b> .....	<b>53</b>
<b>Классы и объекты</b> .....	<b>60</b>
Конструктор .....	60
Методы в классах. ....	62

<b>Инкапсуляция, геттеры и сеттеры .....</b>	<b>69</b>
<b>Наследование, полиморфизм .....</b>	<b>75</b>
<b>Абстрактные классы .....</b>	<b>83</b>
<b>Интерфейсы .....</b>	<b>89</b>
<b>Графика (GUI) .....</b>	<b>92</b>
Создание нашего первого графического окна .....	92
Добавление кнопок, текстового поля и метки (надписи).....	93
<b>Обработка исключений, try – catch .....</b>	<b>97</b>
<b>Коллекции и класс ArrayList .....</b>	<b>105</b>
<b>Чтение и запись текстовых файлов. ....</b>	<b>109</b>
<b>Сериализация, десериализация .....</b>	<b>114</b>
<b>Потоки, многопоточность, интерфейс Runnable.....</b>	<b>119</b>
<b>Сети, использование класса Socket .....</b>	<b>126</b>
Использование потоков (Thread) для мульти-клиентного сервера.....	127

## Что такое язык программирования JAVA?

### История:

**Java** представляет собой язык программирования и платформу вычислений, которая была впервые выпущена Sun Microsystems в 1995 г. Существует множество приложений и веб-сайтов, которые не работают при отсутствии установленной **Java**, и с каждым днем число таких веб-сайтов и приложений увеличивается.

**Java** отличается быстротой, высоким уровнем защиты и надежностью. От портативных компьютеров до центров данных, от игровых консолей до суперкомпьютеров, используемых для научных разработок, от сотовых телефонов до сети Интернет — **Java** повсюду.

## Введение, первая программа. Переменные, примитивные типы данных.

JAVA собою предоставляет набор команд, которая выполняется последовательно.

Приведем простой пример:

### *Main.java*

```
public class Main{  
  
    public static void main(String[] args) {  
  
        int a = 10;  
        int b = 20;  
        System.out.println(a + b);  
  
    }  
}
```

В этом примере мы создаем две переменные с типами данных `int` (целое число), которые равняются 10 и 20 соответственно, и в итоге выводиться их сумма.

### Установка и запуск первой программы.

Для начала нам нужно будет установить библиотеку Java, которую можно скачать с официального сайта. Так как библиотека и сам язык находятся в открытом доступе, то скачать ее можно бесплатно по ссылке:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Соответственно, вам нужно будет выбрать нужный файл относительно вашей операционной системы.

Далее, после скачки, вам нужно будет создать первую программу. Создайте файл **Main.java** так как имя нашего класса тоже называется **Main**. (Это очень важно, чтобы имя класса и имя файла были одинаковыми.)

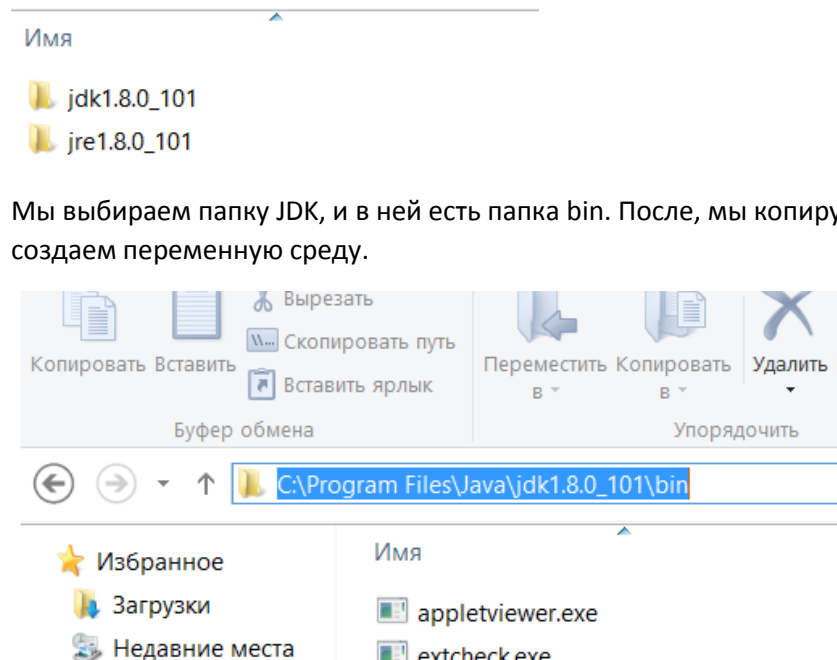
После создания файла, вы должны набрать в ней строки кода, что мы рассматривали выше.

Сохраните файл, и его последнее состояние.

Теперь нам нужно будет создать переменную среду в операционной системе. Скорее всего наша библиотека будет установлена в папке:

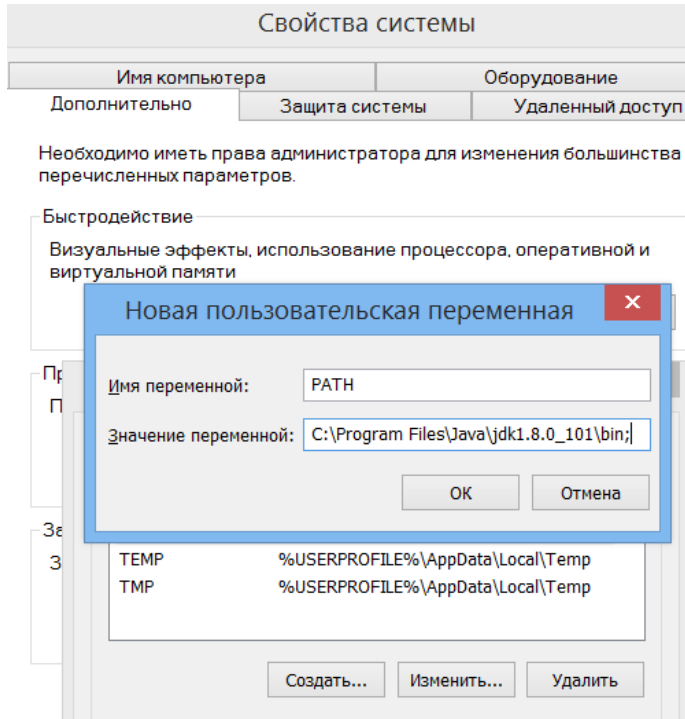
### C:\Program Files\Java

В этой папке у нас установлено две папки, JDK и JRE. JRE (Java Runtime Environment) нам нужен для запуска, а вот основная библиотека установлена в папке JDK (Java Development Kit)



Создаем переменную среду таким образом.

- 1) Заходим в Мой Компьютер, и нажатием правой кнопки мышки выбираем свойства компьютера.
- 2) Далее выбираем во вкладке "Дополнительно", нажмите на "Переменные среды"
- 3) Создайте новую переменную среду под именем "PATH" и в ее значение вставьте полный путь к библиотеке JDK.
- 4) Сохраните и создайте.



Далее, мы возвращаемся в корневую папку, где находится наш **Main.java**

Открываем командную строку, в котором мы заходим в нашу папку, где находится наш **Main.java** файл. (Чтобы войти в директорию, наберите **cd 'наименование папки'**, чтобы выйти из директории, наберите **cd..**)

Далее, компилируем наш файл с помощью команды **javac Main.java**, и если нету ошибок, то запускаем с помощью команды **java Main.java**

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\User>cd Desktop
C:\Users\User\Desktop>cd "JAVA SE"
C:\Users\User\Desktop\JAVA SE>javac Main.java
C:\Users\User\Desktop\JAVA SE>java Main
30
C:\Users\User\Desktop\JAVA SE>_

```

## Переменные, примитивные типы данных.

В Java есть 8 примитивных типов, которые делят на 3 группы, вот они:

Целые числа - byte, short, char, int, long

Числа с плавающей точкой (иначе дробные) - float, double

Логический – boolean

### Целочисленные типы

Целочисленные типы различаются между собой только диапазонами возможных значений.

Тип	Размер (бит)	Диапазон
<b>byte</b>	8 бит	от -128 до 127
<b>short</b>	16 бит	от -32768 до 32767
<b>char</b>	16 бит	без знаковое целое число, представляющее собой символ UTF-16
<b>int</b>	32 бит	от -2147483648 до 2147483647
<b>long</b>	64 бит	от -9223372036854775808 до 9223372036854775807

Пример использования целочисленных типов:

```
public class Main {

    public static void main(String[] args) {

        byte b = 234; // Ошибка, т.к. диапазон от -128 до 127!
        short s = 3323;
        int i = 62536;
        long l = 1347483648L;
        System.out.println(i);
        System.out.println(b);
        System.out.println(s);
        System.out.println(l);

    }

}
```

Символы тоже относят к целочисленным типам из-за особенностей представления в памяти и традиций.

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

        char a = 'a', b, c = 'c';
        b = (char) ((a + c) / 2);
        System.out.println(b);

    }
}
```

### Типы с плавающей точкой

Тип	Размер (бит)	Диапазон
float	32	от -1.4e-45f до 3.4e+38f
double	64	от -4.9e-324 до 1.7e+308

```
public class Main {

    public static void main(String[] args) {

        double a, b = 4.12;
        a = 22.1 + b;
        float pi = 3.14f;
        float anotherPi = (float) 3.14
        double c = 27;
        double d = pi * c;
        System.out.println(d);

    }

}
```

### Логический тип

Тип	Размер (бит)	Значение
boolean	8 (в массивах), 32 (не в массивах используется int)	true (истина) или false (ложь)

### Ссылочные типы

Ссылочные типы - это все остальные типы - классы, перечисления и интерфейсы, например, объявленные в стандартной библиотеке **Java**, а также массивы.



## Строки

Строки - это объекты класса **String**, они очень распространены, поэтому в некоторых случаях обрабатываются по-другому в отличии от всех остальных объектов. Строковые литералы записываются в двойных кавычках.

```
public class Main {  
  
    public static void main(String[] args) {  
  
        String a = "Hello", b = "World";  
        System.out.println(a + " " + b);  
  
    }  
  
}
```

## Типы переменных, локальные переменные, статические переменные.

Переменная предоставляется нам именем хранения, чтобы нашей программой можно было манипулировать. Каждая переменная имеет конкретный тип, который определяет размер и размещение её в памяти; диапазон значений, которые могут храниться в памяти; и набор операций, которые могут быть применены к переменной.

### Локальные переменные

Локальные переменные объявляются в методах, конструкторах или блоках.

Локальные переменные создаются, когда метод, конструктор или блок запускается и уничтожаются после того, как завершиться метод, конструктор или блок.

В **Java** не существует для локальных переменных значения по умолчанию, так что они должны быть объявлены и начальное значение должны быть присвоено перед первым использованием.

### Статические переменные

Статические переменные - которые объявляются со статическим ключевым словом **static** в классе, но за пределами метода, конструктора или блока.

Модификатор **static** в **Java** напрямую связан с классом, если поле статично, значит оно принадлежит классу, если метод статичный, аналогично — он принадлежит классу. Исходя из этого, можно обращаться к статическому методу или полю используя имя класса. Например, если поле **name** статично в классе **User**, значит, вы можете обратиться к переменной запросом вида: **User.name**.

Там будет только одна копия каждой статической переменной в классе, независимо от того, сколько объектов создано из него.

```
public class Main {  
  
    static String country = "Казахстан";  
  
    public static void main(String[] args) {  
  
        System.out.println("Моя страна : " + country);  
  
    }  
  
}
```

В **Java** статические переменные создаются при запуске программы и уничтожаются, когда выполнение программы остановится.

Значения по умолчанию такое же, как и у переменных экземпляра. Для чисел по умолчанию равно **0**, для данных типа **Boolean** - **false**; и для ссылок на объект - **null**. Значения могут быть присвоены при объявлении или в конструкторе. Кроме того, они могут быть присвоены в специальных блоках статического инициализатора.

## Базовые операторы, арифметические, логические.

### Логические операторы

Существует несколько бинарных логических операторов и один унарный. В качестве аргументов для всех этих операторов выступают логические литералы (константы), логические переменные и выражения, имеющие логическое значение. Логические операторы работают только с операндами типа **boolean**.

**!** — логическое **НЕ**, унарный оператор, меняет значение на противоположное.

**&&** — логическое **И**, бинарная операция, возвращает истинное значение тогда и только тогда, когда оба операнда истинны.

**||** — логическое **ИЛИ**, бинарная операция, возвращает истинное значение, когда хотя бы один из операндов истинный.

Приоритет логических операторов: логическое НЕ, логическое И, логическое ИЛИ.

Оператор	Описание
&	Логическое AND (И)
&&	Сокращённое AND
	Логическое OR (ИЛИ)
	Сокращённое OR
^	Логическое XOR (исключающее OR (ИЛИ))
!	Логическое унарное NOT (НЕ)
==	Равно
!=	Не равно

```
public class Main {

    public static void main(String[] args) {

        boolean a = true || false; // true
        boolean b = true && true;    // true
        boolean c = false && true;   // false
        boolean d = a && (c || b);   // true

    }

}
```

## Операторы сравнения

Большинство операторов сравнения применимы к числовым значениям. Всё это бинарные операторы, имеющие два числовых аргумента, но возвращающие логическое значение.

> - оператор «больше».

>= - оператор «больше или равно».

< - оператор «меньше».

<= - оператор «меньше или равно».

!= - оператор «не равно».

== - оператор эквивалентности (равенства)

```
public class Main {  
    public static void main(String[] args) {  
        boolean a = 5>7;    // false  
        boolean b = 17>=9;  // true  
        boolean c = 5!=6;   // true  
        boolean d = 4==4;   // true  
    }  
}
```

## Класс **Scanner**, ввод и вывод данных.

Класс **Scanner** в **Java** позволяет **читать** данные полученные, например, из консоли или файла. Класс находится в пакете **java.util**. Для взаимодействия с консолью нам необходим класс **System**.

Для работы с потоком ввода нам необходимо создать объект класса **Scanner**, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в **Java** представлен объектом — **System.in**. А стандартный поток вывода (дисплей) — уже знакомым вам объектом **System.out**.

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        double b = in.nextDouble();
        String name = in.next();
        System.out.println(a*a);
        System.out.println(b*b);
        System.out.println("My name is " + name);

    }

}
```

Метод **nextInt()** считывает целостные числа типа **int**.

Метод **nextDouble()** считывает дробные числа типа **double**.

Метод **next()** считывает строку типа **String**.

Метод **nextFloat()** считывает числа типа **float**.

Метод **nextLong()** считывает числа типа **long**.

## Упражнения для разминки

### Задание 1:

Напишите программу, где я складываю сумму двух целостных чисел **a** и **b**?

### Задание 2:

Напишите программу, где я складываю сумму двух целостных чисел **a** и **b**?

### Задание 3:

Напишите программу в котором я запрашиваю имя, фамилию, возраст, страну, и в конце программа выводит данные пользователя в консоль.

#### Ввод:

```
Ilyas
Zhuanyshev
28
Kazakhstan
```

#### Вывод:

```
Name: Ilyas
Surname: Zhuanyshev
Age: 28
Country: Kazakhstan
```

### Задание 4:

Напишите программу в котором я ввожу два целостных чисел, и если первое число больше второй, то программа выведет **true**, иначе **false**.

### Задание 5:

Напишите программу в котором я ввожу три целостных чисел **a**, **b** и **c**. Выведите сумму, умножение и разницу всех этих чисел.

#### Ввод:

```
10, 20, 30
```

#### Вывод:

```
a + b + c = 60
a - b - c = -50
a * b * c = 6000
```

**Задание 6:**

Я ввожу через консоль три целостных чисел **a**, **b** и **c**.  
Выполните данную операцию:  $a^3 + 2b^2 - 3ab + c^3$

**Практические задачи****Задание 1:**

Напишите программу, где я складываю сумму двух целостных чисел **a** = 100 и **b** = 200?

**Задание 2:**

Напишите программу, где я умножаю два **a** и **b** из консоли?

**Задание 3:**

Напишите программу в котором я запрашиваю университет, факультет, предмет, и в конце программа выводит данные пользователя в консоль.

**Ввод:**

KBTU

Computer

JAVA

**Вывод:**

University: KBTU

Faculty: Computer

Subject: JAVA

**Задание 4:**

Напишите программу в котором я ввожу три целостных чисел **a**, **b** и **c**, и если сумма **a** и **b** больше **c**, то программа выведет **true**, иначе **false**.

**Задание 5:**

Выполните данную математическую операцию где я ввожу **x**, **y** и **z** с консоли. Выполните данную операцию:  $x^4 + 4xy^2 - 4yz + z^4$



## Условия if, if-else, switch-case.

### if - else

В программировании мы можем создавать условия, для выполнения определенных задач, где мы хотим, чтобы программа действовала по определенным правилам. Условный оператор **if** часто применяется программистами и имеется во всех языках программирования. Оператор **if** является основным оператором выбора в Java и позволяет выборочно изменять ход выполнения программы.

Действия, написанные в операторе **else**, будут выполнены, только если значение **if** равно **false**.

Например, мы вводим какое-то число, оценку за семестр, и программа должна определить прошел ли студент курс или не прошел.

```
import java.util.Scanner;

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

        Scanner in = new Scanner(System.in);
        System.out.println("Введите оценку:");
        int mark = in.nextInt();

        if (mark >= 50) {

            System.out.println("Студент прошел курс");

        } else {

            System.out.println("Студент провалил курс");

        }

    }
}
```

В данном примере мы видим, как программа проверяет прошел ли студент курс или нет?

Чтобы реализовать такую программу, мы ставим определенное условие. Переменная **mark** является нашим баллом за курс, и если он больше или равно чем 50, то выполнится нижний блок кода, которая находится под **if(mark >= 50)**. Иначе выполниться блок кода, который находится под **else**.

## else-if

Далее, программу можно улучшить с помощью оператора **else if** добавляя условия.

Например, мы можем написать программу которая может определить точную оценку за семестр студенту относительно полученного балла за курс.

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);
        System.out.println("Введите оценку за курс:");
        int mark = in.nextInt();

        if(mark >= 90) {

            System.out.println("Студент получил А за курс");

        } else if (mark >= 80) {

            System.out.println("Студент получил В за курс");

        } else if (mark >= 70) {

            System.out.println("Студент получил С за курс");

        } else if (mark >= 60) {

            System.out.println("Студент получил D за курс");

        } else {

            System.out.println("Студент провалил курс");

        }

    }

}
```

Помните, программа выполняет одно из этих условий если вы используете оператор **else if**. То есть, например, студент получил 86 за курс, и оценка 86 подходит под второе, третье и четвертое условие. Но так как второе условие выполнится раньше, то больше под остальные условия оно выполняться не будет. Далее, все зависит от того каким образом вы разработаете программу.

## switch

Команду **switch** называют командой выбора. Выбор осуществляется в зависимости от целочисленного выражения. Например:

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);
        int day = in.nextInt();

        String dayValue = "";

        switch(day) {

            case 1: dayValue = "Понедельник";
                    break;
            case 2: dayValue = "Вторник";
                    break;
            case 3: dayValue = "Среда";
                    break;
            case 4: dayValue = "Четверг";
                    break;
            case 5: dayValue = "Пятница";
                    break;
            case 6: dayValue = "Суббота";
                    break;
            case 7: dayValue = "Воскресенье";
                    break;
            default: dayValue = "Нет такого дня";

        }

        System.out.println(dayValue);

    }

}
```

Данная программа смотрит значение переменной `day`, и соответственно выполняет строку кода после двоеточий. Если ни одно условие не подходит под переменную **day**, то выполняется строка кода после **default**.

## Упражнения для разминки

### Задание 1:

Напишите программу, где я складываю ввожу число **n**, и если оно между 10 и 20, то программа выведет **YES** иначе **NO**.

### Задание 2:

Напишите программу где я ввожу сумму внесенной валюты и выберу на какую валюту хочу перевести. (Курс USD – 320, EUR – 360, GBP - 420)

#### Ввод:

Insert amount in KZT:

**1000**

Choose currency:

[1] USD

[2] EUR

[3] GBP

**2**

#### Вывод:

**2.77 EUR**

### Задание 3:

Напишите программу, где я ввожу целостное число **a** и **b**, и если **a** делиться на **b**, то программа должна вывести **divisible** иначе **not divisible**.

\*\*\* Подсказка: Есть такой оператор **%**, **%** показывает остаток числа.

Например,  $10\%4$  будет равна 2, так как когда мы делим 10 на 4, остаток у нас будет 2

### Задание 4:

Напишите программу где я ввожу логин и пароль. И если они совпадают, то мы выводим Authentication completed, иначе Invalid login or password.

(Логин должен быть user, пароль - qwerty)

\*\*\* Подсказка: Сравнение строк вы делаете с помощью метода **equals()**.

Например, строка **name = "test"**

**name.equals("test")** возвращает **true**

**name.equals("other")** возвращает **false**

**Ввод:**

Введите логин:

**ilyas**

Введите пароль:

**qwerty**

**Вывод:**

Invalid login or password

**Ввод:**

Введите логин:

**user**

Введите пароль:

**qwerty**

**Вывод:**

Authentication completed

**\*\*\* Задание 5:**

Напишите программу, которая может определить вашу будущую профессию по вашим способностям. Программа будет задавать несколько вопросов и примерно по ним определит кем вам нужно стать в будущем.

Логика вопросов такова:

Сфера, которая нравится:

Точная наука, Гуманитарные предметы, Искусство, Спорт

1. Если точная наука, то какие точно предметы:

- a. Математика - Преподаватель по математике, или финансист
- b. Физика – IT программист или инженер

2. Если гуманитарные предметы, то какие точно предметы:

- a. История – дипломат или историк
- b. Иностранные языки - переводчик или журналист

3. Если искусство, то какие точно предметы:

- a. Рисование - художник или архитектор
- b. Пение - певец или музыкант

4. Если спорт, то какие точно предметы:

- a. Командный - футбол или баскетбол
- b. Индивидуальный - Бокс или теннис

**Ввод:**

Введите сферу которая вам нравится:

- [1] Точная наука
- [2] Гуманитарные предметы
- [3] Искусство
- [4] Спорт

**3**

Какой предмет нравится?

- [1] Рисование
- [2] Пение

**2**

**Вывод:**

Скорее вам надо стать певцом или музыкантом

## Практические задачи

**Задание 1:**

Напишите программу, где вы вводите две целостных чисел, и программа должна вывести что они равны, больше, или меньше.

**Ввод:**

7 10

**Вывод:**

7 less than 10

**Ввод:**

45 45

**Вывод:**

They are equal

**Ввод:**

55 6

**Вывод:**

55 is greater than 6

**Задание 2:**

Напишите программу, где вы вводите две целостных чисел, и программа должна вывести что они равны, больше, или меньше.

радиус

**Ввод:**

7 10

**Вывод:**

7 less than 10

**Ввод:**

45 45

**Вывод:**

They are equal

**Ввод:**

55 6

**Вывод:**

55 is greater than 6

## Циклы: while, do-while, for.

### while

Оператор цикла **while** есть практически во всех языках программирования. Он повторяет оператор или блок операторов до тех пор, пока значение его управляющего выражения истинно.

Форма цикла **while**:

```
while (условие) {
    блок кода
}
```

Если условие является верным, то выполнится блок кода, и после выполнения обратно будет проверяться условие, до тех пор, пока условие не будет неверным.

Соответственно, если мы поставим в условие значение **true**, то условие будет выполняться бесконечно. Для этого мы можем использовать целые числа.

Например:

```
public class Main {
    public static void main(String[] args) {
        int i = 0;
        while (i < 10) {
            System.out.println("Hello BITLAB");
            i++;
        }
    }
}
```

В данном примере блок кода будет выполняться до тех пор пока условие **i < 10** не станет неверным. Для этого, мы каждый раз увеличиваем значение **i** на **1**, чтобы после нескольких выполнений блока кода **i++**, значение **i** стало равна **10**, что означает **10 < 10**, то есть неверно. Таким образом мы получим цикл, который будет выполняться 10 раз, то есть 10 раз выведет "Hello BITLAB".

### do-while

Конструкция цикла:

```
do {
    блок кода
} (условие) ;
```



Отличие цикла **do-while** от цикла **while** состоит в том, что цикл **do-while** выполняется по крайней мере один раз, даже если условие изначально ложно. В цикле **while** такое не произойдёт. Цикл **do-while** используется реже, чем **while**.

Например:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int size = in.nextInt();
        int i = 0;
        do{
            System.out.println("Hello BITLAB");
            i++;
        } while(i<size);
    }
}
```

В данном примере, если введенное значение переменной **size** будет равняться 0, что означает  $0 < 0$ , то есть неверно, но все равно один раз блок кода будет выполняться. Таким образом, блок кода все равно как минимум один раз будет выполняться.

## for

Конструкция **for** управляет циклами. Команда выполняется до тех пор, пока управляющее логическое выражение не станет ложным.

Цикл **for** является наиболее распространённым циклом в программировании, поэтому его следует изучить. Цикл **for** проводит инициализацию перед первым шагом цикла. Затем выполняется проверка условия цикла, и в конце каждой итерации происходит изменение управляющей переменной.

```
for(инициализация; условие; итерация) {
    блок кода
}
```

Блок кода будет выполняться, пока условие не станет неверным. Сначала происходит инициализация, затем проверяется условие, и если оно верное, то выполняется блок кода, затем после выполнения блока кода идет итерация (то есть шаг).

Например:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int size = in.nextInt();
        for(int i=0;i<size;i++){
            System.out.println("Hello BITLAB");
        }
    }
}
```

Данная программа будет выполнять блок кода (Вывод текста: Hello BITLAB), столько раз, чему будет равно значение **size**.

## Упражнения для разминки

### Задание 1:

Напишите программу, в котором я ввожу число **a** и **b**. Программа должна вывести мне умножение этих двух чисел. (Не используйте оператор умножения **\***, вместо этого - цикл.)

### Задание 2:

Введите в программу число **n**.  
Программа должна вывести квадраты чисел от 1 до **n**.

**Ввод:**

3

**Вывод:**

1 1

2 4

3 9

**Ввод:**

10

**Вывод:**

1 1

2 4

3 9

4 16

5 25

6 36

7 49

8 64

9 81

10 100

### Задание 3:

Напишите программу, которая запрашивает числа. Программа должна остановиться запрашивать, когда мы вводим 0. Программа должна вывести количество введенных чисел и среднее значение.

**Ввод:**

3 8 9 0

**Вывод:**

3

6.67

**Ввод:**

10 45 87 34 12 0

**Вывод:**

5

37.6

#### Задание 4:

Напишите программу, которая запрашивает числа. Программа должна остановиться запрашивать, когда мы вводим 0. Программа должна вывести максимальное значение из всех чисел.

**Ввод:**

1 2 3 0

**Вывод:**

3

**Ввод:**

3 17 8 9 0

**Вывод:**

17

37.6

#### Задание 5:

Вводим число  $n$ . Программа должна вывести сумму первых  $n$  чисел от последовательности 1,3,5,7...

**Ввод:**

3

**Вывод:**

9

**Ввод:**

15

**Вывод:**

225

#### Задание 6:

Вводим число  $a$  и  $b$ . Программа должна вывести таблицу степени  $a^n$ , где  $n$  это числа которые идут от 0 до  $b$ .

**Ввод:**

7 2

**Вывод:**

0 1

1 7

2 49

**Ввод:**

2 10

**Вывод:**

0 1

1 2

2 4

3 8

4 16

5 32

6 64

7 128

8 256

9 512

10 1024

## Практические задачи

### Задание 1:

Напишите программу, в котором я ввожу число **n**. Программа должна вывести умножение первых **n** чисел.  $1*2*3*4*5...n$

**Ввод:**

3

**Вывод:**

6

**Ввод:**

10

**Вывод:**

3628800

**Задание 2:**

Напишите программу, которая запрашивает числа. Программа должна остановиться запрашивать, когда мы вводим 0.

Программа должна вывести умножение всех введенных чисел.

**Ввод:**

3.1 2.4 1.2 0

**Вывод:**

8.928

**Ввод:**

5.2 -2.3 0

**Вывод:**

-11.96

**Задание 3:**

Напишите программу, которая запрашивает числа. Программа должна остановиться запрашивать, когда мы вводим 0.

Программа должна вывести **сумму нечетных элементов**, которые мы вводили.

**Ввод:**

3 4 0

**Вывод:**

3

**Ввод:**

-3 5 8 2 7 0

**Вывод:**

9

**Задание 4:**

Программа запрашивает число **n**. Программа должна вывести сумму первых **n** чисел в последовательности  $1+1/2+1/3+1/4 \dots$

**Ввод:**

3

**Вывод:**

1.83333

**Ввод:**

15

**Вывод:**

3.31823

**Задание 5:**

Вводим число **a** и **b**. Программа должна вывести  $a^b$  не используя математические операторы степени. Используйте цикл.

**Ввод:**

7 2

**Вывод:**

49

**Ввод:**

2 10

**Вывод:**

1024

## Практические задачи по двумерным циклам

### Задание 1:

Напишите программу в котором я ввожу целостные числа **n** и **m**. Программа должна вывести следующее.

**Ввод:**

5 3

**Вывод:**

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

### Задание 2:

Напишите программу в котором я ввожу целостное число **n**. Программа должна вывести следующее.

**Ввод:**

5

**Вывод:**

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

### Задание 3:

Напишите программу в котором я ввожу целостное число **n**. Программа должна вывести следующее.

**Ввод:**

5

**Вывод:**

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*



**Задание 4:**

Напишите программу в котором я ввожу целостное число **n**. Программа должна вывести следующее.

**Ввод:**

5

**Вывод:**

```
*  
***  
*****  
*****  
*****
```

## Массивы, одномерные массивы

Массив - тип или структура данных в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом. При этом доступ к отдельным элементам массива осуществляется с помощью индексации, то есть через ссылку на массив с указанием номера (индекса) нужного элемента. В языке JAVA массив удобен для работы с большим количеством однотипных данных.

```
int[] books;
```

Мы создали массив с типом **int** и назвали ее **books**.

```
int[] books = new int[7];
```

Мы создали массив с типом **int** и назвали ее **books** у которого размер является 7.

В данном примере, значение элементов равно 0, так как мы не указали значение каждого элемента. Например, для числовых значений начальное значение будет 0. Для массива типа **boolean** начальное значение будет равно **false**, для массива типа **char** - **'\u0000'**, для массива типа класса - **null**.

Каждая переменная в данном массиве называется элементом массива. Чтобы сослаться на определенный элемент в массиве нужно знать имя массива в соединении с целым значением, называемым индексом. Индекс указывает на позицию конкретного элемента относительно начала массива. Индекс начального элемента — 0, следующего за ним — 1 и т. д. Индекс последнего элемента в массиве — на единицу меньше, чем размер массива.

```
int[] books = {1, 4, 5, 7, 8};
```

В данном примере, мы создали массив, сразу заполнив его элементы. Размер данного массива является 5.

```
int[] books = {1, 4, 5, 7, 8};  
System.out.println(books[2]);
```

Программа выведет число 5, то есть третий элемент массива. Таким образом, чтобы вывести первый элемент массива нам надо обратиться к индексу 0. А последний элемент - 4. Чтобы получить размер массива, мы обратимся к полю **length**.

```
int[] books = {1, 4, 5, 7, 8};  
System.out.println(books.length);
```

В данном примере мы получим размер массива, что является 5.

Массивы можно использовать в циклах, например, если мы хотим вывести все элементы.

```
int[] marks = {40, 56, 75, 95, 88};  
for(int i = 0; i < marks.length; i++){  
    System.out.println("marks[" + i + "] = " + marks[i]);  
}
```

Программа выведет:

marks[0] = 40

marks[1] = 56

marks[2] = 75

marks[3] = 95

marks[4] = 88

## Упражнения для разминки

### Задание 1:

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Выведите в конце все элементы в обратном порядке.

**Ввод:**

10

6 19 26 -3 46 8 5 -65 90 25

**Вывод:**

25 90 -65 5 8 46 -3 26 19 6

### Задание 2:

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Программа должна вывести сумму и среднее значение введенных чисел.

**Ввод:**

3

4 9 2

**Вывод:**

15 5

### Задание 3:

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Программа должна вывести максимальное и минимальное значение в массиве.

**Ввод:**

10

6 19 26 9 46 8 5 65 90 25

**Вывод:**

5 90

**Задание 4:**

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Программа должна вывести сумму и среднее значение исключая максимальное и минимальное значение.

**Ввод:**

10  
6 19 26 9 46 8 5 65 90 25

**Вывод:**

204

**Задание 5:**

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Далее, программа запрашивает пользователя число **m**. Если число **m** существует в нашем массиве, программа должна вывести слово "Yes" и вывести индекс (расположение, адрес) данного числа. Иначе вывести слово "No".

**Ввод:**

3  
4 9 2  
8

**Вывод:**

No

**Ввод:**

10  
6 19 26 3 46 8 5 65 90 25  
46

**Вывод:**

Yes  
Index: 4

**Упражнения по двумерным массивам****Задание 1:**

Программа запрашивает число **N**, затем вы создаете двумерный массив с размером **2xN**, заполняете их. Программа должна вывести "Yes", если числа в первом массиве совпадают со вторыми. Иначе "No".

**Ввод:**

3  
1 2 3  
1 2 3

**Вывод:**

Yes

**Ввод:**

5  
1 9 0 5 6  
1 9 1 5 6

**Вывод:**

No

## Задание 2:

Программа запрашивает два числа **N** и **M**, затем мы создаем двумерный массив и заполняем их числами. Программа в конце должна посчитать количество отрицательных чисел, и при выводе их заменить символом "x".

**Ввод:**

2 3  
0 -2 3  
-5 8 -8

**Вывод:**

3  
0 x 3  
x 8 x

**Ввод:**

4 5  
2 -4 -5 6 7  
0 1 -2 9 11  
-1 -1 8 3 0  
3 4 5 6 7

**Вывод:**

```
5
2 x x 6 7
0 1 x 9 11
x x 8 3 0
3 4 5 6 7
```

**Задание 3:**

Программа запрашивает два числа **N** и **M**, затем мы создаем двумерный массив и заполняем их числами. Программа должна вывести количество отрицательных чисел в каждой строке.

**Ввод:**

```
2 3
0 -2 3
-5 8 -8
```

**Вывод:**

```
1
2
```

**Ввод:**

```
4 5
2 -4 -5 6 7
0 1 -2 9 11
-1 -1 8 3 0
3 4 5 6 7
```

**Вывод:**

```
2
1
2
0
```

**Задание 4:**

Программа запрашивает число **N**, затем мы создаем двумерный массив **N x N** и заполняем их числами. Программа должна вывести массив таким образом, как в нижних примерах.

**Ввод:**

```
3
0 -2 3
-5 8 -8
1 2 3
```

**Вывод:**

```
x -2 3
-5 x -8
1 2 x
```

**Ввод:**

```
5
2 -4 -5 6 9
0 1 -2 9 0
-1 -1 8 3 3
3 4 5 6 1
7 11 0 7 12
```

**Вывод:**

```
x -4 -5 6 9
0 x -2 9 0
-1 -1 x 3 3
3 4 5 x 1
7 11 0 7 x
```

**Задание 5:**

Программа запрашивает число **N**, затем мы создаем двумерный массив **N x N** и заполняем их числами. Программа должна вывести **"Yes"**, если диагонали элементов симметричны, иначе **"No"**.  
 \*\*\* Для понятия диагонали, они в примере выделены жирным шрифтом.

**Ввод:**

```
3
1 2 3
2 7 4
3 4 6
```

**Вывод:**

```
Yes
```



**Ввод:**

```

3
1 2 3
2 5 6
3 8 0

```

**Вывод:**

```

No

```

## Практические задачи

### Задание 1:

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Выведите в конце только положительные элементы.

**Ввод:**

```

10
6 19 26 -3 46 8 5 -65 90 25

```

**Вывод:**

```

6 19 26 46 8 5 90 25

```

### Задание 2:

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Программа должна заменить местами вывести максимального и минимального элемента.

**Ввод:**

```

3
4 9 2

```

**Вывод:**

```

4 2 9

```

**Ввод:**

10

6 19 26 9 46 8 5 65 90 25

**Вывод:**

6 19 26 9 46 8 90 65 5 25

**Задание 3:**

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Программа должна вывести умножение всех элементов, не равных 0.

**Ввод:**

4

9 2 4 0

**Вывод:**

72

**Ввод:**

10

6 19 0 -3 4 8 0 -6 9 5

**Вывод:**

2954880

**Задание 4:**

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Программа должна вывести с начала все положительные элементы, затем нули и затем только отрицательные элементы.

**Ввод:**

4

9 -2 4 0

**Вывод:**

9 4 0 -2

**Ввод:**

10

6 19 0 -3 4 8 0 -6 9 5

**Вывод:**

6 19 4 8 9 5 0 0 -3 -6

**Задание 5:**

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Затем мы вводим число **m**. Программа должна вывести среднее значение всех элементов, которые больше **m**.

**Ввод:**

4

0 -2 4 0

1

**Вывод:**

4

**Ввод:**

10

6 19 0 -3 4 8 0 -6 9 5

5

**Вывод:**

10.5

**Задание 6:**

Программа запрашивает число **n**. Далее, мы вводим **n** чисел и сохраняем все введенные числа в массив. Программа должна вывести сумму всех чисел которые находятся между нулями.

**Ввод:**

4

0 -2 4 0

**Вывод:**

2

**Ввод:**

10

6 19 0 -3 4 8 0 -6 9 5

**Вывод:**

9

## Практические задачи по двумерным массивам

### Задание 1:

Напишите программу в котором я ввожу два числа **N** и **M**. Затем создаю двумерный массив **NxM**. Далее мы заполняем этот массив числами. Программа должна вывести максимальный элемент в каждой строке.

**Ввод:**

2 3

0 -2 3

-5 8 -8

**Вывод:**

3

8

**Ввод:**

4 5

2 -4 -5 6 7

0 1 -2 9 11

-1 -1 8 3 0

3 4 5 6 7

**Вывод:**

7

11

8

7

### Задание 2:

Программа запрашивает число **N**, затем мы создаем двумерный массив **N x N** и заполняем их числами. Программа должна вывести массив таким образом, как в нижних примерах.

**Ввод:**

```
3
0 -2 3
-5 8 -8
1 2 3
```

**Вывод:**

```
x -2 x
-5 x -8
x 2 x
```

**Ввод:**

```
5
2 -4 -5 6 9
0 1 -2 9 0
-1 -1 8 3 3
3 4 5 6 1
7 11 0 7 12
```

**Вывод:**

```
x -4 -5 6 x
0 x -2 x 0
-1 -1 x 3 3
3 x 5 x 1
x 11 0 7 x
```

**Задание 3:**

Программа запрашивает число N, затем мы создаем двумерный массив N x N и заполняем их числами. Программа должна заменить первую горизонтальную половину на вторую горизонтальную половину.

**Ввод:**

```
3
1 2 3
2 7 4
3 4 6
```

**Вывод:**

```
3 4 6
2 7 4
1 2 3
```

**Ввод:**

```
5
8 9 0 1 5
5 2 3 7 6
0 7 8 9 4
1 2 3 4 5
7 6 4 2 3
```

**Вывод:**

```
1 2 3 4 5
7 6 4 2 3
0 7 8 9 4
8 9 0 1 5
5 2 3 7 6
```

**Задание 4:**

Напишите программу в котором я ввожу два числа **N** и **M**. Затем создаю двумерный массив **NxM**. Далее мы заполняем этот массив числами. Программа должна заменить максимальный элемент на минимальный элемент по колонке.

**Ввод:**

```
3 3
1 2 3
2 7 4
3 4 6
```

**Вывод:**

```
3 7 6
2 2 4
1 4 3
```

**Ввод:**

```
2 3
1 6 10
2 5 9
```

**Вывод:**

```
2 5 9
1 6 10
```

**Задание 5:**

Напишите программу в котором я ввожу два числа **N** и **M**. Затем создаю двумерный массив **NxM**. Далее мы заполняем этот массив числами. Затем, мы вводим число **k**. Программа должна вывести первые **k** отрицательных элементов в каждой колонке, как показано в примере.

**Ввод:**

```
4 5
2 -4 -5 6 7
0 1 -2 9 11
-1 -1 8 3 0
3 4 5 6 7
1
```

**Вывод:**

```
2 -4 -5 6 7
0 1 -2 9 11
-1 -1 8 3 0
3 4 5 6 7

- - - -
-1 -4 -5 x x
```

## Строки, класс String

Строки - это объекты класса **String**, они очень распространены, поэтому в некоторых случаях обрабатываются по-другому в отличие от всех остальных объектов. Строковые литералы записываются в двойных кавычках.

Создать строку очень просто.

Например:

```
String text = "BITLAB Team";
```

Можно создать массив строк:

```
String[] names = {"Ronaldo", "Modric", "Ramos"};
```

Можно создать строку через массив символов:

```
char[] symbols = { 'R', 'o', 'n', 'a', 'l', 'd', 'o' };  
String player = new String(symbols);
```

Можно использовать конструктор, позволяющий задать диапазон символьного массива. Вам нужно указать начало диапазона и количество символов для использования:

```
char[] symbols = { 'R', 'o', 'n', 'a', 'l', 'd', 'o', 'C', 'R', '7' };  
String logo = new String(symbols, 7, 3);
```

На языке Java знак плюс (+) означает конкатенацию строк.

```
String name = "Cristiano";  
String surname = "Ronaldo";  
String fullname = name + " " + surname; // Cristiano Ronaldo
```

## Методы класса String

### ***public char charAt (int index)***

Данный метод возвращает символ в строке, по индексу от 0 до ее размера - 1.

```
String fullName = "Cristiano Ronaldo";  
System.out.println(fullName.charAt());
```



***public boolean equals (Object string)***

Данный метод проверяет равны ли строки или нет. Если равны, то метод возвращает **true**, иначе **false**.

```
String str1 = "Cristiano";
String str2 = "Ronaldo";

if(str1.equals(str2)) {

    System.out.println("Строки равны");

}else{

    System.out.println("Строки не равны");

}
```

***public int length()***

Данный метод возвращает длину строки.

```
String name = "Cristiano";

System.out.println(name.length()); // возвращает 9
```

***public String trim()***

Удаляет пробелы до и после строки.

```
String text = "    Cristiano Ronaldo    ";

System.out.println(text.trim());
```

## Практические задачи

### Задание 1:

Мы вводим строку (текст) в нашу программу. Программа должна определить, является ли наш текст "Палиндром"-ом или нет. Палиндром - это когда текст читается так же одинаково если ее читать в обратном порядке.

#### Ввод:

kazak

#### Вывод:

Yes

#### Ввод:

assets

#### Вывод:

No

### Задание 2:

Мы вводим строку (текст) в нашу программу. Программа должна вывести каждую букву с ее количеством в тексте.

#### Ввод:

universe

#### Вывод:

u 1  
n 1  
i 1  
v 1  
e 2  
r 1  
s 1

**Ввод:**

macbookair

**Вывод:**

m 1  
a 2  
c 1  
b 1  
o 2  
k 1  
i 1  
r 1

**Задание 3:**

Мы вводим строку (текст) в нашу программу. Программа должна количество гласных букв.  
**Гласные буквы: а, е, і, о, u**

**Ввод:**

kayak

**Вывод:**

2

**Ввод:**

universe

**Вывод:**

4

**Задание 4:**

Вводим в программу две строки **s1** и **s2**. Если **s2** содержится внутри слова **s1**, то программа выводит "Yes", иначе "No".

**Ввод:**

universe  
ivery

**Вывод:**

No

**Ввод:**

macbookair  
book

**Вывод:**

Yes

### Задание 5:

Вводим строки **s1** и **s2** в программу. Программа должна вывести **"Yes"**, если **s2** является противоположным (в обратном чтении) **s1**. Иначе **"No"**.

**Ввод:**

universe  
ivery

**Вывод:**

No

**Ввод:**

macbookair  
irakoobcam

**Вывод:**

Yes

## Методы и аргументы.

**Методы** - это подпрограммы, присоединенные к конкретным определениям классов. Они описываются внутри определения класса на том же уровне, что и переменные объектов. При объявлении метода задаются тип возвращаемого им результата и список параметров.

```
public class Main{

    public static void main(String args[]){

        int sum = getSum(10,20);

        System.out.println(sum);

    }
    public static int getSum(int a, int b){

        return a + b;

    }
}
```

В данном примере мы создаем метод **getSum()** который берет два введенных числа с аргумента и возвращает их сумму. Методы в программе могут упростить задачу разработчику, если он собирается несколько раз использовать ее, или разбить программу на определенные блоки кода. **public** - в данном методе означает что метод доступен всем классам, **void** - это тип метода, который он возвращает, **static** - означает что метод статичный, то есть для его использования, нам не надо создавать отдельный экземпляр объекта класса **Main**, мы можем напрямую к нему обратиться таким образом: **Main.getSum(10,20);** или изнутри главного метода **main** как выше показано на примере.

Тип результата, который должен возвращать метод может быть любым, в том числе и типом **void** - в тех случаях, когда возвращать результат не требуется. Список формальных параметров или аргументы - это последовательность пар тип данных, разделенных запятыми. Если у метода параметры отсутствуют, то после имени метода должны стоять пустые круглые скобки.

```

public class Main{

    public static void main(String args[]){

        String madrid [] = {"Navas", "Ramos", "Ronaldo"};
        String barcelona [] = {"Ter Stegen", "Messi", "Suarez"};
        String juventus [] = {"Buffon", "Dybala", "Higuain"};
        String bayern [] = {"Neuer", "Robben", "Lewandowski"};

        printSquad("Real Madrid", madrid);
        printSquad("Barcelona", barcelona);
        printSquad("Juventus", juventus);
        printSquad("Bayern Munich", bayern);

    }

    public static void printSquad(String team, String squad[]){
        System.out.println("----- ");
        System.out.println("Team : " + team);

        for(int i=0;i<squad.length;i++){
            System.out.println(squad[i]);
        }
    }
}

```

В данном примере мы создаем 4 массива составов команд, и используем специальный метод **printSquad** который принимает в аргументы массив из String и выводит список всего состава. Таким образом, тут мы ничего не возвращаем так как используем **void**.

## Упражнения для разминки

### Задание 1:

Создайте метод который в аргументы принимает 3 значения целостных чисел, и возвращает в самую максимальную из них.

**Ввод:**

12 0 9

**Вывод:**

12

**Ввод:**

89 -34 432

**Вывод:**

432

### Задание 2:

Напишите метод который принимает строку и возвращает количество гласных букв.

**Ввод:**

algorithmization

**Вывод:**

7

**Ввод:**

university

**Вывод:**

4

### Задание 3:

Напишите метод, который принимает в аргументах массив целостных чисел, в итоге программа должна вывести количество элементов не равных 0.

**Ввод:**

3

8 9 2

**Вывод:**

3

**Ввод:**

8

-2 0 3 4 0 12 3 5

**Вывод:**

6

**Задание 4:**

Создайте метод который принимает двумерный массив целостных чисел. Метод должен вывести в какой строке меньше всего содержится элементов не равных 0.

**Ввод:**

3 3

8 5 4

0 0 9

1 0 6

**Вывод:**

1

**Ввод:**

3 4

1 0 0 0

3 0 1 0

6 0 9 7

**Вывод:**

3

**Задание 5:**

Создайте метод который принимает в аргументы двумерный массив целостных чисел, в итоге метод должен возвращать массив, заменив элементы первых строк с последними.

**Ввод:**

3 3

8 5 4

0 0 9

1 0 6



**Вывод:**

1 0 6

0 0 9

8 5 4

**Ввод:**

3 4

-4 -5 -5 5

8 4 -7 6

-6 -8 9 4

**Вывод:**

-6 -8 9 4

8 4 -7 6

-4 -5 -5 5

## Практические задачи

### Задание 1:

Создайте такой метод, который принимает в аргументы массив строк. Метод внутри должен создать массив строк, в котором содержаться строки, у которых количество гласных букв больше 4. В итоге метод должен возвращать этот массив.

**Ввод:**

arman

erbolat

armagedon

avadakedavra

lamusicasuen

**Вывод:**

avadakedavra

lamusicasuen

### Задание 2:

Создайте такой метод, который принимает в аргументы массив целостных чисел, и определенное целостное число **n**. Метод в итоге должен вывести те числа в массиве, значение которых совпадают с их индексами и не делятся на число **n**.

**Ввод:**

3

5 1 6

2

**Вывод:**

1

**Ввод:**

10

0 7 1 3 8 5 2 4 0 9

3

**Вывод:**

0 5

### Задание 3:

Создайте такой метод, который в аргументы принимает две строки **text**, **subtext**. Если **subtext** содержится в строке **text**, то метод возвращает "Yes", иначе "No".

**Ввод:**

blossom loss

**Вывод:**

Yes

**Ввод:**

university sink

**Вывод:**

No

### Задание 4:

Создайте такой метод, который принимает в аргументы массив целостных чисел. Метод должен возвращать массив целостных чисел, в котором содержится уникальные элементы. (Не повторяющийся числа).

**Ввод:**

3

5 1 6

**Вывод:**

5 1 6

**Ввод:**

10

0 7 1 3 8 5 3 4 0 8

**Вывод:**

7 1 5 4

**Задание 5:**

Создайте такой метод, который принимает в аргументы двумерный массив целостных чисел. Метод должен возвращать сумму всех чисел, у которых индексы по колонке и по строке нечетные.

**Ввод:**

```
3 3
8 5 4
0 0 9
1 0 6
```

**Вывод:**

```
0
```

**Ввод:**

```
3 4
-4 -5 -5 5
8 4 -7 6
-6 -8 9 4
```

**Вывод:**

```
10
```

## Классы и объекты

Класс является ключевым понятием в объектно-ориентированном программировании, под которое и заточена **Java**. Класс описывает содержание и поведение некой совокупности данных и действий над этими данными. Объявление класса производится с помощью ключевого слова `class`.

Пример: `class < имя_класса > { // содержимое класса }.`

Например, если брать в теории, то скажем что **“Человек”** является определенным классом (то есть каким-то типом данных), **а вот каждый человек**, является его объектом, то есть экземпляром. В Мире около 7 миллиардов человек, получается у нас 1 класс (тип) под названием **“Человек”**, и около 7 миллиардов экземпляров или объектов класса **“Человек”**.

**Класс – “Человек”** (В общем, с его параметрами. Имя, рост, вес, цвет волос, национальность и.т.д.)

**Объект - “Один конкретный экземпляр человека”** (Криштиану Роналду, 186 см, 78 кг, черные волосы, португалец)

### *Human.java*

```
public class Human {
    String name;
    int age;
    int height;
    public void run(){
        System.out.println(" Имя: " + name + ", возраст: " + age
            + ", рост: " + height + " выполняет пробежку ");
    }
}
```

### *Main.java*

```
public class Main {
    public static void main(String[] args) {
        Human h1 = new Human();
        h1.age = 28;
        h1.name = "Arman";
        h1.height = 184;
        h1.run(); // Имя: Arman, возраст: 28, рост: 184 выполняет пробежку
    }
}
```

## Конструктор

Конструктор — это метод класса, который инициализирует новый объект после его создания. Если мы хотим дать объекту какие-то начальные параметры, то мы можем все это прописать в конструкторе класса. Например, если брать наш пример человека, то мы знаем, что при рождении

ребенка, у него есть изначальный вес, начальный возраст, рост и т.д. Позже, когда уже ребенок растет, у него меняется его вес, возраст и рост, то есть это мы можем уже изменить с помощью методов.

Пример:

### *Human.java*

```
public class Human {
    String name;
    int age;
    int height;
    double weight;
    // Конструктор по умолчанию
    public Human() {
        this.age = 0;
        this.height = 0;
        this.weight = 3.5;
        this.name = " No Name ";
    }
    // Конструктор с параметрами
    public Human(int age, int height, double weight, String name) {
        this.age = age;
        this.height = height;
        this.weight = weight;
        this.name = name;
    }

    public void run() {
        System.out.println("Имя: " + name + ", возраст:" + age + ", вес:" + weight
            + ", рост: " + height + " выполняет пробежку ");
    }
}
```

### *Main.java*

```
public class Main {

    public static void main(String[] args) {

        Human h1 = new Human();

        h1.run(); //Имя: No Name, возраст:0, вес:0, рост:0 выполняет пробежку

        Human h2 = new Human(28, 186, 88, "Arman");

        h2.run(); //Имя: Arman, возраст:28, вес:88, рост:186 выполняет пробежку

    }

}
```

Конструктор инициализирует объект непосредственно во время создания. Имя конструктора совпадает с именем класса, включая регистр, а по синтаксису конструктор похож на метод без

возвращаемого значения. Конструкторов может быть **несколько**, но они по структуре **не должны быть похожи** (порядок и тип аргументов). Программа сама подбирает конструктор который мы сами используем при инициализации экземпляра.

**this** используется внутри любого метода для ссылки на текущий объект. То есть **this** всегда служит ссылкой на объект, для которого был вызван метод. Тем самым я указываю что поля age, height, weight, name являются полями именно текущего класса.

## Методы в классах.

Класс может содержать методы - один, два, три и больше в зависимости от сложности класса. Методы в классе используются в качестве функции, или действия. Например, в нашем примере человека, методом может быть его какие-то функциональные действия: есть, спать, ходить, говорить и.т.д.

Пример:

### *Human.java*

```
public class Human {

    String name;
    int age;
    int height;
    double weight;
    // Конструктор по умолчанию
    public Human() {
        this.age = 0;
        this.height = 0;
        this.weight = 3.5;
        this.name = " No Name ";
    }
    // Конструктор с параметрами
    public Human(int age, int height, double weight, String name) {
        this.age = age;
        this.height = height;
        this.weight = weight;
        this.name = name;
    }
    public void run() {
        System.out.println("Имя: " + name + ", возраст: " + age + ", вес: " + weight
            + ", рост: " + height + " выполняет пробежку ");
    }
    public String getUserData() {
        return "Имя: " + name + ", возраст: " + age + ", вес: " + weight + ", рост: " + height;
    }
}
```

**Main.java**

```

public class Main {

    public static void main(String[] args) {

        Human h1 = new Human();
        h1.run();
        // Имя: No Name, возраст:0, вес:0, рост:0 выполняет пробежку

        System.out.println(h1.getUserData());
        Human h2 = new Human(28,186,88,"Arman");
        h2.run();
        // Имя: Arman, возраст:28, вес:88, рост:186 выполняет пробежку
        System.out.println(h2.getUserData());

    }
}

```

Имя: No Name, возраст:0, вес:0, рост:0 выполняет пробежку

Имя: No Name, возраст:0, вес:0, рост:0

Имя: Arman, возраст:28, вес:88, рост:186 выполняет пробежку

Имя: Arman, возраст:28, вес:88, рост:186

В данном примере у нас вызываются два метода, **run()** - который просто выводит данные объекта и заставляет его бежать и **getUserData()** - который возвращает данные объекта. А выводим мы его с помощью **System.out.println(h1.getUserData());**

## Упражнения для разминки

### Задание 1:

Создайте класс **Student** с параметрами:

(\*\*\*Знак "+" означает что модификатор доступа **public**, "-" означает **private**, а "#" - **protected**)

```
+ int id;
+ String name;
+ String surname;
+ double gpa;

+ Student()
+ Student(int id, String name, String surname, double gpa)
```

```
+ String getStudentData() // Данный метод возвращает все данные студента.
```

В классе **Main**, вы должны создать 5 объектов разных студентов с разными параметрами.

Создайте массив из класса **Student**, заполните массив 5 объектами класса **Student** которые мы создали до этого, и используя цикл, выведите данные по каждому студенту.

### Задание 2:

Используйте предыдущий класс **Student**.

Создайте специальный метод **topStudent(Student students[])**, который в аргументы принимает массив из студентов. Метод должен вывести из списка **данные самого лучшего студента**, у которого **высокий gpa**. Создайте 10 объектов разных студентов с разными параметрами. Примените этот метод.

### Задание 3:

Создайте мини программу, с меню панелью отображенным ниже.

```
PRESS [1] TO ADD STUDENT
PRESS [2] TO LIST STUDENT
PRESS [0] TO EXIT
```

При нажатии [1]:

Insert name?

*Вводите имя с консоли*

Insert surname?

*Вводите фамилию с консоли*

Insert GPA?

*Вводите GPA с консоли*



После введения данных, программа должна добавить в список нашего студента.

При нажатии [2]: Программа должна вывести список всех студентов.

- 1) Ilyas Zhuanyshv 4.0
- 2) Aknur Abubakirova 3.9
- 3) Dauren Mukhametkarim 3.5
- 4) Almat Ybyray 3.4
- 5) Aziza Kamet 3.2

При нажатии [0], программа должна выйти.

#### Задание 4:

Создайте класс **Player** с параметрами:

```
+ int number;
+ String name;
+ String surname;
+ String position;

+ Player()
+ Player(int number, String name, String surname, String position)

+ String toString() // Данный метод возвращает данные по каждому игроку, то есть все поля в один
ряд.
```

Создайте класс **Club** с параметрами:

```
+ String name;
+ String country;
+ int ratingPoints;
+ Player []players;

+ Club()
+ Club(String name, String country, int ratingPoints, Player []players)

+ void printClubData(); // Данный метод выводит данные о клубе, включая список всех игроков,
которые присутствуют в массиве.
```

В главном классе **Main**, создайте 2 массива из разных игроков. (В каждом из них по 5 игроков). Далее, создайте два клуба и присвойте к ним массивы из игроков. Далее, создайте массив из клубов, и добавьте туда те 2 наших клуба. В конце, при помощи цикла, выведите данные каждого клуба.

**Задание 5:**

Используйте предыдущие классы **Player** и **Club**.  
Создайте специальных два метода в классе **Main**:

```
static Player [] sortPlayers(Player[] players)
```

и

```
static Club [] sortClub(Club []club)
```

Каждый из этих методов должен возвращать отсортированный массив по убыванию. То есть, массив должен быть отсортирован по критериям **number** (для **Player**) и **ratingPoints** (для **Club**). Алгоритм сортировки можно использовать (Bubble sort - сортировка пузырьками).

Создайте в главном классе **Main** 5 объектов класса **Club**, в каждом клубе должен быть по 7 игроков. Создайте массив из класса **Club**. Назовите массив – **league**. Отсортируйте всех игроков в каждом клубе, используя метод **sortPlayers**, и отсортируйте массив **league**, используя метод **sortClub**.

**Практические задачи****Задание 1:**

Создайте класс **Car** с параметрами:

```
+ String name;  
+ String model;  
+ int speed;  
+ int weight;
```

```
+ Car()  
+ Car(String name, String model, int speed, int weight)
```

```
+ String toString() // Данный метод возвращает все данные машины.  
+ void ride() // Данный метод выводит на экран что автомобиль с параметрами едет.
```

В классе **Main**, вы должны создать 5 объектов разных машин с разными параметрами.

**Задание 2:**

Создайте класс **Engine** с параметрами:

```
+ String name;
+ int cylinderAmount;
+ double cylinderVolume;
+ int weight;
```

Используйте предыдущий класс **Car**, добавьте в параметры объект класса **Engine**.

Класс **Car**

```
+ String name;
+ String model;
+ int speed;
+ int weight;
+ Engine engine;

+ Car()
+ Car(String name, String model, int speed, int weight, Engine engine)

+ String toString() // Данный метод возвращает все данные машины.
+ void ride() // Данный метод выводит на экран что автомобиль с параметрами едет.

+ int totalWeight(); //Данный метод возвращает общий вес машины вместе с двигателем.
```

В классе **Main**, вы должны создать 3 объекта разных двигателей, 6 объектов разных машин с разными параметрами, используя созданные нами объекты двигателей.

Добавьте все машины в массив, и выведите общий вес всех машин при помощи метода **totalWeight()** и цикла.

**Задание 3:**

Создайте класс **CPU**.

```
+ int cashMemory;
+ int price;
+ int weight;

+ CPU()
+ CPU(int cashMemory, int price, int weight)
```

Создайте класс **HDD**.

```
+ int memory;
+ int price;
+ int weight;

+ HDD()
+ HDD(int memory, int price, int weight)
```

Создайте класс **Laptop**.

```
+ String name;
+ int price;
+ int weight;
+ HDD hardDiskDrive;
+ CPU[] cpuMemory;

+ int getTotalPrice()
+ int getTotalCPUMemory()
+ int getTotalWeight()
```

**getTotalPrice()** - возвращает общую сумму компьютера, включая сумму всех процессоров и хард диска.

**getTotalCPUMemory()** - возвращает сумму памяти кэша в массиве **CPU**.

**getTotalWeight()** - Возвращает общую сумму веса компьютера, включая **CPU** и **HDD**.

Создайте в главном классе Main 2 объекта **CPU**, с разными параметрами. Затем создайте 2 объекта класса **HDD**. Создайте 6 объектов Laptop, чтобы в первых двух было по 2 **CPU**, у вторых два по 4, у третьих два по 8 **CPU**. В Каждом лаптопе должно быть по одному **HDD**. Добавьте все объекты в массив и выведите общий вес, цену и память процессоров используя методы **getTotalPrice()**, **getTotalCPUMemory()** и **getTotalWeight()**.

## Инкапсуляция, геттеры и сеттеры

Инкапсуляцию мы применяем с целью скрыть от непосредственного доступа к параметрам класса извне, а вместо этого дать доступ с помощью специальных методов. Таких методов мы называем **геттерами** и **сеттерами** (**getters** and **setters**). Инкапсуляция предохраняет данные объекта от нежелательного доступа, позволяя объекту самому управлять доступом к своим данным.

```
public class Animal{

    private double weight;
    private String name;

    public Animal(){
        this.weight = 0;
        this.name = "No Name";
    }

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

    public double getWeight(){
        return this.weight;
    }

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

    public void setWeight(double weight){
        this.weight = weight;
    }

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

}
```

В данном примере мы видим, что класс извне не может иметь прямой доступ к параметрам `weight` и `name`, но чтобы иметь к ним доступ мы используем методы `getWeight()` и `setWeight()`. Модификаторы доступа позволяют ограничить нежелательный доступ к членам класса извне.

Модификатор доступа	Область доступа
<b>public</b>	Без ограничений
<b>private</b>	Только из данного класса
<b>protected</b>	Из данного класса и его потомков
<b>Без модификатора</b>	Для всех классов данного пакета

Ключевое слово «**private**» дает самую слабую видимость — только внутри описания/методов самого класса.

Ключевое слово «**protected**» позволяет видеть переменные другим классам, но не всем. Переменные будут видны в классах-наследниках, и классах, находящихся в том же пакете.

Ключевое слово «**public**». Это описание позволяет обращаться к переменной/методу откуда угодно.

## Упражнения для разминки

### Задание 1

Создайте класс **User** с параметрами:

- int id;
- String login;
- String password;
- String role;

- + User()
- + User(int id, String login, String password, String role)

+ String **toString()** // Данный метод возвращает все данные пользователя. **Но, для возврата, используйте геттеры и сеттеры а не поля.**

Геттеры и сеттеры для каждого параметра.

В основном классе создайте несколько объектов класса **Users** и добавьте их в массив. Далее, используя цикл, выведите логин каждого пользователя.

### Задание 2

Во втором задании, вы должны заранее в массив добавить 10 объектов класса **Users**.

Используйте данный массив как базу данных пользователей.

При запуске программы, у вас должны запрашивать логин и пароль:

**INSERT LOGIN:**

*Вводите логин*

**INSERT PASSWORD:**

*Вводите пароль*

При верном раскладе, программа должна вас запустить

Welcome ilyas

PRESS [1] TO EDIT LOGIN

Insert login

PRESS [2] TO CHANGE PASSWORD

Insert old password

Insert new password

Re-Insert new password

PRESS [3] TO DELETE OWN ACCOUNT

PRESS [0] TO EXIT

### Задание 3

Создайте класс **Book** с параметрами:

- String name;
- String author;
- int isbn;
- int price;

Создайте конструкторы для них:

```
Book()
Book(String name, String author, int isbn, int price)
```

Создайте геттеры и сеттеры

Создайте метод **toString()** который возвращает все данные о книге.

Создайте класс **Library** с параметрами:

- String name;
- String city;
- **Book[] books = new Book[100];**

Создайте конструкторы для них:

```
Library()
Library(String name, String city)
```

Создайте геттеры и сеттеры.

Создайте метод, который возвращает количество всех книг в библиотеке:

```
+ int size()
```

Создайте метод который возвращает общую цену всех книг в библиотеке:

```
+ int priceSum()
```

Создайте метод **getBook(int index)**, который возвращает объект книги под индексом index.

Создайте метод **addBook(Book book)** который добавляет книгу в массив. (Создайте специальное приватное поле **int sizeOfBooks = 0**, которая увеличивается при добавлении книги)

В основном классе создайте 10 объектов класса Book.

Создайте два объекта класса **Library**. Используя метод **addBook(Book book)** добавьте **4 первых книг** в первый объект библиотеки, остальные 6 книг во второй объект библиотеки. Выведите общую сумму книг, и цены каждой книги и общую цену всех книг по каждой библиотеке.



## Практические задачи

### Задание 1

Создайте класс **Worker** с параметрами:

- int id;
- String name;
- String surname;
- String department;
- int salary;

+ Worker()  
+ Worker(int id, String name, String surname, String department, int salary)

Геттеры и сеттеры для каждого параметра.

+ String **toString()** // Данный метод возвращает все данные рабочего. **Но, для возврата, используйте геттеры и сеттеры а не поля.**

Создайте класс **BankApplication** с параметрами:

- String name;
- String country;
- **Worker []workers = new Worker[100];** // Для данного поля геттер и сеттер **запрещен**
- int **sizeOfWorkers;** // Для данного поля геттер и сеттер **запрещен**, он работает как счетчик.

Конструкторы:

+ **BankApplication()**  
+ **BankApplication(String name, String country)**

Геттеры и сеттеры для всех полей кроме **sizeOfWorkers**.

Методы:

**void addWorker(String name, String surname, String department, int salary)**

Данный метод добавляет в массив нового рабочего. (Заметьте, я не передаю поле индекса в аргументы, вместо этого я автоматически беру значение поля **sizeOfWorkers**)

**void listWorkers()** // Данный метод выводит на экран весь список рабочих по порядку массива.

В основном классе **Main**:

Используя один объект класса **BankApplication**, и используя его методы, создайте такое приложение с меню:

Welcome to BITLAB – Almaty

PRESS [1] TO ADD WORKER

PRESS [2] TO LIST WORKERS

PRESS [0] TO EXIT

## Наследование, полиморфизм.

Одним из ключевых аспектов объектно-ориентированного программирования является наследование. Это очень мощный инструмент, без которого не обходится ни одна профессионально написанная программа. С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого.

Используя наследование, можно создать общий класс, которые определяет характеристики, общие для набора связанных элементов. Затем вы можете наследоваться от него и создать новый класс, который будет иметь свои уникальные характеристики. Главный наследуемый класс в **Java** называют **суперклассом**. Наследующий класс называют подклассом. Получается, что **подкласс** - это специализированная версия суперкласса, которая наследует все члены суперкласса и добавляет свои собственные уникальные элементы.

Например, возьмем пример животных. Создадим класс **Animal.java**.

### *Animal.java*

```
public class Animal {

    String name;
    int weight;

    public Animal() {
        this.name = "No Name";
        this.weight = 0;
    }
    public Animal(String name, int weight) {
        this.name = name;
        this.weight = weight;
    }
    public void run() {

        System.out.println("Animal "+name+", "+weight+" kg is running");
    }

}
```

Создаем класс **Dog** который наследует от класса **Animal**. Чтобы наследовать класс, достаточно вставить имя наследуемого класса с использованием ключевого слова **extends**.

**Dog.java**

```

public class Dog extends Animal {

    String sound;

    public Dog() {
        super();
        this.sound = "No sound";
    }
    public Dog(String name, int weight, String sound) {
        super(name, weight);
        this.sound = sound;
    }

    public void run() {
        System.out.println(" Dog " + name + ", " + weight + " kg " +
            " with sound " + sound + " is running ");
    }
}

```

Используя слово **extends** мы расширяем класс **Animal**, тем самым нам не приходится заново объявлять параметры **name**, **weight**. Они как бы по умолчанию уже есть.

Далее, мы создаем еще один класс **Cat.java**.

```

public class Cat extends Animal {

    int nails;
    public Cat() {
        super();
        this.nails = 0;
    }
    public Cat(String name, int weight, int nails) {
        super(name, weight);
        this.nails = nails;
    }
    public void run() {
        System.out.println(" Cat " + name + ", " + weight + " kg " +
            " with " + nails + " nails is running ");
    }
}

```

**Main.java**

```
public class Main {
    public static void main(String args[]){
        Animal a1 = new Animal("Some animal", 12);
        Dog d1 = new Dog("Spike", 10, "Bow Bow");
        Cat c1 = new Cat("Garfield", 3, 20);
        a1.run();
        d1.run();
        c1.run();
    }
}
```

В данном примере мы создаем по одному объекту **Animal**, **Cat**, **Dog** и вызываем метод **run()**. Как вы заметили, подклассы **Cat** и **Dog** расширяют супер класс **Animal**. (Присваивают параметры и методы.)

Способность к изменению функциональности, унаследованной от базового класса, называется **полиморфизмом** и является одним из ключевых аспектов объектно-ориентированного программирования наряду с **наследованием** и **инкапсуляцией**.

**Полиморфизмом** называется возможность работать с несколькими типами так, как будто это один и тот же тип и в то же время поведение каждого типа будет уникальным в зависимости от его реализации.

Давайте попробуем создать несколько объектов разных классов, и поместить их в общий массив класса **Animal**.

**Main.java**

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

        Animal a1 = new Animal("Some animal", 12);
        Animal a2 = new Animal("Other animal", 9);
        Dog d1 = new Dog("Spike", 10, "Bow Bow");
        Dog d2 = new Dog("Tuzik", 9, "Gaf Gaf");
        Cat c1 = new Cat("Garfield", 3, 20);
        Cat c2 = new Cat("Tom", 4, 18);

        Animal animals[] = new Animal[6];
        animals[0] = a1;
        animals[1] = a2;
        animals[2] = b1;
        animals[3] = b2;
        animals[4] = c1;
        animals[5] = c2;

        for(int i=0;i<animals.length;i++){
            animals[i].run();
        }
    }
}
```

```

Animal Some animal, 12 kg is running
Animal Other animal, 9 kg is running
Dog Spike, 10 kg with sound Bow Bow is running
Dog Tuzik, 9 kg with sound Gaf Gaf is running "
Cat Garfield, 3 kg with 20 nails is running
Cat Tom, 4 kg with 18 nails is running

```

В данном примере для каждого объекта метод **run()** действует по разному, в зависимости от реализации.

В **Java** существует ключевое слово **super**, которое обозначает суперкласс, т.е. класс, производным от которого является текущий класс. В конструкторе **Dog** и **Cat** мы дублировали поля **name** и **weight**, которые уже есть в классе **Animal**. Это не слишком эффективно. Соответственно, вызвав ключевое слово **super**, мы вызываем действия конструктора класса **Animal**, и дополняем ее уже в конструкторах класса **Cat** и **Dog**. Ключевое слово **super** можно использовать для вызова конструктора суперкласса и для обращения к члену суперкласса, скрытому членом подкласса. Вызов метода **super()** всегда должен быть первым оператором, выполняемым внутри конструктора подкласса. При вызове метода **super()** с нужными аргументами, мы фактически вызываем конструктор **Animal**, который инициализирует переменные **name** и **weight**, используя переданные ему значения соответствующих параметров. Далее, мы дополняем то что нам нужно. У суперкласса могут быть несколько перегруженных версий конструкторов, поэтому можно вызывать метод **super()** с разными параметрами. Программа выполнит тот конструктор, который соответствует указанным аргументам.

## Упражнения для разминки

### Задание 1

Создайте класс **User** с параметрами:

```
+Users() // Конструктор по умолчанию
+Users(int id, String login, String password, String name, String surname)
```

```
protected int id
protected String login
protected String password
protected String name
protected String surname
```

```
+ User()
+ User(int id, String login, String password, String role)
```

Геттеры и сеттеры

```
+String getData() // Данный метод возвращает все данные пользователя
```

Создайте 2 класса которые наследуют от класса **User**:

#### 1 – **Staff**

```
+double salary
+String subjects[]
Геттеры и сеттеры
Переопределите метод getData(), относительно их параметрам
```

#### 2 – **Student**

```
+double gpa
+String coruses[]
Геттеры и сеттеры
Переопределите метод getData(), относительно их параметрам
```

В вашем основном классе **Main**, вы должны создать как минимум по 5 объектов класса **Student**, **Staff** и **Users**, и добавить их в массив из класса **Users**.

## Задание 2

Создайте меню для первого задания, где вы управляете студентами, рабочими и пользователями.

PRESS [1] ADD USER

PRESS [1] TO ADD STUDENT  
PRESS [2] TO ADD STAFF

PRESS [2] TO LIST USERS

PRESS [1] TO LIST STUDENTS  
PRESS [2] TO LIST STAFF

PRESS [0] TO EXIT

(Подсказка: Фильтр вывода студента или рабочего нужно реализовать с помощью ключевого слова: **instanceof**)

**instanceof** - специальное ключевое слово, которая возвращает true, если объект является типом данного класса.

Например:

```
Dog d = new Dog();
```

```
if(d instanceof Dog){  
    System.out.println("I am a Dog");  
}else{  
    System.out.println("I am not a Dog");  
}
```

В данном примере, объект **d** является экземпляром класса **Dog**, соответственно возвращает **true**.



## Практические задачи

### Задание 1

Создайте класс **Book** с параметрами:

```
private String name;
private String code;
private int pages;
```

```
+Book() // Конструктор по умолчанию
+Book(String name, String code, int pages) // Конструкторы с параметрами
```

Геттеры и сеттеры

```
+String getBookData(); // Данный метод возвращает данные о книге
```

Создайте класс **ScientificBook** которая наследует от **Book**:

```
private int price;
private String publisher;
```

```
+ ScientificBook()
+ ScientificBook(String name, String code, int pages, int price, String publicsher)
```

Геттеры и сеттеры

Переопределите метод **getBookData()**

Создайте класс **LiteratureBook** Которая наследует от класса **Book**:

```
private String author;
private int publishedYear;
```

```
+ LiteratureBook()
+ LiteratureBook(String name, String code, int pages, String author, int publishedYear)
```

Геттеры и сеттеры

Переопределите метод **getBookData()**

Создайте класс **Library** с параметрами:

```
private String name;
private String city;
private String country;
private Book[] books; //Для данного массива не создавайте геттер и сеттер
private int sizeOfBooks; //Для данного типа не создавайте геттер и сеттер
```

```
+University() // Конструктор по умолчанию
```

```
+University(String name, String city, String country, Book[] books)
```

Геттеры и сеттеры

```
+ void addBook(Book b) // данный метод добавляет книгу в массив используя курсор sizeOfBooks
```

```
+ void printLibraryData() //Данный метод выведет данные о библиотеке и выведет данные о каждой книге
```

In main class:

Создайте объект класса **Library**.

Добавьте туда разных по 5 объектов класса **ScientificBook** и **LiteratureBook**.

Выведите все данные о библиотеке с ее книгами используя метод **printLibraryData()**

## Задание 2

Создайте программу которая будет хранить книги в вашей библиотеке.

Программа позволит искать книги по определенным критериям:

```
PRESS [1] TO SEARCH BOOK BY NAME
    INSERT NAME OF THE BOOK:
PRESS [2] TO SEARCH BOOK BY CODE
    INSERT CODE OF THE BOOK:
PRESS [3] TO SEARCH BOOK BY PAGES AMOUNT
    INSERT MINIMUM AMOUNT OF PAGES:
    INSERT MAXIMUM AMOUNT OF PAGES:
```

## Абстрактные классы

Бывают моменты, когда нужно определить класс, в котором задана структура какой-либо абстракции, но реализация всех методов отсутствует. В таких случаях мы можем с помощью модификатора **abstract** объявить, что некоторые из методов обязательно должны быть реализованы в подклассах. Класс, содержащий абстрактные методы, называется абстрактным классом. Такие классы помечаются ключевым словом **abstract**.

Абстрактные классы объявляются с ключевым словом **abstract** и содержат объявления абстрактных методов, которые не реализованы в этих классах, а будут реализованы в подклассах. Объекты таких классов создавать нельзя, но можно создать объекты подклассов, которые реализуют эти абстрактные методы. Абстрактные классы могут содержать и реализованные методы, а также конструкторы и поля данных.

### *Shape.java*

```
public abstract class Shape{

    public abstract double getPerimeter();
    public abstract double getArea();

    public void printArea(){

        System.out.println("MY AREA IS : "+getArea());

    }

    public void printPerimeter(){

        System.out.println("MY PERIMETER IS : "+getPerimeter());

    }

}
```

Мы создаем абстрактный класс **Shape** (Фигура) в котором есть два метода **printArea()** и **printPerimeter()** и два абстрактных метода **getArea()** и **getPerimeter()**, которые возвращают значения площади и периметра фигур. Но загвоздка в том, что мы не знаем, что это за фигура, круг, квадрат или треугольник, ибо для каждого из них вычисление периметра и площади по-разному. Соответственно, мы и создали абстрактный класс **Shape** (Фигура), в котором есть абстрактные методы.

Далее мы создаем класс **Circle** (Круг), который наследует от класса **Shape**, где мы уже конкретно можем реализовать два абстрактных метода суперкласса **Shape**.

**Circle.java**

```

public class Circle extends Shape{

    double radius;

    public Circle(){
        this.radius = 0;
    }
    public Circle(double radius){
        this.radius = radius;
    }
    public double getArea(){

        return 3.14*radius*radius;

    }
    public double getPerimeter(){

        return 2*3.14*radius;

    }
}

```

Тут мы видим конкретно что у круга площадь измеряется с помощью радиуса и числа Пи (мы его обозначим дробным числом 3.14).

Далее мы создаем еще один класс **Square** (Квадрат), который тоже наследует от класса **Shape**, и который тоже реализовывает абстрактные методы суперкласса **Shape**.

```

public class Square extends Shape{

    double width;

    public Square(){
        this.width = width;
    }
    public Square(double width){
        this.width = width;
    }
    public double getArea(){

        return width*width;

    }
    public double getPerimeter(){

        return 4*width;

    }
}

```

**\*\*\* Примечание:** Подклассы всегда должны реализовать абстрактные методы суперклассов или сами должны быть объявлены абстрактными.

Абстрактный класс может содержать не только абстрактные, но и обычные методы.

```
public class Main{

    public static void main(String[] args) {

        Circle c = new Circle(5);
        Square s1 = new Square(5);
        Square s2 = new Square(15);

        Shape figures[] = new Shape[3];

        figures[0] = c;
        figures[1] = s1;
        figures[2] = s2;

        for (int i=0;i<figures.length; i++) {
            figures[i].printArea();
        }

    }
}
```

Одним словом, абстракция - это когда мы создаем что-то неконкретное (В данном примере это **Shape** - Фигура, у которой есть площадь и периметр), но позже, наследуя от них, мы реализовываем их абстрактные методы (конкретизируем), и применяем. В нашем примере есть два метода **printArea()** и **printPerimeter()**, которые выводят в консоль значение их периметров и площади. Но, так как реализация для каждого из конкретного класса (Конкретной фигуры) по-разному, мы их оставляем на будущее, то есть для подклассов, тем самым объявляя их абстрактными.

## Упражнения для разминки

### Задание 1

Создайте абстрактный класс **Engine** с параметрами:

- double engineVolume;
- int cylinderAmount;
- double engineWeight;

Конструкторы с параметрами и конструктор по умолчанию  
Геттеры и сеттеры

```
+ abstract double efficiency();
+ abstract double throttleEnergy();
+ abstract double breakEnergy();

+ double getMaxSpeed(){
    return (throttleEnergy()-breakEnergy())*efficiency();
}
```

Создайте класс **FerrariEngine** которая наследует от класса **Engine**.  
Конструкторы с параметрами и конструктор по умолчанию  
Реализуйте все абстрактные методы

В **FerrariEngine** вы вычислите абстрактные методы по формуле:

Efficiency: 0.25

Throttle energy: engineVolume\*cylinderAmount\*100

Break energy: engineWeight\*2

Создайте класс **RenaultEngine** которая наследует от класса **Engine**.

- double extraTurboEnergy;

Конструкторы с параметрами и конструктор по умолчанию  
Реализуйте все абстрактные методы

В **RenaultEngine** вы вычислите абстрактные методы по формуле:

Efficiency: 0.27

Throttle energy: engineVolume\*cylinderAmount\*110 + extraTurboEnergy

Break energy: engineWeight\*2.1

В основном классе Main, создайте 10 объектов класса **FerrariEngine** и **RenaultEngine** с разными значениями.

Добавьте все в массив класса **Engine**.

Выведите максимальные скорости каждого объекта

## Практические задачи

### Задание 1

Создайте абстрактный класс **User** с параметрами:

- int id;
- String login;
- String password;

abstract String getUserData();

Создайте конструкторы по умолчанию и с параметрами  
Создайте геттеры и сеттеры

Создайте класс **Student** которая наследует от класса **User** с параметрами.

- String name;
- String surname;
- String group;
- double gpa;

Создайте конструкторы по умолчанию и с параметрами  
Создайте геттеры и сеттеры

Реализуйте метод **getUserData()** которая должна возвращать все параметры пользователя.

Создайте класс **Teacher** которая наследует от класса **User** с параметрами.

- String nickName;
- String status; //// например: Professor, Lecturer, Tutor, Assistant.
- String subjects[] = new String[10]; //// Предметы которые он ведет, максимум 10
- int sizeOfSubjects = 0;

Создайте конструкторы по умолчанию и с параметрами  
Создайте геттеры и сеттеры  
Для полей **sizeOfSubjects** и **subjects[]** создайте только геттеры.

Создайте специальный метод **addSubject(String subject)**, которая добавляет предмет в массив **subjects**.

Реализуйте метод **getUserData()** которая должна возвращать все параметры пользователя, включая все предметы которые он ведет.

Создайте класс **ERPSystem** которая управляет пользователями, с параметрами и методами:

```
- User memory = new User[1000]; ////максимум 1000 пользователей
- int sizeOfUsers = 0;
```

```
void addUser(User u); //// метод добавляет пользователя
```

```
void printAllUsers(); //// метод выводит на экран всех пользователей
```

```
void printUser(int index); //// метод выводит на экран одного пользователя, но если пользователь
по данному индексу не существует, то выводит сообщение: "No user in this index"
```

В главном классе Main, создайте один объект класса **ERPSystem**, где вы управляете пользователями.

PRESS [1] TO ADD USER

PRESS [1] TO ADD STUDENT

Name:

Surname:

Group:

GPA:

PRESS [2] TO ADD TEACHER

Nick Name:

Status:

PRESS [1] TO ADD SUBJECT

Subject:

PRESS [0] TO FINISH ADDING SUBJECT

PRESS [2] TO LIST USERS

PRESS [1] TO LIST STUDENTS

PRESS [2] TO LIST TEACHERS

PRESS [0] TO EXIT



## Интерфейсы

**Интерфейс** - важная конструкция для реализации объектно-ориентированного программирования в языке программирования **Java**. Один из вариантов абстрактного класса. Оно начинается с заголовка. Сначала указываются модификаторы. Интерфейс может быть объявлен как **public**, и тогда он будет доступен для общего использования, либо модификатор доступа может не указываться, в этом случае интерфейс доступен только для типов своего пакета. Модификатор **abstract** для интерфейса не требуется, поскольку все интерфейсы являются абстрактными классами. Его можно указать, но делать этого не рекомендуется, чтобы не загромождать код. Далее записывается ключевое слово **interface** и имя интерфейса. После этого может следовать ключевое слово **extends** и список интерфейсов, от которых будет наследоваться объявляемый интерфейс. Родительских типов (классов и/или интерфейсов) может быть много — главное, чтобы не было повторений, и чтобы отношение наследования не образовывало циклической зависимости. Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов.

Пример:

### *Graphics.java*

```
public interface Graphics{

    public void draw();

}
```

### *Shape.java*

```
public interface Shape{

    public double getArea();
    public double getPerimeter();

}
```

Мы создаем два интерфейса **Shape** (Фигура) и **Graphics** (Графика), чтобы в которых есть методы **getArea()**, **getPerimeter()** и **draw()** соответственно. Интерфейс **Graphics** отвечает за рисование какой-либо фигуры, а интерфейс **Shape** за саму фигуру и ее параметры.

Давайте теперь создадим конкретный класс, конкретную фигуру **Circle** которая реализует сразу два интерфейса **Shape** и **Graphics**. В реализации данного класса мы должны будем реализовать все абстрактные методы наследуемых интерфейсов. Если нам надо применить в классе несколько интерфейсов, то они все перечисляются через запятую после слова **implements**.

**Circle.java**

```

public class Circle implements Shape, Graphics{

    double radius;

    public Circle(){

        this.radius = 0;

    }

    public Circle(double radius){

        this.radius = radius;

    }

    public void draw(){

        System.out.println("I am drawing Circle");

    }

    public double getArea(){

        return 3.14*radius*radius;

    }

    public double getPerimeter(){

        return 2*3.14*radius;

    }

}

```

**Main.java**

```

public class Main{

    public static void main(String[] args) {

        Circle c = new Circle(5);

        c.draw();

    }

}

```

## Упражнения для разминки

### Задание 1

Создайте класс **Users** с параметрами:

- String name;
- String surname;

Конструкторы по умолчанию и с параметрами.

Создайте геттеры и сеттеры

Создайте интерфейс **UserBeanLocal** с методами:

```
Users[] getAllUsers();  
Users[] getUsersByName(String name);  
Users[] getUsersBySurname(String surname);
```

Создайте класс **UserBean** которая реализовывает интерфейс **UserBeanLocal**.

- Users[] users;

Реализуйте все методы.

В главном классе создайте массив из минимум 10 объектов класса **Users**. Создайте объект класса **UserBean**. Используя данный класс, выведите список всех пользователей, и всех пользователей на имя "John", и выведите список людей на фамилию "Smith".

## Графика (GUI).

### Создание нашего первого графического окна

В программах на **Java** можно создавать графический пользовательский интерфейс (GUI), используя графический компонент библиотеки **Java** под названием **Swing**. Пакет **javax.swing** содержит классы, при помощи которых можно создать разнообразные компоненты, используя стиль операционной системы. Чтобы включить эту возможность, нужно добавить оператор импорта в программу: **import javax.swing.\*;**.

Для создания графического пользовательского интерфейса нужно организовать класс, к которому можно добавлять компоненты, используемые при построении интерфейса. Проще всего это сделать, объявив подкласс класса **JFrame** при помощи ключевого слова **extends** — тем самым наследуя атрибуты и методы, которые позволят пользователю работать с окном: перемещать, изменять размер и закрывать.

#### MainFrame.java

```
import javax.swing.*;
import java.awt.*;

public class MainFrame extends JFrame{
    public MainFrame() {

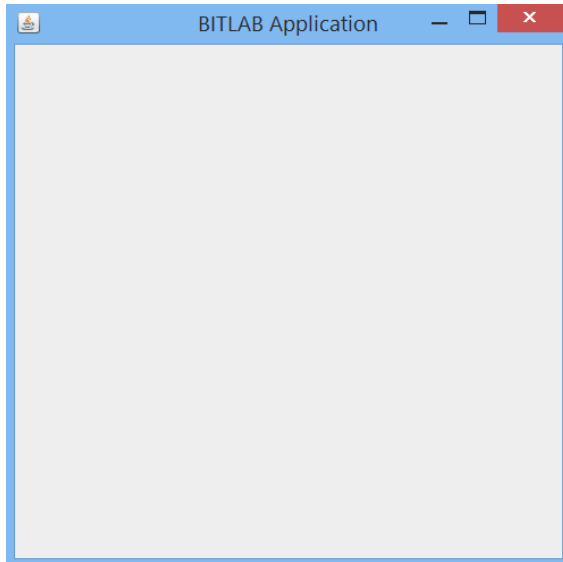
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("BITLAB Application");
        setSize(500,500);

    }
}
```

#### Main.java

```
public class Main{
    public static void main(String[] args) {
        MainFrame frame = new MainFrame();
        frame.setVisible(true);
    }
}
```

В данном примере мы создаем простое окно, где наш класс **MainFrame** наследует от класса **JFrame**.



*Наше первое графическое окно.*

В нашем коде метод `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)` нужен чтобы при нажатии кнопки закрытия окна (крестик в правом углу), то программа должна завершиться. По умолчанию, окно исчезает, но программа не завершается.

Метод `setTitle("BITLAB Application")` - нужен нам для настройки заголовка окна.

Метод `setSize(500, 500)` - нужен нам для настройки размера (ширина и высота) в пикселях.

### Добавление кнопок, текстового поля и метки (надписи).

Для начала попробуем создать простой пример окошки, где мы имеем текстовое поле, метку и кнопку. После введении текста в текстовое поле и нажатии кнопки, текст нашей метки должен измениться, то есть принять значение текстового поля. Но для начала, мы должны создать окно с текстовым полем, меткой и кнопкой.

Окно мы создаем при помощи наследования нашего класса от класса **JFrame**. Далее, мы задаем размер нашей окошки, и размещаем туда наши элементы. При помощи метода `setLayout(null)`, мы даем нашим элементам свободно переставляться по площади нашей окошки.

Для текстового поля мы применяем класс **JTextField**. Метод `textField.setLocation(100,150)` задает расположение кнопки по пиксели по ширине и по высоте. Далее, мы создаем метку, используя класс **JLabel**.

Кнопка создается с помощью класса **JButton**. Мы задаем ей своего слушателя (`ActionListener`), который срабатывает при нажатии на кнопку. В методе `actionPerformed(ActionEvent e)`, мы берем значение текстового поля и задаем это значение к нашей метке.

**MainFrame.java**

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MainFrame extends JFrame{

    private JButton button;
    private JLabel label;
    private JTextField textField;

    public MainFrame() {

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("BITLAB Application");
        setSize(500,500);

        setLayout(null);

        label = new JLabel("Hello BITLAB");
        label.setSize(300,30);
        label.setLocation(100,100);
        add(label);

        textField = new JTextField();
        textField.setSize(300,30);
        textField.setLocation(100,150);
        add(textField);

        button = new JButton("CONFIRM");
        button.setSize(300,30);
        button.setLocation(100,200);
        button.addActionListener(new ActionListener() {

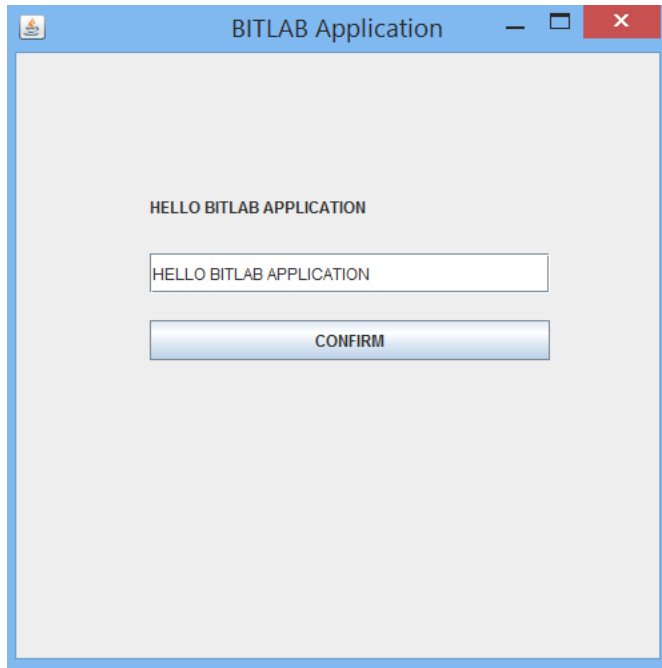
            public void actionPerformed(ActionEvent e) {

                String text = textField.getText();
                label.setText(text);

            }

        });
        add(button);
    }
}

```

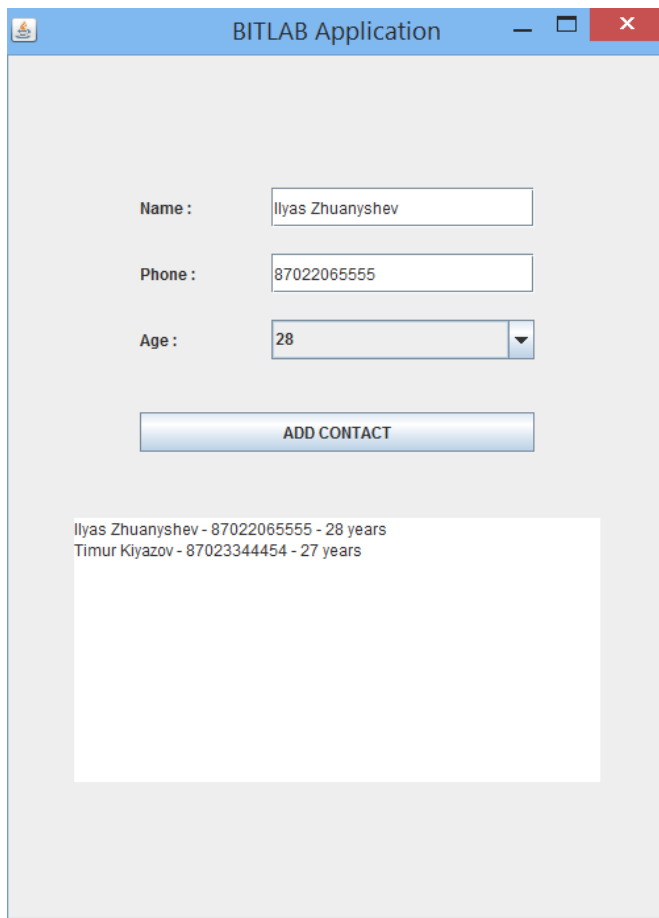


*Наше графическое окно. Пример того, как можно брать значение текстового поля, и задавать это значение в другой элемент как метка. Действие кнопки происходит при добавлении слушателя.*

## Упражнения для разминки

### Задание 1.

Создайте графическое приложение, которая добавляет в список контактов пользователей, введенные в текстовое поле. Для отображения списка пользователей, используйте **JTextArea**.





## Обработка исключений, try – catch.

Исключение в Java — это объект, который описывает исключительное состояние, возникшее в каком-либо участке программного кода. Это нештатная ситуация, ошибка во время выполнения программы.

Самый простой пример - мы создаем объект класса **Scanner** для считывания целостных чисел с консоли. И при запуске программы мы вместо целостного числа вводим какой-либо текст. Так как программа ожидает число, у нас возникает нештатная ситуация. Ошибка, которая возникает при запуске, а не при компиляции. Можно вручную отслеживать возникновение подобных ошибок, а можно воспользоваться специальным механизмом исключений, который упрощает создание больших надёжных программ, уменьшает объём необходимого кода и повышает уверенность в том, что в приложении не будет необработанной ошибки.

Пример:

```
import java.util.Scanner;
public class Main {
    public static void main(String args[]) {

        Scanner in = new Scanner(System.in);
        int num = in.nextInt();
        System.out.println(num*num);
        System.out.println("Program finished");

    }
}
```

```
C:\WINDOWS\system32\cmd.exe
C:\Users\User\Desktop\JAVA SE>java Main
dg
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at Main.main(Main.java:6)

C:\Users\User\Desktop\JAVA SE>java Main
bitlab
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at Main.main(Main.java:6)

C:\Users\User\Desktop\JAVA SE>
```

Если мы с консоли введем вместо цифры **int** какой-либо текст, то у нас программа выбросит ошибку. И далее программа не будет выполняться.

Пример:

```
import java.util.Scanner;
public class Main {

    public static void main(String args[]){

        Scanner in = new Scanner(System.in);
        int num;

        try{

            num = in.nextInt();

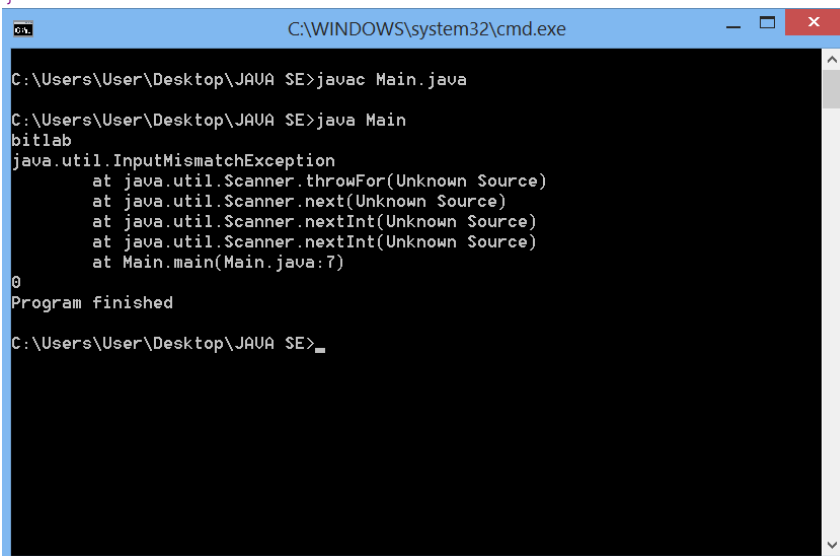
        } catch (Exception e) {

            num = 0;
            e.printStackTrace();

        }

        System.out.println(num*num);
        System.out.println("Program finished");

    }
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\Users\User\Desktop\JAVA SE>javac Main.java
C:\Users\User\Desktop\JAVA SE>java Main
bitlab
java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at Main.main(Main.java:7)
0
Program finished
C:\Users\User\Desktop\JAVA SE>_
```

Во втором примере, при запуске программы, даже если мы введем какой-либо текст, то все равно программа выполнится до конца, так как блок **try** отлавливает исключение или ошибку, затем сразу же выполняется блок **catch**. То есть, при введении текста вместо цифры, в итоге наша переменная **num** будет равна **0**. В конце концов программа дойдет до своего конца.

Исключение может появиться благодаря самой системе, либо вы сами можете создать его вручную. Системные исключения возникают при неправильном использовании языка Java или запрещённых приёмов доступа к системе. Ваши собственные исключения обрабатывают специфические ошибки вашей программы.

В **Java** есть два вида исключений: которые наследуются от **RuntimeException** и которые нет. Те, которые наследуются от **RuntimeException** их не обязательно отлавливать, это ошибки которые, могут возникнуть в результате обычных операций в **Java**. Но те, которые не наследуются от **RuntimeException**, их необходимо отлавливать, иначе компилятор откажется компилировать программу.

Для задания блока программного кода, который требуется защитить от исключений, используется ключевое слово **try**. Сразу же после **try**-блока помещается блок **catch**, задающий тип исключения которое вы хотите обрабатывать.

В некоторых случаях один и тот же блок программного кода может возбуждать исключения различных типов. Для того, чтобы обрабатывать подобные ситуации, Java позволяет использовать любое количество **catch**-разделов для **try**-блока. Наиболее специализированные классы исключений должны идти первыми, поскольку ни один подкласс не будет достигнут, если поставить его после суперкласса. Следующая программа перехватывает два различных типа исключений, причем за этими двумя специализированными обработчиками следует раздел **catch** общего назначения, перехватывающий все подклассы класса **Throwable**.

```
import java.util.Scanner;
import java.util.InputMismatchException;
public class Main {
    public static void main(String args[]) {

        Scanner in = new Scanner(System.in);
        int num;
        int denom;
        int div;
        try{
            num = in.nextInt();
            denom = in.nextInt();
            div = num/denom;

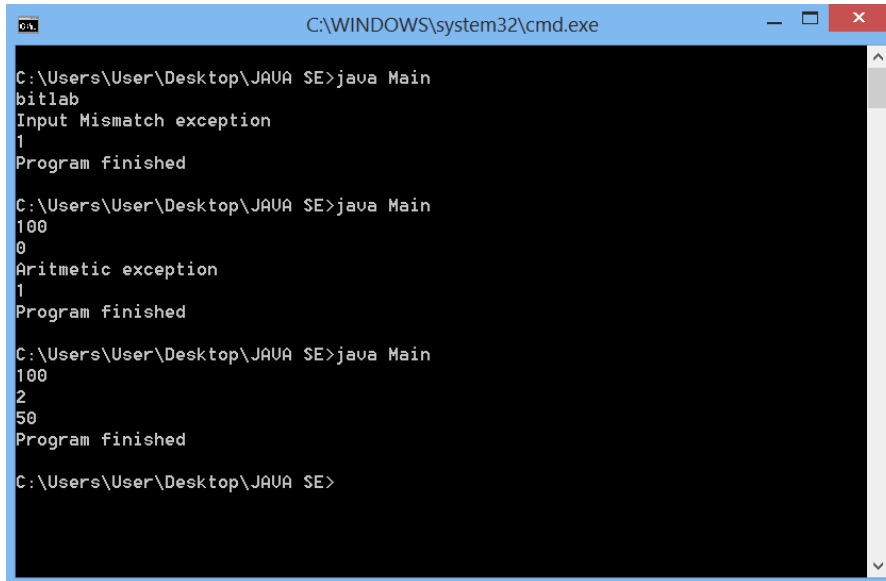
        } catch(ArithmeticException e) {

            System.out.println("Aritmetic exception");
            div = 1;

        } catch(InputMismatchException e) {

            div = 1;
            System.out.println("Input Mismatch exception");

        }
        System.out.println(div);
        System.out.println("Program finished");
    }
}
```



```

C:\WINDOWS\system32\cmd.exe
C:\Users\User\Desktop\JAVA SE>java Main
bitlab
Input Mismatch exception
1
Program finished

C:\Users\User\Desktop\JAVA SE>java Main
100
0
Arithmetic exception
1
Program finished

C:\Users\User\Desktop\JAVA SE>java Main
100
2
50
Program finished

C:\Users\User\Desktop\JAVA SE>

```

В данном примере мы определяем два исключения, **InputMismatchException** (пакет **java.util**) и **ArithmeticException**. Например, если мы делим число на 0, то у нас выбрасывается **ArithmeticException**. А если мы вместо цифры мы вводим строку, то у нас выбрасывается **InputMismatchException**. Одним словом, мы для каждого исключения мы можем прописать свой **catch**.

### Оператор finally

Ключевое слово **finally** создаёт блок кода, который будет выполнен после завершения блока **try/catch**, но перед кодом, следующим за ним. Блок будет выполнен, независимо от того, передано исключение или нет. Оператор **finally** не обязателен, однако каждый оператор **try** требует наличия либо **catch**, либо **finally**.

**Main.java**

```

import java.util.Scanner;
import java.util.InputMismatchException;

public class Main {

    public static void main(String args[]){

        Scanner in = new Scanner(System.in);
        int num;
        int denom;
        int div=0;

        try{

            num = in.nextInt();
            denom = in.nextInt();
            div = num/denom;

        }catch(ArithmeticException e){

            System.out.println("Aritmetic exception");
            div = 1;

        }catch(InputMismatchException e){

            div = 1;
            System.out.println("Input Mismatch exception");

        }finally{

            System.out.println(div);
            System.out.println("Program finished");

        }

    }

}

```

## Упражнения для разминки

### Задание 1.

Создайте класс **Book** с параметрами:

- int id;
- String name;
- String author;

Добавьте геттеры и сеттеры

+ String toString(); ///*///*данный метод возвращает все данные по книге

Создайте класс **Library** с параметрами:

- String name;
- String city;
- **Book** books[] = new **Book**[20];
- int index = 0;

Добавьте геттеры и сеттеры для каждого поля кроме **index** и массива **books**.

Создайте метод **addBook(Book book)** который добавляет книгу в список, и увеличивает значение **index** на 1.

Создайте метод **printBooks()**; ///*///* данный метод списком выводит данные по всем книгам

В ходе разработки и запуска, есть вероятность что человек может добавить в список не инициализированный объект класса **Book** (т.е. **Book b = null**), что приводит выбросу исключения **NullPointerException**, если мы хотим вывести его данные с помощью метода **toString()**. Ваша задача состоит в том, чтобы метод **printBooks()** самостоятельно отлавливала эту ошибку и вместо выброса из цикла программы, метод должен просто вывести сообщение : **"The book is empty"**.

При добавлении книги в список книг при помощи метода **addBook(Book book)**, вы должны отлавливать ошибку **ArrayIndexOutOfBoundsException**, которая выбрасывается при превышении размера массива. и вместо исключения, мы должны вывести на экран **"The library is full"**.

## Практические задачи

### Задание 1.

Создайте класс **Book** с параметрами:

- int id;
- String name;
- String author;

Добавьте геттеры и сеттеры

+ String toString(); ///*данный метод возвращает все данные по книге*

Создайте класс **Library** с параметрами:

- String name;
- String city;
- **Book** books[] = new **Book**[20];
- int index = 0;

Добавьте геттеры и сеттеры для каждого поля кроме **index** и массива **books**.

Создайте метод **addBook(Book book)** который добавляет книгу в список, и увеличивает значение **index** на 1.

Создайте метод **printBooks()**; ///*данный метод списком выводит данные по всем книгам*

В ходе разработки и запуска, есть вероятность что человек может добавить в список не инициализированный объект класса **Book** (т.е. **Book b = null**), что приводит выбросу исключения **NullPointerException**, если мы хотим вывести его данные с помощью метода **toString()**. Ваша задача состоит в том, чтобы метод **printBooks()** самостоятельно отлавливала эту ошибку и вместо выброса из цикла программы, метод должен просто вывести сообщение : **"The book is empty"**.

При добавлении книги в список книг при помощи метода **addBook(Book book)**, вы должны отлавливать ошибку **ArrayIndexOutOfBoundsException**, которая выбрасывается при превышении размера массива. и вместо исключения, мы должны вывести на экран **"The library is full"**.

**Задание 2.**

Создайте класс исключения: **DatabaseIncorrectException** которая наследует от класса **Exception**

Переопределите метод **printStackTrace()** которая должна вывести сообщение:

"Java Connection Exception at DatabaseDriver caused by incorrect data..."

Создайте класс **User** с параметрами:

- name

- surname

- login

- password

Геттеры и сеттеры

Создайте класс **Connection** с параметрами:

+ **User [] getUsersList()**

Данный метод возвращает массив класса **User**

Как список можете сами вручную возвращать массив из трех или четырех объектов класса **User**

Создайте класс **DatabaseDriverUtil** с параметрами:

+ **static Connection getConnection(String url, String dbName, String userName, String password);**

Данный метод возвращает объект класса **Connection**

Данный метод выбрасывает ошибку исключения **DatabaseIncorrectException** если:

Значение **url** не равно "https://bitlab.kz:3306/ "

Значение **dbName** не равно "maindatabase"

Значение **userName** не равно "root"

Значение **password** не равно "bitlab2017"

В основном классе **Main.java** попробуйте создать объект класса **Connection** методом **getConnection** и вывести список при помощи метода **getUsersList**.



## Коллекции и класс ArrayList

Для хранения большого количества однотипных данных могут использоваться массивы, но они не всегда являются идеальным решением.

Для работы с огромным количеством однотипных данных могут использоваться массивы. Но не во всех случаях массивы могут быть идеальным решением. Например, мы зададим длину массива заранее и в случае, если количество элементов заранее неизвестно, придется заранее побольше выделять память, и не факт, что мы все будем это использовать. А переопределять и увеличивать длину массива будет сложнее. Одним словом, массивы у нас статичные по размеру. В голову приходит мысль создать динамический массив, с "**резиновым**" размером, где мы можем при добавлении элементов "**по ходу**" увеличивать ее длину.

В программировании уже давным-давно применяются такие структуры данных как очередь, список, множество, стек и.т.д. **Коллекция** - это группа элементов с готовыми реализациями функции добавления, удаления и поиска определенного элемента. Причем, каждый тип коллекции отличается друг от друга механизмом работы операции, алгоритмами и принципами. Например, элементы стека упорядочены в последовательность, добавление нового элемента может происходить только в конец, и получить можно только элемент, находящийся. Очередь, напротив, позволяет получить лишь первый элемент. Другие коллекции, например, список позволяют получить элемент из любого места последовательности, а множество вообще не упорядочивает элементы и позволяет помимо добавления и удаления только узнать, содержится ли в нем данный элемент.

Список некоторых популярных реализованных классов библиотеки коллекции. (**Пакет: java.util**)

**ArrayList** - индексированная динамически расширяющая и сокращающая последовательность

**LinkedList** - упорядоченная последовательность, допускающая эффективную вставку и удаление на любой позиции

**HashSet** - неупорядоченная коллекция, исключающая дубликаты

**TreeSet** - отсортированное множество

**LinkedHashSet** - множество, запоминающее порядок ввода элементов

**HashMap** - структура данных для хранения связанных вместе пар "ключ-значение"

**TreeMap** - отображение с отсортированными ключами

**LinkedHashMap** - отображение с запоминанием порядка, в котором добавлялись элементы

Один из самых популярных и часто применяемых в разработке классов является **ArrayList**.

**ArrayList** является обобщенной коллекцией, которая наследует от класса **AbstractList** и применяет интерфейс **List**. Одним словом, **ArrayList** является простым списком, похожим на массив, но при этом количество элементов в нем не фиксировано.

Пример:

```
import java.util.ArrayList;

public class Main{

    public static void main(String[] args) {

        ArrayList<Integer> list = new ArrayList<Integer>();
        list.add(10);
        list.add(100);
        list.add(245);
        list.add(16);
        list.add(434);

        for(int i=0;i<list.size();i++){
            System.out.println(" Element in (" + i + ") = " + list.get(i));
        }

        System.out.println(list.toString());

    }

}
```

```
C:\WINDOWS\system32\cmd.exe

C:\Users\User\Desktop\JAVA SE>java Main
Element in (0) = 10
Element in (1) = 100
Element in (2) = 245
Element in (3) = 16
Element in (4) = 434

C:\Users\User\Desktop\JAVA SE>javac Main.java

C:\Users\User\Desktop\JAVA SE>java Main
Element in (0) = 10
Element in (1) = 100
Element in (2) = 245
Element in (3) = 16
Element in (4) = 434
[10, 100, 245, 16, 434]

C:\Users\User\Desktop\JAVA SE>
```

В данном примере, мы создаем объект класса **ArrayList** под названием **list**, у которого тип **Integer**. Мы собираемся туда хранить целостные числа, но обратите внимание, мы не использовали примитивные типы данных **int**, так как **int** не является классом, соответственно, мы всегда будем указывать классы вместо примитивных типов данных. (**Dobule**, **Boolean**, **Integer**, **Long** и.т.д.)

Метод **add(obj)** используется для добавления нового элемента в список. Изначально, размер нашего списка равна 0, затем при каждом добавлении элемента мы можем ее расширять. Метод **size()** возвращает текущий размер списка. Метод **get(int)** возвращает значение элемента в заданном индексе.

Метод **toString()** выводит содержимое коллекции в квадратных скобках, с разделением элементов запятыми.

**Некоторые основные методы интерфейса List, которые часто используются в ArrayList:**

**void add(int index, E obj):** добавляет в список по индексу index объект obj

**void sort(Comparator<? super E> comp):** сортирует список с помощью компаратора comp

**E get(int index):** возвращает объект из списка по индексу index

**int indexOf(Object obj):** возвращает индекс первого вхождения объекта obj в список. Если объект не найден, то возвращается -1

**int lastIndexOf(Object obj):** возвращает индекс последнего вхождения объекта obj в список. Если объект не найден, то возвращается -1

**E remove(int index):** удаляет объект из списка по индексу index, возвращая при этом удаленный объект

**E set(int index, E obj):** присваивает значение объекта obj элементу, который находится по индексу index

5

## Упражнения для разминки

### Задание 1

Создайте класс **Account** с параметрами:

- int id;
- String name;
- String surname;
- double balance;

Геттеры и сеттеры

+ String toString(); /// Данный метод все значения полей

Создайте класс **BankApplication** с параметрами:

- String name;
- ArrayList<Account> accounts;
- + void addAccount(Account a); // Добавляет новый объект Account
- + void removeAccount(int i); // Удаляет объект со списка под индексом i
- + Account getMaxAccount(); // Возвращает объект Account у кого самый высокий баланс
- + double getAverageBalance(); // Возвращает среднее значение баланса пользователей
- + double getTotalBalance(); // Возвращает сумму баланса пользователей
- + int totalAccounts(); // Возвращает количество пользователей
- + String getBankData(); // Возвращает имя банка, количество счетов, сумму баланса и среднее значение баланса

В основном классе Main, создайте 3 объекта **BankApplication**, добавьте туда по 10 новых счетов, с разным балансом.

Добавьте все эти банки в массив из банков **ArrayList<BankApplication> allBanks;**

Выведите данные по каждому банку, отсортировав их по среднему значению баланса, по убыванию.

## Чтение и запись текстовых файлов.

В **Java** есть два метода чтения и записи файлов: **бинарный** (минимальная единица записи - байт) и **строковый** (минимальная единица записи - символ).

Мы сконцентрируемся на строковом методе.

В **Java** есть классы **Reader/Writer** - основа строковой записи. **Reader** - считывает данные, **Writer** записывает данные. От этих классов наследуются классы **FileReader**, **FileWriter** - классы, которые считывают и записывают в файлы.

Но чтение и запись через эти классы не сильно удобна, для этого существует классы, преобразовывающие поток данных в удобное представление **BufferedReader** и **BufferedWriter**.

В итоге, чтобы создать объект для чтения с файла, у нас получается следующий код:

```
BufferedReader reader = new BufferedReader(new FileReader("text.txt"));
```

Класс **BufferedReader** считывает текст из символьного потока ввода, буферизируя прочитанные символы. Использование буфера увеличивает производительность чтения данных из потока.

Класс **BufferedReader** имеет следующие конструкторы:

```
BufferedReader(Reader in)
```

```
BufferedReader(Reader in, int size)
```

Где **Reader in** это поток ввода. Во втором конструкторе, кроме потока ввода, из которого производится чтение, можно определить размер буфера, в который будут считываться символы.

В первом примере мы создаем файл **file.txt** в котором храниться какой-либо текст.

Мы создаем программу где, наш **BufferedReader** считывает данные по одному символу

```
import java.io.*;

public class Main {

    public static void main(String[] args) {
        try{
            BufferedReader br =
                new BufferedReader (new FileReader("file.txt"));

            int symbol;

            while ((symbol=br.read())!=-1) {
                System.out.println((char) symbol);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Блок кода `while((symbol=br.read())!=-1)` проверяет значение считанного символа, и если оно равняется -1, то файл завершен. Далее, после того как мы присвоим значение символа в `int symbol`, мы с помощью "кастинга" преобразовываем `int` в символ `char`.

```
System.out.println((char) symbol);
```

Теперь, мы попытаемся считать построчно с помощью метода `readLine()`.

```
import java.io.*;

public class Main {

    public static void main(String[] args) {

        try{

            BufferedReader br=
                new BufferedReader (new FileReader("file.txt"));

            String s;
            while ((s=br.readLine())!=null) {

                System.out.println(s);
            }

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

Класс **BufferedWriter** записывает текст в поток, предварительно буферизируя записываемые символы.

```
import java.io.*;

public class FilesApp {

    public static void main(String[] args) {

        try{

            BufferedWriter writer =
                new BufferedWriter(new FileWriter("file.txt"))

            String message = "Hello BITLAB";
            writer.write(message);

        }catch(IOException ex) {

            e.printStackTrace();

        }

    }

}
```

## Упражнения для разминки

### Задание 1

Создайте класс **User** с параметрами:

- int id;
- String login;
- String password;

Геттеры и сеттеры

Перегрузите метод String toString(), которая возвращает все поля объекта

В основном классе **Main.java**, создайте два метода:

```
List<Users> getUsersList();  
void saveUsersList(List<Users> users);
```

Метод **getUsersList** возвращает динамический массив пользователей, которых мы загружаем из определенного файла **memory.txt**

Метод **saveUsersList** сохраняет динамический массив пользователей наш файл **memory.txt**

Используя эти методы, создайте мини приложение с интерфейсом:

```
PRESS [1] TO ADD USERS  
PRESS [2] TO LIST USERS  
PRESS [3] TO DELETE USERS  
PRESS [4] TO EXIT
```

## Практические задачи

### Задание 1

#### Мини приложение "Магазин"

Создайте класс **GoodItem** с параметрами:

- String name;
- int price;

Геттеры и сеттеры

Перегрузите метод String toString(), которая возвращает все поля объекта



Создайте класс **BuyHistory** с параметрами:

- String goodName;
- int goodPrice;
- Date buyTime; // java.util.Date;

Поле **buyTime** является типом **java.util.Date**, в котором я могу хранить время покупки. Для получения данного времени системы, нужно просто создать новый объект **new Date()**, в котором по умолчанию устанавливается время системы.

Данный класс нужен для записи продаж товара в магазине.

В основном классе **Main.java**, создайте два метода:

```
List <GoodItem> getGoodItemList();
void saveGoodItems(List<GoodItem> goodItems);

List <BuyHistory> getBuyHistoryList();
void saveBuyHistories(List<BuyHistory> buyHistory);
```

Метод **getGoodItemList** возвращает динамический массив товаров, которых мы загружаем из определенного файла **goods.txt**

Метод **saveBuyHistory** сохраняет динамический массив товаров в наш файл **goods.txt**

Метод **getGoodItemList** возвращает динамический массив истории покупок, которых мы загружаем из определенного файла **history.txt**

Метод **saveUsersList** сохраняет динамический массив истории покупок в наш файл **history.txt**

Создайте бизнес логику приложения (Все действия будут расписаны в методах)

Список методов:

```
addGoodItem(GoodItem g);
deleteGoodItem(int index);

addHistory(GoodItem g);
```

Создайте графический интерфейс, в котором мы можем опрaвлять товарами и их покупками, используя данные методы бизнес логики.

## Сериализация, десериализация

**Сериализация** - объекта это технология сохранения полной копии его объекта на которую он ссылается, при использовании потока ввода и вывода. Например, у нас есть какой-то список динамического массива **ArrayList** или какой-то объект, и мне нужно сохранить его состояние, и в будущем обратно вернуть и использовать его обратно.

Сериализация в **Java** предоставляет возможность для преобразования групп или отдельных объектов, в поток битов или массив байтов, для хранения или передаче по сети. Данный поток битов или массив байтов, можно преобразовать обратно в объекты **Java**. И все это происходит автоматически благодаря классам **ObjectInputStream** и **ObjectOutputStream**. Разработчик может решить применить эту функцию, путем реализации интерфейса **Serializable** при создании шаблона класса. Попробуем с простого примера. Создадим простой шаблон класса **Users**:

### *Users.java*

```
import java.io.Serializable;

public class User implements Serializable{

    private String name;
    private int number;

    public User(){
        this.name = "No Number";
        this.number = 0;
    }

    public User(String name, int number){
        this.name = name;
        this.number = number;
    }

    public void setName(String name){
        this.name = name;
    }
    public void setNumber(int number){
        this.number = number;
    }
    public String getName(){
        return this.name;
    }
    public int getNumber(){
        return this.number;
    }
}
```

**Main.java**

```

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

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

        ArrayList<User> usersList = new ArrayList<User>();

        usersList.add(new User("Ilyas", 123));
        usersList.add(new User("Timur", 321));
        usersList.add(new User("Aibek", 555));
        usersList.add(new User("Dauren", 777));

        try{

            ObjectOutputStream outStream =
                new ObjectOutputStream(new FileOutputStream("bitlab.dat"));

            outStream.writeObject(usersList);

            outStream.close();

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

    }
}

```

Данный пример показывает возможность записи объектов в поток, используя класс **ObjectOutputStream**.

С начала создается класс, который является сериализуемым, так как реализует интерфейс **Serializable**.

В программе мы создаем динамический массив и добавляем туда несколько объектов класса **User**. Мы можем создавать файл с любым расширением, например, **"bitlab.dat"**. Далее, мы записываем наш объект с помощью метода **writeObject(Object)**. После этого, мы закрываем поток с помощью метода **close()** и у нас в файловой системе создается файл **bitlab.dat** в котором и храниться наш сериализуемый объект.

Далее, нашей задачей является десериализовать наш объект динамического массива обратно.

**MainRead.java**

```

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

public class MainRead{

    public static void main(String[] args) {

        ArrayList<User> usersList = new ArrayList<User>();

        try{

            ObjectInputStream inStream =
                new ObjectInputStream(new FileInputStream("bitlab.dat"));

            usersList = (ArrayList<User>)inStream.readObject();

            inStream.close();

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

        for(User u : usersList){
            System.out.println(u.getName() + " - " + u.getNumber());
        }

    }
}

```

В данной программе, мы точно таким же образом ссылаемся на наш **bitlab.dat** файл, и десериализуем его с помощью класса **ObjectInputStream**. В этом классе есть метод **readObject()** который возвращает объектное значение. Для этого на придется применить метод "кастинга", то есть преобразования из одного типа в другой, где мы превращаем объект в **ArrayList**.

```

C:\WINDOWS\system32\cmd.exe

C:\Users\User\Desktop\BITLAB\JAVA SE\examples\serialization>java Main

C:\Users\User\Desktop\BITLAB\JAVA SE\examples\serialization>javac MainRead
error: Class names, 'MainRead', are only accepted if annotation processing is explicitly requested
1 error

C:\Users\User\Desktop\BITLAB\JAVA SE\examples\serialization>java MainRead
Ilyas - 123
Timur - 321
Aibek - 555
Dauren - 777

C:\Users\User\Desktop\BITLAB\JAVA SE\examples\serialization>

```

## Упражнения для разминки

### Задание 1

Создайте сериализуемый класс **Subject**:

- String name;
- int credits;

Геттеры и сеттеры

Переопределите метод **String toString()** где мы выводим все поля объекта

Создайте сериализуемый класс **Student**:

- String name;
- String surname;
- List<Subject> subjects = new ArrayList<Students>();

Геттеры и сеттеры

В главном классе создайте меню:

PRESS [1] TO ADD STUDENT

Name:

Surname:

PRESS [1] TO ADD SUBJECT

Name:

Credits:

PRESS [0] TO GO TO MAIN MENU

PRESS [2] TO LIST STUDENTS

PRESS [0] TO EXIT

Состояние общего массива `ArrayList<Students>` должен сохраняться в сериализуемый файл `memory.dat` и загружаться оттуда при обращении к списку.

## Практические задачи

### Задание 1

Создайте сериализуемый класс **Players**:

- String nickName;
- double rating;

Геттеры и сеттеры

Создайте сериализуемый класс **Game**:

- String gameName;
- String ipAddress;
- int port;
- List <Players> players = new ArrayList<Players>();

Геттеры и сеттеры

В главном классе создайте метод **void createGame()**, который создает новую игру.

Метод **void addPlayers(Player player)** должен добавить игроков в игру.

После запуска каждого метода, мы должны сохранять состояние класса в определенный файл **settings.data**

В главном классе создайте меню:

PRESS 1 TO CREATE GAME

Game name:

IP address:

PORT:

PRESS 2 TO ADD PLAYER TO GAME

Nickname:

Rating:

PRESS 3 TO PLAY GAME

PRESS 0 TO EXIT

При нажатии на 1, мы создаем игру.

При нажатии на 2, мы добавляем игрока

При нажатии на 3 мы запускаем игру, где просто выводим список игроков и данные об игре. Если мы не добавляли игроков, то мы выводим "No players".

## Потоки, многопоточность, интерфейс Runnable

Потоки в мы применяем при организации одновременных выполнении нескольких задач, каждой в независимом потоке. Потоки представляют собой экземпляры классов, каждый из которых запускается самостоятельно, автономно от главного потока выполнения программы. Обычно, когда мы запускаем программу, у нас строки кода выполняются последовательно, то есть пока не выполнится одна часть, мы не выполним вторую. А в многопоточной программе, мы по сути можем выполнять операции одновременно. Это дает нам возможность в выигрыше времени, скорости, распределении ресурсов, и независимости частей программы друг от друга.

В реальной жизни по сути мы живем в потоке. Если брать каждого человека за поток, то у нас в Мире около 7 миллиардов потоков, которые независимо друг от друга могут ходить, дышать, есть, спать, бегать или программировать.

В языке **Java** потоки выполняются двумя способами:

- 1 - Через класс **Thread**
- 2 - Через интерфейс **Runnable**

Рассмотрим первый способ через класс **Thread**.

### *ThreadSample.java*

```
public class ThreadSample extends Thread{

    String name;

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

    public void run(){
        try{

            for(int i=0;i<5;i++){
                System.out.println("Thread "+ name + " : " + i);
                Thread.sleep(500);
            }

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

Тут мы создаем класс **ThreadSample** которая наследует от класса **Thread**.

Далее, замещаем метод **run()** в котором прописываем цикл с задержкой 500 миллисекунд. После завершения цикла, мы выходим из метода **run()** тем самым завершая работу одного потока.

### **Main.java**

```
public class Main{

    public static void main(String[] args) {

        ThreadSample ts1 = new ThreadSample("t1");
        ThreadSample ts2 = new ThreadSample("t2");
        ThreadSample ts3 = new ThreadSample("t3");

        ts1.start();
        ts2.start();
        ts3.start();

    }

}
```

```
C:\WINDOWS\system32\cmd.exe
C:\Users\User\Desktop\JAVA SE>java Main
Thread t1 : 0
Thread t2 : 0
Thread t3 : 0
Thread t2 : 1
Thread t1 : 1
Thread t3 : 1
Thread t1 : 2
Thread t2 : 2
Thread t3 : 2
Thread t3 : 3
Thread t1 : 3
Thread t2 : 3
Thread t3 : 4
Thread t2 : 4
Thread t1 : 4
C:\Users\User\Desktop\JAVA SE>
```

В основном классе мы создаем 3 экземпляра потока класса **ThreadSample**, и запускаем их с помощью метода **start()**. Метод **start()** запускает наш поток, в котором выполняется метод **run()**. Получается метод **run()** каждого созданного потока выполняются параллельно и независимо друг от друга. В результате запуска мы видим, что каждый поток работая отдельно увеличивает значение **i** с задержкой на пол секунды.



Рассмотрим второй способ через интерфейс **Runnable**.

### ***ThreadRunnable.java***

```
public class ThreadRunnable implements Runnable{

    String name;

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

    public void run() {
        try{

            for(int i=0;i<5;i++){
                System.out.println("Thread "+ name + " : " + i);
                Thread.sleep(500);
            }

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

Мы создаем **ThreadRunnable** которая наследует от интерфейса **Runnable**. Далее, мы должны будем реализовать абстрактный метод **run()**, точно таким же способом как и в предыдущем примере.

### ***Main.java***

```
public class Main{

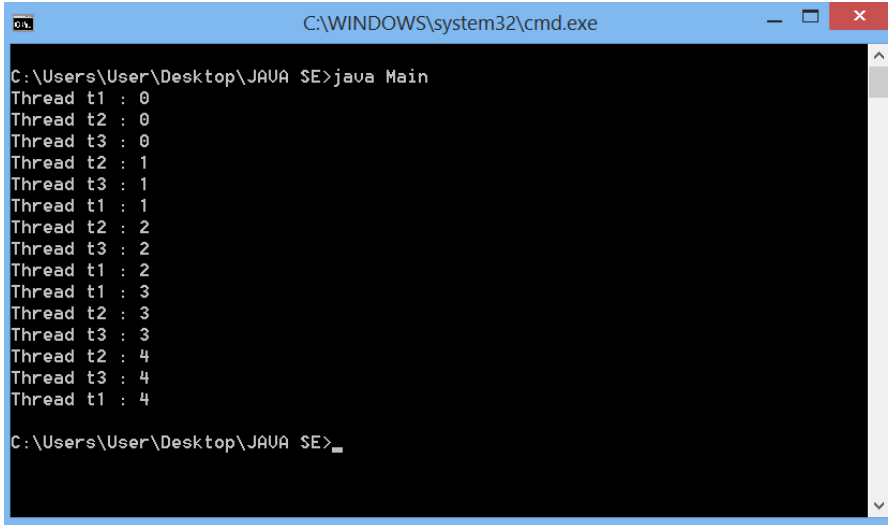
    public static void main(String[] args) {

        ThreadRunnable tr1 = new ThreadRunnable("t1");
        ThreadRunnable tr2 = new ThreadRunnable("t2");
        ThreadRunnable tr3 = new ThreadRunnable("t3");

        Thread t1 = new Thread(tr1);
        Thread t2 = new Thread(tr2);
        Thread t3 = new Thread(tr3);

        t1.start();
        t2.start();
        t3.start();

    }
}
```



```

C:\WINDOWS\system32\cmd.exe
C:\Users\User\Desktop\JAVUA SE>java Main
Thread t1 : 0
Thread t2 : 0
Thread t3 : 0
Thread t2 : 1
Thread t3 : 1
Thread t1 : 1
Thread t2 : 2
Thread t3 : 2
Thread t1 : 2
Thread t1 : 3
Thread t2 : 3
Thread t3 : 3
Thread t2 : 4
Thread t3 : 4
Thread t1 : 4
C:\Users\User\Desktop\JAVUA SE>

```

Данный пример таким же способом запускает потоки, в независимости друг от друга, но обратите внимание, что мы с начала создаем экземпляры нашего класса **ThreadRunnable**, а потом отдельно создаем экземпляры класса **Thread**, где мы переедаем через конструктор наши **ThreadRunnable** объекты. Разница между первым примером и вторым в том, что первый наследует от класса **Thread**, а второй наследует от интерфейса **Runnable**. Мы прекрасно знаем, что мы не можем наследовать от нескольких классов, но наследовать от нескольких интерфейсов мы можем, соответственно, второй подход дает нам больше гибкости, например, при создании игры или какого-либо процесса, где нам надо наследовать от одного класса и нескольких интерфейсов.

У нас есть состояние потока, когда он "спит". Это состояние мы получаем, когда вызываем метод **sleep(int)**, куда в аргументы мы передаем значение времени в миллисекундах.

Выход из программы или остановка потока происходит, когда потоки закончат свою работу. Но потоки можно приостановить досрочно, с помощью метода **interrupt()**;

В классе **Thread** есть два метода: **interrupt()** и **isInterrupted()**.

**interrupt()** - говорит потоку, что нужно завершаться.

**isInterrupted()** - проверяет, должен ли поток завершиться или нет.

Если поток "спит" после **Thread.sleep(int)**, вызвать **interrupt()**, то поток получит **InterruptedException**, но при вызове **isInterrupted()** мы получим **false**. Поэтому необходимо вызвать **interrupt()** повторно внутри **catch**.

## Упражнения для разминки

### Задание 1

Создайте класс **Operation** которая наследует от класса **Thread** с параметрами:

- String operationName;
- int operationTime; //время операции в секундах

Переопределите метод **run()** в котором мы запускаем наш поток, и выводим на экран:

Например operationTime = 4, operationName = "Add Client to Bank"

```
Operation "Add Client to Bank": started
Operation "Add Client to Bank": 1 second
Operation "Add Client to Bank": 2 second
Operation "Add Client to Bank": 3 second
Operation "Add Client to Bank": 4 second
Operation "Add Client to Bank": finished
```

То есть, каждую секунду мы выводим состояние операции на экран, тем самым показывая сколько времени у нас уходит на одно действие.

В итоге, в главном классе создайте 10 объектов операции и назначьте для них разное время. Запустите их одновременно.

### Задание 2

Создайте графическое окно в котором содержится кнопка "FIRE" и метка (JLabel) со значением #.

При нажатии на кнопку "FIRE" наша метка должна переместиться с одного места на другое плавным образом.

\*\*\* Подсказка:

При нажатии мы просто запускаем цикл, где например каждые 100 миллисекунд меняем локацию  
- setLocation(x,y) нашей метки

## Практические задачи

### Задание 1

Создайте класс **Car** которая наследует от класса **Thread** с параметрами:

- String name;
- int speed;
- double distance;

Переопределите метод **void run()** в котором мы пишем дистанцию, которую проехала машина за каждую секунду.

Например:

“Ferrari 1 is in 100 meters”

“Ferrari 1 is in 200 meters”

“Ferrari 1 is in 300 meters”

“Ferrari 1 is in 400 meters”

...

Машина остановиться после того как проедет определенную дистанцию, которую мы задаем в основном классе. Данная дистанция будет финишем, и вы ее зададите в виде статического целостного числа.

В основном классе, создайте 10 объектов класса разных машин и запустите их одновременно, задав дистанцию финишной линии. Ваши машины, после проезда финишной линии должны остановиться, и мы должны получить общий список гонки в таком виде:

**Place 1: Ferrari 1 - 10 seconds**

**Place 2: Mercedes 1 – 11 seconds**

**Place 3: Renault 2 – 13 seconds**

**Place 4: BMW 1 – 15 seconds**

....

**Place 10: Mercedes 3 – 28 seconds**

## Задание 2

Создайте мини приложения секундомера, где мы можем иметь три функции:

"Старт" - в котором мы запускаем наш отчет секундомера

"Пауза" - в котором мы входим в паузу

"Рестарт" - в котором мы останавливаемся и обратно возвращаемся в исходное время

Нарисуйте графический интерфейс для нашего приложения, где мы имеем три кнопки **JButton** и одну метку **JLabel**.

Формат времени – **milliseconds:seconds:minutes** (23:12:11)

Время будет в миллисекундах. 100 миллисекунд - 1 секунда, 60 секунд - 1 минута

## Сети, использование класса Socket

Одна из сильнейших сторон **Java** является сетевое программирование. Один из важнейших компонентов сетевого программирования – программирование на сокетах. Представьте себе сокет как абстракцию, соединение между двумя устройствами через кабель или розетку. У каждого устройства есть свой IP-адрес и порт. Именно по этим данным, мы можем создавать соединение. Сокет – это комбинация IP-адреса и порта. Сокет адрес дает возможность другим компьютерам в сети находить определенную программу, которая выполняется на определенном компьютере. Сокет адрес выглядит таким образом 192.168.1.1:80, где 192.168.1.1– IP-адрес и 80 – порт.

В **Java** вы создаете сокет, чтобы создать соединение с другой машиной, затем вы получаете **InputStream** и **OutputStream** из сокета по которым вы и начинаете передавать данные, как объект потока ввода/вывода. Существует 2 класса сокетов:

1. **ServerSocket**
2. **Socket**

**ServerSocket** - находится на сервере. После, клиент создает подключение к серверу, и после подключения клиента, **ServerSocket** возвращает объект **Socket** при помощи метода **accept()**, через которого может происходить коммуникация на стороне сервера.

Давайте рассмотрим простой пример:

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

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

        Scanner in = new Scanner(System.in);
        try{

            Socket socket = new Socket("127.0.0.1", 1989);

            ObjectOutputStream outputStream =
                new ObjectOutputStream(socket.getOutputStream());

            while(true) {
                outputStream.writeObject(in.next());
            }

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

```

import java.net.*;
import java.io.*;
import java.util.*;

public class Server{

    public static void main(String[] args) {

        try{

            ServerSocket server = new ServerSocket(1989);

            Socket socket = server.accept();

            System.out.println("Client connected");

            ObjectInputStream inStream =
                new ObjectInputStream(socket.getInputStream());

            String message = "";

            while((message = (String)inStream.readObject()) != null) {

                System.out.println(message);

            }

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

    }
}

```

В данном примере клиент подключается к серверу через класс `Socket`. У сервера есть свой IP-Адрес. На данный момент, она равняется **127.0.0.1**, то есть локальный адрес нашего компьютера. Порт у сервера – **1989**. После того как клиент подключается к серверу, мы создаем потоки **ObjectInputStream** и **ObjectOutputStream** для сервера и клиента соответственно. Далее идет классическое отправление данных через потоки, с помощью метода **writeObject()** и считывание через метод **readObject()**.

### Использование потоков (Thread) для мульти-клиентного сервера.

В первом примере, мы создаем один сервер и один клиент, в котором клиент отправляет сообщение серверу. Но возникает вопрос, если мы хотим, чтобы несколько клиентов одновременно могли подключаться к серверу и отправлять сообщения. Для этого мы должны использовать потоки (**Threads**).

Мы создаем дополнительный класс **ClientHandler** которая наследует от класса **Thread**. Одним словом, идея такова что процесс обмена данных будет реализована в методе **run()**, которая может работать параллельно.

#### *ClientHandler.java*

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

public class ClientHandler extends Thread{

    private Socket socket;

    public ClientHandler(Socket socket) {

        this.socket = socket;
    }

    public void run() {

        try{

            ObjectInputStream inStream =
                new ObjectInputStream(socket.getInputStream());

            String message = "";

            while ( (message = (String) inStream.readObject()) != null) {

                System.out.println(message);

            }

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}
```

Теперь наш сервер будет способен обрабатывать несколько подключении, передав всю работу специальному поточному классу **ClientHandler**.



```

import java.net.*;

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

public class Server{

    public static void main(String[] args) {

        try{

            ServerSocket server = new ServerSocket(1989);

            while(true) {

                Socket socket = server.accept();

                ClientHandler clientHandler = new ClientHandler(socket);

                clientHandler.start();

            }

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

    }

}

```

Клиентовская сторона останется точно таким же, так как она не требует несколько обработок подряд. В целом, на ваше усмотрение можно создавать мини приложения для чата, используя библиотеку графического интерфейса **Swing**.