**ПРИЛОЖЕНИЕ. Краткое введение в язык Java SDK**

Настоящее приложение имеет целью способствовать лучшему пониманию текстов программ, помещенных в пособии, и содержит основные сведения по базовому уровню языку Java. Этот язык использует синтаксис языка cи. В связи с этим мы опускаем вопросы, связанные с «азами» языка (типы данных, управляющие конструкции, массивы и пр.). Java не использует указателей и оператора goto. У него нет необходимости использовать деструктор объекта явным образом. Для наших целей эти детали, впрочем, не нужны.

Имеется две основные среды программирования на языке Java – Java SDK (Standard development kit) и JEE (Java Enterprise Edition). Java SDK – это и есть «стандартный» Java. Java EE – это «корпоративный» Java, который используется на предприятиях для разработки распределенных и сетевых систем. Большинство примеров этого пособия относилось к Java EE. Однако знание Java SDK необходимо в качестве первой ступени владения языком. Для освоения используем среду NetBeans (свободно скачиваемую в Интернет).

**П1. Установка Java и NetBeans**

Сначала устанавливаем Java. Скачиваем файл java\_ee\_sdk-6u4-jdk7-windows.exe

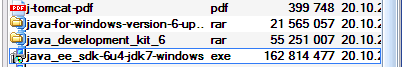


Рисунок 133. Инсталляция Java SDK

и запускаем его. Затем скачиваем и устанавливаем среду выполнения java (JRE):



Рисунок 134. Инсталляция Java JRE

Затем скачиваем и устанавливаем NetBeans:



Рисунок 135. Инсталляция Java NetBeans

Запускаем NetBeans. Его стартовое окно (русско-язычная версия) имеет следующий вид (рисунок 136)

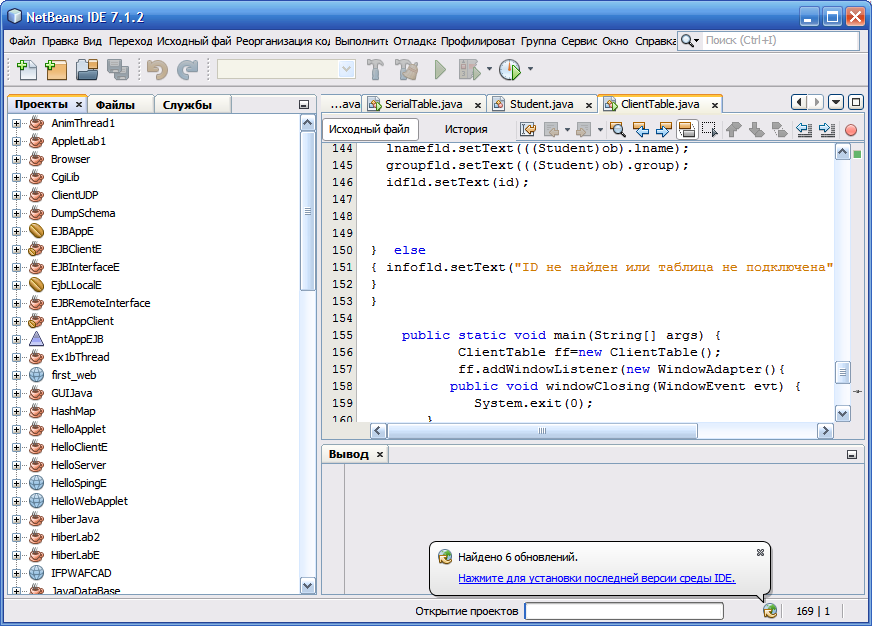


Рисунок 136. Стартовое окно для создания проектов в NetBeans

В левой части экрана имеются три важнейшие закладки: Проекты, Файлы, Службы. Для создания проекта выбираем опцию главного меню Файл, затем выбираем опцию Создать проект. Открывается следующее окно (hbceyjr 137)

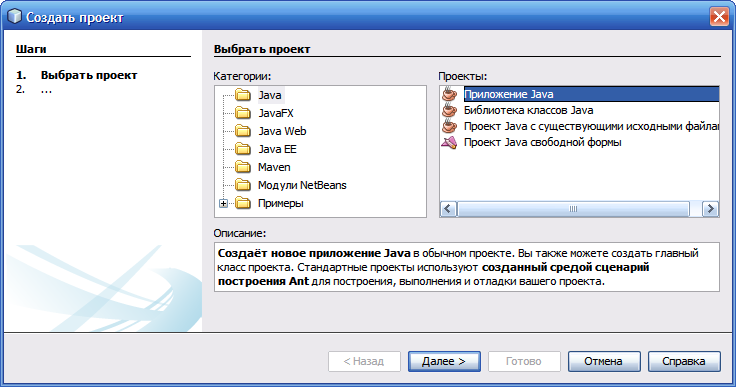


Рисунок 137. Создание простого приложения Java

Указываем тип проекта – Приложение Java (другими типами проектов, которые мы используем в пособии, являются Java Web и Java EE). Затем вводим имя проекта. В дереве проекта новый проект представлен следующим образом (раскрываем значок «+» с именем проекта)

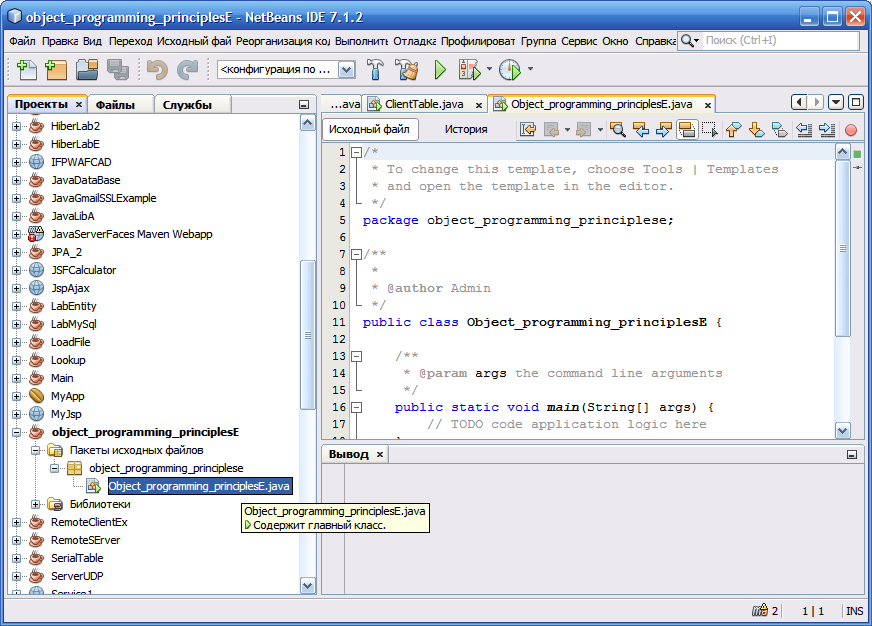


Рисунок 138. Дерево проекта

Щелкаем по имени файла object\_programming\_principlesE.java. Открывается окно редактора, где и набираем нужный нам код.

Для выполнения набранной программы следует в контекстном окне, вызываемом щелчком правой кнопки мыши на имени проекта, выбрать опцию Очистить и построить, а затем – Выполнить. Обычно исходные java-классы помещают в пакеты. Пакет подключается с помощью директивы package. Для создания пакета используем контекстное меню, которое открываем на нужном узле дерева проекта.

Использование закладок Файлы и Службы рассматривалось в основном материале пособия и для знакомства с основами не требуется.

**П2. Объектные принципы Java**

**Консольные приложения**

Java приложение представляет собой один или несколько классов. В качестве первого примера рассмотрим следующий.

**Пример 1**. Простое использование класса.

package object\_programming\_principlese;

import java.awt.\*;

import java.util.Scanner;

class SimpleArithmetic

{

int Add(int par1,int par2)

{

return par1+par2;

}

}

public class Object\_Programming\_PrinciplesE {

public static void main(String[] args) {

System.out.println("Input first number");

Scanner input= new Scanner(System.in);

int n1=input.nextInt();

System.out.println("Input second number");

int n2=input.nextInt();

SimpleArithmetic sa=new SimpleArithmetic();

System.out.printf("The sum of %d and %d is %d",n1,n2,sa.Add(n1, n2));

}

}

Данная программа порождает следующее окно (рисунок 139) с результатами работы

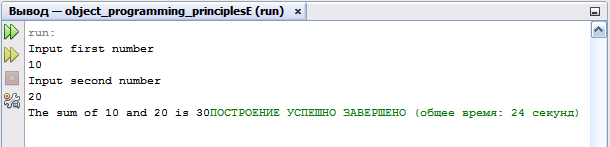


Рисунок 139. Выходные результаты программы

В приведенном приложении объявлена два класса:

SimpleArithmetic и Object\_Programming\_PrinciplesE,

объединенных в пакет object\_programming\_principles. Пакет объявляется как самая первая строка программы. Если пакет не объявлен, то он используется как анонимный (без имени). Следовательно, пакет является вариантом объединения классов. Классы в одном и том же пакете «видят» друг друга, т.е. могут использовать друг друга, если это позволяют их уровни доступа. Уровень доступа public является предельно общим и делает класс доступным для использования в других классах. Уровень доступа private, напротив, делает класс недоступным для использования в других классах. Уровень доступа protected делает класс доступным только в дочерних классах. Если не объявлять спецификатор доступа, то класс будет «видим» в пределах папки (каталога – folder). Класс, объявленный как public, должен содержать точку входа – метод main, с которого начинается его выполнение. Классы состоят из методов и переменных – членов класса. В примере выше класс SimpleArithmetic содержит метод

Add, а класс Object\_Programming\_PrinciplesE – метод main. Метод должен возвращать значение определенного типа и, как правило, использует аргументы. Так, метод int Add(int par1,int par2) возвращает значение целого типа (int) и использует два параметра par1, par2 также целого типа. Метод заканчивается командой return, которая и возвращает ответ. В методе может быть несколько команд return. Классы и методы должны записываться в фигурных скобках.

Классы могут использоваться как сами по себе, так и создавать объекты (экземпляры классов). Если требуется использовать объект класса, то он должен быть создан. Объекты создаются с помощью конструкторов классов – методов, имена которых совпадают с именами классов. В нашем примере объект класса SimpleArithmetic создавался в команде

SimpleArithmetic sa=new SimpleArithmetic();

При обращении к конструктору обязательно указывается ключевое слово new. Конструкторы в общем случае должны объявляться в классах, однако если конструктор не содержит параметров, то его можно не объявлять (действует объявление по умолчанию – by default). В рассматриваемом примере именно это имеет место.

Если хотим использовать классы без объектов, то нужно объявлять методы как static. Этот вариант рассмотрим позже. Имеются стандартные классы Java, подключаемые с помощью библиотечных пакетов. Подключение их реализуется через команду import. В рассматриваемом примере использованы стандартные классы System.out и Scanner. Оба используются для консольного ввода-вывода. Они подключаются посредством команд

import java.awt.\*;

import java.util.Scanner;

Объект класса Scanner используется для ввода значения с клавиатуры. Создаем этот объект таким образом

Scanner input= new Scanner(System.in);

Читаем значения параметров с помощью этого объекта ( с именем ) так

int n1=input.nextInt();

и

int n2=input.nextInt();

Класс используется для вывода результатов/текста на консоль, например

System.out.println("Input second number");

Для вывода используется метод println.

Итак, рассматриваемое приложение начинается с метода main и приглашает ввести два числа. Затем создается объект класса SimpleArithmetic, вызывается метод sa.Add(n1, n2) (с указанием в качестве параметров двух введенных целых чисел n1, n2) этого объекта. Заметим, что обращение к методу sa.Add выполняется внутри другого метода

System.out.printf("The sum of %d and %d is

%d",n1,n2,sa.Add(n1, n2));

Вообще, System.out.printf используется для форматированного вывода. Специальные обозначения %d (place holders) используются для подстановки вместо них параметров, перечисляемых после закрывающих двойных кавычек. Маленькая литера d указывает, что подставляется значение целого типа.

Теперь рассмотрим тот же пример с использованием статических методов (static)

package object\_programming\_principlese;

import java.awt.\*;

import java.util.Scanner;

class SimpleArithmetic

{

static int Add(int par1,int par2)

{

return par1+par2;

}

}

public class Object\_Programming\_PrinciplesE {

public static void main(String[] args) {

System.out.println("Input first number");

Scanner input= new Scanner(System.in);

int n1=input.nextInt();

System.out.println("Input second number");

int n2=input.nextInt();

//SimpleArithmetic sa=new SimpleArithmetic();

System.out.printf("The sum of %d and %d is %d",n1,n2,SimpleArithmetic.Add(n1, n2));

}

}

Здесь имеется статическое объявление

static int Add(int par1,int par2)

{

return par1+par2;

}

Статически объявленные методы нельзя использовать в объектах. Поэтому ранее использованная строка

//SimpleArithmetic sa=new SimpleArithmetic();

Записана через знаки комментария (//). При вызове статически объявленного метода перед его именем указывается имя класса:

SimpleArithmetic.Add(n1, n2))

Важным вариантом класса является интерфейс. В интерфейсе методы только объявляются, но не реализуются. Класс, в котором методы интерфейса должны быть реализованы, подключает интерфейс с помощью директивы implements. Предыдущий пример, реализованный на основе интерфейса, представлен таким образом

package object\_programming\_principlese;

import java.awt.\*;

import java.util.Scanner;

interface SimpleArithmetic

{

public int Add(int par1,int par2);

}

public class Object\_Programming\_PrinciplesE

implements SimpleArithmetic

{

public int Add(int par1, int par2)

{

return par1+par2;

}

public static void main(String[] args) {

System.out.println("Input first number");

Scanner input= new Scanner(System.in);

int n1=input.nextInt();

System.out.println("Input second number");

int n2=input.nextInt();

System.out.printf("The sum of %d and %d is %d",n1,n2,new

Object\_Programming\_PrinciplesE().Add(n1, n2));

}

}

Подключение интерфейса реализовано в строке

public class Object\_Programming\_PrinciplesE

implements SimpleArithmetic

**Приложения на основе окон (Frames)**

Обратимся к приложениям на основе окон (форм). В простейшем случае оконные приложения следует создавать самостоятельно, включая операторы для отображения окон (форм), кнопок, списков, меню и других элементов визуального интерфейса. Ниже помещен пример приложения с кнопкой, которая обеспечивает завершение работы программы.

package win\_lab;

import java.awt.\*;

import java.awt.event.\*;

public class Win\_Lab extends Frame implements ActionListener{

Button bex=new Button("Exit");

public Win\_Lab()

{

super("my window");

setLayout(null);

setBackground(new Color(150,200,100));

setSize(250,250);

add(bex);

bex.setBounds(110,190,100,20);

bex.addActionListener(this);

this.show();

this.setLocationRelativeTo(null);

}

public void actionPerformed(ActionEvent ae)

{

if(ae.getSource()==bex)

System.exit(0);

}

public static void main(String[] args)

{

new Win\_Lab();

}

}

Результат работы этой программы показан на рисунке 140



Рисунок 140. Приложение на основе окна

Приложение содержит единственный класс, но его объявление расширено:

public class lab1\_2 extends Frame implements ActionListener

Указывается, что класс имеет имя Win\_Lab, тип доступа public и расширяет (наследуется от) класс Frame. Класс Frame является стандартным оконным классом. Механизм наследования позволяет расширять одни классы за счет других, автоматически сохраняя методы и члены наследуемого класса с правами доступа типа public (и в ограниченном контексте protected). Приведенное выше объявление позволяет нашему классу Win\_Lab автоматически стать оконным классом, скрывая подробности, связанные с прорисовкой и отображением окна. Класс Win\_Lab использует интерфейс ActionListener. Выше было указано, что интерфейс – это класс, содержащий только объявления методов без их реализации. Интерфейс ActionListener используется для прослушивания событий от кнопок и меню. Для других элементов используются другие интерфейсы.

Обратимся теперь к структуре класса Win\_Lab. Представлено объявление единственной кнопки

Button bex=new Button("Exit");

Тип кнопки Button. Кнопка имеет программное имя bex. Кнопка не только объявлена, но и создана с помощью конструктора класса Button. В вызове конструктора указывается в качестве параметра название кнопки. Конструктор класса Win\_Lab объявлен и реализован следующим образом

public Win\_Lab()

{

super("my window");

setLayout(null);

setBackground(new Color(150,200,100));

setSize(250,250);

add(bex);

bex.setBounds(110,190,100,20);

bex.addActionListener(this);

this.show();

this.setLocationRelativeTo(null);

}

Назначение конструктора – инициализировать форму, т.е. задать размеры, цвет фона, разместить элементы интерфейса и др. Цвет фона задает команда

setBackground(new Color(150,200,100)).

Размеры формы устанавливает команда

setSize(250,250);

Команда

this.show();

делает форму видимой.

Кнопка добавляется на форму командой: add(bex). С помощью команды

bex.setBounds(110,190,100,20);

кнопка отображается на форме в требуемой позиции и с указанными размерами (в пискселах). Аргументы задают в данном случае координаты левого верхнего угла кнопки: (110,190), а также ее размеры (ширину и высоту) – 100,20. Прослушиватель (listener) подключается к кнопке командой:

bex.addActionListener(this);

Прослушиватель позволяет «реагировать» на нажатие кнопки. Если его не подключить, то нажатие на кнопку не вызовет никаких действий. Обработка событий от кнопки (и других элементов, которые «прослушивает» интерфейс ActionListener, реализуется с помощью метода public void actionPerformed(ActionEvent ae). Аргументом этого метода является объект класса ActionEvent. При обработке события от кнопки выполняется проверка источника события:

if(ae.getSource()==bex)

System.exit(0);

Метод getSource() возвращает программное имя источника события. Если имя источника события совпадает с программным именем кнопки (в нашем примере – это bex), то выполняется завершение приложения (System.exit(0)). Следует заметить, что просто щелчком на иконке  нельзя выполнить ее завершение.

**П3. Обработка исключений**

Важную роль играет обработка исключений (ошибок). Как правило, программные ошибки вызывают завершение приложение и выдачу диагностической информации. Во многих случаях при возникновении ошибок следует продолжить программу, предусмотрев выполнение определенных действий. Для обработки исключительных ситуаций используют конструкции типа try {…} catch(Exception e){…}. Написанный здесь образец является предельно общим. Поясним его следующим примером, где выполняется попытка деления на 0:

package zerodivision;

import java.awt.\*;

import java.util.Scanner;

public class ZeroDivision {

public static void main(String[] args) {

System.out.println("Input first number");

Scanner input= new Scanner(System.in);

int n1=input.nextInt();

System.out.println("Input second number");

int n2=input.nextInt();

try

{

double res=(double)(n1 / n2);

System.out.printf("The division of %d by %d is %f",n1,n2,res);

}

catch(Exception ex)

{

System.out.println("Division by zero is forbidden!!! "+ex.getMessage());

}

}

}

Если при вводе второго числа ввести 0, то возникнет исключение, которое будет перехвачено блоком catch, и на экране получим соответствующее сообщение

run:

Input first number

10

Input second number

0

Division by zero is forbidden!!! / by zero

ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 6 секунд)

Программа при этом аварийно не завершается. Имеются разные классы исключений. Класс Exception – наиболее общий. В той же программе можно было использовать класс ArithmeticException:

catch(ArithmeticException ex)

{

System.out.println("Division by zero is forbidden!!! "+

ex.getMessage());

}

Наряду с try и catch используют также конструкцию finally, которая выполняется всегда, есть ошибка или нет. Пример

**П4. Потоки**

Поток – это ветвь основной программы, которая выполняется параллельно ей. Если нужно объявить, что класс использует поток, то в объявлении класса указываем интерфейс implements java.lang.Runnable. В данном интерфейсе объявлен метод run – основной метод потока. Когда поток стартует, то он начинает выполнять метод run. После завершения метода run поток автоматически уничтожается. Следующий фрагмент кода содержит пример метода run, который мы используем в учебном приложении

public void run() {

while ( fThread != null){

try{

Thread.sleep (300);

++bellTolls;

javax.swing.ImageIcon img =

new javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Bella.gif");

aLabel.setIcon(img);

}

catch (InterruptedException e) { }

anOther();

if (this.isDone()) fThread = null;

}

Здесь программное имя потокового объекта fThread. Происходит динамически повторяемое поочередное отображение двух картинок (icon) через паузу. Первая картинка создается в команде

javax.swing.ImageIcon img =

new javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Bella.gif");

и отображается в поле label

aLabel.setIcon(img);

Метод anOther() создает и отображает вторую картинку и имеет следующий вид

public void anOther(){

try{

Thread.sleep (300);

javax.swing.ImageIcon img =

new javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Sun.gif");

aLabel.setIcon(img);

}

catch (InterruptedException e) { }

}

Через паузу (Thread.sleep) в 300 милисекунд на ярлыке отображается другая картинка (Sun.gif).

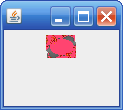
 

Рисунок 141. Смена картинок в потоках

Приводим полный текст программы

package ex1bthread;

import java.awt.\*;

import javax.swing.\*;

public class Ex1bThread extends javax.swing.JFrame implements java.lang.Runnable{

public Ex1bThread() {

aPanel = new javax.swing.JPanel();

aLabel = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setPreferredSize(new java.awt.Dimension(50, 110));

setBackground(Color.PINK);

add(aPanel);

aPanel.add(aLabel);

pack();

start();

}

public final void start() {

if (fThread == null){

fThread = new java.lang.Thread (this);

fThread.start();

}

}

public void run() {

while ( fThread != null){

try{

Thread.sleep (300);

++bellTolls;

javax.swing.ImageIcon img =

new javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Bellb.gif");

aLabel.setIcon(img);

}

catch (InterruptedException e) { }

anOther();

if (this.isDone()) fThread = null;

}

}

public void anOther(){

try{

Thread.sleep (300);

javax.swing.ImageIcon img =new javax.swing.ImageIcon("E:\\work5\\Ex1bThread\\src\\Sun.gif");

aLabel.setIcon(img);

}

catch (InterruptedException e) { }

}

public boolean isDone() {

boolean temp=false;

if(bellTolls==COUNTS) {

temp=true;

bellTolls=0;

}

return temp;

}

public static void main(String args[]) {

Ex1bThread z=

new Ex1bThread();

z.setVisible(true);

z.setBackground(Color.PINK);

}

private javax.swing.JLabel aLabel;

private javax.swing.JPanel aPanel;

Thread fThread;

int bellTolls=0;

final int COUNTS=50;

}

Конструктор основного класса Ex1bThread объявлен и реализован следующим образом

public Ex1bThread() {

aPanel = new javax.swing.JPanel();

aLabel = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);//размещает окно по центру

//экрана

setPreferredSize(new java.awt.Dimension(50, 110));

setBackground(Color.PINK);

add(aPanel);

aPanel.add(aLabel);

pack();

start();

}

Создается панель (вид окна)

aPanel = new javax.swing.JPanel(),

на которую добавляем текстовый ярлык (Label)

aPanel.add(aLabel).

Вызываем метод start для запуска потока

public final void start() {

if (fThread == null){

fThread = new java.lang.Thread (this);

fThread.start();//поток запускается командой start

}

Смена картинок будет выполняться число раз, регулируемое переменной bellToll в методе isDone() (в приложении – 50 раз).

**П5. Работа с файлами**

Java предоставляет средства для низкоуровневого (на уровне байтов) доступа к файлам и средства высокоуровневого доступа при этом высокоуровневая объектная поточная переменная создается на основе низкоуровневой. Следующий фрагмент служит пояснением

FileOutputStream fout=new FileOutputStream("e:/my.txt");

DataOutputStream dout=new DataOutputStream(fout);

dout.writeUTF("hello, File World! ");

dout.close();

Переменная fout является низкоуровневой файловой поточной переменной. Она используется в конструкторе new DataOutputStream для создания высокоуровневой переменной dout,с помощью которой и осуществляется запись в файл e:/my.txt. Метод writeUTF выполняет запись текстовых данных информации в формате UNICODE. Чтобы прочитать записанные в файл текстовые данные, используем следующий код

FileInputStream finp=FileInputStream("e:/my.txt");

DataInputStream dinp=DataInputStream(finp);

System.out.println(""+dinp.readUTF());

Dis.close();

Создадим оконное приложение для иллюстрации возможностей работы с файлами.

package fileexample;

import java.awt.event.\*;

import java.io.\*;

import java.awt.\*;

public class FileExample extends Frame implements ActionListener

{

Button bt=new Button("Выход");

Button bt1=new Button("Записать");

Button bt2=new Button("Прочитать");

TextField tf=new TextField();

public FileExample()

{

super("Work with Files");

setLayout(null);

setBackground(new Color(250,200,120));

setSize(250,200);

setVisible(true);

add(bt);

add(bt1);

add(bt2);

add(tf);

bt.addActionListener(this);

bt1.addActionListener(this);

bt2.addActionListener(this);

bt.setBounds(20,40,100,20);

bt1.setBounds(20,65,100,20);

bt2.setBounds(20,90,100,20);

tf.setBounds(20,115,150,20);

}

public void actionPerformed(ActionEvent ae)

{

if(ae.getSource()==bt1)

{

try{

FileOutputStream fout=new FileOutputStream("e:/work/my.txt");

DataOutputStream dout=new DataOutputStream(fout);

dout.writeUTF("Hello, File World!");

dout.close();

}

catch(Exception ex){}

}

else

if(ae.getSource()==bt)

System.exit(0);

else

if(ae.getSource()==bt2)

{

try{

FileInputStream finp= new FileInputStream("e:/work/my.txt");

DataInputStream dinp=new DataInputStream(finp);

String s=dinp.readUTF();

dinp.close();

tf.setText(s);

}

catch(Exception ex) {}

}

}

public static void main(String[] args)

{

new FileExample();

}

}

Результат работы приложения показан на рисунке 142

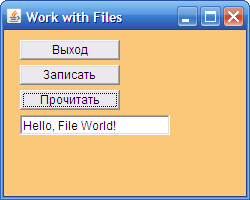


Рисунок 142. Чтение и запись в файлы

Для записи строк в формате ASCII используется поточный класс PrintStream, как показано ниже

try{

FileOutputStream fout=new FileOutputStream("e:/work/my.txt");

PrintStream pstr=new PrintStream(fout);

pstr.println("Hello, File World");

pstr.close();

}

catch(Exception ex){}

}

else

if(ae.getSource()==bt2)

{

try{

FileInputStream finp= new FileInputStream("e:/work/my.txt");

DataInputStream dinp=new DataInputStream(finp);

String s=dinp.readLine();

dinp.close();

tf.setText(s);

}

catch(Exception ex) {}

}

Изменим несколько наше приложение с тем, чтобы выбирать файл из окна файлового диалога. Окно файлового диалога создается и отображается следующим образом

FileDialog fd=new FileDialog(this, "FileOpening");

fd.show();

В конструкторе FileDialog указывается режим открытия файла ("FileOpening"). Имя выбранного файла получаем с помощью команды

String s=fd.getFile();

С учетом сказанного приложение можно переписать следующим образом.

package fileexample;

import java.awt.event.\*;

import java.io.\*;

import java.awt.\*;

public class FileExample extends Frame implements ActionListener

{

Button bt=new Button("Выход");

Button bt1=new Button("Записать");

Button bt2=new Button("Прочитать");

TextField tf=new TextField();

public FileExample()

{

super("Work with Files");

setLayout(null);

setBackground(new Color(250,200,120));

setSize(250,200);

setVisible(true);

add(bt);

add(bt1);

add(bt2);

add(tf);

bt.addActionListener(this);

bt1.addActionListener(this);

bt2.addActionListener(this);

bt.setBounds(20,40,100,20);

bt1.setBounds(20,65,100,20);

bt2.setBounds(20,90,100,20);

tf.setBounds(20,115,150,20);

}

public void actionPerformed(ActionEvent ae)

{

if(ae.getSource()==bt1)

{

try{

FileOutputStream fout=new FileOutputStream("e:/work/my.txt");

DataOutputStream dout=new DataOutputStream(fout);

dout.writeUTF("Hello, File World!");

dout.close();

}

catch(Exception ex){}

}

else

if(ae.getSource()==bt)

System.exit(0);

else

if(ae.getSource()==bt2)

{

try{

FileDialog fd=new FileDialog(this,"FileOpening");

fd.show();

String s1=fd.getFile();

FileInputStream finp= new FileInputStream("e:/work/my.txt");

DataInputStream dinp=new DataInputStream(finp);

String s=dinp.readUTF();

dinp.close();

tf.setText(s);

}

catch(Exception ex) {}

}

}

public static void main(String[] args)

{

new FileExample();

}}

Окно приложения показано на рисунке 143.

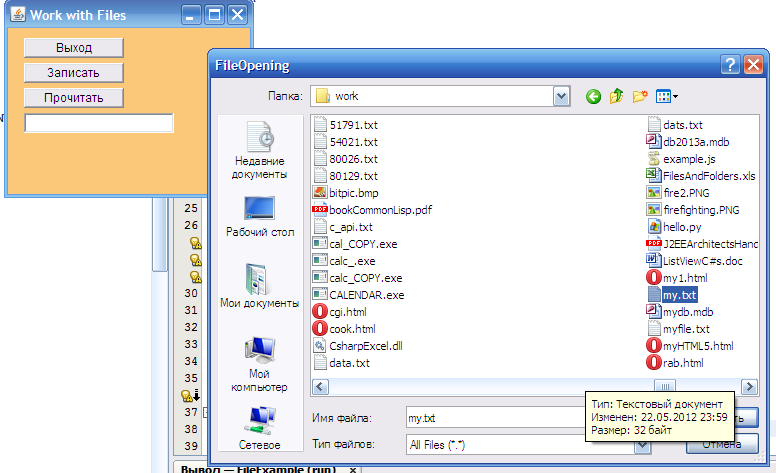


Рисунок 143. Окно приложения с файловым диалогом

Для произвольного доступа используется класс RandomAccessFile. Для доступа к данным используем метод seek(long position). Положение указателя возвращает метод long getFilePointer(). Пример записи в файл целых чисел командой writeInt дает следующий код

package mylabs;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

class lab1\_10 extends Frame implements ActionListener

{

Button b=new Button("Exit");

Button b1=new Button("Write to File");

Button b2=new Button(" Read from File");

public lab1\_10()

{

setLayout(null);

setBackground(new Color(240,230,100));

setSize(300,300);

setVisible(true);

add(b);

add(b1);

add(b2);

b.addActionListener(this);

b1.addActionListener(this);

b2.addActionListener(this);

b.setBounds(20,30,100,20);

b1.setBounds(20,60,100,20);

b2.setBounds(20,90,100,20);

}

public void actionPerformed(ActionEvent ae)

{

if(ae.getSource()==b)

System.exit(0);

else

if(ae.getSource()==b1)

{

try{

RandomAccessFile raf= new RandomAccessFile("temp.dat","rw");

for (int i=0;i<10;i++)

raf.writeInt(i);

raf.close();

}

catch(Exception ex){}

}

else

if(ae.getSource()==b2)

{

try{

RandomAccessFile raf= new RandomAccessFile("temp.dat","rw");

Graphics g=getGraphics();

for (int i=0;i<10;i++)

{

raf.seek(i\*4);

int z=raf.readInt();

g.drawString(""+z,130,50+20\*i);

}

raf.close();

}

catch(Exception ex) {}

}

}

}

public class lab1\_11{

public static void main(String[] args)

{

lab1\_10 app=new lab1\_10();

}

}

Для записи в файл используем фрагмент

try{

RandomAccessFile raf= new RandomAccessFile("temp.dat","rw");

for (int i=0;i<10;i++)

raf.writeInt(i);

raf.close();

}

catch(Exception ex){}

}

Запись целых чисел выполняет метод writeInt.

Для записи чисел в формате с плавающей точкой используют метод writeFloat. Для записи чисел с фиксированной точкой используют метод writeDouble. Для записи строки в формате Unicode – метод writeUTF. Чтение осуществляется соответственно методами readInt, readFloat, readDouble, readUTF. Однако чтение нужно выполнять с позиционированием головки чтения/записи. В нашем примере позиционирование реализовано в фрагменте:

for (int i=0;i<10;i++)

{

raf.seek(i\*4);

int z=raf.readInt();

……

Метод seek устанавливает указатель на указанное смещение в байтах относительно начала. Целое число занимает 4 байта. Поэтому выполняем умножение i\*4. Вещественные числа занимают 8 байтов. Следующий фрагмент дает иллюстрацию

try{

RandomAccessFile raf= new RandomAccessFile("temp.dat","rw");

Graphics g=getGraphics();

for (int i=0;i<10;i++)

{

raf.seek(i\*8);

double z=raf.readDouble();

g.drawString(""+z,130,50+20\*i);

}

raf.close();

}

**П6. Массивы, коллекции и итераторы**

Массив − это набор элементов одного типа, каждый из которых имеет номер (индекс). Следующий пример показывает создание массива строк и их вывод на консоль.

package collectionsa;

import java.util.Iterator;

import java.util.ArrayList;

public class CollectionsA {

public static String[] str\_arr = {"oleg", "nick", "adam"};

public static void main(String[] args) {

int size = str\_arr.length;

for (int i=0; i<size; i++)

{

System.out.println(str\_arr[i]);

}

}

}

Массив создается «прямым» перечислением

String[] str\_arr = {"oleg", "nick", "adam"};

Результат работы программы представлен выходным окном

run:

oleg

nick

adam

ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 0 секунд)

Коллекции – это наборы значений различных типов. В следующем примере демонстрируется использование коллекции на основе типа ArrayList:

package collectionsa;

import java.util.Iterator;

import java.util.ArrayList;

public class CollectionsA {

public static void main(String[] args) {

ArrayList arr = new ArrayList();

arr.add("one");

arr.add("two");

arr.add("three");

arr.add(4);

arr.add("5");

Iterator iter = arr.iterator();

while(iter.hasNext())

System.out.println(""+iter.next());

}

}

Значения в коллекцию добавляем с помощью метода add. Из примера видно, что в коллекцию добавлены как строки, так и целое число. Для выборки данных из коллекции нужно создать итератор (iterator). Это выполняется в строке

Iterator iter = arr.iterator();

Теперь с помощью итератора можно «пройтись» по коллекции и последовательно выбирать значения элементов:

while(iter.hasNext())

System.out.println(""+iter.next());

Удаление элемента из коллекции выполняется следующий код

package collectionsa;

import java.util.Iterator;

import java.util.ArrayList;

public class CollectionsA {

public static void main(String[] args) {

ArrayList arr = new ArrayList();

arr.add("one");

arr.add("two");

arr.add("three");

arr.add(4);

arr.add("5");

Iterator iter = arr.iterator();

while(iter.hasNext())

System.out.println(""+iter.next());

iter = arr.iterator();

while(iter.hasNext()){

if("three".equals(iter.next())){

iter.remove();

}

}

System.out.println("After remove:");

System.out.println(arr);

} }

Результат работы программы дает следующее содержимое окна вывода

run:

one

two

three

4

5

After remove:

[one, two, 4, 5]

ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 0 секунд)

Команда для удаления элемента из коллекции

iter.remove();

Коллекция с встроенным итератором относится к типу Enumeration. Пример дает следующий код

package collectionsa;

import java.util.Iterator;

import java.util.ArrayList;

import java.util.Enumeration;

import java.util.Vector;

import java.util.\*;

public class CollectionsA {

public static void main(String[] args) {

Vector<String> vl = new Vector<String>();

Enumeration<String> enm = null;

vl.add("one");

vl.add("two");

vl.add("three");

vl.add("four");

enm = vl.elements();

while(enm.hasMoreElements()){

System.out.println(enm.nextElement());

}

System.out.println("new collection starts");

Enumeration enm2 = null;

Vector v2=new Vector();

v2.add(1);

v2.add("two");

v2.add(new Date());

enm2=v2.elements();

while(enm2.hasMoreElements()){

System.out.println(""+enm2.nextElement());

}

}

}

В этом примере представлены две коллекции. Одна типизированная на базе типа String:

Enumeration<String> enm = null;

Вторая содержит элементы разных типов

Enumeration enm2 = null;

Результат работы программы представлен ниже

run:

one

two

three

four

new collection starts

1

two

Thu Aug 06 14:06:31 MSK 2015

ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 1 секунда)

**П7. Обработка событий**

Должно быть ясно, что обработка событий в приложении Java требует подключения прослушивателя событий. Для событий от кнопок требуется прослушиватель ActionListener (этот прослушиватель также используется для текстового поля и меню), для событий от клавиатуры − KeyListener, для событий от мыши − MouseListener и т.д. Следует иметь в виду, что прослушиватель событий представляет собой интерфейс – т.е. абстрактный класс, в котором методы только объявлены, но не определены. Так, в интерфейсе ActionListener нужно самостоятельно определить метод actionPerformed(ActionEvent e). В интерфейсе KeyListener следует определить методы keyPressed, keyReleased, keyTyped. Аргументом этих трех последних методов является объект класса KeyEvent, который имеет ряд полезных методов. Метод getKeyCode() позволяет получить числовой код нажатой клавиши, а метод getKeyChar() – символ, соответствующий нажатой клавише. Подключение прослушивателя от клавиатуры реализует метод addKeyListener класса Frame. Метод keyReleased активизируется при отпускании кнопки. Методы keyPressed, keyTyped вызываются при нажатии на кнопку. Метод keyTyped не позволяет получить значений клавиш-модификаторов (ALT, SHIFT,CTRL). Обработчик событий от мыши содержит и вовсе пять методов:

* public void mouseEntered(MouseEvent ev); - возникает при входе курсора мыши на элемент формы.
* public void mouseExited(MouseEvent ev); - возникает при выходе курсора мыши из элемента формы.
* public void mousePressed(MouseEvent ev); - событие, связанное с удерживанием кнопки мыши в нажатом состоянии
* public void mouseReleased(MouseEvent ev); - освобождение клавиши мыши
* public void mouseClicked(MouseEvent ev); - нажатие клавиши мыши.

(во всех этих примерах нажатие правой или левой кнопки мыши не различимо).

Следующий пример показывает, как в месте щелчка мыши на форме отображаются координаты точки, в которой произведен щелчок (рисунок 144).

package mylabs;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

class lab1\_12 extends Frame

implements ActionListener,MouseListener

{

Button b=new Button("Exit");

public lab1\_12()

{

setLayout(null);

setBackground(new Color(240,230,100));

setSize(300,300);

setVisible(true);

add(b);

b.addActionListener(this);

b.setBounds(20,30,100,20);

addMouseListener(this);

}

public void actionPerformed(ActionEvent ae)

{

if(ae.getSource()==b)

System.exit(0);

}

public void mouseEntered(MouseEvent ev)

{};

public void mouseExited(MouseEvent ev)

{};

public void mousePressed(MouseEvent ev)

{

int x=ev.getX();

int y=ev.getY();

Graphics g=getGraphics();

g.drawString("X="+x+"; Y="+y,x,y);

};

public void mouseReleased(MouseEvent ev)

{};

public void mouseClicked(MouseEvent ev)

{};

}

public class lab1\_13{

public static void main(String[] args)

{

lab1\_12 app=new lab1\_12();

}

}

Получение x-координаты (y-координаты) реализует метод getX() (getY()) класса MouseEvent в методе mousePressed.

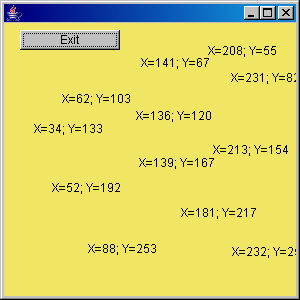


Рисунок 144. Вывод координат мыши по месту щелчка

**СПИСОК ЛИТЕРАТУРЫ**

1. Хабибуллин, И. Создание распределенных приложений на Java 2. – СПб.: БХВ–Петербург, 2002. – 692 с. – ББК 32.973.26-018.1.
2. Перроун, П. Дж. Создание корпоративных систем на основе Java 2 Enterprise Edition. Руководство разработчика /Пол Дж. Перроун, Венката С.Р. Кришна, Р. Чаганти – М.: Издательский дом “Вильямс”, 2001. – 1184 с. – ISBN 5-8459-0168-5.
3. Цимбал, А.А. Технологии создания распределенных систем. Для профессионалов / А.А. Цимбал, М.Л. Аншина. – СПб.:Питер, 2003. – 576с. ББК 32.973.202
4. Чапел, Л. TCP/IP. Учебный курс / Л. Чапел, Э. Титтел. – СПб.: БХВ–Петербург, 2003. – 976 с.
5. Хеффельфингер, Д. Java EE и сервер приложений GlassFish 3 –М.: ДМК, 2013. – 416 с.
6. Герман, О.В.Java и Интернет-бизнес /О.В. Герман, Ю.О. Герман. –Мн.: Бестпринт, 2010. – 384 с. – ББК 65-240-21.
7. Герман, О.В. Программирование на Java и с# для студента / О.В. Герман, Ю.О. Герман. – СПб.: БХВ–Петербург, 2005. – 512 с.
8. Холл, М. Программирование для WEB / М. Холл, Л. Браун – М.: Издательский дом “Вильямс”, 2002. – 1264 с.
9. Томас, М. Секреты программирования для Интернет на Java / М. Томас, П. Пател, А. Хадсон, Д. Болл. – www. Books-Shop.com (электронное издание). 2002. – 390 с.
10. Блинов, И.Н. Java. Промышленное программирование / И.Н. Блинов, В.С. Романчик – Мн.: УниверсалПресс, 2007. – 704 с.
11. Сетевые операционные системы / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2001. – 544 с. – ISBN 5-272-0120-6.
12. Гарнаев, А.Ю. Excel, VBA, Internet в экономике и финансах. – СПб.: БХВ–Петербург, 2002. – 816 с. – ISBN 5-94157-050-3.
13. Кровчик, Э .NET. Сетевое программирование для профессионалов / Э. Кровчик, В. Кумар –М.: Лори, 207. – 417 с. – ISBN 5-85582-170-2.
14. Мак-Дональд, М. ASP.NET 4 с примерами на с# 2010 для профессионалов/ М. Мак-Дональд, А. Фримен, М. Шпушта. – Издательский дом “Вильямс”, 2011. – 1424 с. – ISBN 978-5-8459-1702-7.

*Учебное издание*

**Герман** Олег Витольдович

**«АДМИНИСТРИРОВАНИЕ И ПРОГРАММИРОВАНИЕ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ»**

Пособие

для студентов специальности

«Автоматизированные системы обработки информации»

дневной и дистанционной форм обучения

Редактор Т. П. Андрейченко

Корректор А. В. Тюхай

|  |  |  |
| --- | --- | --- |
| Подписано в печать 2015г. | Формат 60х84 1/16 | Бумага офсетная |
| Гарнитура «Таймс» | Отпечатано на ризографе | Усл.печ.л. 15,0 |
| Уч.-изд. л. 5,0 | Тираж 100 экз. | Заказ 833 |

Издатель и полиграфическое исполнение: учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009

220013, Минск, П.Бровки,6