

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Локтионова Оксана Геннадьевна
Должность: проректор по учебной работе
Дата подписания: 18.06.2023 15:30:03
Уникальный программный ключ:
0b817ca911e6668abb13a5d426d39e5f1c11eabbf73e943df4a4851fda56d089

1

МИНОБРАЗОВАНИЯ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
"Юго-Западный государственный университет"
(ЮЗГУ)

Кафедра биомедицинской инженерии



ТЕОРИЯ И ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ ДЛЯ БИОТЕХНИЧЕСКИХ СИСТЕМ

Методические указания к проведению практических занятий для
студентов направления подготовки 12.03.04 - "Биотехнические
системы и технологии"

Курск 2023

УДК 615.478

Составитель М.В.Артеменко

Рецензент

Кандидат технических наук, доцент *С.С. Шевелев*

Теория и технология программирования для биотехнических систем: методические указания к поведению практических занятий / ЮгоЗап. гос. ун-т; сост. М.В. Артеменко. - Курск, 2023. – 37 с.: ил. 9, табл.2.

Содержатся сведения, необходимые для выполнения практических работ по теории и технологии программирования для биотехнических систем.

Методические указания по структуре, содержанию и стилю изложения материала соответствуют методическим и научным требованиям, предъявляемым к учебным и методическим пособиям.

Предназначены для студентов направления подготовки 12.03.04 очной формы обучений.

Текст печатается в авторской редакции

Подписано в печать

Формат 60x84 1/16

Усл. печ.л. ____ . Уч. -изд.л. __ Тираж ____ экз. Заказ. ____ Бесплатно.

Юго-Западный государственный университет

305040, г.Курск, ул. 50 лет Октября, 94

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
Практическая работа №1. Динамические структуры данных.....	5
Практическая работа №2. GUI, обработка событий	11
Практическая работа №3. Изучение массивов	24
Практическая работа №5. Объектно-ориентированное программирование: классы, объекты и методы	35

ВВЕДЕНИЕ

Основной целью проведения практических занятий является формирование умений и навыков по программированию с использованием современных технологий, включая теорию и технологию программирования для биотехнических систем.

Проведению практических занятий предшествует самостоятельная работа студентов, направленная на ознакомление с соответствующим теоретическим материалом. При необходимости, студенты по заданиям преподавателей выполняют подготовительную работу, обеспечивающую более эффективный процесс закрепления умений и навыков.

Практические занятия проводятся в специализированном классе, оснащенном персональными ЭВМ не ниже Intel i3, с операционной системой не ниже Windows 7. При проведении практического занятия рекомендуется сочетание интерактивного и практического обучения.

Контроль умений и навыков, приобретаемых на практических занятиях осуществляется в форме собеседования.

Содержание практических занятий и объем в часах на каждую тему приведены в таблице 1

Таблица 1 - Практические занятия

№ п/п	Наименование практического занятия	Объем, час.
1.	Динамические структуры данных	4
2.	GUI, обработка событий	4
3.	Изучение массивов	4
4.	Разнородные структуры данных	4
5.	Объекто-ориентированное программирование: классы, объекты и методы	2
Итого		18

Практическая работа №1. Динамические структуры данных

1.1. Цель работы: Введение в использование динамических структур в языке C++.

1.2. Краткие теоретические сведения

Указатель - это переменная, содержащая адрес переменной. Указатели широко применяются в Си - отчасти потому, что в некоторых случаях без них просто не обойтись, а отчасти потому, что программы с ними обычно короче и эффективнее. Указатели и массивы тесно связаны друг с другом: в данной главе мы рассмотрим эту зависимость и покажем, как ею пользоваться. Наряду с goto указатели когда-то были объявлены лучшим средством для написания малопонятных программ. Так оно и есть, если ими пользоваться бездумно. Ведь очень легко получить указатель, указывающий на что-нибудь совсем нежелательное. При соблюдении же определенной дисциплины с помощью указателей можно достичь ясности и простоты. Мы попытаемся убедить вас в этом.

Изменения, внесенные стандартом ANSI, связаны в основном с формулированием точных правил, как работать с указателями. Стандарт узаконил накопленный положительный опыт программистов и удачные нововведения разработчиков компиляторов. Кроме того, взамен `char*` в качестве типа обобщенного указателя предлагается тип `void*` (указатель на `void`). Начнем с того, что рассмотрим упрощенную схему организации памяти. Память типичной машины подставляет собой массив последовательно пронумерованных или проадресованных ячеек, с которыми можно работать по отдельности или связными кусками. Применительно к любой машине верны следующие утверждения:

один байт может хранить значение типа `char`, двухбайтовые ячейки могут рассматриваться как целое типа `short`, а четырехбайтовые - как целые типа `long`. Указатель - это группа ячеек (как правило, две или четыре), в которых может храниться адрес

Унарный оператор `*` есть оператор косвенного доступа. Примененный к указателю он выдает объект, на который данный указатель указывает.

Предположим, что `x` и `y` имеют тип `int`, а `ip` - указатель на `int`. Следующие несколько строк придуманы специально для того, чтобы показать, каким образом объявляются указатели и как используются

```
операторы & и *. int x = 1, y = 2, z[10];
int *ip; /* ip - указатель на int */
ip = &x; /* теперь ip указывает на x */
y = *ip; /* y теперь равен 1 */
*ip = 0; /* x теперь равен 0 */
ip = &z[0]; /* ip теперь указывает на z[0] */
```

Объявления `x`, `y` и `z` нам уже знакомы. Объявление указателя `ip`

```
int *ip;
```

мы стремились сделать мнемоничным - оно гласит: "выражение `*ip` имеет тип `int`". Синтаксис объявления переменной "подстраивается" под синтаксис выражений, в которых эта переменная может встретиться. Указанный принцип применим и в объявлениях функций. Например, запись

```
double *dp, atof(char *);
```

означает, что выражения `*dp` и `atof(s)` имеют тип `double`, а аргумент функции `atof` есть указатель на `char`.

Вы, наверное, заметили, что указателю разрешено указывать только на объекты определенного типа. (Существует одно исключение: "указатель на `void`" может указывать на объекты любого типа, но к

такому указателю нельзя применять оператор косвенного доступа.) Если `ip` указывает на `x` целочисленного типа, то `*ip` можно использовать в любом месте, где допустимо применение `x`; например, `*ip = *ip + 10;` увеличивает `*ip` на 10.

Унарные операторы `*` и `&` имеют более высокий приоритет, чем арифметические операторы, так что присваивание

```
y = *ip + 1;
```

берет то, на что указывает `ip`, и добавляет к нему 1, а результат присваивает переменной `y`. Аналогично

```
*ip += 1;
```

увеличивает на единицу то, на что указывает `ip`; те же действия выполняют

```
++*ip;
```

и

```
(*ip)++;
```

В последней записи скобки необходимы, поскольку если их не будет, увеличится значение самого указателя, а не то, на что он указывает. Это обусловлено тем, что унарные операторы `*` и `++` имеют одинаковый приоритет и порядок выполнения - справа налево.

И наконец, так как указатели сами являются переменными, в тексте они могут встречаться и без оператора косвенного доступа.

Например, если `iq` есть другой указатель на `int`, то `iq = ip;`

копирует содержимое `ip` в `iq`, чтобы `ip` и `iq` указывали на один и тот же объект.

В Си существует связь между указателями и массивами, и связь эта настолько тесная, что эти средства лучше рассматривать вместе.

Любой доступ к элементу массива, осуществляемый операцией

индексирования, может быть выполнен с помощью указателя. Вариант с указателями в общем случае работает быстрее, но разобраться в нем, особенно непосвященному, довольно трудно.

Объявление

```
int a[10];
```

Определяет массив `a` размера 10, т. е. блок из 10 последовательных объектов с именами `a[0]`, `a[1]`, ..., `a[9]`.

Запись `a[i]` отсылает нас к i -му элементу массива. Если `pa` есть указатель на `int`, т. е. объявлен как

```
int *pa;
```

то в результате присваивания `pa = &a[0]`;

`pa` будет указывать на нулевой элемент `a`, иначе говоря, `pa` будет содержать адрес элемента `a[0]`.

Теперь присваивание `x = *pa`;

будет копировать содержимое `a[0]` в `x`.

Если `pa` указывает на некоторый элемент массива, то `pa+1` по определению указывает на следующий элемент, `pa+i` - на i -й элемент после `pa`, а `pa-i` - на i -й элемент перед `pa`. Таким образом, если `pa` указывает на `a[0]`, то

```
*(pa+1)
```

есть содержимое `a[1]`, `pa+i` - адрес `a[i]`, а `*(pa+i)` - содержимое `a[i]`.

Сделанные замечания верны безотносительно к типу и размеру элементов массива `a`. Смысл слов "добавить 1 к указателю", как и смысл любой арифметики с указателями, состоит в том, чтобы `pa+1` указывал на следующий объект, а `pa+i` - на i -й после `pa`.

Между индексированием и арифметикой с указателями существует очень тесная связь. По определению значение переменной или

выражения типа массив есть адрес нулевого элемента массива.

После присваивания

```
pa = &a[0];
```

pa и a имеют одно и то же значение. Поскольку имя массива является синонимом расположения его начального элемента, присваивание $pa = \&a[0]$ можно также записать в следующем виде:

```
pa = a;
```

Еще более удивительно (по крайней мере на первый взгляд) то, что $a[i]$ можно записать как $*(a+i)$. Вычисляя $a[i]$, Си сразу преобразует его в $*(a+i)$; указанные две формы записи эквивалентны. Из этого следует, что полученные в результате применения оператора $\&$ записи $\&a[i]$ и $a+i$ также будут эквивалентными, т. е. и в том и в другом случае это адрес i -го элемента после a . С другой стороны, если pa - указатель, то его можно использовать с индексом, т. е. запись $pa[i]$ эквивалентна записи $*(pa+i)$. Короче говоря, элемент массива можно изображать как в виде указателя со смещением, так и в виде имени массива с индексом.

Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель - это переменная, поэтому можно написать $pa = a$ или $pa++$. Но имя массива не является переменной, и записи вроде $a = pa$ или $a++$ не допускаются.

Если имя массива передается функции, то последняя получает в качестве аргумента адрес его начального элемента. Внутри вызываемой функции этот аргумент является локальной переменной, содержащей адрес. Мы можем воспользоваться отмеченным фактом и написать еще одну версию функции `strlen`, вычисляющей длину строки.

```
/* strlen: возвращает длину строки */
```

```
int strlen(char *s)
{
    int n;
    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}
```

Так как переменная `s` - указатель, к ней применима операция `++`; `s++` не оказывает никакого влияния на строку символов функции, которая обратилась к `strlen`. Просто увеличивается на 1 некоторая копия указателя, находящаяся в личном пользовании функции `strlen`. Это значит, что все вызовы, такие как:

```
strlen("Здравствуй, мир"); /* строковая константа */ strlen(array); /* char
array[100]; */
strlen(ptr); /* char *ptr; */
```

правомерны.

1.3. Задание на практическую работу

Составьте текст программы, которая запрашивает информацию об оценках студентов вашей группы и заносит их в динамический массив, после чего производит поиск максимального, минимального и среднего значения, а также ранжирует оценки по признакам: отлично, хорошо, удовлетворительно и неудовлетворительно

1.4. Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы с динамическим массивом
4. Скриншот экрана

Практическая работа №2. GUI, обработка событий

2.1 Цель работы: Создание приложений форм с использованием компонентов JAVA. Обработка событий и событий с использованием интерфейсов Java.

2.2 Краткие теоретические сведения

Java включает библиотеки, обеспечивающие многоплатформенную поддержку объектов графического интерфейса пользователя. «Мультиплатформенный» аспект заключается в том, что вы можете написать программу для разных операционных систем, таких как Linux, Unix, Mac OS, и у вас появятся одни и те же графические объекты, когда программа будет запущена в UNIX или Windows или Mac OS. Компоненты GUI Java включают метки, текстовые поля, текстовые области, кнопки, всплывающие меню и т.д. В SWING Toolkit также входят контейнеры, которые могут включать эти компоненты. Контейнеры включают фреймы (окна или формы), холсты (которые используются для рисования) и панели (которые используются для группировки компонентов). Панели, кнопки и другие компоненты могут быть размещены либо непосредственно в кадрах, либо в панелях внутри фреймов. Эти компоненты GUI автоматически рисуются всякий раз, когда отображается окно, в котором они находятся. Таким образом, нам не нужно явно вводить команды для их рисования. Действия с этими компонентами GUI обрабатываются с использованием модели событий Java. Когда пользователь взаимодействует с компонентом (щелкает по кнопке, вводит в поле, выбирает из всплывающего меню и т.д.), Событие генерируется компонентом, с которым вы взаимодействуете. Для каждого компонента вашей программы программист должен указывать один

или несколько объектов для «прослушивания» событий из этого компонента. Если обнаружено какое-либо событие, код, связанный с этим событием, должен выполняться автоматически. Название кода, которое будет выполнено в случае какого-либо события, - это «обработчик событий».

Остальная часть этой практической работы обсуждает некоторые компоненты GUI, которые позволяют разработчикам приложений создавать полную программу. В таблице 2 показаны несколько базовых компонентов GUI Swing, которые мы обсуждаем сегодня.

Таблица 2. Список компонентов графического интерфейса пользователя

Название компонента	Описание	Внешний вид
JFrame	Это класс-контейнер, база программы, окно, содержащее все остальные компоненты программы. В .NET (Microsoft Visual Studio)	Hello world .
JButton	JButton - это класс в пакете javax.swing, который представляет кнопки на экране. Этот компонент запускает событие	Click me
JLabel	Отображает неотредактируемый текст и / или значки. В .NET	Hello world
JTextField	Обычно получает вход от пользователя, строку	<u>My name is Ahmad</u>

Компонент JFrame является базой программы, он выглядит как окно, в котором расположены кнопки, метки и другие компоненты. Размер JFrame, цвет, местоположение можно контролировать с помощью

свойств и методов JFrame. Свойства - это некоторые переменные внутри класса, которые отвечают за его внешнее представление. Методы - это команды, которые должны выполняться объектом класса. Наиболее популярными методами класса JFrame являются:

Show(); - показать форму на экране;

GetWidth (); - этот метод возвращает ширину формы, количество пикселей;

SetSize (int, int); - настроить новый размер формы. Два целочисленных параметра описывают соответственно ширину и высоту формы;

GetHeight (); - этот метод возвращает фактическую ширину формы, представленной в пикселях;

SetLocation (int, int); - установить новое положение формы на экране, два целочисленных параметра - X и Y формы;

SetTitle (string); - этот метод изменит название формы, как показано в таблице 1, название «Hello world».

Наиболее популярные свойства и методы компонентов JButton, JLabel, JTextField:

GetText (); - метод, возвращающий текст, отображаемый компонентом; **SetText ("String object");** Определяет текст, который будет отображаться компонентом;

SetCursor (Cursor cursor); Определяет форму курсора, когда указатель мыши будет находиться в границах компонента;

setLocation (x, y); Определяет новое положение компонента в форме согласно параметрам x и y. Этот метод действителен, если ранее не были определены макеты. Тема о макетах (распределение компонентов в форме) будет поднята в следующей практической работе.

getLocation (); Возвращает точку (переменная с типом данных

«point»), которая содержит информацию о местоположении компонента, x и y Все другие методы будут рассмотрены в будущем. Теперь мы можем создать наше первое графическое приложение, как уже упоминалось ранее, JAVA-программа является дочерней (подклассом) класса JFrame, чтобы создать форму, которую вы должны создать подкласс класса JFrame. Следующий пример иллюстрирует создание окна вашей программы:

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

```
import javax.swing.*;
```

```
class MyFrame extends JFrame
```

```
{
```

```
    public MyFrame()
```

```
    { //Default constructor of your class
```

```
        //you can place any code here
```

```
    }
```

```
}
```

```
public class Example
```

```
{ //The class-driver
```

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

```
{
```

```
    MyFrame f = new MyFrame(); // create object of the class
```

```
    f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE
```

```
);
```

```
    f.setSize(250,150); //Set size of the frame f.setVisible( true ); // display  
    frame
```

```
    } // end main
```

```
}
```

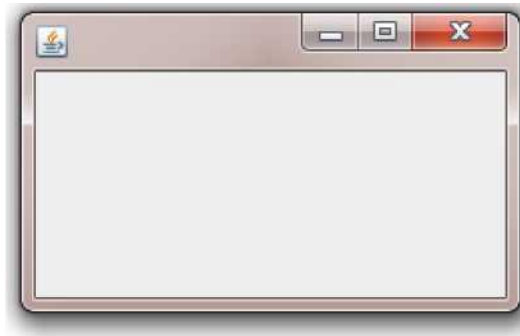


Рисунок 4.1. Форма (окно), созданная кодом

Во-первых, в программе мы создали класс «MyFrame», используя наследование от класса «JFrame». Затем в основном методе класса «Example» был создан объект «f» класса «MyFrame», этот объект является нашей программой. Вы можете управлять размером формы, используя метод `setSize` (ширина, высота), заголовок, используя метод `setTitle` («String object»), положение в соответствии с набором `KOMaHgLocation` (x, y) и многими другими функциями вашей программы. инструкция `f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`; Требуется, чтобы виртуальная машина Java закрывала программу в случае нажатия X формы или нажатия клавиш ALT + F4.

Потому что любая профессиональная программа содержит некоторые другие компоненты, выделенные в Форме. В следующем примере показано, как создать программу, содержащую форму (класс `JFrame`), две кнопки (`JButton`), метку (`JLabel`) и поле для ввода текста (`JTextField`).

(`JButton`), label (`JLabel`) and field to type a text (`JTextField`).

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

```

class MyFrame extends JFrame
{
    JButton button1=new JButton("Click me");//1 JButton button2=new
    JButton("Hello world");//2 JLabel label1=new JLabel("This is a label");//3
    JTextField text1=new JTextField("Type text
here",20);//4
    public MyFrame()
    { super("Hello world");//5
        setLayout(new FlowLayout());//6 this.add(label1);//7
        this.add(text1);//8
        this.add(button1);//9 this.add(button2);//10 }
    }

public class Example
{ //The class-driver
public static void main( String[] args )
{
    MyFrame f = new MyFrame(); // create object of the
class
    f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE
);
    f.setSize(250,150);//Set size of the frame
    f.setVisible( true ); // display frame

} // end main
}

```



Рисунок 4.2. Графический интерфейс, созданный программой

Все важные строки программы нумеруются с использованием

комментариев. Строки 1-4 иллюстрируют определение и создание кнопок, меток и текстовых полей. Строка 5, расположенная внутри конструктора класса, вы можете увидеть инструкцию `super` («Hello words»), которая используется здесь для определения названия формы («Hello world»). Строка 6 `setLayout` (новый `FlowLayout` ()) создает раскладку кадра. Макет - это метод, описывающий, как компоненты будут распределены по форме. Планирование потока означает, что все компоненты распределены слева направо и сверху вниз. Детали макетов будут обсуждаться в следующей практической работе. Строки 7-10 иллюстрируют инструкции по добавлению компонентов в форму: `This.add` («Название компонента»); Распределение компонентов, показанных на рисунке 7.2.

Теперь пользователи вашей программы могут вводить текст внутри текстового поля или кнопки, но никаких действий, связанных с нажатием кнопки, не происходит. Другими словами у нас нет обработчиков событий для нажатия кнопки. Это означает, что очень важной частью графического интерфейса является программирование обработчиков событий.

Событие - это действие, обнаруженное программой. Существует много разных действий, например, нажатие кнопки мыши или перемещение мыши внутри формы или ввод текста с клавиатуры. Иногда некоторые события требуют выполнения некоторого Java-кода. В этом случае код должен быть выделен в специальной функции, которая имеет имя «Обработчик событий». Как правило, в Java-программах существует два способа создания обработчика событий: использование интерфейсов и использование адаптеров классов. Сегодня мы покажем вам, как обрабатывать события с помощью интерфейсов. Интерфейс в Java - это класс, который содержит только конечные переменные и абстрактные методы. В

Java существует много интерфейсов, предназначенных для обработки событий, потому что существует много групп событий. Ниже мы покажем вам наиболее популярные интерфейсы, которые могут быть полезны для вас в будущем.

Интерфейс **ActionListener** находится в пакете `java.awt.event`. Этот интерфейс позволяет обнаруживать действия, связанные с нажатием кнопки. Класс, который заинтересован в обработке события, реализует этот интерфейс, а объект, созданный с помощью этого класса, регистрируется компонентом с

использованием метода

«`addActionListener`» компонента. Когда происходит событие действия, вызывается метод «`actionPerformed`» этого объекта:

Public void actionPerformed (ActionEvent e);

Интерфейс **KeyListener** расположен в пакете `java.awt.event`. *. Этот интерфейс предназначен для приема нажатий клавиш. Интерфейс содержит четыре метода, которые должны быть реализованы внутри вашего класса.

Void keyTyped (KeyEvent e) Вызывается при вводе ключа

Void keyPressed (KeyEvent e) Вызывается при нажатии клавиши.

Void keyReleased (KeyEvent e) Вызывается, когда ключ был освобожден.

Интерфейс **MouseListener**. Этот интерфейс прослушивателя для получения «интересных» событий мыши (нажмите, отпустите, нажмите, введите и выйдите) на компонент. Ваш класс должен реализовать 5 методов, если вы хотите использовать этот интерфейс:

Void mouseClicked (MouseEvent e); Вызывается, когда кнопка

мышь была нажата (нажата и отпущена) на компоненте.

Void mousePressed (MouseEvent e); Вызывается, когда мышь нажата и удерживается

Void mouseReleased (MouseEvent e); Вызывается, когда мышь выпущена
Void mouseEntered (MouseEvent e); - Вызывается, когда мышь входит в компонент.

Void mouseExited (MouseEvent e); - Вызывается, когда мышь выходит из компонента.

Интерфейс MouseMotionListener. Этот интерфейс для приема событий движения мыши на компоненте. Для кликов и других событий мыши используйте интерфейс «MouseListener». Затем объект-слушатель, созданный из этого класса, регистрируется компонентом с использованием метода «addMouseMotionListener» компонента. Событие движения мыши генерируется, когда мышь перемещается или перетаскивается. (Многие такие события будут сгенерированы). Когда происходит событие движения мыши, вызывается соответствующий метод в объекте прослушивателя, и ему передается «MouseEvent». Этот интерфейс содержит два метода:

Void mouseDragged (MouseEvent e); Вызывается, когда кнопка мыши нажата на компоненте, а затем перетаскивается.

Void mouseMoved (MouseEvent e); Вызывается, когда курсор мыши перемещен на компонент, но кнопки не были нажаты.

Следующий пример иллюстрирует программу, которая позволяет нажимать кнопки1 и кнопку2. Если нажата кнопка1, в окне отображается сообщение «Button1 clicked», если кнопка2 нажата, на метке появится сообщение «Button2 clicked».

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

```

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

class MyFrame extends JFrame implements
ActionListener
{
    JButton button1=new JButton("Click me");//1
    JButton button2=new JButton("Hello world");//2
    JLabel label1=new JLabel("This is a label");//3
    JTextField text1=new JTextField("Type text
here",20);//4
    public MyFrame()
    {super("Hello world");//5
      setLayout(new FlowLayout());//6
      this.add(label1);//7
      this.add(text1);//8
      this.add(button1);//9
      this.add(button2);//10
      button1.addActionListener(this);//11
      button2.addActionListener(this);//12
    }
    public void actionPerformed(ActionEvent event) //13
    {if (event.getSource()==button1)//14
      label1.setText("Button1 clicked");//15
      if (event.getSource()==button2)//16
        label1.setText("Button2 was clicked");//17
      text1.setText(event.getActionCommand());//18
    }
}

public class Example
{ //The class-driver
  public static void main( String[] args )
  {
    MyFrame f = new MyFrame(); // create object of the class
    f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    f.setSize(250,150);//Set size of the frame f.setVisible( true ); // display
    frame

  } // end main
}

```



Рисунок 4.3. Внешний вид программы и выполнение обработчиков событий

В приведенной выше программе содержится обработчик событий, связанный с интерфейсом «ActionListener». Внутри интерфейса есть один метод `actionPerformed`, который будет выполняться автоматически, если кнопка будет нажата мышью. Строки № 11 и 12 предоставляют регистрацию обработчика событий для кнопок 1 и 2 кнопок. Внутри обработчика событий с именем `actionPerformed` вы должны распознать объект, который генерирует это событие. Это может быть `кнопка1` или `кнопка2`. Обработчик событий содержит параметр «событие», поле «`getSource ()`» этого параметра возвращает объект, который выполняет это событие. Строки 14 и 16 сравнивают «`getSource ()`» с `кнопкой 1` и `кнопкой2`, если условие становится равным «`true`», объект распознается. Кроме того, вы можете использовать поле `getActionCommand ()`, чтобы получить заголовок нажатой кнопки.

2.3 Задание для практической работы.

1. В приведенном ниже коде представлена простая Java-игра под названием «виртуальная лягушка». Форма пуста, но если вы попытаетесь щелкнуть ее мышью, она перейдет в случайную позицию. Класс «`MyFrame`» реализует интерфейс `MouseMotionListener`, который позволяет обнаруживать движение мыши. Интерфейс содержит два метода `mouseDragged` и

mouseMoved, оба метода должны быть реализованы в классе, даже если вы не используете какой-либо метод. Протестируйте эту программу с помощью Eclipse.

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

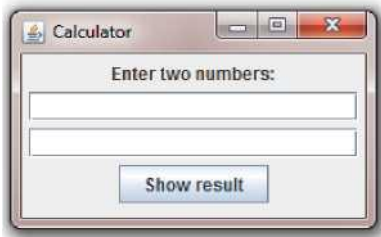
```
class MyFrame extends JFrame implements
    MouseMotionListener
{
    public MyFrame()
    { super("Virtual frog");//5
      this.addMouseMotionListener(this);
    }
    public void mouseDragged(MouseEvent arg0) { }

    public void mouseMoved(MouseEvent arg0) {
    double x=Math.random()*800;
    double y=Math.random()*600;
    this.setLocation((int)x, (int)y);
    } }
}
```

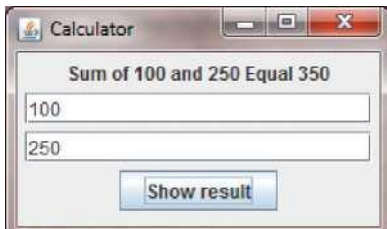
```
public class Example
{ public static void main( String[] args )
  {
    MyFrame f = new MyFrame(); // create object of the class
    f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    f.setSize(250,150);//Set size of the frame
    f.setVisible( true ); // display frame

  } // end main
}
```

2. Создайте простой калькулятор Java, графический интерфейс программы, представленный ниже.



В программе есть метка, которая указывает сообщение «Введите два номера», два поля (JTextField) и кнопку «Показать результат». Пользователь может набирать числа в текстовых полях, затем нажимает кнопку, программа вычисляет сумму и показывает результат в метке, см. Рисунок ниже:



2.4 Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы
4. Скриншот работы приложения

Практическая работа №3. Изучение массивов

3.1. **Цель работы:** изучить и получить практические навыки проектирования программ с использованием массивов данных

3.2. Краткие теоретические сведения

Java предоставляет структуру данных, массив, в котором хранится последовательный набор фиксированного размера элементов того же типа. Массив используется для хранения коллекции данных, но часто бывает полезно придумать массив как набор переменных того же типа.

Вместо объявления отдельных переменных, таких как `number0`, `number1`, ... и `number99`, вы объявляете одну переменную массива, такую как числа, и используете числа `[0]`, числа `[1]` и ..., цифры `[99]` для представления отдельных переменных.

В этом руководстве описывается, как объявлять переменные массива, создавать массивы и массивы процессов с помощью индексированных переменных.

Объявление переменных массива:

Чтобы использовать массив в программе, вы должны объявить переменную для ссылки на массив, и должны указать тип массива, который может ссылаться на переменную. Вот синтаксис объявления переменной массива: `dataType[] arrayRefVar; // preferred way.`

or

`dataType arrayRefVar[]; // works but not preferred way.`

Примечание. Стиль `dataType [] arrayRefVar` является предпочтительным.

Стиль `dataType arrayRefVar []` исходит из языка C / C ++ и был принят на Java для размещения программистов на C / C ++.

Создание массивов:

Вы можете создать массив, используя «новый» оператор со следующим синтаксисом: `ArrayRefVar = new dataType [arraySize];`

Вышеприведенное утверждение делает две вещи:

Он создает массив с использованием нового `dataType [arraySize];`

Он присваивает ссылку вновь созданному массиву переменной `arrayRefVar`.

Объявление переменной массива, создание массива и назначение

ссылки массива переменной могут быть объединены в один оператор, как показано ниже: `DataType [] arrayRefVar = new dataType [arraySize];`

В качестве альтернативы вы можете создавать массивы следующим образом: `DataType [] arrayRefVar = {value0, value1, ..., valuek};`

Элементы массива получают доступ через индекс. Индексы массивов основаны на 0;

То есть они начинаются от 0 до **`arrayRefVar.length-1`**.

Примеры правильного и неправильного объявления и создания массивов:

```
int[] k;//Declaring array k of int data type
```

```
k[0]=5;//Mistake,array declared but not created
```

```
k=new int[5];//Creation of array, k, length equal 5 elements
```

```
k[0]=2;//No mistake, array was created, cell 0 assigned
```

```
int[] k1=new int[5];//No mistake, array k1 declared and created
```

```
//using one line of Java code k1[0]=4;//No mistake, array
created, you can
```

```
//assign a value to the cell 0
```

Обработка массивов:

При обработке элементов массива мы можем использовать любой цикл для доступа ко всем элементам массива, которые имеют одинаковый тип данных. Язык Java- компьютера имеет 4 разных цикла, которые можно использовать для доступа к элементам массива: для `while` и для каждого цикла:

```
for (int i=0;i<10;i++) {/*body of the loop*/}
```

```
while (/*condition*/) {/*body of the loop*/}
```

```
do {/*body of the loop*/} while(/*condition*/);
```

```
for(int i:name_of_array){/*body of the loop*/}
```

В следующем примере сначала мы создадим массив целых чисел, тогда мы будем использовать разные циклы для доступа и распечатки всех элементов заданного массива:

```
public class ArrayTest
```

```
{
```

```

public static void main(String[] args) {
int[] k1={5,4,3,2,1,6,8,9};
//k1 is array of 8 integer numbers
//Example of "for" loop:
for (int i=0;i<k1.length;i++) System.out.print(k1[i]);
System.out.println();//go to next line
//Example of "while" loop:
int j=0;//create integer variable j, set value of j to zero
while(j<k1.length)
{
    System.out.print(k1[j]);
    j++;
}
System.out.println();//go to next line
//Example of "do while" loop:
int a=0;
do
{
    System.out.print(k1[a]);
    a++;
}
while(a<k1.length);
System.out.println();//go to next line
//Example of "for each" loop:
for(int b:k1) System.out.print(b);
System.out.println();//go to next line
} //End of method "main"

```

```
}//End of the class ArrayTest
```

Результатом этой программы будет:

54321689

54321689

54321689

54321689

Вы можете видеть, что каждый цикл предоставляет один и тот же вывод, и в будущем вы можете использовать любой цикл по своему вкусу для доступа к элементам массива, но обычно наиболее популярными циклами в Java являются цикл «for» и «for each» цикла. В приведенной выше программе имя массива k1, вы можете использовать переменную «length », если хотите знать длину массива: k1.length - это целое число, которое возвращает фактическую длину массива.

Сортировка массива очень типичная задача на работу с массивами, наверное каждому (или почти каждому) придет сразу идея отсортировать массив простым перебором. Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N- 1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма).

3.3. Задание на практическую работу

Составьте текст программы, которая запрашивает информацию об оценках студентов вашей группы, после чего производит поиск максимального, минимального и среднего значения, а также ранжирует оценки по признакам: отлично, хорошо, удовлетворительно и неудовлетворительно

3.4. Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы

Практическая работа №4. Разнородные структуры данных

3.1. **Цель работы:** Введение в использование динамических структур в языке Java, позволяющих хранить разнородные данные.

3.2. Краткие теоретические сведения

Разнородные данные - пример реальных задач, когда требуется обеспечить хранение нетипизированной последовательности данных и обеспечить доступ к каждой из записи в случайном порядке. Указатель - это переменная, содержащая адрес переменной. Указатели широко применяются в Си - отчасти потому, что в некоторых случаях без них просто не обойтись, а отчасти потому, что программы с ними обычно короче и эффективнее. Указатели и массивы тесно связаны друг с другом: в данной главе мы рассмотрим эту зависимость и покажем, как ею пользоваться. Наряду с goto указатели когда-то были объявлены лучшим средством для написания малопонятных программ. Так оно и есть, если ими пользоваться бездумно. Ведь очень легко получить указатель, указывающий на что-нибудь совсем нежелательное. При соблюдении же определенной дисциплины с помощью указателей можно достичь ясности и простоты. Мы попытаемся убедить вас в этом.

Изменения, внесенные стандартом ANSI, связаны в основном с формулированием точных правил, как работать с указателями. Стандарт узаконил накопленный положительный опыт программистов и удачные нововведения разработчиков компиляторов. Кроме того, взамен `char*` в качестве типа обобщенного указателя предлагается тип `void*` (указатель на `void`). Начнем с того, что рассмотрим упрощенную схему организации памяти. Память типичной машины подставляет собой массив

последовательно пронумерованных или проадресованных ячеек, с которыми можно работать по отдельности или связными кусками. Применительно к любой машине верны следующие утверждения: один байт может хранить значение типа `char`, двухбайтовые ячейки могут рассматриваться как целое типа `short`, а четырехбайтовые - как целые типа `long`. Указатель - это группа ячеек (как правило, две или четыре), в которых может храниться адрес

Унарный оператор `*` есть оператор косвенного доступа. Примененный к указателю он выдает объект, на который данный указатель указывает. Предположим, что `x` и `y` имеют тип `int`, а `ip` - указатель на `int`. Следующие несколько строк придуманы специально для того, чтобы показать, каким образом объявляются указатели и как используются операторы `&` и `*`.

```
int x = 1, y = 2, z[10];
```

```
int *ip; /* ip - указатель на int */
```

```
ip = &x; /* теперь ip указывает на x */
```

```
y = *ip; /* y теперь равен 1 */
```

```
*ip = 0; /* x теперь равен 0 */
```

```
ip = &z[0]; /* ip теперь указывает на z[0] */
```

Объявления `x`, `y` и `z` нам уже знакомы. Объявление указателя `ip` `int *ip;`

мы стремились сделать mnemonicным - оно гласит: "выражение `*ip` имеет тип `int`". Синтаксис объявления переменной "подстраивается" под синтаксис выражений, в которых эта переменная может встретиться. Указанный принцип применим и в объявлениях функций. Например, запись `double *dp, atof(char *)`; означает, что выражения `*dp` и `atof(s)` имеют тип `double`, а аргумент функции `atof` есть указатель на `char`.

Вы, наверное, заметили, что указателю разрешено указывать только

на объекты определенного типа. (Существует одно исключение: "указатель на void" может указывать на объекты любого типа, но к такому указателю нельзя применять оператор косвенного доступа.) Если `ip` указывает на `x` целочисленного типа, то `*ip` можно использовать в любом месте, где допустимо применение `x`; например, `*ip = *ip + 10;` увеличивает `*ip` на 10.

Унарные операторы `*` и `&` имеют более высокий приоритет, чем арифметические операторы, так что присваивание

```
y = *ip + 1;
```

берет то, на что указывает `ip`, и добавляет к нему 1, а результат присваивает переменной `y`. Аналогично

```
*ip += 1;
```

увеличивает на единицу то, на что указывает `ip`; те же действия выполняют `++*ip;`

и

```
(*ip)++;
```

В последней записи скобки необходимы, поскольку если их не будет, увеличится значение самого указателя, а не то, на что он указывает. Это обусловлено тем, что унарные операторы `*` и `++` имеют одинаковый приоритет и порядок выполнения - справа налево.

И наконец, так как указатели сами являются переменными, в тексте они могут встречаться и без оператора косвенного доступа.

Например, если `iq` есть другой указатель на `int`, то `iq = ip;` копирует содержимое `ip` в `iq`, чтобы `ip` и `iq` указывали на один и тот же объект.

В Си существует связь между указателями и массивами, и связь эта

настолько тесная, что эти средства лучше рассматривать вместе. Любой доступ к элементу массива, осуществляемый операцией индексирования, может быть выполнен с помощью указателя. Вариант с указателями в общем случае работает быстрее, но разобраться в нем, особенно непосвященному, довольно трудно.

Объявление

```
int a[10];
```

Определяет массив `a` размера 10, т. е. блок из 10 последовательных объектов с именами `a[0]`, `a[1]`, ..., `a[9]`.

Запись `a[i]` отсылает нас к i -му элементу массива. Если `pa` есть указатель на `int`, т. е. объявлен как

```
int *pa;
```

то в результате присваивания

```
pa = &a[0];
```

`pa` будет указывать на нулевой элемент `a`, иначе говоря, `pa` будет содержать адрес элемента `a[0]`.

Теперь присваивание

```
x = *pa;
```

будет копировать содержимое `a[0]` в `x`.

Если `pa` указывает на некоторый элемент массива, то `pa+1` по определению указывает на следующий элемент, `pa+i` - на i -й элемент после `pa`, а `pa-i` - на i -й элемент перед `pa`. Таким образом, если `pa` указывает на `a[0]`, то

```
*(pa+1)
```

есть содержимое `a[1]`, `pa+i` - адрес `a[i]`, а `*(pa+i)` - содержимое `a[i]`.

Сделанные замечания верны безотносительно к типу и размеру элементов массива `a`. Смысл слов "добавить 1 к указателю", как и смысл любой арифметики с указателями, состоит в том, чтобы `ra+1` указывал на следующий объект, а `ra+i` - на *i*-й после `ra`.

Между индексированием и арифметикой с указателями существует очень тесная связь. По определению значение переменной или выражения типа массив есть адрес нулевого элемента массива. После присваивания `ra = &a[0]`; `ra` и `a` имеют одно и то же значение. Поскольку имя массива является синонимом расположения его начального элемента, присваивание `ra=&a[0]` можно также записать в следующем виде: `ra = a`;

Еще более удивительно (по крайней мере на первый взгляд) то, что `a[i]` можно записать как `*(a+i)`. Вычисляя `a[i]`, Си сразу преобразует его в `*(a+i)`; указанные две формы записи эквивалентны. Из этого следует, что полученные в результате применения оператора `&` записи `&a[i]` и `a+i` также будут эквивалентными, т. е. и в том и в другом случае это адрес *i*-го элемента после `a`. С другой стороны, если `ra` - указатель, то его можно использовать с индексом, т. е. запись `ra[i]` эквивалентна записи `*(ra+i)`. Короче говоря, элемент массива можно изображать как в виде указателя со смещением, так и в виде имени массива с индексом.

Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель - это переменная, поэтому можно написать `ra=a` или `ra++`. Но имя массива не является переменной, и записи вроде `a=ra` или `a++` не допускаются.

Если имя массива передается функции, то последняя получает в качестве аргумента адрес его начального элемента. Внутри вызываемой функции этот аргумент является локальной переменной, содержащей адрес. Мы можем воспользоваться

отмеченным фактом и написать еще одну версию функции `strlen`, вычисляющей длину строки.

```
/* strlen: возвращает длину строки */ int strlen(char *s)
{
    int n;
    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}
```

Так как переменная `s` - указатель, к ней применима операция `++`; `s++` не оказывает никакого влияния на строку символов функции, которая обратилась к `strlen`. Просто увеличивается на 1 некоторая копия указателя, находящаяся в личном пользовании функции `strlen`.

Это значит, что все вызовы, такие как:

```
strlen("Здравствуй, мир"); /* строковая константа */ strlen(array); /* char
array[100]; */
strlen(ptr); /* char *ptr; */
```

правомерны.

3.3. Задание на практическую работу

Составьте текст программы, которая запрашивает информацию об оценках студентов вашей группы и заносит их в динамический массив, после чего производит поиск максимального, минимального и среднего значения, а также ранжирует оценки по признакам: отлично, хорошо, удовлетворительно и неудовлетворительно

3.4. Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы с динамическим массивом
4. Скриншот экрана

Практическая работа №5. Объектно-ориентированное программирование: классы, объекты и методы

5.1. Цель работы: изучить основы объектно-ориентированного программирования, классов, объектов, методов, приватных, защищенных и публичных членов классов

5.2. Краткие теоретические сведения

В практическом случае для любого класса компьютерного языка используется набор элементов данных и методов под именем. Элементами данных являются любые структуры данных внутри класса: переменные, такие как `int`, `float`, `double`, `short`; Массивы некоторых элементов; Объекты некоторых других классов, таких как `String`, `StringBuilder`, `ArrayList`. Методы класса - это члены функции любой функции внутри класса. Итак, просто вы можете сказать, что класс - это набор переменных и набор функций, расположенных вместе в одном месте под некоторым именем. Класс имеет имя; Это похоже на некоторый тип данных. В другой руке у нас есть объект - экземпляр класса, объект выглядит как переменная некоторого типа данных. Пример: `Int a;` // `a` - переменная типа данных `int`;

`String st;` // `st` - объект класса `String`.

Пример объявления класса в Java:

```
class MyClass {
    // field, // constructor, and declarations }
    method
```

В примере выше описан новый класс с именем «`MyClass`». Внутри класса вы можете выделить набор переменных, массивов и объектов других классов. «`class`» - это ключевое слово Java. Перед этим ключевым словом вы можете написать префикс

«public». Слово «public» для классов в Java означает, что этот класс доступен из других классов, из других пакетов и из других программ.

Класс может иметь конструктор. Конструктор - это метод класса (функция внутри класса), который имеет одно и то же имя с классом. Конструктор будет выполняться автоматически в любое время, когда будет создан объект класса. Класс может не иметь конструктора, может иметь один конструктор из нескольких конструкторов. Последний факт, называемый перегрузкой конструктора - ситуация, в которой у вас есть много конструкторов внутри класса с тем же именем (что совпадает с именем класса), но с разными параметрами.

Теперь вы знаете, что в Java, как в C ++, есть конструкторы. Но в отличие от C ++ в Java нет деструкторов. Деструктор в C ++ - это метод, который имеет одно и то же имя с префиксом класса плюс ~ перед именем. Деструктор в C ++ - это функция, которая будет выполняться автоматически в любое время, когда объект класса будет уничтожен (используя ключевое слово «удалить»). В Java нет указателей, исключение ключевых слов, отсутствие множества памяти и никаких деструкторов. Все объекты выделены не в множестве памяти, а в некоторой другой памяти виртуальной машины Java, называемой «сбор мусора». Это означает, что программист может создавать объект, но не может его уничтожить. Программист может сообщить о сборе мусора, что какой-то объект больше не нужен, назначив значение «null» объекту, но физически этот объект будет уничтожен последним, может быть через один час, время разрушения непредсказуемо на Java. Вот почему никакие деструкторы в Java не гарантируют, что объект будет уничтожен в будущем. Если вы хотите, то можете использовать деструктор,

представленный на Java, с помощью метода «finalize», но ни один орган не может гарантировать, что деструктор будет выполнен в вашей программе, поэтому в практических программах Java нет причин использовать деструкторы.

Пример прояснит ситуацию с классами, конструкторами, деструкторами и объектами.

5.3. Задание для практической работы.

Создать класс, который позволяет хранить оценки студентов вашей группы. В классе должна быть структура для хранения, а так же набор методов, обеспечивающих следующие функции:

1. Ввод данных
2. Сортировка
3. Печать данных
4. Поиск минимума, максимума и среднего

5.4 Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы
4. Скриншот работы приложения