

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«Иркутский государственный университет путей сообщения»

Сибирский колледж транспорта и строительства

## МЕТОДИЧЕСКИЕ УКАЗАНИЯ

### К ПРАКТИЧЕСКИМ РАБОТАМ

ПМ. 02 Проектирование управляющих программ компьютерных систем и  
комплексов

МДК.02.03. Разработка прикладных приложений

по специальности

09.02.01 Компьютерные системы и комплексы

базовая подготовка

среднего профессионального образования

Иркутск 2023

РАССМОТРЕНО:

Цикловой методической  
комиссией 09.02.01 Компьютерные  
системы и комплексы

Протокол № 9

«26» мая 2023 г.

Председатель ЦМК: Арефьева Н.В.

Разработчик: Саквенко Т.В., преподаватель первой категории Сибирского колледжа транспорта и строительства ФГБОУ ВО «Иркутский государственный университет путей сообщения».

## **Содержание**

Практическая работа № 1. Создание учебного проекта по индивидуальным заданиям.	4
Практическая работа № 2. Методы без параметров и с параметрами в учебном проекте. ....	8
Практическая работа № 3. Оператор SWITCH, цикл FOR, цикл WHILE в учебном проекте. ....	19
Практическая работа № 4. Объявление и обработка одномерного массива. ....	32
Практическая работа № 5. Объявление и обработка двумерного массива. ....	40
Практическая работа № 6. Ввод массивов. ....	44
Практическая работа № 7. Обработка строк: поиск, сравнение. ....	47
Практическая работа № 8. Обработка символов. ....	56
Практическая работа № 9. Включение класса в учебный проект. ....	59
Практическая работа № 10. Разработка приложения в соответствии с принципами объектно-ориентированного программирования по индивидуальным заданиям (начальный этап). ....	62
Практическая работа № 11. Обработка потоков в учебном проекте. ....	67
Практическая работа № 12. Обработка файлов в учебном проекте. ....	74
Практическая работа № 13. Доработка приложения с учетом обработки файлов и потоков. ....	84
Практическая работа № 14. Использование коллекций в учебном проекте .....	91
Практическая работа № 15. Реализация параметризованного интерфейса в учебном проекте. ....	99
Практическая работа № 16. Создание форм. ....	104
Практическая работа № 17. Добавление кнопок, меток, текстовых полей. ....	110
Практическая работа № 18. Интерфейс формы и размещение компонентов. ....	115
Практическая работа № 19. Разработка кода обработки событий в учебном проекте. ....	120
Практическая работа № 20. Разработка учебного проекта в Android Studio (начальный этап). ....	132
Практическая работа № 21. Модификация учебного проекта в Android Studio. ....	140
Практическая работа № 22. Разработка меню в учебном проекте. ....	145

## **Практическая работа № 1. Создание учебного проекта по индивидуальным заданиям.**

### **Задача 1**

Дни недели.

Передать на вход программы число от 1 до 7 в качестве аргумента.

Если число равно 1, выводим на консоль “Понедельник”, 2 –”Вторник” и так далее. Если 6 или 7 – “Выходной”.

Используем конструкцию if-else-if.

```
public class IfElseIfDemo {  
    public static void main(String[] args) {  
        if ("1".equals(args[0])) {  
            System.out.println("Monday");  
        } else if ("2".equals(args[0])) {  
            System.out.println("Tue");  
        } else if ("3".equals(args[0])) {  
            System.out.println("Wed");  
        } else if ("4".equals(args[0])) {  
            System.out.println("Thu");  
        } else if ("5".equals(args[0])) {  
            System.out.println("Fri");  
        } else if ("6".equals(args[0]) || "7".equals(args[0])) {  
            System.out.println("Weekend");  
        } else {  
            System.out.println("Error");  
        }  
    }  
}
```

### **Задача 2**

Калькулятор

Создать метод принимающий на вход две переменные типа int.

Метод вычисляет их сумму и возвращает значение.

Вызвать этот метод из метода main.

Сделать методы для вычисления разницы, произведения и деления.

```

import java.util.Scanner;
public class Calculator {
    public static void main(String[] args) {
        double num1;
        double num2;
        double ans;
        char op;
        Scanner reader = new Scanner(System.in);
        System.out.print("Enter two numbers: ");
        num1 = reader.nextDouble();
        num2 = reader.nextDouble();
        System.out.print("\nEnter an operator (+, -, *, /): ");
        op = reader.next().charAt(0);
        switch(op) {
            case '+': ans = num1 + num2;
                break;
            case '-': ans = num1 - num2;
                break;
            case '*': ans = num1 * num2;
                break;
            case '/': ans = num1 / num2;
                break;
            default: System.out.printf("Error! Enter correct operator");
                return;
        }
        System.out.print("\nThe result is given as follows:\n");
        System.out.printf(num1 + " " + op + " " + num2 + " = " + ans);
    }
}

```

### Задача 3

Создать программу где пользователю предлагается ввести количество процентов заряда батареи на его смартфоне. В случае, если осталось менее 10 процентов заряда, программа предупредит пользователя о необходимости зарядить смартфон.

```

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

```

```

    System.out.print("Сколько процентов заряда батареи осталось на вашем
смартфоне?");
    int a = scanner.nextInt();

```

```

    if (a < 10) {

```

```
        System.out.println("Осталось менее 10 процентов, подключите ваш  
        смартфон к зарядному устройству");  
    }  
}
```

#### **Задача 4**

Проверка четности числа

Создать программу, которая будет сообщать, является ли целое число, введённое пользователем, чётным или нет.

```
public class isEven {  
    public static void main(String args[]) {  
        int n = 9;  
        System.out.print("Число " + n + " ");  
        if( n % 2 == 0) {  
            System.out.println("четное");  
        } else {  
            System.out.println("нечетное");  
        }  
    }  
}
```

#### **Задача 5**

Меньшее по модулю число

Создать программу, которая будет выводить на экран меньшее по модулю из трёх введённых пользователем вещественных чисел.

Для вычисления модуля используйте тернарную условную операцию.

Для ввода чисел используем класс Scanner.

```
public class Swi {
```

```
    public static void main(String...args) throws InterruptedException{  
        Swi obj = new Swi();  
        obj.start();
```

```
}
```

```
public void start(){
```

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

```
    System.out.println("Введите 3 вещественных числа:");
```

```
    float a = scan.nextFloat();
```

```
    float b = scan.nextFloat();
```

```
    float c = scan.nextFloat();
```

```
    a = Math.abs(a);
```

```
    b = Math.abs(b);
```

```
    c = Math.abs(c);
```

```
    System.out.println(a + " "+ b+ " "+c);
```

```
    float tmp = Math.min(a,b);
```

```
    float min = Math.min(tmp, c);
```

```
    System.out.println("Наименьшее вещественное число по модулю:\n" +  
min);
```

```
}
```

```
}
```

## **Практическая работа № 2. Методы без параметров и с параметрами в учебном проекте.**

### **1 Цель занятия**

Научиться создавать методы, которые принимают на вход параметры.

### **2 Теоретический материал**

Как известно, классы могут содержать данные и методы. В Java данные еще называют переменные экземпляра или объекты. Методы еще называются подпрограммы или функции. Метод – это подпрограмма обозначенная именем. по этому имени можно вызывать код подпрограммы (код метода). Вызов метода по имени может осуществляться из другого метода многократно в зависимости от выполняемой задачи. При каждом вызове имени метода вместо этого имени выполняется код метода.

Методы в классе – это именованный программный код, который определяет некоторую работу над данными класса. Класс может содержать любое количество методов. Кроме того, класс может быть реализован без методов. Чаще всего методы класса обрабатывают данные (переменные экземпляров) класса.

Параметры методов. Возврат значений из методов

Методы могут получать параметры. По количеству получаемых параметров методы делятся на:

методы, получающие параметры (методы с параметрами или параметризованные методы). В этом случае, перечень параметров указывается в скобках после имени метода;

методы без параметров. В этом случае после имени метода указываются пустые круглые скобки ( ).

Более подробно о параметрах методов описывается в следующей теме:

Передача параметров в методах класса. Передача переменных примитивных типов и объектов в метод в качестве параметра

Метод может возвращать значение. Перед именем метода указывается тип значения, которое метод возвращает. С точки зрения возвращения значения из метода, методы могут быть двух видов:

методы, возвращающие значение. В этом случае метод должен содержать оператор `return` с кодом возврата;

методы, которые не возвращают значения. Перед именем метода, который не возвращает значение, указывается ключевое слово `void`. В этом случае метод не обязательно должен содержать оператор `return`. Если такой метод содержит оператор `return`, то после этого оператора ничего уже не указывается.

Общая форма объявления метода возвращающего значение

Метод может возвращать значение. Такой метод может использоваться в выражениях. Так, в языке программирования Паскаль, методы, которые возвращают значение имеют название функция.

Общая форма объявления метода, который возвращает значение и получает параметры, имеет следующий вид:

```
return_type MethodName(type1 param1, type2 param2, ..., typeN paramN) {  
    // тело метода - программный код метода  
    // ...  
}
```

где

`MethodName` – имя метода;

`return_type` – тип, возвращаемый методом. Это может быть примитивный (простой) тип, или тип (класс) объявленный дополнительно в программе;

`param1, param2, paramN` – имена параметров, которые имеют соответственно типы `type1, type2, typeN`. Количество параметров у методов может быть произвольным. Кроме того, параметры могут отсутствовать. Если метод не содержит параметров, то вместо списка параметров указываются пустые скобки ( ).

Если метод возвращает значение и не получает параметров, то общая форма такого метода следующая:

```
return_type MethodName() {  
    // тело метода  
    // ...  
}
```

Методы, которые не возвращают значение. Общая форма. Ключевое слово void

Если метод не возвращает значения, то перед его объявлением указывается ключевое слово void. Такой метод не может использоваться в выражениях. Так, в языке программирования Паскаль, методы, которые не возвращают значение имеют название процедуры.

Общая форма объявления метода, который не возвращает значения но получает параметры имеет вид:

```
void MethodName(type1 param1, type2 param2, ..., type param) {  
    // тело метода - программный код метода  
    // ...  
}
```

где

void – зарезервированное ключевое слово (спецификатор), указывающий на то, что метод не возвращает никакого значения;

MethodName – имя метода;

param1, param2, paramN – имена параметров, которые имеют соответственно типы type1, type2, typeN.

Если метод не возвращает значения и не получает параметров, то общая форма такого метода следующая:

```
void MethodName() {  
    // тело метода  
    // ...  
}
```

Примеры методов, которые не получают параметры и не возвращают значения

Пример 1. Демонстрируется реализация метода ShowInfo() в классе DemoMethods, который выводит информацию о Web-ресурсе.

```
class DemoMethods {  
    // метод, который выводит общую информацию о данном Web-ресурсе  
    void ShowInfo() {  
        System.out.println("Website Information:");  
        System.out.println("Main page: https://www.bestprog.net");  
        System.out.println("Programming: theory and practice");  
        System.out.println("The website was created on December 08, 2015");  
    }  
}
```

Использование метода ShowInfo() класса DemoMethods в другом коде может быть следующим:

```
// использование метода ShowInfo() класса DemoMethods  
DemoMethods dm = new DemoMethods(); // создать экземпляр класса  
dm.ShowInfo(); // вызвать метод
```

В результате вызова метода в консоли будет выведен следующий результат:

Website Information:

Main page: https://www.bestprog.net

Programming: theory and practice

The website was created on December 08, 2015

Примеры методов, которые не получают параметров и возвращают значение

**Пример 1.** В классе MathConstants объявляется метод Pi(), возвращающий значение числа  $\pi$ . Метод Pi() возвращает значение типа double с помощью оператора return.

```
// класс, который содержит метод без параметров, который возвращает значение
```

```
class MathConstants {  
    // метод, который возвращает число Пи  
    double Pi() {  
        return 3.1415926535;  
    }  
}
```

Ниже демонстрируется использование метода Pi() для вычисления площади круга:

```
// использование метода Pi() класса MathConstants  
MathConstants mc = new MathConstants();  
double area, radius;  
radius = 2; // вычислить площадь круга радиуса 2  
area = mc.Pi() * radius * radius; // area = 12.566370614
```

**Пример 2.** Объявляется класс Circle, содержащий внутренние данные (x, y, radius) и методы Area(), LenOrigin() которые выполняют над этими данными вычисления

```
// класс, реализующий окружность  
class Circle {  
    double x, y; // координаты центра окружности  
    double radius; // радиус окружности
```

```
// метод, возвращающий площадь круга радиуса radius
double Area() {
    final double pi = 3.141592f; // константа Пи
    return pi * radius * radius;
}

// метод, возвращающий расстояние от центра окружности (x,y) до начала
координат (0; 0)
double LenOrigin() {
    double len = Math.sqrt(x*x + y*y);
    return len;
}
```

Использование класса Circle может быть следующим

```
// использование методов класса Circle
Circle cr = new Circle();
double area, length;

// заполнить экземпляр класса значениями
cr.x = 2.5;
cr.y = 4.2;
cr.radius = 3;

// вызов методов
area = cr.Area(); // площадь окружности, area = 28.274328231811523
length = cr.LenOrigin(); // length = 4.887739763939975
```

Примеры методов, которые получают параметры и не возвращают значения

**Пример 1.** В примере объявляется класс Month, реализующий месяц года. В классе реализован метод SetMonth(), получающий входным параметром номер месяца. Метод SetMonth() не возвращает значения. В зависимости от номера месяца метод устанавливает следующие внутренние переменные:

номер месяца month;  
количество дней в месяце days;  
название месяца name.

Объявление класса Month следующее:

```
// класс, реализующий месяц года
class Month {
    int month; // номер месяца в году 1..12
    int days; // максимальное количество дней в месяце
    String name; // название месяца

    // метод, который устанавливает новый месяц по его номеру
    void SetMonth(int _month) {
        // проверка, корректны ли данные
        if ((_month<1)||(_month>12))
            return;

        // номер месяца
        month = _month;

        // количество дней в месяце
        switch (_month) {
            case 2:
                days = 29;
                break;
```

```
case 4: case 6: case 9: case 11:  
    days = 30;  
    break;  
default:  
    days = 31;  
}
```

```
// название месяца  
switch (_month) {  
    case 1: name = "Jan"; break;  
    case 2: name = "Feb"; break;  
    case 3: name = "Mar"; break;  
    case 4: name = "Apr"; break;  
    case 5: name = "May"; break;  
    case 6: name = "Jun"; break;  
    case 7: name = "Jul"; break;  
    case 8: name = "Aug"; break;  
    case 9: name = "Sep"; break;  
    case 10: name = "Oct"; break;  
    case 11: name = "Nov"; break;  
    case 12: name = "Dec"; break;  
    default: name = "";  
}  
}  
}
```

Демонстрация использования метода следующая:

```
// демонстрация метода, который получает параметры и не возвращает  
значения
```

```
Month mn = new Month(); // создать экземпляр класса
mn.SetMonth(5); // установить данные для месяца "Май"

// проверка
int days;
String name;
days = mn.days; // days = 31
name = mn.name; // name = "May"
```

**Пример 2.** В классе Circle объявляется перегруженный метод SetData(), который устанавливает значение внутренних переменных экземпляра. Метод не возвращает значения – перед именем метода указывается ключевое слово void. Метод SetData() имеет две реализации:

реализация SetData(double, double, double). Метод получает 3 параметра типа double которые устанавливают значения соответствующих внутренних переменных класса;

реализация SetData(Circle). В этом случае метод получает экземпляр такого же класса в качестве источника данных.

```
// класс, реализующий окружность
class Circle {

    double x, y; // координаты центра окружности
    double radius; // радиус окружности

    // метод, который получает 3 параметра типа double
    // метод не возвращает значения (void)
    void SetData(double x, double y, double radius) {
        this.x = x;
        this.y = y;
        this.radius = radius;
    }
}
```

```
// метод, который получает экземпляр типа Circle  
// метод не возвращает значения  
void SetData(Circle cr) {  
    x = cr.x;  
    y = cr.y;  
    radius = cr.radius;  
}  
}
```

### Задача 1

Создать пользовательскую функцию, которая вернёт нам значение переменной, а также присвоит это значение переменной в основном коде.

```
public class Main {
```

```
    public static int function1(){  
        int a = 5;  
        return a;  
    }
```

```
    public static void function2(){      System.out.println("Записывайтесь на  
        курсы Java!");  
    }
```

```
    public static void main(String[] args) { //основной блок программы
```

```
        int b = function1(); //присваиваем переменной значение, которое  
        //возвратит первая функция
```

```
        System.out.println(b); //выводим на экран значение нашей переменной
```

```
        function2(); //вызываем по имени вторую функцию
```

```
    }  
}  
}
```

## Задача 2

Создать функцию с параметрами, а потом вызвать её в основном коде с помощью аргументов. Возвести переменную в определённую степень, а потом вернуть значение в переменную.

```
public class Main {  
  
    public static int function3(int var_1, int var_2){ //функция с параметрами  
        int a = 1; //создаём переменную, в которой будет находиться расчётное  
        //значение  
        for(int i = 0; i < var_2; i++){ //используем цикл для возведения в степень  
            a = a * var_1; //перемножаем новое значение со старым, возводя тем  
            //самым в степень  
        }  
        return a; //возвращаем посчитанное значение  
    }  
  
    public static void main(String[] args) { //основной блок программы  
        int result = function3(5,3); //вызываем функцию, передав 2 аргумента  
        // (возводим 5 в третью степень)  
        System.out.println(result); //выводим значение переменной  
    }  
}
```

## **Практическая работа № 3. Оператор SWITCH, цикл FOR, цикл WHILE в учебном проекте.**

### **1 Цель занятия**

Научиться создавать программы с использованием оператора SWITCH, цикла FOR, цикла WHILE.

### **2 Теоретический материал**

#### **Оператор SWITCH**

Оператор switch в Java- это условный оператор, который дает возможность сравнивать переменную со списком значений.

Рассмотрим общую форму написания оператора switch:

```
switch(выражение) {  
    case значение1:  
        // Блок кода 1  
        break;  
    case значение2:  
        // Блок кода 2  
        break;  
    ...  
    case значениеN:  
        // Блок кода N  
        break;  
    default :  
        // Блок кода для default  
}
```

В круглых скобках мы указываем выражение, значение которого будет сравниваться со списком значений, перечисленных после ключевого слова

case. Если выражение совпадает со значением1, то выполняется блок кода, указанный после значение1. Если выражение не совпадает ни с одним из значений, то выполняется блок кода для default. Этот блок кода является необязательным - можно написать оператор switch и без него.

Выражение в switch должно иметь тип char, byte, short, int, enum (начиная с Java 6) или String(начиная с Java 7). Использование любого другого типа, например float, приведет к ошибке компиляции.

Оператор switch может только проверять на равенство. Такие операторы как  $\geq$ ,  $\leq$  недопустимы.

Рассмотрим первый пример - задается число от 1 до 3. Если 1 - включается зеленый цвет светофора, 2 - желтый и 3 - красный. Если любое другое число, то выполнится блок кода default:

```
public class TrafficLight {  
    public static void main(String[] args) {  
        int x = 3;  
        switch (x) {  
            case 1:  
                System.out.println("Зеленый");  
                break;  
            case 2:  
                System.out.println("Желтый");  
                break;  
            case 3:  
                System.out.println("Красный");  
                break;  
            default:  
                System.out.println("Неправильно введено число");  
        }  
    }  
}
```

```
        break;  
    }  
}  
}
```

Для нескольких значений case можно указывать один блок, как показано в следующем примере:

```
public class SwitchDemo1 {  
    public static void main(String[] args) {  
        int month = 4;  
        String season;  
        switch (month) {  
            case 12:  
            case 1:  
            case 2:  
                season = "Winter";  
                break;  
            case 3:  
            case 4:  
            case 5:  
                season = "Spring";  
                break;  
            case 6:  
            case 7:  
            case 8:  
                season = "Summer";  
                break;  
            case 9:  
            case 10:
```

```
        case 11:
            season = "Autumn";
            break;
        default:
            season = "Not a Month";
        }
    System.out.println("April is in the " + season + ".");
}
}
```

Если выражение в switch имеет тип меньший чем int (byte, short или char), case константа должна соответствовать этому типу. Например следующий пример не откомпилируется:

```
byte number = 2;
switch (number) {
    case 13:
    case 129://compiler error
}
```

Также нельзя использовать несколько case констант с одним и тем же значением. Следующий блок кода не откомпилируется:

```
int number = 90;
switch (number) {
    case 50:
        System.out.println("50");
    case 50:
        System.out.println("50"); //compile error
    case 140:
        System.out.println("140");
    default:
```

```
        System.out.println("default");
    }
```

Выражение в switch может содержать Character, Byte, Short и Integer значения:

```
switch (new Integer(9)) {
    case 9:
        System.out.println("9");
}
```

Case константы просматриваются сверху вниз, и первая совпавшая case константа в switch является точкой входа. Соответствующий ей блок и все последующие блоки будут выполняться. Например:

```
public class SwitchDemo2 {
    public static void main(String[] args) {
        String str = "potato";
        switch (str) {
            case "tomato":
                System.out.print("tomato ");
            case "potato":
                System.out.print("potato ");
            case "cucumber":
                System.out.print("cucumber ");
            default:
                System.out.println("any");
        }
    }
}
```

Результат выполнения программы:

potato cucumber any

Для того чтобы выйти из switch выражения, используйте ключевое слово break. Например, изменим предыдущий пример, чтобы выводилось только одно слово:

```
Vegetable p = Vegetable.potato;  
switch (p) {  
    case tomato:  
        System.out.print("tomato ");  
        break;  
    case potato:  
        System.out.print("potato ");  
        break;  
    case cucumber:  
        System.out.print("cucumber ");  
        break;  
    default:  
        System.out.println("any");  
}
```

Результат будет:

potato

Секция default в switch обрабатывает все значения, которые не указаны явно в одной из case секций. Секция default может размещаться в любом месте, необязательно в конце:

```
public class SwitchDemo3 {  
    public static void main(String[] args) {  
        int z = 8;  
        switch (z) {
```

```
case 1:  
    System.out.println("Fall to one");  
default:  
    System.out.println("default");  
case 3:  
    System.out.println("Fall to three");  
case 4:  
    System.out.println("Fall to four");  
}  
}  
}
```

Начиная с Java 7 можно использовать объекты типа String в switch.  
Например:

```
String exam = null;  
switch (exam) {  
    case "OCPJP 7":  
        System.out.print(exam + ": 1Z0-804");  
        break;  
    case "OCPJP 8":  
        System.out.print(exam + ": 1Z0-809");  
        break;  
    default:  
        System.out.print(exam + ": ----");  
        break;  
}
```

## Цикл for

Цикл for используется в случаях, когда необходимо несколько раз выполнить блок кода и точно известно количество этих повторений.

Общая форма цикла for:

```
for(инициализация; условие; итерация){/*операторы*/ }
```

Пример цикла for:

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

В первом параметре обычно выбирают какую-то переменную, с помощью которой будет подсчитываться количество повторений цикла. Её называют счетчиком. Эту переменную обычно называют *i*. Счётчику задают некоторое начальное значение. Для цикла for не рекомендуется в цикле изменять индекс цикла.

Во втором параметре указывают некоторое ограничение на счётчик (указывают, до какого значения он будет изменяться).

В третьем параметре указывают выражение, изменяющее счётчик после каждого шага цикла. Обычно это инкремент или декремент, но можно использовать любое выражение, где счётчику будет присваиваться некоторое новое значение.

Перед каждым шагом цикла (но после инициализации) проверяется условие повторения, если оно истинно, то в очередной раз выполняется тело цикла. При этом, тело цикла может не выполниться ни разу, если условие будет ложным в момент первой же проверки.

В первом параметре можно через запятую инициализировать несколько переменных, как это показано в следующем примере:

```
public class Comma {  
    public static void main(String[] args) {  
        for (int i = 1, j = 4; i < j; i++, j--) {  
            System.out.println("i = " + i);  
            System.out.println("j = " + j);  
        }  
    }  
}
```

### Цикл while

Цикл while - это цикл типа "пока".

Он используется в случае, когда один фрагмент кода должен выполняться, пока выполняется какое-то условие.

Рассмотрим синтаксис на следующем примере.

После ключевого слова while в круглых скобочках указывается выражение, которое должно возвращать значение типа boolean - это условие цикла. Далее в фигурных скобочках указываем тело цикла - код, который будет повторяться, пока условие цикла возвращает значение true. Пока выполняется условие  $n > 0$ , будет выполняться `System.out.println`. Мы видим, что переменная n меняется внутри тела цикла, что позволяет в конце концов выйти из него.

```
public class While1 {  
    public static void main(String[] args) {  
        int n = 10;  
        while (n > 0) {
```

```
        System.out.println("Тик " + n--);  
    }  
}  
}
```

Условие цикла while проверяется перед выполнением тела цикла. В этом главное отличие цикла while от do-while.

Следующий пример показывает вариант использования цикла while без тела. Даны два числа 100 и 200 и необходимо найти середину между ними. Значение i увеличивается на 1 каждую итерацию цикла, а j уменьшается до тех пор, пока они не станут равны. Изменение i и j происходит в условии цикла, поэтому тело цикла не нужно. Вместо тела просто ставится точка с запятой.

```
public class NoBody {  
    public static void main(String[] args) {  
        int i = 100;  
        int j = 200;      // найти середину между i и j  
        while (++i < --j); // цикл без тела  
        System.out.println("Середина: " + i);  
    }  
}
```

Цикл while может применяться для организации бесконечных циклов в виде while(true):

```
public class EndlessLoop {  
    public static void main(String[] args) {  
        int i = 0;  
        while (true) {  
            System.out.println(i++);  
        }  
    }  
}
```

```
}
```

## Задача 1

Напишите программу, которая выводит на экран числа от 1 до 10.

```
public class NumbersFromOneToTen {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.print(i + " ");
        }
    }
}
```

## Задача 2

Напишите программу, которая находит сумму всех чисел от 1 до 100.

```
public class SumOfNumbers {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 1; i <= 100; i++) {
            sum += i;
        }
        System.out.println("Сумма чисел от 1 до 100: " + sum);
    }
}
```

## Задача 3

Найдите факториал заданного числа.

```
public class FactorialCalculator {
    public static void main(String[] args) {
        int number = 5; // Заданное число, для которого мы хотим найти факториал
        int factorial = 1; // Изначально присваиваем факториалу значение 1

        // Используем цикл for для нахождения факториала
        for (int i = 1; i <= number; i++) {
            factorial *= i; // Умножаем текущее значение факториала на текущее значение i
        }

        // Выводим результат
        System.out.println("Факториал числа " + number + " равен " + factorial);
    }
}
```

## Задача 4

Напишите проверку, является ли заданное число простым.

```
public class PrimeNumberCheck {  
    public static void main(String[] args) {  
        int number = 17;  
        boolean isPrime = true;  
        for (int i = 2; i <= Math.sqrt(number); i++) {  
            if (number % i == 0) {  
                isPrime = false;  
                break;  
            }  
        }  
        System.out.println(number + " является простым числом: " + isPrime);  
    }  
}
```

### Задача 5

Напишите программу, которая выводит на консоль простые числа в промежутке от [2, 100].

Используйте для решения этой задачи оператор "%" (остаток от деления) и циклы.

```
for(int i = 2; i <= 100; i ++){  
    boolean isPrime = true;  
  
    for(int j = 2; j < i; j++){  
        if(i % j == 0){  
            isPrime = false;  
            break;  
        }  
    }  
  
    if(isPrime){  
        System.out.println(i);  
    }  
}
```

### Задача 6

Распечатать 5 символов в одну строку начиная с “h”.

Используя цикл for.

```
public class ForDemo {  
    public static void main(String[] args) {  
        char symbol = 'h';  
        for (int i = 0; i < 15; i++) {  
            System.out.print(symbol++ + " ");  
        }  
    }  
}
```

## Задача 7

Распечатать 10 строк:

“Task1”.

“Task2”.

...

“Task10”.

Используя цикл while.

```
public class While1 {  
    public static void main(String[] args) {  
        int n = 1;  
        while (n < 11) {  
            System.out.println("Task " + n++);  
        }  
    }  
}
```

## **Практическая работа № 4. Объявление и обработка одномерного массива.**

### **1 Цель занятия**

Подробно изучить массивы одномерные.

### **2 Теоретический материал**

Одномерные массивы в Java представляют собой список однотипных переменных. Чтобы создать массив, нужно сначала объявить переменную массива требуемого типа.

Общая форма объявления одномерного массива выглядит следующим образом:

тип[] имяПеременной;

где параметр тип обозначает тип элемента массива, называемый также базовым типом.

Квадратные скобки можно ставить перед переменной или после нее. Но более правильным вариантом считается указание скобок перед переменной - таким образом тип и скобки находятся в одном месте, что позволяет с первого взгляда понять, что перед вами массив такого-то типа:

```
int monthDays[];  
double[] monthSalaries;
```

#### **2. Инициализация массива с помощью ключевого слова new**

Когда массив объявлен, память под него еще не выделена. Для выделения памяти под массив, используется ключевое слово new, после которого опять указывается тип массива и в квадратных скобках его размер:

```
имяПеременной = new тип[размер];
```

Массив может быть объявлен и инициализирован одной строкой:

```
int[] values = new int[45];
```

Рассмотрим пример объявления массива типа int размером 12 на данном примере. После выполнения строки `int[] monthDays = new int[12]` массив из 12 элементов создан. Каждому элементу присваивается значение по умолчанию для заданного типа. Для типа int это ноль. Для обращения к отдельному элементу массива после имени массива в квадратных скобочках задаем индекс элемента. Таким образом мы можем обратиться к элементу массива для изменения или получения его значения:

```
public class Array1 {  
    public static void main(String[] args) {  
        int[] monthDays = new int[12];  
        monthDays[0] = 31;  
        monthDays[1] = 28;  
        monthDays[2] = 31;  
        monthDays[3] = 30;  
        monthDays[4] = 31;  
        monthDays[5] = 30;  
        monthDays[6] = 31;  
        monthDays[7] = 31;  
        monthDays[8] = 30;  
        monthDays[9] = 31;  
        monthDays[10] = 30;  
        monthDays[11] = 31;  
        System.out.println("В апреле " + monthDays[3] + " дней.");  
    }  
}
```

### 3. Инициализация массива с помощью блока для инициализации

Если заранее известны значения для каждого элемента массива, можно использовать блок для инициализации массива. Вместо `new int[12]`, в

фигурных скобках через запятую перечисляются значения элементов массива. Размер массива выводится компилятором из количества указанных элементов:

```
public class Array2 {  
    public static void main(String[] args) {  
        int[] monthDays = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
        System.out.println("В апреле " + monthDays[3] + " дней.");  
    }  
}
```

#### 4. Безымянный массив

Существует еще и третья форма объявления массива - безымянный массив. Он может использоваться в двух случаях.

Первый - вы объявили и инициализировали массив testScores размера четыре, но потом по какой-то причине он должен быть изменен - он должен содержать три элемента. Использовать повторно форму для инициализации массива нельзя - будет ошибка компиляции:

```
int[] testScores = {1, 2, 3, 4};  
...  
testScores = {4, 7, 2}; //ошибка компиляции
```

Но можно использовать безымянный массив, который создаст новый массив в памяти. Форма написания безымянного массива - это микс первых двух:

```
testScores = new int[]{4, 7, 2};
```

Второй случай использования безымянного массива - это передача массива в метод. В следующем примере метод print принимает на вход массив типа int. При вызове метода в качестве аргумента можно передать безымянный массив.

```
public class Array3 {
```

```
public static void main(String[] args) {  
    int[] testScores = {1, 2, 3, 4};  
    for (int element : testScores) {  
        System.out.print(element + " ");  
    }  
    System.out.println();  
    testScores = new int[]{4, 7, 2};  
    for (int element : testScores) {  
        System.out.print(element + " ");  
    }  
    System.out.println();  
  
    print(new int[]{4, 6, 2, 3});  
}
```

```
public static void print(int[] array) {  
    for (int element : array) {  
        System.out.print(element + " ");  
    }  
}
```

### **Задача 1**

Создать массив типа String с размером 7.

Записать в него значения дней недели.

Вывести на консоль значение последнего элемента.

```
public class ArrayDemo1 {
    public static void main(String[] args) {
        String[] array = new String[7];
        array[0] = "Mon";
        array[1] = "Tue";
        array[2] = "Wed";
        array[3] = "Thu";
        array[4] = "Fri";
        array[5] = "Sat";
        array[6] = "Sun";

        System.out.println(array[7]);
    }
}
```

## Задача 2

Создать массив типа double с размером 4.

Записать в него любые значения с помощью блока для инициализации.

Вывести на консоль значение первого элемента.

```
public class ArrayDemo2 {
    public static void main(String[] args) {
        double[] array = {5.4, 4, 7, 6.8};
        System.out.println(array[0]);
    }
}
```

## Задача 3

Отсортируйте массив по значению в порядке возрастания и убывания.

```

import java.util.Arrays;
import java.util.Comparator;

public class ArraySorting {
    public static void main(String[] args) {
        int[] array = {5, 2, 8, 1, 3};

        // Сортировка массива по возрастанию
        Arrays.sort(array);

        System.out.println("Массив, отсортированный по возрастанию:");
        for (int num : array) {
            System.out.print(num + " ");
        }

        // Сортировка массива по убыванию
        Integer[] boxedArray = Arrays.stream(array).boxed().toArray(Integer[]::new);
        Arrays.sort(boxedArray, Comparator.reverseOrder());

        System.out.println("\nМассив, отсортированный по убыванию:");
        for (int num : boxedArray) {
            System.out.print(num + " ");
        }
    }
}

```

#### Задача 4

Напишите программу, которая инвертирует массив (меняет порядок элементов на противоположный).

```

public class ArrayReverse {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};
        int temp;
        int length = array.length;
        for (int i = 0; i < length / 2; i++) {
            temp = array[i];
            array[i] = array[length - i - 1];
            array[length - i - 1] = temp;
        }
        System.out.print("Инвертированный массив: ");
        for (int element : array) {
            System.out.print(element + " ");
        }
    }
}

```

#### Задача 5

Заполните массив случайным числами и выведите максимальное, минимальное и среднее значение.

Для генерации случайного числа используйте метод Math.random(), который возвращает значение в промежутке [0, 1].

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

    int n = 100;
    double[] array = new double[n];
    for (int i = 0; i < array.length; i++) {
        array[i] = Math.random();
    }

    double max = array[0]; // Массив не должен быть пустым
    double min = array[0];
    double avg = 0;
    for (int i = 0; i < array.length; i++) {
        if(max < array[i])
            max = array[i];
        if(min > array[i])
            min = array[i];
        avg += array[i]/array.length;
    }

    System.out.println("max = " + max);
    System.out.println("min = " + min);
    System.out.println("avg = " + avg);
}
```

Реализуйте алгоритм сортировки пузырьком для сортировки массива

```

for (int i = 0; i < array.length; i++) {
    for (int j = 0; j < array.length - i - 1; j++) {
        if (array[j] > array[j + 1]) {
            double temp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = temp;
        }
    }
}

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

```

## Задача 6

Дан массив целых чисел и ещё одно целое число. Удалите все вхождения этого числа из массива (пропусков быть не должно).

```

public static void main(String[] args) {
    int test_array[] = {0,1,2,2,3,0,4,2};
    /*
        Arrays.toString:
        см. https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html
    */
    System.out.println(Arrays.toString(removeElement(test_array, 3)));
}

public static int[] removeElement(int[] nums, int val) {
    int offset = 0;

    for(int i = 0; i < nums.length; i++){
        if(nums[i] == val){
            offset++;
        } else{
            nums[i - offset] = nums[i];
        }
    }

    // Arrays.copyOf копирует значение из массива nums в новый массив
    // с длиной nums.length - offset
    return Arrays.copyOf(nums, nums.length - offset);
}

```

## **Практическая работа № 5. Объявление и обработка двумерного массива.**

### **1 Цель занятия**

Подробно изучить массивы двумерные.

### **2 Теоретический материал**

Многомерные массивы представляют собой массивы массивов.

При объявлении переменной многомерного массива для указания каждого дополнительного индекса используется отдельный ряд квадратных скобок.  
Например:

```
int[][] twoD = new int[5][4];
```

Следующий рисунок показывает как можно визуально представить двумерный массив 5 на 4. Левый индекс определяет строку, а правый столбец:

Концептуальное представление массива размерностью 5 на 4 фото

Следующий пример демонстрирует каким образом можно установить значения в двухмерный массив 5x4. Для перебора строк используется внешний цикл for, для перебора столбцов - внутренний. Каждому следующему элементу присваивается значение на единицу большее чем предыдущее.

```
public class TwoDArray1 {  
    public static void main(String[] args) {  
        int[][] twoD = new int[5][4];  
        int i, j, k = 0;  
        for (i = 0; i < 5; i++) {  
            for (j = 0; j < 4; j++) {  
                twoD[i][j] = k;  
                k++;  
            }  
        }  
    }  
}
```

```

        twoD[i][j] = k++;
        System.out.print(twoD[i][j] + " ");
    }
    System.out.println();
}
}

```

Подобным образом может храниться массив любой размерности в памяти.

В двумерных массивах, которые мы рассматривали до сих пор, количество элементов в каждой строке одинаково - чаще всего так и бывает. Но это не обязательно, каждая строка может содержать разное количество элементов.

Посмотрим код, реализующий такой массив. При объявлении двумерного массива необходимо задать количество элементов только для первой размерности - `int[][] array = new int[4][]`. Таким образом, мы указываем количество строк в массиве, но под каждую строку память не выделяем. Далее выделяем отдельно память под каждую строку массива. Например, строка с индексом ноль будет размера 1 - `array[0] = new int[1]`.

```

public class TwoDArray2 {
    public static void main(String[] args) {
        int[][] array = new int[4][];
        array[0] = new int[1];
        array[1] = new int[2];
        array[2] = new int[3];
        array[3] = new int[4];
        int i, j, k = 0;
        for (i = 0; i < 4; i++) {
            for (j = 0; j < i + 1; j++) {
                array[i][j] = k++;
                System.out.print(array[i][j] + " ");
            }
        }
    }
}

```

```
    }
    System.out.println();
}
}
```

Для многомерных массивов можно также использовать блок для инициализации, если значения всех элементов заранее известны. Каждая отдельная строка заключается в фигурные скобки:

```
public class TwoDArray3 {
    public static void main(String[] args) {
        double[][] arrayTwoD = {
            {0, 1, 2, 3},
            {4, 5, 6, 7},
            {8, 9, 10, 11},
            {12, 13, 14, 15}
        };
        for (double[] arrayOneD : arrayTwoD) {
            for (double element : arrayOneD) {
                System.out.print(element + " ");
            }
            System.out.println();
        }
    }
}
```

### Задача 1

Создать массив типа String размером 3x6 и распечатать.

Записать в него значения:

a1 a2 a3 a4 a5 a6

b1 b2 b3 b4 b5 b6

c1 c2 c3 c4 c5 c6

```
public class TwoDArrayDemo {  
    public static void main(String[] args) {  
        char symbol = 'a';  
        String[][] array = new String[3][6];  
        for (int i = 0; i < 3; i++) {  
            for (int j = 0; j < 6; j++) {  
                array[i][j] = symbol + "" + (j + 1);  
                System.out.print(array[i][j] + " ");  
            }  
            symbol++;  
            System.out.println();  
        }  
    }  
}
```

## Задача 2

Создать двумерный массив типа char размером 4x2.

И записать туда значения с помощью блока для инициализации.

Распечатать значения.

```
public class CharArrayDemo {  
    public static void main(String[] args) {  
        char[][] array = {{'r', 'h'},  
                          {'y', 'g'},  
                          {'b', 'f'},  
                          {'s', 'e'}};  
  
        for (char[] row : array) {  
            for (char element : row) {  
                System.out.print(element + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

## **Практическая работа № 6. Ввод массивов.**

### **1 Цель занятия**

Научиться работать с массивами в Java.

### **2 Теоретический материал**

Массив в Java (Java Array) — это структура данных, которая хранит набор пронумерованных значений одного типа (элементы массива).

Допустим, у нас есть класс из десяти учеников и нам нужно сохранить их оценки. Для этого можно создать десять переменных:

```
int mark1 = 4;  
int mark2 = 3;  
int mark3 = 5;  
int mark4 = 3;  
int mark5 = 2;  
int mark6 = 4;  
int mark7 = 4;  
int mark8 = 3;  
int mark9 = 4;  
int mark10 = 2;
```

А если в нашем классе будет не десяток учеников, а в десять раз больше, не заводить же нам 100 переменных! На выручку приходят массивы.

Как создать одномерный массив

В Java массив объявляется с помощью квадратных скобок и специального слова new.

Такой вот синтаксис пришёл из языка С:

```
тип_массива название_переменной[] = new тип_массива[размер_массива];
```

Но в Java предпочтительнее делать так:

```
тип_массива[] название_переменной = new тип_массива[размер_массива];
```

Тип массива может быть любым (int, Integer, String, Date, byte, char, Long и так далее).

Инициализация массива по умолчанию

Объявим массив типа int из 10 элементов:

```
int[] marks = new int[10];
```

При подобной инициализации все элементы массива будут иметь значение по умолчанию. Для int это 0; для float и double — 0.0; для char — \0; для boolean — false, а для String и любого другого класса это null.

В Java размер массива (длина, протяжённость) определяется при объявлении, а изменить его можно только пересоздав массив.

### **Задача 1**

Поиск максимального и минимального значения в каждой строке массива.

Создать двумерный массив 5x8 типа int и инициализировать его с помощью блока для инициализации.

Найти максимальное и минимальное значение в каждой "строке" и записать эти значения в двухмерный массив 5x2.

Распечатать массив, содержащий максимальное и минимальное значение.

### **Задача 2**

#### **Гирлянда №1**

Имеется гирлянда, состоящая из 32 лампочек. Каждая лампочка имеет два состояния - включена или выключена. В начале работы программы случайным образом задайте какие лампочки будут включены. Напишите следующие методы:

метод `blink`, который будет мигать лампочками гирлянды один раз (операция `~`);

метод `run`, который будет запускать гирлянду в режим бегущей строки (операция `<<` или `>>`);

метод `isFirstLampOn`, который будет выяснять включена ли лампочка в первой позиции (наложение маски с помощью `&`);

метод, который будет распечатывать текущее состояние гирлянды. Для получения двоичного формата числа используйте метод `Integer.toBinaryString(a)`.

Используйте побитовые операции.

### **Задача 3**

#### **Гирлянда №2**

Имеется гирлянда, состоящая из 32 лампочек. Каждая лампочка имеет два состояния - включена или выключена. В начале работы программы случайным образом задайте какие лампочки будут включены. Используя массивы. Для реализации бегущей строки используйте метод `System.arraycopy(...)`.

## **Практическая работа № 7. Обработка строк: поиск, сравнение.**

### **1 Цель занятия**

Научиться работать со строками.

### **2 Теоретический материал**

Соединение строк

Для соединения строк можно использовать операцию сложения ("+"):

```
String str1 = "Java";
String str2 = "Hello";
String str3 = str1 + " " + str2;
```

```
System.out.println(str3); // Hello Java
```

При этом если в операции сложения строк используется нестроковый объект, например, число, то этот объект преобразуется к строке:

```
String str3 = "Год " + 2015;
```

Фактически же при сложении строк с нестроковыми объектами будет вызываться метод `valueOf()` класса `String`. Данный метод имеет множество перегрузок и преобразует практически все типы данных к строке. Для преобразования объектов различных классов метод `valueOf` вызывает метод `toString()` этих классов.

Другой способ объединения строк представляет метод `concat()`:

```
String str1 = "Java";
String str2 = "Hello";
str2 = str2.concat(str1); // HelloJava
```

Метод `concat()` принимает строку, с которой надо объединить вызывающую строку, и возвращает соединенную строку.

Еще один метод объединения - метод `join()` позволяет объединить строки с учетом разделителя. Например, выше две строки сливались в одно слово

"HelloJava", но в идеале мы бы хотели, чтобы две подстроки были разделены пробелом. И для этого используем метод join():

```
String str1 = "Java";  
String str2 = "Hello";  
String str3 = String.join(" ", str2, str1); // Hello Java
```

Метод join является статическим. Первым параметром идет разделитель, которым будут разделяться подстроки в общей строке, а все последующие параметры передают через запятую произвольный набор объединяемых подстрок - в данном случае две строки, хотя их может быть и больше

### Извлечение символов и подстрок

Для извлечения символов по индексу в классе String определен метод charAt(int index). Он принимает индекс, по которому надо получить символов, и возвращает извлеченный символ:

```
String str = "Java";  
char c = str.charAt(2);  
System.out.println(c); // v
```

Как и в массивах индексация начинается с нуля.

Если надо извлечь сразу группу символов или подстроку, то можно использовать метод getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin). Он принимает следующие параметры:

srcBegin: индекс в строке, с которого начинается извлечение символов

srcEnd: индекс в строке, до которого идет извлечение символов

dst: массив символов, в который будут извлекаться символы

dstBegin: индекс в массиве dst, с которого надо добавлять извлеченные из строки символы

```
String str = "Hello world!";
int start = 6;
int end = 11;
char[] dst=new char[end - start];
str.getChars(start, end, dst, 0);
System.out.println(dst); // world
```

### Сравнение строк

Для сравнения строк используются методы equals() (с учетом регистра) и equalsIgnoreCase() (без учета регистра). Оба метода в качестве параметра принимают строку, с которой надо сравнить:

```
String str1 = "Hello";
```

```
String str2 = "hello";
```

```
System.out.println(str1.equals(str2)); // false
```

```
System.out.println(str1.equalsIgnoreCase(str2)); // true
```

В отличие от сравнения числовых и других данных примитивных типов для строк не применяется знак равенства ==. Вместо него надо использовать метод equals().

Еще один специальный метод regionMatches() сравнивает отдельные подстроки в рамках двух строк. Он имеет следующие формы:

```
boolean regionMatches(int toffset, String other, int ooffset, int len)
```

```
boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
```

Метод принимает следующие параметры:

ignoreCase: надо ли игнорировать регистр символов при сравнении. Если значение true, регистр игнорируется

`toffset`: начальный индекс в вызывающей строке, с которого начнется сравнение

`other`: строка, с которой сравнивается вызывающая

`oofset`: начальный индекс в сравниваемой строке, с которого начнется сравнение

`len`: количество сравниваемых символов в обеих строках

Используем метод:

```
String str1 = "Hello world";
String str2 = "I work";
boolean result = str1.regionMatches(6, str2, 2, 3);
System.out.println(result); // true
```

В данном случае метод сравнивает 3 символа с 6-го индекса первой строки ("wor") и 3 символа со 2-го индекса второй строки ("wor"). Так как эти подстроки одинаковы, то возвращается `true`.

И еще одна пара методов `int compareTo(String str)` и `int compareToIgnoreCase(String str)` также позволяют сравнить две строки, но при этом они также позволяют узнать больше ли одна строка, чем другая или нет. Если возвращаемое значение больше 0, то первая строка больше второй, если меньше нуля, то, наоборот, вторая больше первой. Если строки равны, то возвращается 0.

Для определения больше или меньше одна строка, чем другая, используется лексикографический порядок. То есть, например, строка "A" меньше, чем строка "B", так как символ 'A' в алфавите стоит перед символом 'B'. Если первые символы строк равны, то в расчет берутся следующие символы.

Например:

```
String str1 = "hello";
String str2 = "world";
```

```
String str3 = "hell";
```

```
System.out.println(str1.compareTo(str2)); // -15 - str1 меньше чем str2
```

```
System.out.println(str1.compareTo(str3)); // 1 - str1 больше чем str3
```

### Поиск в строке

Метод `indexOf()` находит индекс первого вхождения подстроки в строку, а метод `lastIndexOf()` - индекс последнего вхождения. Если подстрока не будет найдена, то оба метода возвращают -1:

```
String str = "Hello world";
int index1 = str.indexOf('l'); // 2
int index2 = str.indexOf("wo"); //6
int index3 = str.lastIndexOf('l'); //9
```

Метод `startsWith()` позволяют определить начинается ли строка с определенной подстроки, а метод `endsWith()` позволяет определить заканчивается строка на определенную подстроку:

```
String str = "myfile.exe";
boolean start = str.startsWith("my"); //true
boolean end = str.endsWith("exe"); //true
```

### Замена в строке

Метод `replace()` позволяет заменить в строке одну последовательность символов на другую:

```
String str = "Hello world";
String replStr1 = str.replace('l', 'd'); // Heddo wordd
String replStr2 = str.replace("Hello", "Bye"); // Bye world
```

### Обрезка строки

Метод `trim()` позволяет удалить начальные и конечные пробелы:

```
String str = " hello world ";
str = str.trim(); // hello world
```

Метод `substring()` возвращает подстроку, начиная с определенного индекса до конца или до определенного индекса:

```
String str = "Hello world";
String substr1 = str.substring(6); // world
String substr2 = str.substring(3,5); //lo
```

### Изменение регистра

Метод `toLowerCase()` переводит все символы строки в нижний регистр, а метод `toUpperCase()` - в верхний:

```
String str = "Hello World";
System.out.println(str.toLowerCase()); // hello world
System.out.println(str.toUpperCase()); // HELLO WORLD
```

### Split

Метод `split()` позволяет разбить строку на подстроки по определенному разделителю. Разделитель - какой-нибудь символ или набор символов передается в качестве параметра в метод. Например, разобьем текст на отдельные слова:

```
String text = "FIFA will never regret it";
String[] words = text.split(" ");
for(String word : words){
    System.out.println(word);
}
```

### Задача 1

У гусей и кроликов вместе  $2n$  лап. Сколько может быть гусей и кроликов?

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

```
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
```

```

int g,k;
System.out.println("Всего лап: " + 2*n);
g =(2*n % 4) / 2;
k = (2*n-2*g) / 4;
do{
    System.out.println(g + " гусей и "+ k+ "кроликов");
    g+=2;
    k--;
} while (k >= 0);
}
}

```

## **Задача 2**

Поиск наименьшего и наибольшего числа из 3 чисел в массиве.

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

```

class Main {
    public static void main(String[] args) {
        // declare and create array object
        // declare smallest and largest int variables
        int[] numbers;
        numbers = new int[3];

        // create Scanner object
        Scanner input = new Scanner(System.in);

        // prompt user
        System.out.print("Введите 3 числа: \n");
        // use for loop to obtain user input
        for (int counter = 0; counter < numbers.length; counter++) {

```

```

        numbers[counter] = input.nextInt();

    } // end obtaining input

    //Use in built Math.min and Math.max to get smallest and largest numbers
    System.out.printf("%s: %d%n", "Наименьшее число ",
Math.min(numbers[0], Math.min(numbers[1], numbers[2])));

    System.out.printf("%s: %d%n", "Наибольшее число ",
Math.max(numbers[0], Math.max(numbers[1], numbers[2])));

}
}

```

### **Задача 3**

Напишите программу на Java, которая читает целое число и проверяет, является ли оно отрицательным, нулевым или положительным.

```

import java.util.Scanner;

public class Main {

    public static void main(String[] args)

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

        System.out.print("Введите число: ");

        int n = in.nextInt();

        if (n > 0)

        {
            System.out.println("Число положительное");
        }

        else if (n < 0)

        {
            System.out.println("Число отрицательное");
        }

        else

```

```
{  
    System.out.println("Число нулевое");  
}  
}  
}
```

## **Практическая работа № 8. Обработка символов.**

### **1 Цель занятия**

Научиться манипулировать символами в строке.

### **2 Теоретический материал**

String Класс имеет ряд методов для проверки содержимого строк, поиска символов или подстрок внутри строки, изменения регистра и других задач.

**Получение символов и подстрок по индексу**

Вы можете получить символ с определенным индексом внутри строки, вызвав charAt() метод доступа. Индекс первого символа равен 0, а индекс последнего символа равен length()-1. Например, следующий код возвращает символ с индексом 9 в строке:

```
String anotherPalindrome = "Ниагара. О, снова рев!";
```

```
char aChar = anotherPalindrome.charAt(9);
```

Индексы начинаются с 0, поэтому символом с индексом 9 является "О".

Если вы хотите получить из строки более одного последовательного символа, вы можете использовать substring метод. substring Метод имеет две версии, как показано в следующей таблице:

#### **substringМетоды в String классе**

<b>Метод</b>	<b>Описание</b>
<code>String substring(int beginIndex, int endIndex)</code>	Возвращает новую строку, которая является подстрокой этой строки. Подстрока начинается с указанного beginIndex и продолжается до символа с индексом endIndex - 1.
<code>String substring(int beginIndex)</code>	Возвращает новую строку, которая является подстрокой этой строки. Аргумент integer указывает индекс первого символа. Здесь

возвращаемая подстрока продолжается до конца исходной строки.

## Поиск символов и подстрок в строке

Вот несколько других String методов поиска символов или подстрок внутри строки. String Класс предоставляет методы доступа, которые возвращают позицию в строке определенного символа или подстроки: indexOf() и lastIndexOf(). indexOf()Методы выполняют поиск в прямом направлении от начала строки, а lastIndexOf() методы - в обратном направлении от конца строки. Если символ или подстрока не найдены, indexOf() и lastIndexOf() верните значение -1.

StringКласс также предоставляет метод поиска, contains который возвращает true, если строка содержит определенную последовательность символов. Используйте этот метод, когда вам нужно только знать, что строка содержит последовательность символов, но точное местоположение не важно.

## Замена символов и подстрок в строке

StringКласс имеет очень мало методов для вставки символов или подстрок в строку. В общем, они не нужны: вы можете создать новую строку путем конкатенации строки, вы должны удалить из строки с подстрокой, которую вы хотите вставить.

### Задача 1

Создать программу для получения кол-ва уникальных символов.

```
public class Main {  
    public static void main(String []args){  
        String line = "aaabccdddc";  
        System.out.println( line.chars().distinct().count() );  
    }  
}
```

### Задача 2

Перевести строку в верхний регистр.

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

```
String line = "организация объединённых наций";
char[] chars = line.toCharArray();
for (int i = 0; i < chars.length; i++) {
    if (i == 0 || chars[i - 1] == ' ') {
        chars[i] = Character.toUpperCase(chars[i]);
    }
}
System.out.println(new String(chars));
}
```

## **Практическая работа № 9. Включение класса в учебный проект.**

### **1 Цель занятия**

Разобраться в устройстве классов из которых создают объекты.

### **2 Теоретический материал**

Класс в Java - это шаблон для создания объекта, а объект - это экземпляр класса. Класс определяет структуру и поведение, которые будут совместно использоваться набором объектов. Класс содержит переменные и методы, которые называются элементами класса, членами класса. Он составляет основу инкапсуляции в Java. Каждый объект данного класса содержит структуру и поведение, которые определены классом. Иногда объекты называют экземплярами класса.

Методы используются для описания того, что объект класса умеет делать или что можно с ним сделать. Переменные - для описания свойств или характеристик объекта.

Как создать класс в Java

Рассмотрим как создать класс в языке Java. Упрощенная общая форма определения класса:

```
class ИмяКласса{  
  
    тип переменнаяЭкземпляра1;  
    тип переменнаяЭкземпляра2;  
    // ...  
    тип переменнаяЭкземпляраН;  
  
    тип имяМетода 1 ( список параметров) {  
        // тело метода  
    }  
}
```

```
тип имяМетода2 (список параметров) {  
    // тело метода  
}  
  
...  
  
тип имяМетодaN (список параметров ) {  
    // тело метода  
}  
}
```

После ключевого слова `class` пишется имя класса. В теле класса объявляются переменные и методы класса. Их может быть сколько угодно.

Опишем класс для объекта `Box` (коробка). У коробки есть три главные характеристики: ширина, высота и глубина, описанные с помощью переменных:

```
public class Box {  
    double width;  
    double height;  
    double depth;  
}
```

### Задача 1

Одежда

Создать перечисление, содержащее размеры одежды (`XXS`, `XS`, `S`, `M`, `L`). Перечисление содержит метод `getDescription`, возвращающий строку "Взрослый размер". Переопределить метод `getDescription` - для константы `XXS` метод должен возвращать строку "Детский размер". Также перечисление должно содержать числовое значение `euroSize(32, 34, 36, 38, 40)`, соответствующее каждому размеру. Создать конструктор, принимающий на вход `euroSize`.

Создать интерфейсы "Мужская Одежда" с методом "одетьМужчину" и "Женская Одежда" с методом "одетьЖенщину".

Создать абстрактный класс Одежда, содержащий переменные - размер одежды, стоимость, цвет.

Создать классы наследники Одежды - Футболка (реализует интерфейсы "Мужская Одежда" и "Женская Одежда"), Брюки (реализует интерфейсы "Мужская Одежда" и "Женская Одежда"), Юбка (реализует интерфейсы "Женская Одежда"), Галстук (реализует интерфейсы "Мужская Одежда").

Создать массив, содержащий все типы одежды. Создать класс Ателье, содержащий методы одетьЖенщину, одетьМужчину, на вход которых будет поступать массив, содержащий все типы одежды. Метод одетьЖенщину выводит на консоль всю информацию о женской одежде. То же самое для метода одетьМужчину.

## Задача 2

Интерфейс Printable

Определить интерфейс Printable, содержащий метод void print().

Определить класс Book, реализующий интерфейс Printable.

Определить класс Magazine, реализующий интерфейс Printable.

Создать массив типа Printable, который будет содержать книги и журналы.

В цикле пройти по массиву и вызвать метод print() для каждого объекта.

Создать статический метод printMagazines(Printable[] printable) в классе Magazine, который выводит на консоль названия только журналов.

Создать статический метод printBooks(Printable[] printable) в классе Book, который выводит на консоль названия только книг. Используем оператор instanceof.

# **Практическая работа № 10. Разработка приложения в соответствии с принципами объектно-ориентированного программирования по индивидуальным заданиям (начальный этап).**

## **1 Цель занятия**

Научиться разработки приложений в соответствии с принципами ООП.

## **2 Теоретический материал**

Объектно-ориентированное программирование (ООП) - методика программирования, в которой основными концепциями являются понятия объектов и классов. Прежде чем начать писать инструкции для решения задачи, в задаче выделяются объекты и описываются с помощью классов. В классе прописывается поведение объектов с помощью методов и характеристики или свойства объекта с помощью переменных класса. Одной из ключевых особенностей языка Java является ООП.

Выделяют три принципа ООП:

- Инкапсуляция — это свойство системы, позволяющее объединить данные и методы в классе, и скрыть детали реализации от пользователя.
- Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. В Java класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником или производным классом.
- Полиморфизм — буквально означает много форм. Это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. “Один интерфейс, множество методов”. Реализации полиморфизма в языке Java - это перегрузка и переопределение методов, интерфейсы.

Выделяют еще один четвертый принцип:

- Абстракция данных — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения не значимые. Соответственно, абстракция — это набор всех таких характеристик.

## **Задача 1**

Класс Phone.

Создайте класс Phone, который содержит переменные number, model и weight.

Создайте три экземпляра этого класса.

Выведите на консоль значения их переменных.

Добавить в класс Phone методы: receiveCall, имеет один параметр – имя звонящего. Выводит на консоль сообщение “Звонит {name}”. Метод getNumber – возвращает номер телефона. Вызвать эти методы для каждого из объектов.

Добавить конструктор в класс Phone, который принимает на вход три параметра для инициализации переменных класса - number, model и weight.

Добавить конструктор, который принимает на вход два параметра для инициализации переменных класса - number, model.

Добавить конструктор без параметров.

Вызвать из конструктора с тремя параметрами конструктор с двумя.

Добавьте перегруженный метод receiveCall, который принимает два параметра - имя звонящего и номер телефона звонящего. Вызвать этот метод.

Создать метод sendMessage с аргументами переменной длины. Данный метод принимает на вход номера телефонов, которым будет отправлено сообщение. Метод выводит на консоль номера этих телефонов.

Изменить класс Phone в соответствии с концепцией JavaBean.

## **Задача 2**

Класс Person

Создать класс Person, который содержит:

переменные fullName, age;

методы move() и talk(), в которых просто вывести на консоль сообщение - "Такой-то Person говорит".

Добавьте два конструктора - Person() и Person(fullName, age).

Создайте два объекта этого класса. Один объект инициализируется конструктором Person(), другой - Person(fullName, age).

Вызовите методы move() и talk().

### **Задача 3**

Класс Матрица

Создать класс "Матрица". Класс должен иметь следующие переменные:

двумерный массив вещественных чисел;

количество строк и столбцов в матрице.

Класс должен иметь следующие методы:

сложение с другой матрицей;

умножение на число;

вывод на печать;

умножение матриц.

### **Задача 3**

Читатели библиотеки

Определить класс Reader, хранящий такую информацию о пользователе библиотеки:

ФИО,

номер читательского билета,

факультет,

дата рождения,

телефон.

Методы takeBook(), returnBook().

Разработать программу, в которой создается массив объектов данного класса.

Перегрузить методы takeBook(), returnBook():

- takeBook, который будет принимать количество взятых книг. Выводит на консоль сообщение "Петров В. В. взял 3 книги".
- takeBook, который будет принимать переменное количество названий книг. Выводит на консоль сообщение "Петров В. В. взял книги: Приключения, Словарь, Энциклопедия".
- takeBook, который будет принимать переменное количество объектов класса Book (создать новый класс, содержащий имя и автора книги). Выводит на консоль сообщение "Петров В. В. взял книги: Приключения, Словарь, Энциклопедия".

Аналогичным образом перегрузить метод returnBook(). Выводит на консоль сообщение "Петров В. В. вернул книги: Приключения, Словарь, Энциклопедия". Или "Петров В. В. вернул 3 книги".

## Задача 4

### Животные

Создать класс Animal и расширяющие его абстрактные классы Dog, Cat, Bear.

Класс Animal содержит переменную name и абстрактные методы makeNoise, eat, getDescription. Метод makeNoise, например, может выводить на консоль звуки животных. Метод eat выводит на консоль список того, чем питается данное животное. Метод getDescription возвращает описание животного.

Dog, Cat, Bear переопределяют методы makeNoise, eat, getDescription.

Создайте класс Ветеринар, в котором определите метод void treatAnimal(Animal animal). Пусть этот метод распечатывает name и описание пришедшего на прием животного.

В методе main создайте массив типа Animal, в который запишите животных всех имеющихся у вас типов. В цикле отправляйте их на прием к ветеринару. В отдельном цикле вызовите методы makeNoise, eat для каждого животного.

## Задача 5

### Фигуры

Создайте супер класс Shape и его наследников Circle, Rectangle.

Класс Shape содержит абстрактный метод draw() и переменную хранящую цвет.

Классы Circle, Rectangle содержат координаты точек.

Создать массив содержащий эти фигуры.

В цикле нарисовать их (вызвать метод draw).

### **Задача 6**

Интернет магазин, часть 1

Создать класс Товар, имеющий переменные имя, цена, рейтинг.

Создать класс Категория, имеющий переменные имя и массив товаров.

Создать несколько объектов класса Категория.

Создать класс Basket, содержащий массив купленных товаров.

Создать класс User, содержащий логин, пароль и объект класса Basket.

Создать несколько объектов класса User.

Вывести на консоль каталог продуктов.

Вывести на консоль покупки посетителей магазина.

# **Практическая работа № 11. Обработка потоков в учебном проекте.**

## **1 Цель занятия**

Научиться разрабатывать программы с использованием потоков выполнения.

## **2 Теоретический материал**

К большинству современных распределенных приложений (Rich Client) и веб-приложений (Thin Client) выдвигаются требования одновременной поддержки многих пользователей, каждому из которых выделяется отдельный поток, а также разделения и параллельной обработки информационных ресурсов.

**Потоки** — средство, которое помогает организовать одновременное выполнение нескольких задач, каждой в независимом потоке. Потоки представляют собой экземпляры классов, каждый из которых запускается и функционирует самостоятельно, автономно (или относительно автономно) от главного потока выполнения программы. Существует два способа создания и запуска потока: на основе расширения класса **Thread** или реализации интерфейса **Runnable**:

```
package by.bsu.threads;  
public class TalkThread extends Thread {  
    @Override  
    public void run() {  
        for (int i = 0; i < 10; i++) {  
            System.out.println("Talking");  
        }  
    }  
}
```

```
    Thread.sleep(7); // остановка на 7 миллисекунд
} catch (InterruptedException e) {
System.err.print(e);
}
}
}
}
```

При реализации интерфейса Runnable необходимо определить его единственный абстрактный метод run(). Например:

```
package by.bsu.threads;
public class WalkRunnable implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("Walking");
            try {
                Thread.sleep(7);
            } catch (InterruptedException e) {
                System.err.println(e);
            }
        }
    }
}

package by.bsu.threads;
public class WalkTalk {
    public static void main(String[ ] args) {
        // новые объекты потоков
        TalkThread talk = new TalkThread();
```

```
Thread walk = new Thread(new WalkRunnable());  
// запуск потоков  
talk.start();  
walk.start();  
  
// WalkRunnable w = new WalkRunnable(); // просто объект, не поток  
// w.run(); или talk.run(); // выполнится метод, но поток не запустится!  
}  
}
```

Запуск двух потоков для объектов классов TalkThread непосредственно и WalkRunnable через инициализацию экземпляра Thread приводит к выводу строк: Talking Walking. Порядок вывода, как правило, различен при нескольких запусках приложения. Интерфейс Runnable не имеет метода start(), а только единственный метод run(). Поэтому для запуска такого потока, как WalkRunnable следует создать экземпляр класса Thread с передачей экземпляра WalkRunnable его конструктору. Однако при прямом вызове метода run() поток не запустится, выполнится только тело самого метода.

## Жизненный цикл потока

При выполнении программы объект класса Thread может быть в одном из четырех основных состояний: «новый», «рабочоспособный», «нерабочоспособный» и «пассивный». При создании потока он получает состояние «новый» (NEW) и не выполняется. Для перевода потока из состояния «новый» в состояние «рабочоспособный» (RUNNABLE) следует выполнить метод start(), который вызывает метод run() — основной метод потока.

Поток может находиться в одном из состояний, соответствующих элементам статически вложенного перечисления Thread.State:

**NEW** — поток создан, но еще не запущен;

**RUNNABLE** — поток выполняется;

**BLOCKED** — поток блокирован;

**WAITING** — поток ждет окончания работы другого потока;

**TIMED\_WAITING** — поток некоторое время ждет окончания другого потока;

**TERMINATED** — поток завершен.

Получить текущее значение состояния потока можно вызовом метода `getState()`. Поток переходит в состояние «неработоспособный» в режиме ожидания (`WAITING`) вызовом методов `join()`, `wait()`, `suspend()` (deprecated-метод) или методов ввода/вывода, которые предполагают задержку. Для задержки потока на некоторое время (в миллисекундах) можно перевести его в режим ожидания по времени `TIMED_WAITING` с помощью методов `yield()`, `sleep(long millis)`, `join(long timeout)` и `wait(long timeout)`, при выполнении которых может генерироваться прерывание `InterruptedException`. Вернуть потоку работоспособность после вызова метода `suspend()` можно методом `resume()` (deprecated-метод), а после вызова метода `wait()` — методами `notify()` или `notifyAll()`. Поток переходит в «пассивное» состояние (`TERMINATED`), если вызваны методы `interrupt()`, `stop()` (deprecated-метод) или метод `run()` завершил выполнение, и запустить его повторно уже невозможно. После этого, чтобы запустить поток, необходимо создать новый объект потока.

Метод `interrupt()` успешно завершает поток, если он находится в состоянии «рабочоспособный». Если же поток неработоспособен, например, находится в состоянии

`TIMED_WAITING`, то метод инициирует исключение

`InterruptedException`. Чтобы это не происходило, следует предварительно вызвать метод `isInterrupted()`, который проверит возможность завершения работы потока. При разработке не следует использовать методы принудительной остановки потока, так как возможны проблемы с закрытием ресурсов и другими внешними объектами. Методы `suspend()`, `resume()` и `stop()` являются `deprecated`-методами и запрещены к использованию, так как они не являются в полной мере «потокобезопасными».

### **Задания для самостоятельного решения.**

Разработать многопоточное приложение. Использовать возможности, предоставляемые пакетом `java.util.concurrent`. Не использовать слово `synchronized`. Все сущности, желающие получить доступ к ресурсу, должны быть потоками. Использовать возможности ООП. Не использовать графический интерфейс. Приложение должно быть консольным.

**1** Порт. Корабли заходят в порт для разгрузки/загрузки контейнеров. Число контейнеров, находящихся в текущий момент в порту и на корабле, должно быть неотрицательным и превышающим заданную грузоподъемность судна и вместимость порта. В порту работает несколько причалов. У одного причала может стоять один корабль. Корабль может загружаться у причала, разгружаться или выполнять оба действия.

**2** Маленькая библиотека. Доступны для чтения несколько книг. Однаковых книг в библиотеке нет. Некоторые выдаются на руки, некоторые только в читальный зал. Читатель может брать на руки и в читальный зал несколько книг.

**3 Автостоянка.** Доступно несколько машиномест. На одном месте может находиться только один автомобиль. Если все места заняты, то автомобиль нестанет ждать больше определенного времени и уедет на другую стоянку.

**4 CallCenter.** В организации работает несколько операторов. Оператор может обслуживать только одного клиента, остальные должны ждать своей очереди. Клиент может положить трубку и перезвонить еще раз через некоторое время.

**5 Автобусные остановки.** На маршруте несколько остановок. На одной остановке может останавливаться несколько автобусов одновременно, но не более заданного числа.

**6 Свободная касса.** В ресторане быстрого обслуживания есть несколько касс. Посетители стоят в очереди в конкретную кассу, но могут перейти в другую очередь при уменьшении или исчезновении там очереди.

**7 Тоннель.** В горах существует два железнодорожных тоннеля, по которым поезда могут двигаться в обоих направлениях. По обоим концам тоннеля собралось много поездов. Обеспечить безопасное прохождение тоннелей в обоих направлениях. Поезд можно перенаправить из одного тоннеля в другой при превышении заданного времени ожидания на проезд.

**8 Банк.** Имеется банк с кассирами, клиентами и их счетами. Клиент может снимать/пополнять/переводить/оплачивать/обменивать денежные средства. Кассир последовательно обслуживает клиентов. Поток-наблюдатель следит, чтобы в кассах всегда были наличные, при скоплении денег более определенной суммы, часть их переводится в хранилище, при истощении

запасов наличных происходит пополнение из хранилища.

**9 Аукцион.** На торги выставляется несколько лотов. Участники аукциона делают заявки. Заявку можно корректировать в сторону увеличения несколько раз за торги одного лота. Аукцион определяет победителя и переходит к следующему лоту. Участник, не заплативший за лот в заданный промежуток времени, отстраняется на несколько лотов от торгов.

**10 Биржа.** На торгах брокеры предлагают акции нескольких фирм. На бирже совершаются действия по купле-продаже акций. В зависимости от количества

проданных-купленных акций их цена изменяется. Брокеры предлагают к продаже некоторую часть акций. От активности и роста-падения котировок акций изменяется индекс биржи. Биржа может приостановить торги при резком падении индекса.

**11 Аэропорт.** Посадка/высадка пассажиров может осуществляться через конечное число терминалов и наземным способом через конечное число трапов. Самолеты бывают разной вместимости и дальности полета. Организовать функционирование аэропорта, если пунктов назначения 4–6, и зон дальности 2–3.

## **Практическая работа № 12. Обработка файлов в учебном проекте.**

### **1 Цель занятия**

Научиться разрабатывать программы с использованием обработки файлов.

### **2 Теоретический материал**

В Java мы можем работать с файлами с помощью класса File. Этот класс File находится внутри пакета java.io . Класс File можно использовать, создав объект класса и затем указав имя файла.

#### **Зачем требуется обработка файлов?**

Обработка файлов является неотъемлемой частью любого языка программирования, поскольку обработка файлов позволяет нам сохранять выходные данные любой конкретной программы в файле и выполнять с ним определенные операции.

Простыми словами, обработка файлов означает чтение и запись данных в файл.

```
// Importing File Class  
  
import java.io.File;  
  
  
class GFG {  
  
    public static void main(String[] args)  
    {  
  
        // File name specified  
  
        File obj = new File("myfile.txt");
```

```

        System.out.println("File Created!");

    }

}

```

В Java концепция Stream используется для выполнения операций ввода-вывода над файлом. Итак, сначала давайте познакомимся с концепцией, известной в Java как Stream.

## Методы файлового класса Java

В следующей таблице представлены несколько методов файлового класса:

Имя метода	Описание	Возвращаемый тип
<b>CanRead()</b>	Проверяет, доступен ли файл для чтения или нет.	Логическое значение
<b>CanWrite()</b>	Проверяет, доступен ли файл для записи или нет.	Логическое значение
<b>createNewFile()</b>	Это создает пустой файл.	Логическое значение
<b>удалить()</b>	Это приводит к удалению файла.	Логическое значение

Имя метода	Описание	Возвращаемый тип
<b>существует()</b>	Проверяет, существует файл или нет.	Логическое значение
<b>длина()</b>	Возвращает размер файла в байтах.	Длинный
<b>getName()</b>	Возвращает имя файла.	Строка
<b>список()</b>	Возвращает массив файлов в каталоге.	Строка[]
<b>mkdir()</b>	Создает новый каталог.	Логическое значение
<b>getAbsolutePath()</b>	Возвращает абсолютный путь к файлу.	Строка

## Файловые операции на Java

Ниже приведены несколько операций, которые можно выполнить с файлом на Java :

- Создайте файл
- Чтение из файла
- Запись в файл
- Удалить файл

Теперь давайте подробно изучим каждую из вышеперечисленных операций.

### 1. Создайте файл

Для того, чтобы создать файл на Java, вы можете использовать метод `createNewFile()`.

Если файл успешно создан, он вернет логическое значение `true` и `false`, если файл уже существует.

Ниже приведена демонстрация того, как создать файл на Java :

```
// Import the File class
import java.io.File;

// Import the IOException class to handle errors
import java.io.IOException;

public class GFG {
    public static void main(String[] args)
    {
        try {
            File Obj = new File("myfile.txt");
            if (Obj.createNewFile()) {
                System.out.println("File created: "
                        + Obj.getName());
            }
            else {

```

```
        System.out.println("File already exists.");  
    }  
}  
  
catch (IOException e) {  
  
    System.out.println("An error has occurred.");  
  
    e.printStackTrace();  
}  
}  
}
```

## 2. Чтение из файла

Мы будем использовать класс Scanner для чтения содержимого из файла.

Ниже приведена демонстрация того, как читать содержимое из файла в Java :

```
// Import the File class  
  
import java.io.File;  
  
// Import this class for handling errors  
  
import java.io.FileNotFoundException;  
  
// Import the Scanner class to read content from text files  
  
import java.util.Scanner;  
  
public class GFG {  
  
    public static void main(String[] args)  
    {  
        try {  
            File Obj = new File("myfile.txt");  
  
            Scanner Reader = new Scanner(Obj);  
  
            while (Reader.hasNextLine()) {  
  
                String data = Reader.nextLine();  
  
                System.out.println(data);  
            }  
  
            Reader.close();  
        }  
  
        catch (FileNotFoundException e) {  
  
            System.out.println("An error has occurred.");  
        }  
    }  
}
```

```
        e.printStackTrace();

    }

}

}
```

### 3. Запись в файл

Мы используем класс `FileWriter` вместе с его методом `write()` для записи некоторого текста в файл. Ниже приведена демонстрация того, как записать текст в файл на Java

```
// Import the FileWriter class

import java.io.FileWriter;

// Import the IOException class for handling errors

import java.io.IOException;

public class GFG {

    public static void main(String[] args)

    {

        try {

            FileWriter Writer

                = new FileWriter("myfile.txt");

            Writer.write(

                "Files in Java are seriously good!!");

        }
```

```
        Writer.close();

        System.out.println("Successfully written.");

    }

    catch (IOException e) {

        System.out.println("An error has occurred.");

        e.printStackTrace();

    }

}

}
```

#### 4. Удаление файла

Для удаления файла мы используем метод `delete()`. Ниже приведена демонстрация того, как удалить файл на Java :

```
// Import the File class

import java.io.File;

public class GFG {

    public static void main(String[] args)

    {

        File Obj = new File("myfile.txt");

        if (Obj.delete()) {

            System.out.println("The deleted file is : "
```

```
+ Obj.getName());  
}  
  
else {  
  
    System.out.println(  
        "Failed in deleting the file.");  
  
}  
  
}  
}
```

### **Задачи для самостоятельного решения**

1. Написать программу, которая считывает содержимое текстового файла и выводит его на экран.
2. Реализовать программу, которая считывает числа из текстового файла, суммирует их и выводит результат на экран.
3. Создать программу, которая считывает содержимое текстового файла, удаляет из него все символы пунктуации и сохраняет результат в новый файл.
4. Разработать программу, которая считывает содержимое нескольких текстовых файлов, объединяет их в один файл и сохраняет результат.
5. Написать программу, которая считывает содержимое текстового файла, заменяет в нем все вхождения одного слова на другое и сохраняет результат.
6. Реализовать программу, которая считывает содержимое текстового файла, определяет количество строк и слов в файле и выводит результат на экран.

7. Создать программу, которая считывает содержимое текстового файла и сортирует строки в алфавитном порядке, сохраняя результат.
8. Разработать программу, которая считывает содержимое нескольких текстовых файлов, объединяет их в один файл и сортирует строки по длине, сохраняя результат.
9. Написать программу, которая считывает содержимое текстового файла, определяет частоту встречаемости каждого слова и выводит на экран самые часто встречающиеся слова.
10. Реализовать программу, которая считывает содержимое текстового файла, определяет количество уникальных слов и выводит результат на экран.

# **Практическая работа № 13. Доработка приложения с учетом обработки файлов и потоков.**

## **1 Цель занятия**

Научиться разрабатывать программы с использованием потоков.

## **2 Теоретический материал**

### **Потоки на Java**

- В Java последовательность данных известна как поток.
- Эта концепция используется для выполнения операций ввода-вывода над файлом.
- Существует два типа потоков :

#### **1. Входной поток:**

Класс Java `InputStream` является суперклассом всех входных потоков. Входной поток используется для чтения данных с многочисленных устройств ввода, таких как клавиатура, сеть и т.д. `InputStream` - это абстрактный класс, и из-за этого он бесполезен сам по себе. Однако его подклассы используются для чтения данных.

Существует несколько подклассов класса `InputStream`, к которым относятся следующие:

1. `AudioInputStream`
2. `ByteArrayInputStream`
3. `FileInputStream`
4. `FilterInputStream`
5. `StringBufferInputStream`
6. `ObjectInputStream`

#### **Создание входного потока**

```
// Creating an InputStream
```

```
InputStream obj = new FileInputStream();
```

Здесь поток ввода создается с помощью `FileInputStream`.

## Методы InputStream

S No.	Метод	Описание
1	читать()	Считывает один байт данных из входного потока.
2	чтение (массив байт[])()	Считывает байт из потока и сохраняет этот байт в указанном массиве.
3	отметить()	Он отмечает позицию во входном потоке до тех пор, пока данные не будут прочитаны.
4	доступно ()	Возвращает количество байтов, доступных во входном потоке.
5	Функция markSupported()	Проверяется, поддерживаются ли в потоке методы mark() и reset().
6	сброс ()	Возвращает элемент управления в точку, где была установлена метка внутри потока.
7	пропускает ()	Пропускает и удаляет определенное количество байтов из входного потока.
8	закрыть()	Закрывает входной поток.

## 2. Выходной поток:

Выходной поток используется для записи данных на многочисленные устройства вывода, такие как монитор, файл и т.д. OutputStream - это абстрактный суперкласс, представляющий выходной поток. OutputStream - это абстрактный класс, и из-за этого он бесполезен сам по себе. Однако его подклассы используются для записи данных.

Существует несколько подклассов класса OutputStream, которые заключаются в следующем:

1. ByteArrayOutputStream
2. FileOutputStream
3. StringBufferOutputStream
4. ObjectOutputStream
5. Поток вывода данных
6. PrintStream

### **Создание потока вывода**

```
// Creating an OutputStream  
OutputStream obj = new FileOutputStream();
```

Здесь выходной поток создается с помощью FileOutputStream.

### **Методы OutputStream**

S. Нет.	Метод	Описание
1.	write()	Записывает указанный байт в выходной поток.
2.	запись (массив байт[])	Записывает байты, находящиеся внутри определенного массива, в выходной поток.
3.	закрыть()	Закрывает выходной поток.
4.	flush()	Принудительно записывает все данные, присутствующие в выходном потоке, в пункт назначения.

**В зависимости от типа данных существует два типа потоков :**

#### **1. Поток байтов:**

Этот поток используется для чтения или записи байтовых данных. Поток байтов снова подразделяется на два типа, которые являются следующими:

- **Поток ввода байтов:** используется для чтения байтовых данных с разных устройств.
- **Поток вывода байтов:** используется для записи байтовых данных на различные устройства.

## 2. Поток символов:

Этот поток используется для чтения или записи символьных данных. Символьный поток снова подразделяется на 2 типа, которые следующие:

- **Поток ввода символов:** используется для чтения символьных данных с разных устройств.
- **Поток вывода символов:** используется для записи символьных данных на различные устройства.

Поскольку вы знаете, что такое stream, давайте усовершенствуем обработку файлов на Java, углубив понимание различных методов, полезных для выполнения операций с файлами, таких как создание, чтение и запись файлов.

### Индивидуальные задания

#### 1) Класс FileCopy

Переписать класс FileCopy:

```
package io;
```

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

```
public class FileCopy {  
    public static void main(String[] args) throws IOException {  
        FileInputStream fileIn = null;  
        FileOutputStream fileOut = null;  
  
        try {
```

```
fileIn = new FileInputStream("src\\io\\file.txt");
fileOut = new FileOutputStream("src\\io\\copied_file.txt");

int a;
while ((a = fileIn.read()) != -1) {
    fileOut.write(a);
}
} finally {
    if (fileIn != null) {
        fileIn.close();
    }
    if (fileOut != null) {
        fileOut.close();
    }
}
```

1. Добавить блок try-with-resources.
2. Добавить catch блок для обработки IOException.

## 2) Класс FileInputStreamDemo

**Переписать класс FileInputStreamDemo:**

```
package io;
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
```

```

public class FileInputStreamDemo {
    public static void main(String[] args) {
        try {
            char[] symbols = {'a', 'b', 'c'};
            OutputStream output = new FileOutputStream("a.txt");
            for (int i = 0; i < symbols.length; i++) {
                // Запись каждого символа в текстовый файл
                output.write(symbols[i]);
            }
            output.close();
        }

        InputStream input = new FileInputStream("a.txt");
        int size = input.available();

        for (int i = 0; i < size; i++) {
            // Чтение текстового файла посимвольно
            System.out.print((char) input.read() + " ");
        }
        input.close();
    } catch (IOException e) {
        System.out.print("Exception");
    }
}

```

1. Добавить блок try-with-resources.
2. Замените for (int i = 0; i < c.length; i++) на блок for-each.
3. Пусть file.txt записывается в каталог src/io.

### **3) Копирование файла в другой файл**

1. Написать класс, который копирует содержимое из одного файла в другой.
2. Используем  
классы BufferedReader, FileReader, BufferedWriter, FileWriter.

#### **4) Список каталога**

1. Написать метод который циклически просматривает содержимое заданного каталога и выводит на печать информацию о всех файлах и каталогах, находящихся в нем и в его подкаталогах.
2. Используем рекурсию.

## **Практическая работа № 14. Использование коллекций в учебном проекте.**

### **1 Цель занятия**

Научиться разрабатывать программы с использованием коллекций.

### **2 Теоретический материал**

Коллекции или контейнеры — это классы позволяющие хранить и производить операции над множеством объектов. Коллекции используются для сохранения, получения, манипулирования данными и обеспечивают агрегацию одних объектов другими.

При программировании на Java операций над группой однотипных объектов важно выбирать наиболее эффективную структуру данных (класс) для хранения этих объектов. В языке java определены специальные классы для хранения однотипных объектов, которые

называются коллекциями, определяющими такие структуры как список, множество, очередь.

Выбор определенного класса для работы с коллекциями определяет набор методов, которые будут доступны для объекта этого класса. Например, если используется список (который определяет интерфейс List), то существует выбор для его реализации: ArrayList, LinkedList, Vector, Stack. Конкретный выбор реализации списка сказывается на эффективности манипуляций с объектами списка. Так, ArrayList хранит элементы в виде массива, а значит, доступ и замена будет выполняться относительно быстро. В то же время LinkedList хранит элементы в виде связного списка, что влечет за собой относительно медленный поиск элементов и быструю операцию добавления/удаления.

Рассмотрим пример

Управление списком студентов в университете, включая добавление новых студентов, удаление старых, поиск конкретного студента по идентификатору и вывод списка студентов определенного курса

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

class Student {
    private int id;
    private String name;
    private int course;

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

    // геттеры и сеттеры
    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public int getCourse() {
        return course;
    }
}
```

```
class University {  
    private List<Student> students;  
  
    public University() {  
        students = new ArrayList<>();  
    }  
  
    public void addStudent(Student student) {  
        students.add(student);  
    }  
  
    public void removeStudent(int id) {  
        students.removeIf(s -> s.getId() == id);  
    }  
  
    public Student findStudentById(int id) {  
        for (Student student : students) {  
            if (student.getId() == id) {  
                return student;  
            }  
        }  
        return null;  
    }  
  
    public List<Student> getStudentsByCourse(int course) {  
        List<Student> studentsByCourse = new ArrayList<>();  
        for (Student student : students) {  
            if (student.getCourse() == course) {  
                studentsByCourse.add(student);  
            }  
        }  
        return studentsByCourse;  
    }  
}
```

```
        }

    }

    return studentsByCourse;
}

}

public class Main {
    public static void main(String[] args) {
        University university = new University();

        // Добавление нового студента
        university.addStudent(new Student(1, "Иванов", 3));
        university.addStudent(new Student(2, "Петров", 2));
        university.addStudent(new Student(3, "Сидоров", 1));

        // Удаление студента
        university.removeStudent(2);

        // Поиск студента по идентификатору
        System.out.println(university.findStudentById(1).getName());

        // Вывод списка студентов определенного курса
        List<Student> studentsByCourse = university.getStudentsByCourse(3);
        for (Student student : studentsByCourse) {
            System.out.println(student.getName());
        }
    }
}
```

## **Индивидуальные задания**

- 1)** В кругу стоят N человек, пронумерованных от 1 до N. При ведении счета по кругу вычеркивается каждый второй человек, пока не останется один. Составить две программы, моделирующие процесс. Одна из программ должна использовать класс ArrayList, а вторая – LinkedList. Какая из двух программ работает быстрее? Почему?
- 2)** Задан список целых чисел и число X. Не используя вспомогательных объектов и не изменяя размера списка, переставить элементы списка так, чтобы сначала шли числа, не превосходящие X, а затем числа, больше X.
- 3)** Написать программу, осуществляющую сжатие английского текста. Построить для каждого слова в тексте оптимальный префиксный код по алгоритму Хаффмена. Использовать класс PriorityQueue.
- 4)** Реализовать класс Graph, представляющий собой неориентированный граф. В конструкторе класса передается количество вершин в графе. Методы должны поддерживать быстрое добавление и удаление ребер.
- 5)** На базе коллекций реализовать структуру хранения чисел с поддержкой следующих операций:
  - добавление/удаление числа;
  - поиск числа, наиболее близкого к заданному (т. е. модуль разницы минимален).
- 6)** Реализовать класс, моделирующий работу N-местной автостоянки. Машина подъезжает к определенному месту и едет вправо, пока не встретится свободное место. Класс должен поддерживать методы, обслуживающие приезд и отъезд машины.
- 7)** Во входном файле хранятся две разреженные матрицы – A и B. Построить циклически связанные списки CA и CB, содержащие ненулевые элементы соответственно матриц A и B. Просматривая списки, вычислить: а) сумму  $S = A + B$ ; б) произведение  $P = A \times B$ .
- 8)** Во входном файле хранятся наименования некоторых объектов. Построить список C1,

элементы которого содержат наименования и шифры данных объектов, причем элементы списка должны быть упорядочены по возрастанию шифров. Затем «сжать» список С1, удаляя дублирующие наименования объектов.

**9)** Во входном файле расположены два набора положительных чисел; между наборами стоит отрицательное число. Построить два списка С1 и С2, элементы которых содержат соответственно числа 1-го и 2-го набора таким образом, чтобы внутри одного списка числа были упорядочены по возрастанию. Затем объединить списки С1 и С2 в один упорядоченный список, изменения только значения полей ссылочного типа.

**10)** Во входном файле хранится информация о системе главных автодорог, связывающих г. Минск с другими городами Беларуси. Используя эту информацию, построить дерево, отображающее систему дорог республики, а затем, продвигаясь по дереву, определить минимальный по длине путь из г. Минска в другой заданный город. Предусмотреть возможность сохранения дерева в виртуальной памяти.

**11)** Один из способов шифрования данных, называемый «двойным шифрованием», заключается в том, что исходные данные при помощи некоторого преобразования последовательно шифруются на некоторые два ключа – К1 и К2. Разработать и реализовать эффективный алгоритм, позволяющий находить ключи К1 и К2 по исходной строке и ее зашифрованному варианту. Проверить, оказался ли разработанный способ действительно эффективным, протестировав программу для случая, когда оба ключа являются 20-битными (время ее работы не должно превосходить одной минуты).

**12)** На плоскости задано N точек. Вывести в файл описания всех прямых, которые проходят более чем через одну точку из заданных. Для каждой прямой указать, через сколько точек она проходит. Использовать класс HashMap.

**13)** На клетчатой бумаге нарисован круг. Вывести в файл описания всех клеток, целиком лежащих внутри круга, в порядке возрастания расстояния от клетки до центра круга. Использовать класс PriorityQueue.

**14)** На плоскости задано N отрезков. Найти точку пересечения двух отрезков, имеющую минимальную абсциссу. Использовать класс TreeMap.

**15)** На клетчатом листе бумаги закрашена часть клеток. Выделить все различные фигуры, которые образовались при этом. Фигурой считается набор закрашенных клеток, достижимых друг из друга при движении в четырех направлениях. Две фигуры являются различными, если их нельзя

совместить поворотом на угол, кратный 90 градусам, и параллельным переносом. Используйте класс HashSet.

**16)** Данна матрица из целых чисел. Найти в ней прямоугольную подматрицу, состоящую из максимального количества одинаковых элементов.

Использовать класс Stack.

**17)** Реализовать структуру «черный ящик», хранящую множество чисел и имеющую внутренний счетчик K, изначально равный нулю. Структура должна поддерживать операции добавления числа в множество и возвращение K-го по минимальности числа из множества.

**18)** На прямой гоночной трассе стоит N автомобилей, для каждого из которых известны начальное положение и скорость. Определить, сколько произойдет обгонов.

**19)** На прямой гоночной трассе стоит N автомобилей, для каждого из которых известны начальное положение и скорость. Вывести первые K обгонов.

**20)** Ввести строки из файла, записать в список ArrayList. Выполнить сортировку строк, используя метод sort() из класса Collections.

**21)** Задана строка, состоящая из символов «(», «)», «[», «]», «{», «}». Проверить правильность расстановки скобок. Использовать стек.

**22)** Задан файл с текстом на английском языке. Выделить все различные слова. Слова, отличающиеся только регистром букв, считать одинаковыми. Использовать класс HashSet.

**23)** Задан файл с текстом на английском языке. Выделить все различные слова. Для каждого слова подсчитать частоту его встречаемости. Слова, отличающиеся регистром букв, считать различными. Использовать класс HashMap.

**24)** Ввести строки из файла, записать в список. Вывести строки в файл в обратном порядке.

**25)** Сложить два многочлена заданной степени, если коэффициенты многочленов хранятся в объекте HashMap.

**26)** С использованием множества выполнить попарное суммирование произвольного конечного ряда чисел по следующим правилам: на первом этапе суммируются попарно рядом стоящие числа, на втором этапе суммируются результаты первого этапа и т. д. до тех пор, пока не останется одно число.

**27)** Не используя вспомогательных объектов, переставить отрицательные элементы данного списка в конец, а положительные – в начало списка.

**28)** Списки (стеки, очереди) I(1..n) и U(1..n) содержат результаты n-измерений тока и напряжения на неизвестном сопротивлении R. Найти приближенное число R методом наименьших квадратов.

**29)** Создать стек из номеров записи. Организовать прямой доступ к элементам записи.

**30)** Задать два стека, поменять информацию местами.

### **Контрольные вопросы**

1)Что такое коллекции? Перечислите типы коллекции.

2)Чем отличается ArrayList от LinkedList? Приведите достоинства и недостатки каждой из

них.

3) Как устроена HashMap?

4) Роль equals и hashCode?

# **Практическая работа № 15. Реализация параметризованного интерфейса в учебном проекте.**

## **1 Цель занятия**

Научиться разрабатывать программы с использованием параметризованного интерфейса .

## **2 Теоретический материал**

Параметризованные типы позволяют объявлять классы, интерфейсы и методы, где тип данных, которыми они оперируют, указан в виде параметра. Используя дженерики, можно создать единственный класс, например, который будет автоматически работать с разными типами данных.

Классы, интерфейсы или методы, имеющие дело с параметризованными типами, называются параметризованными или обобщениями, параметризованными (обобщенными) классами или параметризованными (обобщёнными) методами.

В методах параметризованного класса можно использовать параметр типа, а следовательно, они становятся параметризованными относительно параметра типа. Но можно объявить параметризованный метод, в котором непосредственно используется один или несколько параметров типа. Более того, можно объявить параметризованный метод, входящий в не параметризованный класс. Например:

```
public class GenMethodDemo {  
    /**  
     * Является ли объект x элементом массива array  
     * @param x  
    */  
}
```

```
* @param array
* @param <T>
* @param <V>
* @return
*/
public static <T, V> boolean isIn(T x, V[] array) {
    for (V element : array) {
        if (x.equals(element)) {
            return true;
        }
    }
    return false;
}

public static void main(String[] args) {
    Integer[] intArray = {1, 2, 3, 4, 5};

    if (isIn(2, intArray)) {
        System.out.println("2 входит в массив intArray");
    }

    if (!isIn(7, intArray)) {
        System.out.println("7 не входит в intArray");
    }
}
```

```
 }

System.out.println();
```

```
String[] strArray = {"one", "two", "three", "four", "five"};
```

```
if (isIn("two", strArray)) {

    System.out.println("two входит в массив strArray");

}
```

```
if (!isIn("seven", strArray)) {

    System.out.println("seven не входит в массив strArray");

}

}
```

Конструкторы также могут быть обобщенными, даже если их классы таковыми не являются. Например:

```
public class GenConstructor {

    private double value;

    public <T extends Number> GenConstructor(T arg) {

        value = arg.doubleValue();

    }
}
```

```
public void showValue() {  
    System.out.println("value: " + value);  
}  
}  
  
public class GenConstructorDemo {  
    public static void main(String[] args) {  
        GenConstructor genConstructor1 = new GenConstructor(100);  
        GenConstructor genConstructor2 = new GenConstructor(123.5F);  
  
        genConstructor1.showValue();  
        genConstructor2.showValue();  
    }  
}
```

## Задачи

- 1) Создать параметризованный интерфейс для работы со списком и реализовать его.
- 2) Создание сортировки с использованием параметризованного интерфейса.
- 3) Создание обобщенного стека с использованием параметризованного интерфейса.
- 4) Создание обобщенного списка с использованием параметризованного интерфейса.
- 5) отсортировать список объектов по определенному полю.
- 6) создать класс-контейнер, который будет хранить элементы определенного типа данных и предоставлять возможность выполнения

операций над этими элементами.

# Практическая работа № 16. Создание форм.

## 1 Цель занятия

Научиться создавать формы.

## 2 Теоретический материал

Формы являются важным элементом взаимодействия пользователей с приложениями, особенно веб-приложениями. В Java существуют различные способы создания форм, включая использование Swing, JavaFX или HTML.

**1) Swing:** Swing - это набор библиотек, которые позволяют создавать графический интерфейс пользователя (GUI) в Java. Для создания формы с помощью Swing, вы можете использовать классы, такие как JFrame, JPanel, JTextField, JButton и другие. Ниже приведен пример создания простой формы с помощью Swing:

```
import javax.swing.*;  
  
public class MyForm extends JFrame {  
    public MyForm() {  
        // Настройка окна  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setSize(300, 200);  
  
        // Создание панели  
        JPanel panel = new JPanel();  
  
        // Создание компонентов формы  
        JLabel label = new JLabel("Введите ваше имя:");  
        JTextField textField = new JTextField(20);  
        JButton button = new JButton("Отправить");
```

```
// Добавление компонентов на панель
panel.add(label);
panel.add(textField);
panel.add(button);

// Добавление панели на окно
add(panel);
}

public static void main(String[] args) {
    // Создание и отображение формы
    SwingUtilities.invokeLater(() -> {
        MyForm form = new MyForm();
        form.setVisible(true);
    });
}
```

**2) JavaFX:** JavaFX представляет собой платформу для создания богатых клиентских приложений. Для создания формы с помощью JavaFX, вы можете использовать классы, такие как Stage, Scene, TextField, Button и другие. Ниже приведен пример создания простой формы с помощью JavaFX:

```
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.VBox;  
import javafx.stage.Stage;  
  
public class MyForm extends Application {  
    public static void main(String[] args) {  
        launch(args);  
    }  
  
    @Override  
    public void start(Stage primaryStage) {  
        // Настройка окна  
        primaryStage.setTitle("Моя форма");  
  
        // Создание компонентов формы  
        Label label = new Label("Введите ваше имя:");  
        TextField textField = new TextField();  
        Button button = new Button("Отправить");  
  
        // Создание контейнера  
        VBox vbox = new VBox(10);
```

```
vbox.getChildren().addAll(label, textField, button);
```

```
// Создание сцены
```

```
Scene scene = new Scene(vbox, 300, 200);
```

```
// Назначение сцены на окно и отображение окна
```

```
primaryStage.setScene(scene);
```

```
primaryStage.show();
```

```
}
```

```
}
```

**3) HTML:** HTML является языком разметки, используемым для создания веб-страниц. Для создания формы с помощью HTML, вы можете использовать теги, такие как form, input, label, button, и другие. Ниже приведен пример создания простой формы с использованием HTML:

```
<!DOCTYPE html>
<html>
<head>
<title>Моя форма</title>
</head>
<body>
<form>
<label for="name">Введите ваше имя:</label>
<input type="text" id="name" name="name">
<button type="submit">Отправить</button>
</form>
</body>
</html>
```

## **Задачи для самостоятельного решения**

- 1.** Создайте программу, которая запрашивает у пользователя его имя, возраст и адрес, а затем выводит эту информацию в окне формы Java.
  
- 2.** Создайте программу, которая позволяет пользователю вводить числа в два текстовых поля, а затем находит и выводит результат их суммы и произведения в окне формы Java.
  
- 3.** Создайте программу, которая позволяет пользователю выбрать из двух вариантов (например, "да" или "нет") с помощью кнопок на форме Java. После выбора, программа должна выводить сообщение о выборе пользователя.
  
- 4.** Создайте программу, которая позволяет пользователю вводить текст в текстовое поле, а затем сохраняет этот текст в файл на компьютере пользователя.
  
- 5.** Создайте программу, которая предлагает пользователю выбрать из нескольких вариантов (например, "один", "два" или "три") с помощью выпадающего списка на форме Java. После выбора, программа должна выводить сообщение о выборе пользователя.
  
- 6.** Создайте программу, которая позволяет пользователю выбрать из списка предложенных вариантов с помощью радиокнопок на форме Java. После выбора, программа должна выводить сообщение о выборе пользователя.
  
- 7.** Создайте программу, которая предлагает пользователю ввести свою оценку от 1 до 10 в текстовое поле на форме Java. После ввода, программа должна выводить сообщение, соответствующее оценке пользователя (например, "отлично", "хорошо" или "плохо").
  
- 8.** Создайте программу, которая предлагает пользователю ввести свое имя и отметку (A, B, C, D или F) в текстовые поля на форме Java. После ввода,

программа должна выводить сообщение, соответствующее отметке пользователя (например, "отлично", "хорошо" или "плохо").

**9.** Создайте программу, которая позволяет пользователю выбрать из предложенных вариантов (например, "один", "два" или "три") с помощью флажков на форме Java. После выбора, программа должна выводить сообщение о выборе пользователя.

**10.** Создайте программу, которая предлагает пользователю ввести свою дату рождения (день, месяц и год) в текстовые поля на форме Java. После ввода, программа должна выводить сообщение о возрасте пользователя в годах.

# **Практическая работа № 17. Добавление кнопок, меток, текстовых полей.**

## **1 Цель занятия**

Научиться добавлять кнопки, метки и текстовые поля.

## **2 Теоретический материал**

Для добавления кнопок, меток и текстовых полей в графический интерфейс пользователя используются различные технологии и средства программирования. В данном теоретическом материале мы рассмотрим основные концепции и подходы к добавлению элементов интерфейса в некоторых популярных платформах, таких как веб-приложения и мобильные приложения.

### **1. HTML и CSS для веб-приложений:**

- Для создания кнопки в HTML используется элемент <button>, например:  
<button>Нажать</button>.
- Метки в HTML могут быть добавлены с помощью элемента <label> и атрибута "for", например: <label for="name">Имя</label>.
- Текстовое поле в HTML может быть создано с использованием элемента <input> с атрибутом "type" равным "text", например: <input type="text" id="name" name="name">

### **2. Android для мобильных приложений:**

- В Android Studio кнопка может быть создана с помощью элемента Button в разметке XML, например: <Button android:id="@+id/button" android:text="Нажать" />
- Метка может быть добавлена с помощью элемента TextView, например:  
<TextView android:id="@+id/label" android:text="Имя" />
- Текстовое поле может быть создано с использованием элемента EditText, например: <EditText android:id="@+id/textField" android:inputType="text" />

### **3. iOS для мобильных приложений:**

- В Xcode кнопка может быть создана с помощью элемента UIButton в Interface Builder или программно с использованием кода на Objective-C или Swift.
- Метка может быть добавлена с помощью элемента UILabel в Interface Builder или программно.
- Текстовое поле в iOS может быть создано с использованием элемента UITextField в Interface Builder или программно. Кроме того, для каждой из этих платформ существуют различные библиотеки и фреймворки, которые предоставляют более гибкие и расширенные возможности для создания интерфейсов. Важно также учитывать ограничения и рекомендации по дизайну, которые могут быть определены для конкретной платформы или целевой аудитории приложения.

В Java существует несколько способов добавления кнопок, меток и текстовых полей в графическом интерфейсе пользователя. Самыми популярными способами являются использование классов JButton, JLabel и JTextField из библиотеки javax.swing.

1) Добавление кнопки: Класс JButton предоставляет возможность создания кнопки. Пример:

```
import javax.swing.JButton;
import javax.swing.JFrame;

public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Пример кнопки");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton button = new JButton("Нажми меня");
        frame.getContentPane().add(button);

        frame.pack();
        frame.setVisible(true);
    }
}
```

```
}
```

В данном примере мы создаем окно приложения JFrame, добавляем на него кнопку JButton с надписью "Нажми меня" и отображаем окно.

2) Добавление метки: Класс JLabel позволяет создавать метки. Метки используются для отображения текста или изображения на форме. Пример:

```
import javax.swing.JFrame;
import javax.swing.JLabel;

public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Пример метки");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel label = new JLabel("Привет, мир!");
        frame.getContentPane().add(label);

        frame.pack();
        frame.setVisible(true);
    }
}
```

В данном примере мы создаем окно приложения JFrame и добавляем на него метку JLabel с текстом "Привет, мир!".

3) Добавление текстового поля: Класс JTextField используется для создания текстовых полей, в которые пользователь может вводить текст. Пример:

```
import javax.swing.JFrame;
import javax.swing.JTextField;

public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Пример текстового поля");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JTextField textField = new JTextField(20);
        frame.getContentPane().add(textField);

        frame.pack();
        frame.setVisible(true);
    }
}
```

В данном примере мы создаем окно приложения JFrame и добавляем на него текстовое поле JTextField с шириной 20 символов.

### **Задачи для самостоятельного решения**

1. Создайте графический интерфейс пользователя с помощью Java Swing, который содержит кнопку "Нажми меня".
2. Создайте графический интерфейс пользователя с помощью JavaFX, который содержит метку с текстом "Привет, мир!".
3. Создайте графический интерфейс пользователя с помощью Swing, который содержит текстовое поле и кнопку "Отправить". По нажатию на кнопку программа должна выводить в консоль содержимое текстового поля.

4. Создайте графический интерфейс пользователя с помощью JavaFX, который содержит текстовое поле и кнопку "Посчитать". По нажатию на кнопку программа должна складывать числа, введенные в текстовом поле, и выводить результат на экран.
5. Создайте графический интерфейс пользователя с помощью Swing, который содержит метку "Выберите язык" и выпадающий список с вариантами выбора языка. По выбору определенного языка программа должна выводить "Привет, мир!" на соответствующем языке.
6. Создайте графический интерфейс пользователя с помощью JavaFX, который содержит текстовое поле и две кнопки "Вверх" и "Вниз". По нажатию на кнопки программа должна увеличивать или уменьшать значение, введенное в текстовом поле, на 1.
7. Создайте графический интерфейс пользователя с помощью Swing, который содержит метку, текстовое поле и кнопку "Сохранить". При нажатии на кнопку программа должна сохранять содержимое текстового поля в файл.

## **Практическая работа № 18. Интерфейс формы и размещение компонентов.**

### **1 Цель занятия**

Научиться создавать интерфейс формы и размещения компонентов.

### **2 Теоретический материал**

Интерфейс формы в программировании на Java представляет собой способ взаимодействия пользователя с программой. Он включает в себя размещение компонентов, таких как кнопки, текстовые поля, метки и другие элементы, а также определение их поведения и внешнего вида. Размещение компонентов в Java может происходить с использованием различных способов:

1. Диспетчер размещения **BorderLayout**: Он размещает компоненты на панели в пяти областях: NORTH, SOUTH, EAST, WEST и CENTER. Этот диспетчер обычно используется, когда нужно разместить компоненты по всему экрану, например, для создания приложения с главным окном.
2. Диспетчер размещения **FlowLayout**: Он располагает компоненты один за другим в ряду или столбце в зависимости от ориентации компонента. Этот диспетчер обычно используется, когда нужно разместить компоненты последовательно и в одном ряду.
3. Диспетчер размещения **GridLayout**: Он размещает компоненты в виде сетки со строками и столбцами. Каждая ячейка сетки содержит один компонент. Этот диспетчер часто используется, когда нужно разместить компоненты в виде таблицы.
4. Диспетчер размещения **GridBagLayout**: Он позволяет достичь более гибкого и сложного размещения компонентов. Он использует сетку с ячейками, но позволяет задавать различные параметры для каждой ячейки, такие как ширина, высота, растяжение и выравнивание. Этот диспетчер обычно используется, когда нужно создать сложный и гибкий интерфейс.

Интерфейс формы и размещение компонентов в Java регулируют визуальное представление пользовательского интерфейса приложения. Для создания форм и работе с компонентами в Java обычно используется фреймворк Swing или JavaFX. Основные компоненты пользовательского интерфейса Java включают в себя:

- 1) **JFrame**: Главное окно приложения, на котором размещаются все остальные компоненты. Пример использования JFrame:

```
import javax.swing.JFrame;

public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My App");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

2) **JPanel**: Контейнер, который может содержать другие компоненты. Часто используется для разделения формы на отдельные секции. Пример использования JPanel:

```
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My App");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        frame.add(panel);
    }
}
```

```
    frame.setVisible(true);  
}  
}
```

3) **JButton**: Кнопка, с помощью которой пользователь может взаимодействовать с приложением. Пример использования JButton:

```
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
  
public class Main {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("My App");  
        frame.setSize(400, 300);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        JPanel panel = new JPanel();  
        frame.add(panel);  
  
        JButton button = new JButton("Click me");  
        panel.add(button);  
  
        frame.setVisible(true);  
    }  
}
```

4) **JLabel**: Текстовая метка, используемая для отображения статического текста. Пример использования JLabel:

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My App");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        frame.add(panel);

        JLabel label = new JLabel("Hello, World!");
        panel.add(label);

        frame.setVisible(true);
    }
}
```

### **Задачи для самостоятельного решения**

1. Создайте форму с кнопкой "Добавить" и текстовым полем. При нажатии на кнопку текст из текстового поля должен добавляться в список на форме.
2. Создайте форму с полем для ввода логина и пароля, кнопкой "Войти" и ссылкой на "Регистрацию". При нажатии на кнопку "Войти" программа

должна проверить введенные данные и вывести сообщение об успешной аутентификации или об ошибке.

3. Создайте форму с выпадающим списком, кнопкой "Выбрать" и меткой. При выборе определенного элемента из списка и нажатии на кнопку "Выбрать", текст выбранного элемента должен отображаться в метке.
4. Создайте форму с текстовыми полями для ввода имени, фамилии и адреса. При нажатии на кнопку "Сохранить" программа должна сохранять введенные данные в файл или базу данных.
5. Создайте форму с полем для ввода числа, кнопками "+" и "-", меткой и полем для результатов. При нажатии на кнопку "+" или "-" программа должна увеличивать или уменьшать число на 1 и отображать его в метке.
6. Создайте форму с радиокнопками для выбора пола (мужской/женский), текстовым полем для ввода имени и кнопкой "Отправить". При нажатии на кнопку "Отправить" программа должна сохранять выбранный пол и введенное имя в файл или базу данных.
7. Создайте форму с текстовым полем для ввода URL-адреса, кнопкой "Открыть" и панелью для отображения веб-страницы. При нажатии на кнопку "Открыть" программа должна загружать введенный URL-адрес и отображать содержимое веб-страницы на панели.

## **Практическая работа № 19. Разработка кода обработки событий в учебном проекте.**

### **1 Цель занятия**

Научиться разрабатывать код обработки событий.

### **2 Теоретический материал**

Событие можно определить как изменение состояния объекта или поведения путем выполнения действий. Действиями могут быть нажатие кнопки, перемещение курсора, нажатие клавиши с помощью клавиатуры или прокрутка страницы и т.д. Пакет `java.awt.event` может использоваться для предоставления различных классов событий.

#### **Классификация событий**

- События переднего плана
- Фоновые события

#### **1. События переднего плана**

События переднего плана - это события, для генерации которых требуется взаимодействие с пользователем, т.е. События переднего плана генерируются в результате взаимодействия пользователя с компонентами графического пользовательского интерфейса (GUI). Взаимодействия - это не что иное, как нажатие на кнопку, прокручивание полосы прокрутки, наведение курсора и т.д.

#### **2. Фоновые события**

События, для генерации которых не требуется взаимодействия пользователей, называются фоновыми событиями. Примерами таких событий являются сбои / прерывания операционной системы, завершение операции и т.д. Обработка событий -это механизм для управления событиями и принятия решения о том, что должно произойти после того, как событие произойдет. Для обработки событий Java следует модели делегирования событий.

#### **Модель делегирования событий**

У нее есть исходники и прослушиватели.

- Источник: События генерируются из исходного кода. Для генерации событий существуют различные источники, такие как кнопки, флажки, список, пункт меню, выбор, полоса прокрутки, текстовые компоненты, окна и т.д.
- Прослушиватели: Прослушиватели используются для обработки событий, сгенерированных из исходного кода. Каждый из этих прослушивателей представляет интерфейсы, которые отвечают за обработку событий.

## Классы событий в Java

Класс событий	Интерфейс прослушивателя	Описание
ActionEvent	ActionListener	Событие, указывающее, что произошло определенное компонентом действие, такое как нажатие кнопки или выбор элемента из списка элементов меню.
AdjustmentEvent	AdjustmentListener	Событие настройки генерируется настраиваемым объектом, таким как полоса прокрутки.
ComponentEvent	ComponentListener	Событие, указывающее, что компонент переместился, изменился размер или изменилась его видимость.
ContainerEvent	ContainerListener	Когда компонент добавляется в контейнер (или) удаляется из него, это событие генерируется объектом

<b>Класс событий</b>	<b>Интерфейс прослушивателя</b>	<b>Описание</b>
		контейнера.
FocusEvent	FocusListener	Это события, связанные с фокусировкой, которые включают focus, focusin, focusout и blur.
ItemEvent	ItemListener	Событие, указывающее, был ли выбран элемент или нет.
KeyEvent	KeyListener	Событие, возникающее из-за последовательности нажатий клавиш на клавиатуре.
MouseEvent	MouseListener и MouseMotionListener	События, которые происходят из-за взаимодействия пользователя с мышью (указывающим устройством).
MouseWheelEvent	MouseWheelListener	Событие, указывающее, что колесико мыши было повернуто в компоненте.
TextEvent	TextListener	Событие, возникающее при изменении текста объекта.
WindowEvent	WindowListener	Событие, которое указывает, изменило ли окно свой статус или нет.

**Различные интерфейсы состоят из разных методов, которые указаны ниже.**

Интерфейс прослушивателя	Методы
ActionListener	<ul style="list-style-type: none"><li>• actionPerformed()</li></ul>
AdjustmentListener	<ul style="list-style-type: none"><li>• adjustmentValueChanged()</li></ul>
ComponentListener	<ul style="list-style-type: none"><li>• componentResized()</li><li>• componentShown()</li><li>• componentMoved()</li><li>• componentHidden()</li></ul>
ContainerListener	<ul style="list-style-type: none"><li>• componentAdded()</li><li>• componentRemoved()</li></ul>
FocusListener	<ul style="list-style-type: none"><li>• focusGained()</li><li>• focusLost()</li></ul>
ItemListener	<ul style="list-style-type: none"><li>• itemStateChanged()</li></ul>
KeyListener	<ul style="list-style-type: none"><li>• keyTyped()</li><li>• Нажатая клавиша()</li><li>• keyReleased()</li></ul>
MouseListener	<ul style="list-style-type: none"><li>• mousePressed()</li><li>• mouseClicked()</li><li>• mouseEntered()</li><li>• mouseExited()</li><li>• mouseReleased()</li></ul>
MouseMotionListener	<ul style="list-style-type: none"><li>• mouseMoved()</li><li>• mouseDragged()</li></ul>
MouseWheelListener	<ul style="list-style-type: none"><li>• mouseWheelMoved()</li></ul>

Интерфейс прослушивателя	Методы
TextListener	<ul style="list-style-type: none"> <li>• TextChanged()</li> </ul>
WindowListener	<ul style="list-style-type: none"> <li>• windowActivated()</li> <li>• windowDeactivated()</li> <li>• Открыто окно ()</li> <li>• windowClosed()</li> <li>• windowClosing()</li> <li>• windowIconified()</li> <li>• windowDeiconified()</li> </ul>

## Процесс обработки событий

- Для генерации события требуется взаимодействие пользователя с компонентом.
- Объект соответствующего класса событий создается автоматически после генерации события и содержит всю информацию об источнике события.
- Вновь созданный объект передается методам зарегистрированного прослушивателя.
- Метод выполняется и возвращает результат.

## Подходы к кодированию

Три подхода к выполнению обработки событий заключаются в размещении кода обработки событий в одном из указанных ниже мест.

- Внутри класса
- Другой класс
- Анонимный класс

## Обработка событий внутри класса

```
// Java program to demonstrate the
// event handling within the class
```

```
import java.awt.*;
import java.awt.event.*;

class GFGTop extends Frame implements ActionListener {

    TextField textField;

    GFGTop()
    {
        // Component Creation
        textField = new TextField();

        // setBounds method is used to provide
        // position and size of the component
        textField.setBounds(60, 50, 180, 25);

        Button button = new Button("click Here");
        button.setBounds(100, 120, 80, 30);

        // Registering component with listener
        // this refers to current instance
        button.addActionListener(this);

        // add Components
    }

    public void actionPerformed(ActionEvent e)
    {
        if(e.getSource() == button)
        {
            String str = textField.getText();
            System.out.println(str);
        }
    }
}
```

```
    add(textField);

    add(button);

    // set visibility
    setVisible(true);

}

// implementing method of ActionListener
public void actionPerformed(ActionEvent e)
{
    // Setting text to field
    textField.setText("GFG!");

}

public static void main(String[] args)
{
    new GFGTop();
}

}
```

## Выход



После щелчка значение текстового поля устанавливается равным GFG!

### Обработка событий другим классом

```
// Java program to demonstrate the
// event handling by the other class

import java.awt.*;
import java.awt.event.*;

class GFG1 extends Frame {

    TextField textField;

    GFG2()
    {
        // Component Creation
        textField = new TextField();

        // setBounds method is used to provide
```

```
// position and size of component  
textField.setBounds(60, 50, 180, 25);  
  
Button button = new Button("click Here");  
button.setBounds(100, 120, 80, 30);  
  
Other other = new Other(this);  
  
// Registering component with listener  
// Passing other class as reference  
button.addActionListener(other);  
  
// add Components  
add(textField);  
add(button);  
  
// set visibility  
setVisible(true);  
}  
  
public static void main(String[] args)  
{  
    new GFG2();  
}  
}
```

```
/// import necessary packages  
import java.awt.event.*;
```

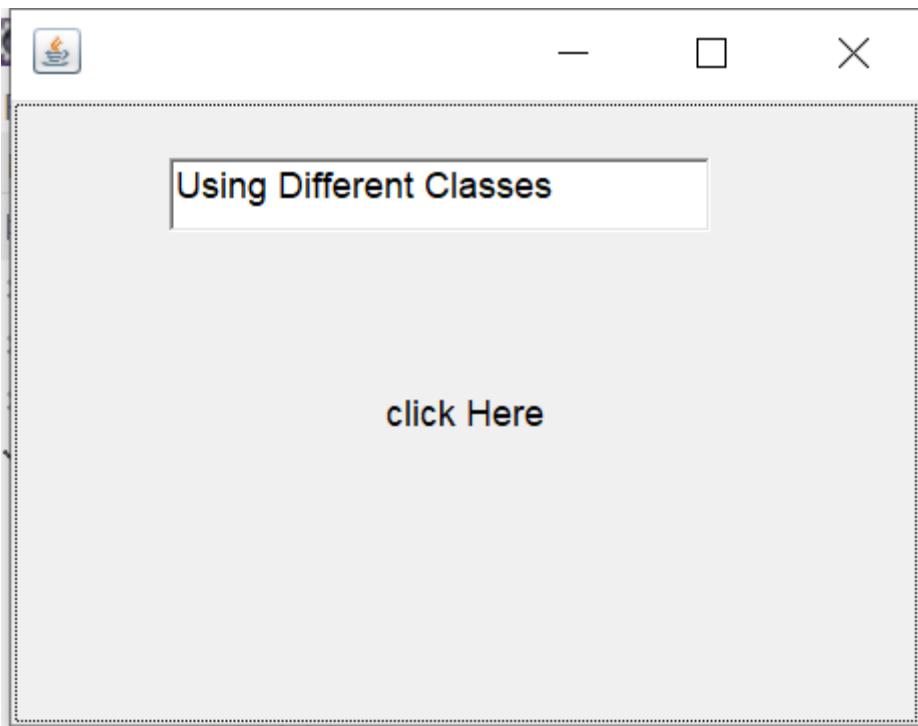
```
// implements the listener interface
class Other implements ActionListener {

    GFG2 gfgObj;

    Other(GFG1 gfgObj) {
        this.gfgObj = gfgObj;
    }

    public void actionPerformed(ActionEvent e)
    {
        // setting text from different class
        gfgObj.textField.setText("Using Different Classes");
    }
}
```

## Выход



Обработка событий из другого класса

### **Обработка событий анонимным классом**

```
// Java program to demonstrate the  
// event handling by the anonymous class  
  
import java.awt.*;  
import java.awt.event.*;  
  
class GFG3 extends Frame {  
  
    TextField textField;  
  
    GFG3()  
    {
```

```
// Component Creation
textField = new TextField();

// setBounds method is used to provide
// position and size of component
textField.setBounds(60, 50, 180, 25);

Button button = new Button("click Here");
button.setBounds(100, 120, 80, 30);

// Registering component with listener anonymously
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        // Setting text to field
        textField.setText("Anonymous");
    }
});

// add Components
add(textField);
add(button);

//make size viewable
setSize(300,300);
// set visibility
setVisible(true);

}

public static void main(String[] args)
```

```
{  
    new GFG3();  
}  
}
```

## Вывод



Обработка анонимно

## Задачи для самостоятельного решения

1. Создание алгоритма сортировки массива целых чисел.
2. Разработка кода для поиска наибольшего общего делителя двух чисел.
3. Написание программы для проверки является ли число простым.
4. Разработка кода для поиска всех простых чисел в заданном диапазоне.
5. Создание алгоритма для вычисления факториала числа.
6. Написание программы для проверки является ли строка палиндромом.
7. Разработка кода для поиска всех подстрок в строке, удовлетворяющих определенному условию.

## 1 Цель занятия

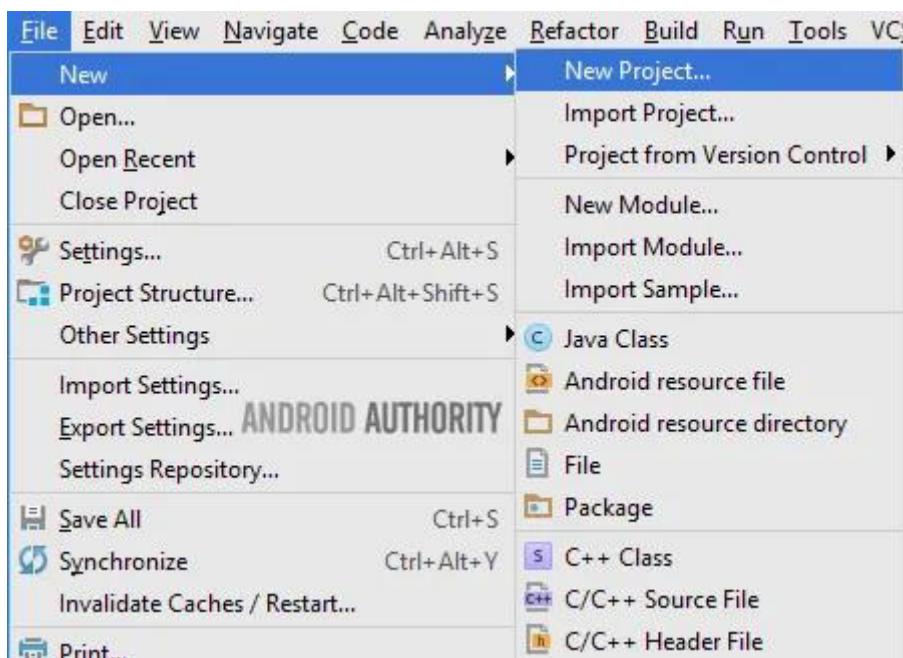
Научиться разрабатывать учебный проект.

## 2 Теоретический материал

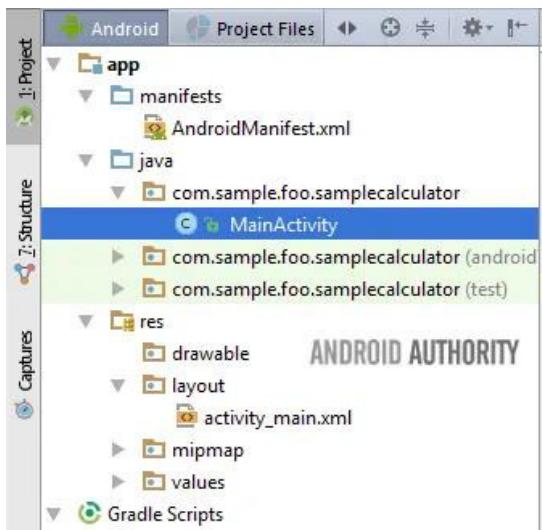
Возьмем к примеру калькулятор и рассмотрим пошаговую инструкцию

### 1) Создание проекта:

Первое, что нужно сделать - это создать в **Android Studio** новый проект: **Start a new Android Studio project** или **File - New - New Project**:



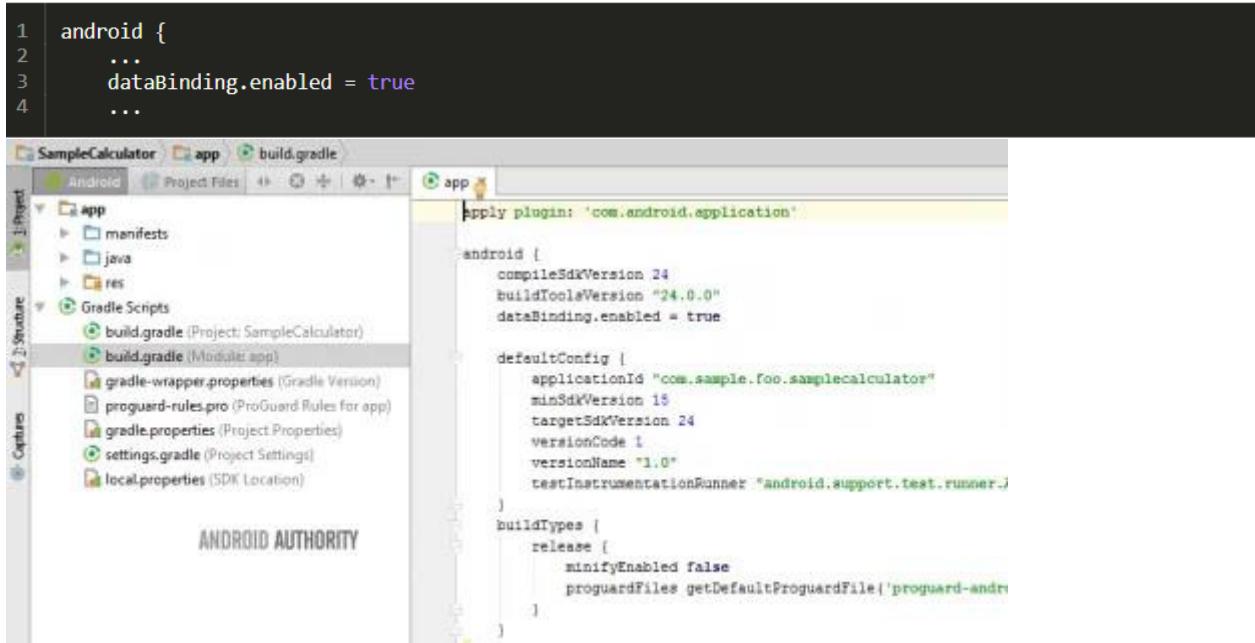
Для этого руководства мы выбрали в панели «*Add an Activity to Mobile*» опцию «*EmptyActivity*», для «*MainActivity*» мы оставили имя по умолчанию – «*Activity*». На этом этапе структура должна выглядеть, как показано на рисунке ниже. У вас есть **MainActivity** внутри пакета проекта и файл **activity\_main.xml** в папке **layout**:



## 2) Включение привязки данных в проекте

Перед тем, как создать приложение для Андроид с нуля, нужно уяснить, что использование привязки данных помогает напрямую обращаться к виджетам (*Buttons*, *EditText* и *TextView*), а не находить их с помощью методов **findViewById()**. Чтобы включить привязку данных, добавить следующую строку кода в файл **build.gradle**.

### JAVA



## 3) Разработка макета калькулятора

Для включения привязки данных в файле **activity\_main.xml** требуется еще одно изменение. Оберните сгенерированный корневой тег (*RelativeLayout*) в **layout**, таким образом сделав его новым корневым тегом.

### XML

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <layout>
3     <RelativeLayout>
4     ...
5     </RelativeLayout>
6 </layout>
```

Тег **layout** - это предупреждает систему построения приложения, что этот файл макета будет использовать привязку данных. Затем система генерирует для этого файла макета класс **Binding**. Поскольку целевой XML-файл называется **activity\_main.xml**, система построения приложения создаст класс **ActivityMainBinding**, который можно использовать в приложении, как и любой другой класс **Java**. Имя класса составляется из имени файла макета, в котором каждое слово через подчеркивание будет начинаться с заглавной буквы, а сами подчеркивания убираются, и к имени добавляется слово «*Binding*».

Теперь перейдите к файлу **MainActivity.java**. Создайте закрытый экземпляр **ActivityMainBinding** внутри вашего класса, а в методе **onCreate()** удалите строку **setContentView()** и вместо нее добавьте **DataBindingUtil.setContentView()**, как показано ниже.

## JAVA

```
1 public class MainActivity extends AppCompatActivity {
2     private ActivityMainBinding binding;
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
7     }
8 }
```

## 4)Общие принципы создания виджетов макета

В приложении калькулятора есть четыре основных элемента:

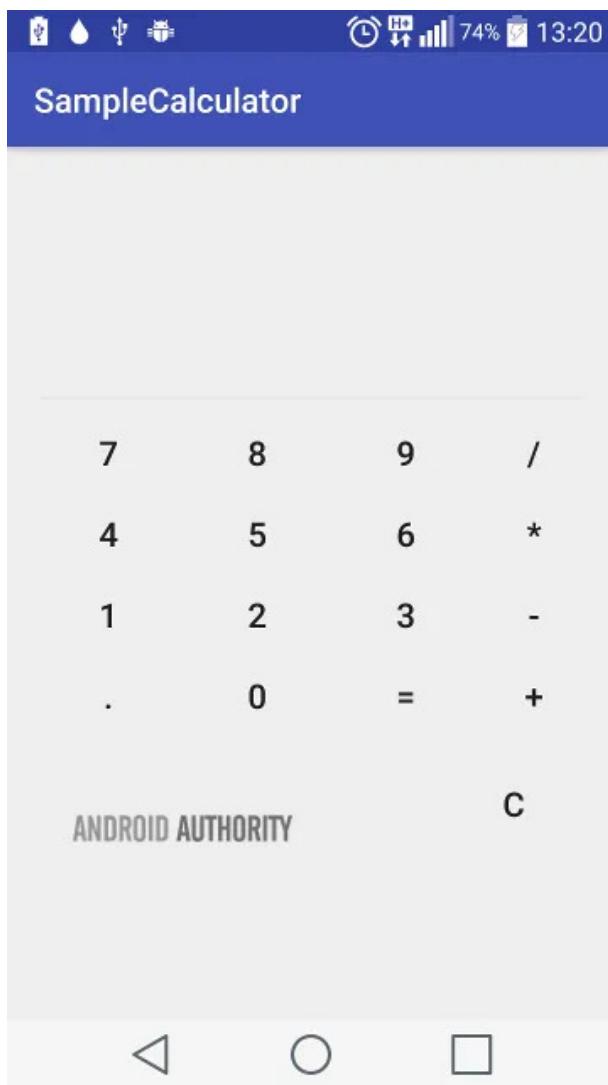
**RelativeLayout** - определяет, как другие элементы будут укладываться или отображаться на экране. **RelativeLayout** используется для позиционирования дочерних элементов по отношению друг к другу или к самим себе.

**TextView** - элемент используется для отображения текста. Пользователи не должны взаимодействовать с этим элементом. С помощью **TextView** отображается результат вычислений.

**EditText** - похож на элемент **TextView**, с той лишь разницей, что пользователи могут взаимодействовать с ним и редактировать текст. Но поскольку калькулятор допускает только фиксированный набор вводимых данных, мы устанавливаем для него статус «*не редактируемый*». Когда пользователь нажимает на цифры, мы выводим их в **EditText**.

**Button** - реагирует на клики пользователя. При создании простого приложения для Андроид мы используем кнопки для цифр и операторов действий в калькуляторе.

## 5) Создание макета калькулятора



Код макета калькулятора объемный. Это связано с тем, что мы должны явно определять и тщательно позиционировать каждую из кнопок интерфейса. Ниже представлен фрагмент сокращенной версии файла

## MAKETA activity\_main:

```
1 <layout>
2   <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:id="@+id/activity_main"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:paddingBottom="@dimen/activity_vertical_margin"
8     android:paddingLeft="@dimen/activity_horizontal_margin"
9     android:paddingRight="@dimen/activity_horizontal_margin"
10    android:paddingTop="@dimen/activity_vertical_margin"
11    tools:context="com.sample.foo.samplecalculator.MainActivity">
12
13   <TextView
14     android:id="@+id/infoTextView"
15     android:layout_width="match_parent"
16     android:layout_height="wrap_content"
17     android:layout_marginBottom="30dp"
18     android:textSize="30sp" />
19
20   <EditText
21     android:id="@+id/editText"
22     android:layout_width="match_parent"
23     android:layout_height="wrap_content"
24     android:layout_below="@id/infoTextView"
25     android:enabled="false"
26     android:gravity="bottom"
27     android:lines="2"
28     android:maxLines="2"
29     android:textSize="20sp" />
30
31   <Button
32     android:id="@+id/buttonSeven"
33     style="@style/Widget.AppCompat.Button.Borderless"
34     android:layout_width="wrap_content"
35     android:layout_height="wrap_content"
36     android:layout_below="@id/editText"
37     android:text="7"
38     android:textSize="20sp" />
39
40   <Button
41     android:id="@+id/buttonEight"
42     style="@style/Widget.AppCompat.Button.Borderless"
43     android:layout_width="wrap_content"
44     android:layout_height="wrap_content"
45     android:layout_below="@id/editText"
46     android:layout_toRightOf="@+id/buttonSeven"
47     android:text="8"
48     android:textSize="20sp" />
49
50   <Button
51     android:id="@+id/buttonNine"
52     style="@style/Widget.AppCompat.Button.Borderless"
53     android:layout_width="wrap_content"
54     android:layout_height="wrap_content"
55     android:layout_below="@id/editText"
56     android:layout_toRightOf="@+id/buttonEight"
57     android:text="9"
58     android:textSize="20sp" />
59
60   ...
61
62   ...
```

```
63
64    <Button
65        android:id="@+id/buttonDot"
66        style="@style/Widget.AppCompat.Button.Borderless"
67        android:layout_width="wrap_content"
68        android:layout_height="wrap_content"
69        android:layout_below="@+id/buttonOne"
70        android:text=". "
71        android:textSize="20sp" />
72
73    <Button
74        android:id="@+id/buttonZero"
75        style="@style/Widget.AppCompat.Button.Borderless"
76        android:layout_width="wrap_content"
77        android:layout_height="wrap_content"
78        android:layout_alignRight="@+id/buttonEight"
79        android:layout_below="@+id/buttonTwo"
80        android:text="0"
81        android:textSize="20sp" />
82
83    <Button
84        android:id="@+id/buttonEqual"
85        style="@style/Widget.AppCompat.Button.Borderless"
86        android:layout_width="wrap_content"
87        android:layout_height="wrap_content"
88        android:layout_alignRight="@+id/buttonNine"
89        android:layout_below="@+id/buttonThree"
90        android:text="="
91        android:textSize="20sp" />
92
93    <Button
94        android:id="@+id/buttonDivide"
95        style="@style/Widget.AppCompat.Button.Borderless"
96        android:layout_width="wrap_content"
97        android:layout_height="wrap_content"
98        android:layout_alignTop="@+id/buttonNine"
99        android:layout_toRightOf="@+id/buttonNine"
100       android:text="/"
101       android:textSize="20sp" />
102
103   <Button
104      android:id="@+id/buttonMultiply"
105      style="@style/Widget.AppCompat.Button.Borderless"
106      android:layout_width="wrap_content"
107      android:layout_height="wrap_content"
108      android:layout_alignTop="@+id/buttonSix"
109      android:layout_toRightOf="@+id/buttonSix"
110      android:text="*"
111      android:textSize="20sp" />
112
113   <Button
114      android:id="@+id/buttonSubtract"
115      style="@style/Widget.AppCompat.Button.Borderless"
116      android:layout_width="wrap_content"
117      android:layout_height="wrap_content"
118      android:layout_alignTop="@+id/buttonThree"
119      android:layout_toRightOf="@+id/buttonThree"
120      android:text="-"
121      android:textSize="20sp" />
122
123   <Button
124      android:id="@+id/buttonAdd"
125      style="@style/Widget.AppCompat.Button.Borderless"
```

```
126         android:layout_width="wrap_content"
127         android:layout_height="wrap_content"
128         android:layout_alignTop="@+id/buttonEqual"
129         android:layout_toRightOf="@+id/buttonEqual"
130         android:text="+"
131         android:textSize="20sp" />
132
133     <Button
134         android:id="@+id/buttonClear"
135         style="@style/Widget.AppCompat.Button.Borderless"
136         android:layout_width="wrap_content"
137         android:layout_height="wrap_content"
138         android:layout_alignRight="@+id/buttonAdd"
139         android:layout_below="@+id/buttonAdd"
140         android:layout_marginTop="@dimen/activity_vertical_margin"
141         android:text="C"
142         android:textSize="20sp" />
143     </RelativeLayout>
144 </layout>
```

## 6)Внутренние компоненты калькулятора

Перед тем, как создать приложение на телефон Android, отметим, что **valueOne** и **valueTwo** содержат цифры, которые будут использоваться. Обе переменные имеют тип **double**, поэтому могут содержать числа с десятичными знаками и без них. Мы устанавливаем для **valueOne** специальное значение **NaN** (*не число*) - подробнее это будет пояснено ниже.

```
1 private double valueOne = Double.NaN;
2     private double valueTwo;
```

Этот простой калькулятор сможет выполнять только операции сложения, вычитания, умножения и деления. Поэтому мы определяем четыре статических символа для представления этих операций и переменную **CURRENT\_ACTION**, содержащую следующую операцию, которую мы намереваемся выполнить.

```
1 private static final char ADDITION = '+';
2     private static final char SUBTRACTION = '-';
3     private static final char MULTIPLICATION = '*';
4     private static final char DIVISION = '/';
5     private char CURRENT_ACTION;
```

Затем мы используем класс **DecimalFormat** для форматирования результата. Конструктор десятичного формата позволяет отображать до десяти знаков после запятой.

```
1 DecimalFormat = new DecimalFormat("#.##########");
```

# Практическая работа № 21. Модификация учебного проекта в Android Studio.

## 1 Цель занятия

Научиться модифицировать учебный проект.

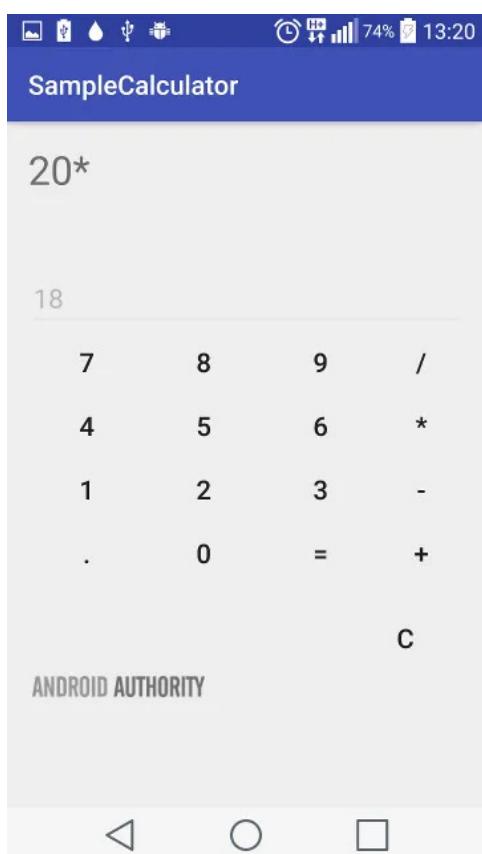
## 2 Теоретический материал

### 7)Обработка нажатий на цифры

В нашем создаваемом простом приложении для Андроид всякий раз, когда пользователь нажимает на цифру или точку, нам нужно добавить эту цифру в **editText**. Пример кода ниже иллюстрирует, как это делается для цифры ноль (0).

```
1 binding.buttonZero.setOnClickListener(new View.OnClickListener() {  
2     @Override  
3     public void onClick(View view) {  
4         binding.editText.setText(binding.editText.getText() + "0");  
5     }  
6 });
```

### 8)Обработка кликов по кнопкам операторов



Обработка нажатия кнопок операторов (*действий*) выполняется по-другому. Сначала нужно выполнить все ожидающие в очереди вычисления. Поэтому мы определяем метод **computeCalculation**. В **computeCalculation**, если **valueOne** является допустимым числом, мы считываем **valueTwo** из **editText** и выполняем текущие операции в очереди. Если же **valueOne** является **NaN**, для **valueOne** присваивается цифра в **editText**.

## JAVA

```
1 private void computeCalculation() {
2     if(!Double.isNaN(valueOne)) {
3         valueTwo = Double.parseDouble(binding.editText.getText().toString());
4         binding.editText.setText(null);
5         if(CURRENT_ACTION == ADDITION)
6             valueOne = this.valueOne + valueTwo;
7         else if(CURRENT_ACTION == SUBTRACTION)
8             valueOne = this.valueOne - valueTwo;
9         else if(CURRENT_ACTION == MULTIPLICATION)
10            valueOne = this.valueOne * valueTwo;
11        else if(CURRENT_ACTION == DIVISION)
12            valueOne = this.valueOne / valueTwo;
13    }
14    else {
15        try {
16            valueOne = Double.parseDouble(binding.editText.getText().toString());
17        }
18        catch (Exception e){}
19    }
20 }
```

Продолжаем создавать копию приложения на Андроид. Для каждого оператора мы сначала вызываем **computeCalculation()**, а затем устанавливаем для выбранного оператора **CURRENT\_ACTION**. Для оператора равно (=) мы вызываем **computeCalculation()**, а затем очищаем содержимое **valueOne** и **CURRENT\_ACTION**.

## JAVA

```

1 binding.buttonAdd.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View view) {
4         computeCalculation();
5         CURRENT_ACTION = ADDITION;
6         binding.infoTextView.setText(decimalFormat.format(valueOne) + "+");
7         binding.editText.setText(null);
8     }
9 });
10 binding.buttonSubtract.setOnClickListener(new View.OnClickListener() {
11     @Override
12     public void onClick(View view) {
13         computeCalculation();
14         CURRENT_ACTION = SUBTRACTION;
15         binding.infoTextView.setText(decimalFormat.format(valueOne) + "-");
16         binding.editText.setText(null);
17     }
18 });
19 binding.buttonMultiply.setOnClickListener(new View.OnClickListener() {
20     @Override
21     public void onClick(View view) {
22         computeCalculation();
23         CURRENT_ACTION = MULTIPLICATION;
24         binding.infoTextView.setText(decimalFormat.format(valueOne) + "*");
25         binding.editText.setText(null);
26     }
27 });
28 binding.buttonDivide.setOnClickListener(new View.OnClickListener() {
29     @Override
30     public void onClick(View view) {
31         computeCalculation();
32         CURRENT_ACTION = DIVISION;
33         binding.infoTextView.setText(decimalFormat.format(valueOne) + "/");
34         binding.editText.setText(null);
35     }
36 });
37 binding.buttonEqual.setOnClickListener(new View.OnClickListener() {
38     @Override
39     public void onClick(View view) {
40         computeCalculation();
41         binding.infoTextView.setText(binding.infoTextView.getText().toString() +
42             decimalFormat.format(valueTwo) + " = " + decimalFormat.format(valueOne));
43         valueOne = Double.NaN;
44         CURRENT_ACTION = '0';
45     }
46 });

```

## 9)Модификация калькулятора

В Android Studio для модификации калькулятора можно использовать различные команды и функции. Например, можно изменить дизайн калькулятора, добавить новые функции или улучшить существующие. Вот несколько примеров команд:

**1. findViewById(R.id.buttonId):** Эта команда используется для получения ссылки на кнопку или другой виджет, используя идентификатор элемента в файле макета. Например, если у вас есть кнопка с идентификатором "button", то вы можете получить ссылку на эту кнопку с помощью следующей команды:

Button button = findViewById(R.id.button);

**2. setOnClickListener():** Эта команда используется для назначения обработчика нажатия на кнопку. Вы можете определить, что будет происходить при нажатии на кнопку, используя анонимный класс OnClickLisener. Например, вы можете добавить следующий код, чтобы выполнить действие при нажатии на кнопку:

```
less Copy code  
  
button.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // Код для выполнения действия при нажатии на кнопку  
    }  
});
```

**3. setText():** Эта команда используется для установки текста в текстовое поле или другой виджет, который может отображать текст. Например, если у вас есть текстовое поле с идентификатором "textView", вы можете установить значение текста с помощью следующей команды:

```
python Copy code  
  
textView.setText("Новый текст");
```

### **Пример модификации калькулятора в Android Studio:**

Допустим, у вас есть калькулятор с кнопками, текстовым полем, отображающим введенные числа и отображающим результат вычислений. Вы можете добавить функциональность к кнопке, чтобы она выполняла определенное вычисление при нажатии.

1) Определите кнопку в файле макета XML, например:

```
python Copy code  
  
<Button  
    android:id="@+id/buttonAdd"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="+"  
/>>
```

2) В вашей активности, найдите ссылку на кнопку и назначьте обработчик нажатия:

SCSS

 Copy code

```
Button buttonAdd = findViewById(R.id.buttonAdd);

buttonAdd.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Получите значения чисел для выполнения операции сложения
        int num1 = Integer.parseInt(editTextNum1.getText().toString());
        int num2 = Integer.parseInt(editTextNum2.getText().toString());

        // Выполните операцию сложения
        int sum = num1 + num2;

        // установите результат в текстовое поле
        textViewResult.setText(String.valueOf(sum));
    }
});
```

Таким образом, при нажатии на кнопку "+", значения чисел из текстовых полей будут сложены, и результат будет отображен в текстовом поле "textViewResult".

## **Практическая работа № 22. Разработка меню в учебном проекте.**

### **1 Цель занятия**

Научиться разрабатывать меню в учебном проекте.

### **2 Теоретический материал**

Меню для калькулятора разрабатывается в Android Studio с использованием XML-файлов и Java-кода. Вот основные шаги для создания меню:

- 1) Создайте новый XML-файл в папке ресурсов вашего проекта. Например, назовите его "menu\_calculator.xml".
- 2) Внутри XML-файла определите элементы меню, используя теги , и т.д. Например, создайте элементы меню "Очистить" и "Выход":

```
php Copy code  
  
<menu xmlns:android="http://schemas.android.com/apk/res/android">  
    <item  
        android:id="@+id/action_clear"  
        android:title="Очистить"/>  
    <item  
        android:id="@+id/action_exit"  
        android:title="Выход"/>  
</menu>
```

- 3) В вашей активности, где расположен калькулятор, создайте метод, который будет обрабатывать выбор элемента меню. Добавьте этот метод в соответствующий Java-код для вашей активности:

java

 Copy code

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.action_clear:  
            // Действия при выборе "Очистить"  
            return true;  
        case R.id.action_exit:  
            // Действия при выборе "Выход"  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

4) Переопределите метод onCreateOptionsMenu() в активности, чтобы создать меню на основе XML-файла:

typescript

 Copy code

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    getMenuInflater().inflate(R.menu.menu_calculator, menu);  
    return true;  
}
```

Теперь, при запуске вашего приложения, вверху экрана появится меню с элементами "Очистить" и "Выход". При выборе элемента будет вызываться соответствующий метод onOptionsItemSelected().

При этом разработка меню в калькуляторе в Android Studio может быть выполнена следующим образом:

- 1)Создайте новый проект в Android Studio с названием "CalculatorApp".
- 2)В файле activity\_main.xml добавьте следующий код для разметки калькулятора:

```
php Copy code

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingTop="16dp"
    android:paddingRight="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/resultTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="24sp"
        android:layout_marginBottom="16dp"
    />

    <Button
        android:id="@+id/number1Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1"
        android:layout_below="@id/resultTextView"
    />
```

```
<Button
    android:id="@+id/number2Button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="2"
    android:layout_toRightOf="@+id/number1Button"
    android:layout_alignTop="@+id/number1Button"
/>

<!-- Добавьте остальные кнопки для остальных чисел -->

</RelativeLayout>
```

3) В файле MainActivity.java добавьте следующий код для обработки нажатия кнопок и для создания меню:

```
public class MainActivity extends AppCompatActivity {

    private TextView resultTextView;
    private Button number1Button;
    private Button number2Button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        resultTextView = findViewById(R.id.resultTextView);
        number1Button = findViewById(R.id.number1Button);
        number2Button = findViewById(R.id.number2Button);

        number1Button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                resultTextView.setText("1");
            }
        });

        number2Button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                resultTextView.setText("2");
            }
        });
    }
}
```

```
// Добавьте методы для обработки нажатий остальных кнопок

// Создание меню
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.calculator_menu, menu);
    return true;
}

// Обработка выбора пункта меню
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.resetMenuItem:
            resultTextView.setText("");
            return true;
        case R.id.exitMenuItem:
            finish();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
}
```

4) В папке res создайте папку menu и внутри нее файл calculator\_menu.xml со следующим содержимым:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/resetMenuItem"
        android:title="Reset" />
    <item
        android:id="@+id/exitMenuItem"
        android:title="Exit" />
</menu>
```

В результате вы получите калькулятор с двумя кнопками для ввода чисел и с двумя пунктами меню: "Reset" и "Exit".

Литература, интернет- издания

Пирская, Л. В. Разработка мобильных приложений в среде Android Studio : учебное пособие / Л. В. Пирская ; Южный федеральный университет. - Ростов-на-Дону ; Таганрог : Издательство Южного федерального университета, 2019. - 123 с. - ISBN 978-5-9275-3346-6. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1894469> (дата обращения: 20.06.2023). – Режим доступа: по подписке.