Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
Омский государственный технический университет

Объектно-ориентированное программирование на JAVA

Методические указания к выполнению лабораторных работ

Омск – 2012

Составитель: Евгений Борисович Юдин (udinev@asoiu.com)

Методические указания предназначены для выполнения студентами лабораторных работ по дисциплине «Объектно-ориентированное программирование на JAVA». Рассматриваются разработка консольного приложения в JAVA, использование регулярных выражений, использование коллекций, разработка визуальных интерфейсов, работа с сетевыми протоками, создание простейших веб-приложений на основе сервлетов.

Методические указания предназначены для направления подготовки бакалавриата 220100.62 – «Системный анализ и управление» по профилю подготовки «Системный анализ и управление в сложных больших системах», а также для направления подготовки бакалавриата, 230100.62 – «Информатика и вычислительная техника» по профилю подготовки «Автоматизированные системы обработки информации и управления»

Печатается по решению редакционно-издательского совета Омского государственного технического университета.

Редактор Т.А. Жирнова
Свод. темплан 2011 г.
И Д №06039 от 12.10.01
Подписано в печать 1.03.11 Бумага офсетная. Формат 64 × 84 $^{1}/_{16}$.
Отпечатано на дупликаторе. Усл. печ. л. 3,75. Уч. - изд.л. 3,75.
Тираж 50 . Заказ 50 .

Издательство ОмГТУ. 644050, Омск, пр. Мира 11
Типография ОмГТУ

Лабораторная работа №1

РАЗРАБОТКА КОНСОЛЬНЫХ ПРИЛОЖЕНИЙ

В данной лабораторной работе разрабатывается консольное приложение для реализации простейшего приложения с использованием массивов, строк и файлов.

Консольные приложения JAVA представляет собой созданный и откомпилированный программистом класс, содержащий точку входа.

Рассмотрим простой пример:

```
public class First {
	public static void main(String[] args) {
		System.out.println("Первая программа на Java!");
	}
}
```

Здесь класс First используется только для того, чтобы определить метод main(), который и является точкой входа и с которого начинается выполнения программы интерпретатором Java. Метод main() содержит аргументы-параметры командной строки String[] args в виде массива строк и является открытым (public) членом класса. Это означает, что метод main() виден и доступен любому классу. Ключевое слово static объявляет методы и переменные класса, используемые для работы с классом в целом, а не только с объектом класса. Символы верхнего и нижнего регистров в JAVA различаются.

Вывод строки «Первая программа на Java!» в примере осуществляет метод println() (ln – переход к новой строке после вывода) свойства out класса System, который доступен в программе автоматически вместе с пакетом java.lang. Приведенную программу необходимо поместить в файл, имя которого совпадает с именем класса и с расширением java. Простейший способ компиляции написанной программы – вызов строчного компилятора:

```
javac First.java.
```

При успешной компиляции создается файл First.class. Этот файл можно запустить на исполнение с помощью интерпретатора Java следующим образом:

```
java First.
```

Для разработки программы возможно использование и специальных средств разработчика.

NetBeans IDE – свободная интегрированная среда разработки для всех платформ Java – Java ME, Java SE и Java EE. Пропагандируется Sun Microsystems, разработчиком Java, как базовое средство для разработки ПО на языке Java).

Eclipse IDE – свободная интегрированная среда разработки для Java SE, Java EE и Java ME. Пропагандируется IBM, одним из важнейших разработчиков корпоративного ПО, как базовое средство для разработки ПО на языке Java.

IntelliJ IDEA – среда разработки для платформ Java SE, Java EE и Java ME. Разработчик – компания JetBrains. Распространяется в двух версиях: свободной бесплатной (Community Edition) и коммерческой проприетарной (Ultimate Edition).

JDeveloper – среда разработки для платформ Java SE, Java EE и Java ME. Разработчик — компания Oracle.

BlueJ – Среда разработки программного обеспечения на языке Java, созданная в основном для использования в обучении, но также подходящая для разработки небольших программ.

Ниже рассмотрены основные классы, используемые при выполнении лабораторной работы, рассмотрен пример решения одного из заданий.

Класс java.io.File

Для работы с файлами в приложениях Java могут быть использованы классы из пакета java.io, одним из которых является класс File.

Класс File служит для хранения и обработки в качестве объектов каталогов и имен файлов. Этот класс не описывает способы работы с содержимым файла, но позволяет манипулировать такими свойствами файла, как права доступа, дата и время создания, путь в иерархии каталогов, создание, удаление, изменение имени файла и каталога и т.д.

Основные методы класса File и способы их применения рассмотрены в следующем примере.

```
import java.io.*;
import java.util.*;
public class Main {
public static void main(String[] args) throws
IOException /*отказ от обработки исключения в main()*/ {
//с объектом типа File ассоциируется файл на диске
     File fp = new File("com\\learn\\FileTest.java");
//другие способы создания объекта
     //File fp = new File("\\com\\learn", "FileTest.java");
     //File fp=new File("d:\\temp\\demo.txt");
     //File fp=new File("demo.txt");
         if(fp.isFile()){//если объект дисковый файл
     System.out.println("Имя файла:\t" + fp.getName());
     System.out.println("Путь к файлу:\t" + fp.getPath());
     System.out.println("Абсолютный путь:\t" + fp.getAbsolutePath());
     System.out.println("Канонический путь:\t" + fp.getCanonicalPath());
     System.out.println("Размер файла:\t" + fp.length());
     System.out.println("Последняя модификация файла:\t" + fp.lastModified());
     System.out.println("Файл доступен для чтения:\t" + fp.canRead());
     System.out.println("Файл доступен для записи:\t" + fp.canWrite());
     System.out.println("Файл удален:\t" + fp.delete());          }
         if(fp.createNewFile()){
System.out.println("Файл " + fp.getName() + " создан");
         }
         if(fp.exists()){
System.out.println("temp файл "+ fp.getName() + " существует");
         }
         else
System.out.println("temp файл " + fp.getName() + " не существует");
//в объект типа File помещается каталог\директория
             File dir = new File("com\\learn");
if (dir.isDirectory())/*если объект объявлен как каталог на диске*/
System.out.println("Директория!");
```

```
        if(dir.exists()){//если каталог существует
     System.out.println("Dir " + dir.getName() + " существует");
              File [] files = dir.listFiles();
              System.out.println("");
         for(int i=0; i < files.length; i++){
         Date date = new Date(files[i].lastModified());
System.out.println(files[i].getPath() + " \t| " + files[i].length() + "\t| " +
date.toString());//toLocaleString());//toGMTString())
              }
         }
     }
}
```

У каталога (директории) как объекта класса File есть дополнительное свойство – просмотр списка имен файлов с помощью методов list(), listFiles(), listRoots().

Класс System

Класс System содержит набор полезных статических методов и полей системного уровня. Экземпляр этого класса не может быть создан или получен.

Наиболее широко используемой возможностью, предоставляемой System, является стандартный вывод, доступный через переменную System.out. Стандартный вывод можно перенаправить в другой поток (файл, массив байт и т.д., главное, чтобы это был объект PrintStream, смотри документацию JSDK: http://docs.oracle.com/javase/6/docs/api/):

```
public static void main(String[] args) {
         System.out.println("Study Java");
         try {
            PrintStream print = new PrintStream(new
               FileOutputStream("d:\\file2.txt"));
            System.setOut(print);
            System.out.println("Study well");
         } catch(FileNotFoundException e) {
            e.printStackTrace();
         }
      }
```

При запуске этого кода на экран будет выведено только:

```
Study Java.
```

И в файл "d:\file2.txt" будет записано:

```
Study well.
```

Аналогично могут быть перенаправлен стандартный ввод System.in – вызовом System.setIn(InputStream) и поток вывода сообщений об ошибках System.err – вызовом System.setErr (по умолчанию все потоки – in, out, err – работают с консолью приложения).

Класс String

Класс String содержит основные методы для работы со строками:

concat(String s) или + – слияние строк;

equals(Object ob), equalsIgnoreCase(String s) – сравнение строк с учетом и без учета регистра;

compareTo(String s), compareToIgnoreCase (String s) – лексикографическое сравнение строк с учетом и без учета регистра;

contentEquals(StringBuffer ob) – сравнение строки и содержимого объекта типа StringBuffer;

charAt(int n)– извлечение из строки символа с указанным номером (нумерация с нуля);

substring(int n, int m)- извлечение из строки подстроки длины m-n, начиная с позиции n;

length() – определение длины строки;

valueOf(объект) – преобразование примитивного объекта к строке;

toUpperCase() / toLowerCase() – преобразование всех символов вызывающей строки в верхний/нижний регистр;

replace(char c1, char c2) – замена в строке всех вхождений первого символа вторым символом;

getBytes(параметры), getChars(параметры) – извлечение символов строки в виде массива байт или символов.

В следующем примере массив символов и целое число преобразуются в объекты типа String с использованием методов этого класса.

```
public class DemoString {
      public static void main(String[] args) {
            char s[] = { 'J', 'a', 'v', 'a' };
            int i = 2;
            // комментарий содержит результат выполнения кода
            String str = new String(s); // str="Java"
            i = str.length(); // i=4
            String num = String.valueOf(2); // num="2"
            str = str.toUpperCase(); // str="JAVA"
            num = str.concat(num); // num="JAVA2"
            str = str + "C";// str="JAVAC";
            char ch = str.charAt(2); // ch='V'
            i = str.lastIndexOf('A'); // i=3 (-1 если отсутвствует)
            num = num.replace('2', 'H'); // num="JAVAH"
            i = num.compareTo(str); // i=5 (между символами 'H' и 'C')
            str.substring(0, 3).toLowerCase(); // java
      }
}
```

Пример решения одного из заданий на лабораторную работу №1.

Задание: Ввести n строк с консоли. Вывести на консоль строки и их длины, упорядоченные по возрастанию.

Решение:

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

```java
public class Main
{
	public static void main(String[] args)
	{
		int n = 0;
		while (true) // ввод числа строк
		{
			System.out.println("Введите число строк");
			Scanner sc1 = new Scanner(System.in);
			try
			{
				n = sc1.nextInt();
				break;
			}
			catch(InputMismatchException fg)
			{
			// если введенное значение не является числом
				System.out.print("Вы ввели не число. ");
			}
		}
		// создание массива строк
		String[] str = new String[n];
		Scanner sc2 = new Scanner(System.in);
		for (int i = 0; i < n; i++)
		{
			System.out.println("Введите строку №" + (i+1));
			str[i] = sc2.nextLine();
		}
		//сортировка массива строк по длине
		for (int i = 0; i < str.length-1; i++)
		{
			for (int j = i+1; j < str.length; j++)
			{
				if (str[i].length()<str[j].length())
				{
					String temp = str[i];
					str[i] = str[j];
					str[j] = temp;
				}
			}
		}
		int maxlength = str[0].length();
		System.out.println("Строки в порядке убывания длины:");
		for (int i = 0; i < str.length; i++)
		{
			System.out.print(str[i]);
			for (int j = 0; j < maxlength - str[i].length(); j++)
			System.out.print(" ");
			System.out.println(" её длина = " + str[i].length());
		}
	}
}
```

Примеры индивидуальных заданий для выполнения лабораторной работы № 1

1. Ввести n строк с консоли, найти самую короткую строку. Вывести эту строку и ее длину.

2. Ввести n строк с консоли. Упорядочить и вывести строки в порядке возрастания их длин, а также (второй приоритет) значений этих их длин.

3. Ввести n строк с консоли. Вывести на консоль те строки, длина которых меньше средней, также их длины.

4. В каждом слове текста k-ю букву заменить заданным символом. Если k больше длины слова, корректировку не выполнять.

5. В русском тексте каждую букву заменить ее номером в алфавите. В одной строке печатать текст с двумя пробелами между буквами, в следующей строке внизу под каждой буквой печатать ее номер.

6. Из небольшого текста удалить все символы, кроме пробелов, не являющиеся буквами. Между последовательностями подряд идущих букв оставить хотя бы один пробел.

7. Из текста удалить все слова заданной длины, начинающиеся на согласную букву.

8. В тексте найти все пары слов, из которых одно является обращением другого.

9. Найти и напечатать, сколько раз повторяется в тексте каждое слово.

10. Найти, каких букв, гласных или согласных, больше в каждом предложении текста.

11. Выбрать три разные точки заданного на плоскости множества точек, составляющие треугольник наибольшего периметра.

12. Найти такую точку заданного на плоскости множества точек, сумма расстояний от которой до остальных минимальна.

13. Выпуклый многоугольник задан на плоскости перечислением координат вершин в порядке обхода его границы. Определить площадь многоугольника.

Лабораторная работа №2

ИСПОЛЬЗОВАНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Во второй лабораторной работе необходимо реализовать консольное приложение, позволяющее манипулировать строкой, разбив ее на элементы путем использования регулярных выражений.

Регулярные выражения – эта система обработки текста, основанная на специальной системе записи образцов для поиска. Образец (англ. pattern), задающий правило поиска, по-русски также иногда называют «шаблоном», «маской». Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил. Язык программирования JAVA также поддерживает регулярные выражения для работы со строками.

Основными классами для работы с регулярные выражения являются класс java.util.regex.Pattern и класс java.util.regex.Matcher.

Класс java.util.regex.Pattern применяется для определения регулярных выражений, для которого ищется соответствие в строке, файле или другом объекте представляющем собой некоторую последовательность символов. Для определения шаблона применяются специальные синтаксические конструкции. О каждом соответствии можно получить больше информации с помощью класса java.util.regex.Matcher. Далее приведены основные логические конструкции для задания шаблона. Если в строке, проверяемой на соответствие, необходимо, чтобы в

какой-либо позиции находился один из символов некоторого символьного набора, то такой набор (класс символов) можно объявить, используя одну из конструкций, представленных в табл.1.

Таблица 1 – Способы определения классов символов

[abc]	a, b или c
[^abc]	символ, исключая a, b и c
[a-z]	символ между a и z
[a-d[m-p]]	либо между a и d, либо между m и p
[e-z&&[dem]]	e либо m (конъюнкция)

Кроме стандартных классов символов существуют предопределенные классы символов (табл. 2)

Таблица 2 – Дополнительные способы определения классов символов

.	любой символ
\d	[0-9]
\D	[^0-9]
\s	[\t\n\x0B\f\r]
\S	[^ \s]
\w	[a-zA-Z_0-9]
\W	[^\w]
\p{javaLowerCase}	тоже, что и Character.isLowerCase()
\p{javaUpperCase}	тоже, что и Character.isUpperCase()

При создании регулярного выражения могут использоваться логические операции (табл.3).

Таблица 3 – Способы задания логических операций

XY	После X следует Y
X|Y	X либо Y
(X)	X

Скобки, кроме их логического назначения, также используются для выделения групп. Для определения регулярных выражений недостаточно одних классов символов, т. к. в шаблоне часто нужно указать количество повторений. Для этого существуют квантификаторы (табл. 4).

Таблица 4 – Квантификаторы

X?	X один раз или ни разу
X*	X ноль или более раз
X+	X один или более раз
X{n}	X n раз
X{n,}	X n или более раз
X{n,m}	X от n до m

Существует еще два типа квантификаторов, которые образованы прибавлением суффикса ? (слабое или неполное совпадение) или + («жадное» или собственное совпадение) к вышеперечисленным квантификаторам. Неполное совпадение соответствует выбору с наименее возможным количеством символов, а собственное – с максимально возможным.

Класс Pattern используется для простой обработки строк. Для более сложной обработки строк используется класс Matcher, рассматриваемый ниже.

В классе Pattern объявлены следующие методы:

compile(String regex) – возвращает Pattern, который соответствует regex;

matcher(CharSequence input) – возвращает Matcher, с помощью которого можно находить соответствия в строке input;

matches(String regex, CharSequence input) – проверяет на соответствие строки input шаблону regex;

pattern() – возвращает строку, соответствующую шаблону;

split(CharSequence input) – разбивает строку input, учитывая, что разделителем является шаблон;

split(CharSequence input, int limit) – разбивает строку input на не более чем limit частей.

С помощью метода matches() класса Pattern можно проверять на соответствие шаблону целой строки, но если необходимо найти соответствия внутри строки, например, определять участки, которые соответствуют шаблону, то класс Pattern не может быть использован. Для таких операций необходимо использовать класс Matcher.

Начальное состояние объекта типа Matcher не определено. Попытка вызвать какой-либо метод класса для извлечения информации о найденном соответствии приведет к возникновению ошибки IllegalStateException. Для того чтобы начать работу с объектом Matcher нужно вызвать один из его методов:

matches() – проверяет, соответствует ли вся строка шаблону;

lookingAt() – пытается найти последовательность символов, начинающуюся с начала строки и соответствующую шаблону;

find() или find(int start) – пытается найти последовательность символов, соответствующих шаблону, в любом месте строки. Параметр start указывает на начальную позицию поиска.

Иногда необходимо сбросить состояние объекта класса Matcher в исходное, для этого применяется метод reset() или reset(CharSequence input), который также устанавливает новую последовательность символов для поиска.

Для замены всех подпоследовательностей символов, удовлетворяющих шаблону, на заданную строку можно применить метод replaceAll(String replacement).

Для того чтобы ограничить поиск границами входной последовательности применяется метод region(int start, int end), а для получения значения этих границ – regionEnd() и regionStart(). С регионами связано несколько методов:

useAnchoringBounds(boolean b) – если установлен в true, то начало и конец региона соответствуют символам ^ и $ соответственно;

hasAnchoringBounds() – проверяет закрепленность границ.

В регулярном выражении для более удобной обработки входной последовательности применяются группы, которые помогают выделить части найденной подпоследовательности. В шаблоне они обозначаются скобками «(« и «)». Номера групп начинаются с единицы. Нулевая группа совпадает со всей найденной подпоследовательностью. Далее приведены методы для извлечения информации о группах:

end() – возвращает индекс последнего символа подпоследовательности, удовлетворяющей шаблону;

end(int group) – возвращает индекс последнего символа указанной группы;

group() – возвращает всю подпоследовательность, удовлетворяющую шаблону;

group(int group) – возвращает конкретную группу;

groupCount() – возвращает количество групп;

start() – возвращает индекс первого символа подпоследовательности, удовлетворяющей шаблону;

start(int group) – возвращает индекс первого символа указанной группы;

hitEnd() – возвращает истину, если был достигнут конец входной последовательности.

Следующий пример показывает использование возможностей классов Pattern и Matcher, для поиска, разбора и разбиения строк.

```
import java.util.regex.*;

public class DemoRegular {
	public static void main(String[] args) {
		// проверка на соответствие строки шаблону
		Pattern p1 = Pattern.compile("a*y");
		Matcher m1 = p1.matcher("aaay");
		boolean b = m1.matches();
		System.out.println(b);
		// поиск и выбор подстроки, заданной шаблоном
		String regex = "(\\w+)@(\\w+\\.)(\\w+)(\\.\\w+)*";
		String s = "адреса эл.почты: mymail@tut.by и rom@bsu.by";
		Pattern p2 = Pattern.compile(regex);
		Matcher m2 = p2.matcher(s);
		while (m2.find()) {
			System.out.println("e-mail: " + m2.group());
		}
		// разбиение строки на подстроки с применением шаблона в качестве
		// разделителя
		Pattern p3 = Pattern.compile("\\d+\\s?");
		String[] words = p3.split("java5tiger 77 java6mustang");
		for (String word : words)
			System.out.println(word);

		}

	}
```

```
}
```

В результате будет выведено:

```
true
e-mail: mymail@tut.by
e-mail: rom@bsu.by
java
tiger
java
mustang
```

Следующий пример демонстрирует возможности использования групп, а также собственных и неполных квантификаторов.

```
import java.util.regex.*;
public class Groups {
    public static void main(String[] args) {
        String input = "abdcxyz";
        myMatches("([a-z]*)([a-z]+)", input);
        myMatches("([a-z]?)([a-z]+)", input);
        myMatches("([a-z]+)([a-z]*)", input);
        myMatches("([a-z]?)([a-z]?)", input);
    }
    public static void myMatches(String regex,
            String input) {
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(input);
        if(matcher.matches()) {
            System.out.println("First group: "
                + matcher.group(1));
            System.out.println("Second group: "
                + matcher.group(2));
        } else
            System.out.println("nothing");
        System.out.println();
    }}
```

Результат работы программы:

```
First group: abdcxy
Second group: z
First group: a
Second group: bdcxyz
First group: abdcxyz
Second group: nothing
```

В первом случае к первой группе (First group) относятся все возможные символы, но при этом остается минимальное количество символов для второй группы (Second group). Во втором случае для первой группы выбирается наименьшее количество символов, т. к. используется слабое совпадение. В третьем случае первой группе будет соответствовать вся строка, а для второй не остается ни одного символа, так как

вторая группа использует слабое совпадение. В четвертом случае строка не соответствует регулярному выражению, т. к. для двух групп выбирается наименьшее количество символов.

В классе Matcher объявлены два полезных метода для замены найденных подпоследовательностей во входной строке.

Matcher appendReplacement(StringBuffer sb, String replacement) – метод читает символы из входной стоки и добавляет их в sb. Чтение останавливается на start() – 1 позиции предыдущего совпадения, после чего происходит добавление в sb строки replacement. При следующем вызове этого метода, производится добавление символов, начиная с символа с индексом end() предыдущего совпадения.

Примеры индивидуальных заданий для выполнения лабораторной работы № 2

1. Написать регулярное выражение, определяющее является ли данная строка строкой "abcdefghijklmnopqrstuv18340" или нет.

Пример правильных выражений: abcdefghijklmnopqrstuv18340.

Пример неправильных выражений: abcdefghijklmnoasdfasdpqrstuv18340.

2. Написать регулярное выражение, определяющее является ли данная строка GUID с или без скобок. Где GUID это строчка, состоящая из 8, 4, 4, 4, 12 шестнадцатеричных цифр разделенных тире.

Пример правильных выражений: e02fd0e4-00fd-090A-ca30-0d00a0038ba0.

Пример неправильных выражений: e02fd0e400fd090Aca300d00a0038ba0.

3. Написать регулярное выражение, определяющее является ли заданная строка правильным MAC-адресом.

Пример правильных выражений: aE:dC:cA:56:76:54.

Пример неправильных выражений: 01:23:45:67:89:Az.

4. Написать регулярное выражение, определяющее является ли данная строчка валидным URL адресом. В данной задаче правильным URL считаются адреса http и https, явное указание протокола также может отсутствовать. Учитываются только адреса, состоящие из символов, т.е. IP адреса в качестве URL не присутствуют при проверке. Допускаются поддомены, указание порта доступа через двоеточие, GET запросы с передачей параметров, доступ к подпапкам на домене, допускается наличие якоря через решетку. Однобуквенные домены считаются запрещенными. Запрещены спецсимволы, например «–» в начале и конце имени домена. Запрещен символ «_» и пробел в имени домена. При составлении регулярного выражения ориентируйтесь на список правильных и неправильных выражений заданных ниже.

Пример правильных выражений: http://www.zcontest.ru, http://zcontest.ru.

Пример неправильных выражений: Just Text, http://a.com.

5. Написать регулярное выражение, определяющее является ли данная строчка шестнадцатиричным идентификатором цвета в HTML. Где #FFFFFF для белого, #000000 для черного, #FF0000 для красного и т.д.

Пример правильных выражений: #FFFFFF, #FF3421, #00ff00.

Пример неправильных выражений: 232323, f#fddee, #fd2.

6. Написать регулярное выражение, определяющее является ли данная строчка датой в формате dd/mm/yyyy. Начиная с 1600 года до 9999 года.

Пример правильных выражений: 29/02/2000, 30/04/2003, 01/01/2003.

Пример неправильных выражений: 29/02/2001, 30-04-2003, 1/1/1899.

7. Написать регулярное выражение, определяющее является ли данная строчка валидным E-mail адресом согласно RFC под номером 2822.

Пример правильных выражений: mail@mail.ru, valid@megapochta.com.

Пример неправильных выражений: bug@@@com.ru, @val.ru, Just Text2.

8. Составить регулярное выражение, определяющее является ли заданная строка IP адресом, записанным в десятичном виде.

Пример правильных выражений: 127.0.0.1, 255.255.255.0.

Пример неправильных выражений: 1300.6.7.8, abc.def.gha.bcd.

9. Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов. Где символом может быть английская буква, цифра и знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру.

Пример правильных выражений: C00l_Pass, SupperPas1.

Пример неправильных выражений: Cool_pass, C00l.

10. Проверить является ли заданная строка шестизначным числом, записанным в десятичной системе счисления без нулей в старших разрядах.

Пример правильных выражений: 123456, 234567.

Пример неправильных выражений: 1234567, 12345.

11. Есть текст со списками цен. Извлечь из него цены в USD, RUR, EU.

Пример правильных выражений: 23.78 USD.

Пример неправильных выражений: 22 UDD, 0.002 USD.

12. Проверить существуют ли в тексте цифры, за которыми не стоит «+».

Пример правильных выражений: (3 + 5) – 9 × 4.

Пример неправильных выражений: 2 * 9 – 6 × 5.

13. Создать запрос для вывода только правильно написанных выражений со скобками (количество открытых и закрытых скобок должно быть одинаково).

Пример правильных выражений: (3 + 5) – 9 × 4.

Пример неправильных выражений: ((3 + 5) – 9 × 4.

Лабораторная работа №3

ИСПОЛЬЗОВАНИЕ КОЛЛЕКЦИЙ

При программировании на Java, при реализации операций над группой однотипных объектов, важно выбирать наиболее эффективную структуру данных (класс) для хранения этих объектов. Эти специальные классы для хранения однотипных объектов называются коллекциями (список, множество, карты).

Н:апример, если используется список (который определяет интерфейс List), то существуют богатый 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. Рекомендуется ознакомиться с описаниями следующих коллекций по JSDK-документации:

– java.util.Collection;
– java.util.ArrayList;
– java.util.HashMap;
– java.util.HashSet.

Дополнительную информацию можно получить по следующим ссылкам:
http://java.sun.com/j2se/1.5.0/docs/api/;
http://www.uic.rsu.ru/doc/programming/java/TIJ2e.ru/Chapter09.html;
http://www.bruceeckel.by.ru/tij/Chapter09.html;
http://ru.wikipedia.org/wiki/Хеширование.

Исходный код некоторых классов и интерфейсов пакета java.util (поставляется вместе с Oracle JDK):

Важно также знание основных классов для работы с коллекциями. Особое внимание при изучении коллекций заслуживают классы Comporator, Collections, Iterator.

Ниже приведен краткий разбор наиболее важных классов при работе с коллекциями, а также решения одного из индивидуальных заданий.

Интерфейс Collection

Интерфейс Collection содержит набор общих методов, которые используются в большинстве коллекций. Рассмотрим основные из них:

add(Object item) – добавляет в коллекцию новый элемент, если элементы коллекции каким-то образом упорядочены, новый элемент добавляется в конец коллекции;

clear() – удаляет все элементы коллекции;

contains(Object obj) – возвращает true, если объект obj содержится в коллекции и false, если нет;

isEmpty() – проверяет, пуста ли коллекция;

remove(Object obj) – удаляет из коллекции элемент obj, возвращает false, если такого элемента в коллекции не нашлось;

size() – возвращает количество элементов коллекции.

Интерфейс List

Интерфейс List описывает упорядоченный список. Элементы списка пронумерованы, начиная с нуля и к конкретному элементу можно обратиться по целочисленному индексу. Интерфейс List является наследником интерфейса Collection, поэтому содержит все его методы и добавляет к ним несколько своих:

add(int index, Object item) – вставляет элемент item в позицию index, при этом список раздвигается (все элементы, начиная с позиции index, увеличивают свой индекс на 1);

get(int index) – возвращает объект, находящийся в позиции index;

indexOf(Object obj) – возвращает индекс первого появления элемента obj в списке;

lastIndexOf(Object obj) – возвращает индекс последнего появления элемента obj в списке;

add(int index, Object item) – заменяет элемент, находящийся в позиции index объектом item;

subList(int from, int to) – возвращает новый список, представляющий собой часть данного (начиная с позиции from до позиции to-1 включительно).

Интерфейс Set

Интерфейс Set описывает множество. Элементы множества не упорядочены, множество не может содержать двух одинаковых элементов. Интерфейс Set унаследован от интерфейса Collection, но никаких новых методов не добавляет. Изменяется только смысл метода add(Object item) – он не добавляет объект item, если он уже присутствует во множестве.

Интерфейс Queue

Интерфейс Queue описывает очередь. Элементы могут добавляться в очередь только с одного конца, а извлекаться с другого (аналогично очереди в магазине). Интерфейс Queue так же унаследован от интерфейса Collection. Специфические для очереди методы:

poll() – возвращает первый элемент и удаляет его из очереди.

о методах интерфейса Queue

peek() – возвращает первый элемент очереди, не удаляя его.

offer(Object obj) – добавляет в конец очереди новый элемент и возвращает true, если вставка удалась.

Класс Vector

Vector (вектор) – набор упорядоченных элементов, к каждому из которых можно обратиться по индексу. По сути эта коллекция представляет собой обычный список.

Класс Vector реализует интерфейс List, основные методы которого названы выше. К этим методам добавляется еще несколько. Например, метод firstElement() позволяет обратиться к первому элементу вектора, метод lastElement() – к его последнему элементу. Метод removeElementAt(int pos) удаляет элемент в заданной позиции, а метод removeRange(int begin, int end) удаляет несколько подряд идущих элементов. Все эти операции можно было бы осуществить комбинацией базовых методов интерфейса List, так что функциональность принципиально не меняется.

Класс ArrayList

Класс ArrayList – аналог класса Vector. Он представляет собой список и может использоваться в тех же ситуациях. Основное отличие в том, что он не синхронизирован и одновременная работа нескольких параллельных процессов с объектом этого класса не рекомендуется. В обычных же ситуациях он работает быстрее.

Класс Stack

Stack – коллекция, объединяющая элементы в стек. Стек работает по принципу LIFO (последним пришел – первым ушел). Элементы кладутся в стек «друг на дру-

га», причем взять можно только «верхний» элемент, т.е. тот, который был положен в стек последним. Для стека характерны операции, реализованные в следующих методах класса Stack:

push(Object item) – помещает элемент на вершину стека;

pop() – извлекает из стека верхний элемент;

peek() – возвращает верхний элемент, не извлекая его из стека;

empty() – проверяет, не пуст ли стек;

search(Object item) – ищет «глубину» объекта в стеке. Верхний элемент имеет позицию 1, находящийся под ним – 2 и т.д. Если объекта в стеке нет, возвращает –1.

Класс Stack является наследником класса Vector, поэтому имеет все его методы (и, разумеется, реализует интерфейс List). Однако если в программе нужно моделировать именно стек, рекомендуется использовать только пять вышеперечисленных методов.

Интерфейс Iterator

Преимущество использования массивов и коллекций заключается не только в том, что можно поместить в них произвольное количество объектов и извлекать их при необходимости, но и в том, что все эти объекты можно комплексно обрабатывать. Например, вывести на экран все шашки, содержащиеся в списке checkers. В случае массива мы пользуемся циклом:

```
for (int i = 1; i < array.length; i++){
// обрабатываем элемент array[i]
}
```

Имея дело со списком, мы можем поступить аналогичным образом, только вместо array[i] писать array.get(i). Но мы не можем поступить так с коллекциями, элементы которых не индексируются (например, очередью или множеством). А в случае индексированной коллекции надо хорошо знать особенности ее работы: как определить количество элементов, как обратиться к элементу по индексу, может ли коллекция быть разреженной (т.е. могут ли существовать индексы, с которыми не связано никаких элементов) и т.д.

Для навигации по коллекциям в Java предусмотрено специальное архитектурное решение, получившее свою реализацию в интерфейсе Iterator. Идея заключается в том, что к коллекции «привязывается» объект, единственное назначение которого — выдать все элементы этой коллекции в некотором порядке, не раскрывая ее внутреннюю структуру.

Интерфейс Iterator имеет всего три метода:

next() – возвращает очередной элемент коллекции, к которой «привязан» итератор (и делает его текущим). Порядок перебора определяет сам итератор.

hasNext() – возвращает true, если перебор элементов еще не закончен

remove() – удаляет текущий элемент.

Интерфейс Collection помимо рассмотренных ранее методов, имеет метод iterator(), который возвращает итератор для данной коллекции, готовый к ее обходу. С помощью такого итератора можно обработать все элементы любой коллекции следующим простым способом:

```
Iterator iter = coll.iterator(); // coll - коллекция
```

while (iter.hasNext()) {
// обрабатываем объект, возвращаемый методом iter.next()
}.

Для коллекций, элементы которых проиндексированы, определен более функциональный итератор, позволяющий двигаться как в прямом, так и в обратном направлении, а также добавлять в коллекцию элементы. Такой итератор имеет интрефейс ListIterator, унаследованный от интерфейса Iterator и дополняющий его следующими методами:

previous() – возвращает предыдущий элемент (и делает его текущим);

hasPrevious() – возвращает true, если предыдущий элемент существует (т.е. текущий элемент не является первым элементом для данного итератора);

add(Object item) – добавляет новый элемент перед текущим элементом;

set(Object item) – заменяет текущий элемент;

nextIndex() и previousIndex() – служат для получения индексов следующего и предыдущего элементов соответственно.

В интерфейсе List определен метод listIterator(), возвращающий итератор ListIterator для обхода данного списка.

Интерфейс Map

Интерфейс Map из пакета java.util описывает коллекцию, состоящую из пар «ключ – значение». У каждого ключа только одно значение, что соответствует математическому понятию однозначной функции или отображения (Map). Интерфейс Map содержит следующие методы, работающие с ключами и значениями:

boolean containsKey (Object key) — проверяет наличие ключа key;

boolean containsValue (Object value) — проверяет наличие значения value;

Set entrySet() – представляет коллекцию в виде множества, каждый элемент которого – пара из данного отображения, с которой можно работать методами вложенного интерфейса Map.Entry;

Object get(Object key) – возвращает значение, отвечающее ключу key; Set key-

Set() – представляет ключи коллекции в виде множества;

Object put(Object key, Object value) — добавляет пару «key— value», если такой пары не было, и заменяет значение ключа key, если такой ключ уже есть в коллекции;

void putAll (Map m) – добавляет к коллекции все пары из отображения m;

collection values () – представляет все значения в виде коллекции.

В интерфейс Map вложен интерфейс Map.Entry, содержащий методы работы с отдельной парой.

Вложенный интерфейс Map.Entry

Этот интерфейс описывает методы работы с парами, полученными методом entrySet() из объекта типа Map.

Методы getKey() и getValue() позволяют получить ключ и значение пары, метод setValue (Object value) меняет значение в данной паре.

Пример решения одного из заданий на лабораторную работу №3.

Задание: Вычислить сколько раз каждая буква встречается в тексте.

Решение:

```
import java.util.HashMap;
import java.util.*;
public class Main {
public static void main(String[] args) {
      String txt = "лабораторная работа";
    HashMap<Character, Integer> map = new HashMap<Character, Integer>(40);
    for (int i = 0; i < txt.length(); ++i) {
        char c = txt.charAt(i);
        //проверяем является ли символ буквой
        if (Character.isLetter(c)) {
            if (map.containsKey(c)) {
                map.put(c, map.get(c) + 1);
            } else {
                map.put(c, 1);
            }
        }
    }

    //вывод на экран букв и с частотой их появления
    for (Entry<Character, Integer> entry : map.entrySet()) {System.out.println("бук-
ва: "+entry.getKey()+" кол-во:"+entry.getValue());}
}
}
```

Примеры индивидуальных заданий для выполнения лабораторной работы № 3

1. Ввести строки из файла, записать их в стек. Вывести строки в файл в обратном порядке.
2. Ввести число, занести его цифры в стек. Вывести в число, у которого цифры идут в обратном порядке.
3. Сложить два многочлена заданной степени, если коэффициенты многочленов хранятся в объекте HashMap.
4. Создать стек из элементов каталога.
5. Не используя вспомогательных объектов, переставить отрицательные элементы данного списка в конец, а положительные - в начало этого списка.
6. Организовать вычисления в виде стека.
7. Выполнить попарное суммирование произвольного конечного ряда чисел следующим образом: 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.
8. Задать два стека, поменять информацию местами.
9. Определить класс Stack. Объявить объект класса. Ввести последовательность символов и вывести ее в обратном порядке.
10. Умножить два многочлена заданной степени, если коэффициенты многочленов хранятся в списках.

11. Определить класс Set на основе множества целых чисел, n = размер. Создать методы для определения пересечения и объединения множеств.

12. Программа получает N параметров вызова (аргументы командной строки). Эти параметры – элементы вектора. Строится массив типа double, а на базе этого массива – объект класса DoubleVector. Далее программа выводит в консоль значения элементов вектора в виде: Вектор: 2.3 5.0 7.3.

13. Списки (стеки) I(1..N) и U(1..N) содержат результаты N измерений тока и напряжения на неизвестном сопротивлении R. Найти приближённое число R методом наименьших квадратов.

Лабораторная работа №4

РАЗРАБОТКА ВИЗУАЛЬНЫХ ИНТЕРФЕЙСОВ

Лабораторная работа №4 посвящена построению приложений с использованием графического интерфейса библиотек java.awt и javax.swing. По данной теме из сети Интернет можно рекомендовать следующие ссылки:

http://www.ibm.com/developerworks/ru/edu/j-dw-java-intswing-i.html;

http://www.mexmat.sgu.ru/sites/chairs/prinf/materials/java/lesson8.htm;

http://www.uic.rsu.ru/doc/programming/java/TIJ2e.ru/Chapter13.html;

Графические инструменты в языке Java реализованы с помощью двух пакетов:

– AWT (для доступа загружается пакет java.awt) содержит набор классов, позволяющих выполнять графические операции и создавать элементы управления;

– Swing (для доступа загружается пакет javax.swing) содержит новые классы, в основном аналогичные AWT. К именам классов добавляется J (JButton, JLabel и др).

На данный момент основные классы для построения визуальных интерфейсов содержатся в пакете Swing. Из пакета AWT используются классы для обработки сообщений. Простейшее графическое приложение приведено ниже.

```
import javax.swing.*;
 public final class HelloWorld implements Runnable {
   public static void main(String[] args) {
//Swing имеет собственный управляющий поток (т.н. dispatching thread),
//который работает параллельно с основным (в котором выполняется main())
//потоком. Если основной поток закончит работу (метод main завершится),
//поток отвечающий за работу Swing-интерфейса может продолжать свою работу.
//И даже если пользователь закрыл все окна, программа продолжит свою работу
//(до тех пор, пока жив данный поток). Начиная с Java 6, когда все //компоненты уни-
чтожены, управляющий поток останавливается автоматически.
//Запускаем весь код, работающий в управляющем потоке, даже инициализацию:
          SwingUtilities.invokeLater (new HelloWorld());
     }
     public void run() {
// Создаем окно с заголовком "Hello, World!"
Frame f = new JFrame ("Hello, World!");
// Ранее практиковалось следующее: создавался listener и регистрировался
// на экземпляре главного окна, который реагировал на windowClosing()
// принудительной остановкой виртуальной машины вызовом System.exit()
// Теперь же есть более "правильный" способ задав реакцию на закрытие окна.
// Данный способ уничтожает текущее окно, но не останавливает приложение. Тем
// самым приложение будет работать пока не будут закрыты все окна.
         f.setDefaultCloseOperation (JFrame.DISPOSE_ON_CLOSE);
```

```
// однако можно задать и так:
//            f.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

// Добавляем на панель окна нередактируемый компонент с текстом.
// f.getContentPane().add (new JLabel("Hello, World!")); - старый стиль
f.add(new JLabel("Hello World"));
// pack() "упаковывает" окно до оптимального размера
// всех расположенных в нем компонентов.
f.pack();
// Показать окно
f.setVisible(true);
}}
```

Технология Swing предоставляет механизмы для управления следующими аспектами представления:

– Клавиатура (Swing предоставляет способ перехвата пользовательского ввода);

– Цвета (Swing предоставляет способ менять цвета, которые вы видите на экране);

– Текстовое поле для ввода (Swing предоставляет текстовые компоненты для обработки всех повседневных задач).

JComponent

Базовым классом всей библиотеки визуальных компонентов Swing является JComponent. Это суперкласс других визуальных компонентов. Он является абстрактным классом, поэтому в действительности вы не можете создать JComponent, но он содержит сотни функций, которые каждый компонент Swing может использовать как результат иерархии классов. Класс JComponent обеспечивает инфраструктуру окрашивания для всех компонентов, он знает, как обрабатывать все нажатия клавиш на клавиатуре, его подклассы, следовательно, должны только прослушивать определенные клавиши. Класс JComponent также содержит метод add(), который позволяет добавить другие объекты класса JComponent, так можно добавить любой Swing-компонент к любому другому для создания вложенных компонентов (например, JPanel, содержащую JButton, или даже более причудливые комбинации, например JMenu, содержащее JButton).

JLabel

Самым простым и в то же время основным визуальным компонентом в библиотеке Swing является JLabel, или «метка». К методам этого класса относится установка текста, изображения, выравнивания и других компонентов, которые описывает метка:

get/setText() – получить/установить текст в метке;

get/setIcon() – получить/установить изображение в метке;

get/setHorizontalAlignment – получить/установить горизонтальную позицию текста;

get/setVerticalAlignment() – получить/установить вертикальную позицию текста;

get/setDisplayedMnemonic() – получить/установить мнемонику (подчеркнутый символ) для метки;

get/setLabelFor() – получить/установить компонент, к которому присоединена данная метка; когда пользователь нажимает комбинацию клавиш Alt + мнемоника, фокус перемещается на указанный компонент.

JButton

Основным активным компонентом в Swing является JButton

Методы, используемые для изменения свойств JButton, аналогичны методам JLabel (вы обнаружите, что они аналогичны для большинства Swing-компонентов). Они управляют текстом, изображениями и ориентацией:

get/setText() – получить/установить текст в кнопке;

get/setIcon() – получить/установить изображение в кнопке;

get/setHorizontalAlignment() – получить/установить горизонтальную позицию текста;

get/setVerticalAlignment() – получить/установить вертикальную позицию текста;

get/setDisplayedMnenomic() – получить/установить мнемонику (подчеркнутый символ), которая в комбинации с кнопкой Alt вызывает нажатие кнопки.

JFrame

КлассJFrame является контейнером, позволяющим добавлять к себе другие компоненты для их организации и предоставления пользователю.

JFrame выступает в качестве моста между независимыми от конкретной операционной системы Swing-частями и реальной операционной системой, на которой они работают. JFrame регистрируется как окно и таким образом получает многие свойств окна операционной системы: минимизация/максимизация, изменение размеров и перемещение. Хотя выполнении лабораторной работы достаточно считать JFrame палитрой, на которой вы размещаете компоненты. Перечислим некоторые из методов, которые вы можете вызвать в JFrame для изменения его свойств:

get/setTitle() – получить/установить заголовок фрейма;

get/setState() – получить/установить состояние фрейма (минимизировать, максимизировать и т.д.);

is/setVisible() – получить/установить видимость фрейма, другими словами, отображение на экране;

get/setLocation() – получить/установить месторасположение в окне, где фрейм должен появиться;

get/setSize() – получить/установить размер фрейма;

add() – добавить компоненты к фрейму.

Схемы, модели и события

При построении визуальных приложений в Java нельзя просто случайно разместить их на экране и ожидать от них немедленной работы. Компоненты необходимо разместить в определенные места, реагировать на взаимодействие с ними, обновлять их на основе этого взаимодействия и заполнять данными. Для эффективной работы с визуальными компонентами необходима установка следующих трех архитектурных составляющих Swing.

1. Схемы (layout). Swing содержит множество схем, которые представляют собой классы, управляющие размещением компонентов в приложении и тем, что должно произойти с ними при изменении размеров окна приложения или при удалении или добавлении компонентов.

2. События (event). Программа должна реагировать на нажатия клавиш, нажатия кнопки мыши и на все остальное, что пользователь может сделать.

3. Модели (model). Для более продвинутых компонентов (списки, таблицы, деревья) и даже для некоторых более простых, например, JComboBox, модели – это самый эффективный способ работы с данными. Они удаляют большую часть работы по обработке данных из самого компонента (вспомните обсуждение MVC) и предоставляют оболочку для общих объектных классов данных (например, Vector и ArrayList).

Особое внимание, в связи с необходимостью изображения динамических сцен на визуальных компонентах необходимо уделить классу Graphics2D.

Пример решения одного из заданий на лабораторную работу №4

Задание: Отобразить вращение треугольника вокруг своего центра тяжести

Решение:

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

import javax.swing.*;
public class Main {
public static void main(String[] args) {
JFrame fr=new JFrame("Вращение треугольника вокруг своего центра тяжести");
      fr.setPreferredSize(new Dimension(300,300));
      final JPanel pan= new JPanel();
      fr.add(pan);
      fr.setVisible(true);
      fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      fr.pack();
      Timer tm= new Timer(500, new ActionListener(){
            int i=0;
            @Override
            public void actionPerformed(ActionEvent arg0) {
                  Graphics2D gr=(Graphics2D)pan.getRootPane().getGraphics();
                  pan.update(gr);

                  GeneralPath path=new GeneralPath();
path.append(new Polygon(new int []{60,-80,50},new int[]{-60,-50,40},3),true);
                  int x=(60-80+50)/3,y=(-60-50+40)/3;
                  gr.translate(150, 150);

                  AffineTransform tranforms = AffineTransform.getRotateInstance((i+
+)*0.07, x, y);
            gr.transform(tranforms);
                  gr.draw(path);
            }});
      tm.start();
}
}
```

Примеры индивидуальных заданий для выполнения лабораторной работы №4

1. Задать движение по экрану строк (одна за другой) из массива строк. Направление движения по апплету и значение каждой строки выбирается случайным образом.

2. Задать движение окружности по апплету так, чтобы при касании границы окружность отражалась от нее с эффектом упругого сжатия.

3. Изобразить в апплете приближающийся издали шар, удаляющийся шар. Шар должен двигаться с постоянной скоростью.

4. Изобразить в окне приложения отрезок, вращающийся в плоскости экрана вокруг одной из своих концевых точек. Цвет прямой должен изменяться при переходе от одного положения к другому.

5. Изобразить в окне приложения отрезок, вращающийся в плоскости фрейма вокруг точки, движущейся по отрезку.

6. Изобразить четырехугольник, вращающийся в плоскости апплета вокруг своего центра тяжести.

7. Создать фрейм с областью для рисования «пером». Создать меню для выбора цвета и толщины линии.

8. Составить программу для управления скоростью движения точки по апплету. Одна кнопка увеличивает скорость, другая – уменьшает. Каждый щелчок изменяет скорость на определенную величину.

9. Изобразить в окне гармонические колебания точки вдоль некоторого горизонтального отрезка. Если длина отрезка равна q, то расстояние от точки до левого конца в момент времени t можно считать равным q(1 + cos(wt))/2, где w – некоторая константа. Предусмотреть поля для ввода указанных величин и кнопку для остановки и пуска процесса.

10. Создать апплет со строкой движущейся по диагонали. При достижении границ апплета все символы строки случайным образом меняют регистр. При этом шрифт меняется на шрифт, выбранный из списка.

11. Создать апплет со строкой движущейся горизонтально, отражаясь от границ апплета и меняя при этом свой цвет, на цвет выбранный из выпадающего списка.

12. Промоделировать вращение спутника вокруг планеты по эллиптической орбите. Когда скрывается за планетой – спутник не виден.

13. Промоделировать аналоговые часы (со стрелками) с кнопками для увеличения/уменьшения времени на час/минуту.

Лабораторная работа №5 – Работа по сети

В лабораторной работе №5 необходимо реализовать сетевое приложение. Язык Java делает сетевое программирование простым благодаря наличию специальных средств и классов. Рассмотрим некоторые виды сетевых приложений. Internet-приложения включают Web-броузер, e-mail, сетевые новости, передачу файлов и telnet. Основной используемый протокол – TCP/IP.

Приложения клиент/сервер используют компьютер, выполняющий специальную программу - сервер, которая предоставляет услуги другим программам - клиентам.

Клиент – это программа, получающая услуги от сервера. Клиент-серверные приложения основаны на использовании верхнего уровня протоколов. На TCP/IP основаны следующие протоколы:

HTTP – Hypertext Transfer Protocol (WWW);
NNTP – Network News Transfer Protocol (группы новостей);
SMTP – Simple Mail Transfer Protocol (посылка почты);
POP3 – Post Office Protocol (чтение почты с сервера);
FTP – File Transfer Protocol (протокол передачи файлов);
TELNET – Удаленное управление компьютерами.

Каждый компьютер по протоколу TCP/IP имеет уникальный IP-адрес. Это 32-битовое число, обычно записываемое как четыре числа, разделенные точками, каждое из которых изменяется от 0 до 255. IP-адрес может быть временным и выделяться динамически для каждого подключения или быть постоянным, как для сервера. Обычно при подключении к компьютеру вместо числового IP адреса используются символьные имена (например - www.bsu.iba.by), называемые именами домена. Специальная программа DNS (Domain Name Sever) преобразует имя домена в числовой IP-адрес. Получить IP-адрес в программе можно с помощью объекта класса InetAddress из пакета java.net.

```
import java.net.*;
public class MyLocal {
	public static void main(String[] args) {
		InetAddress myIP = null;
		try {
			myIP = InetAddress.getLocalHost();
		} catch (UnknownHostException e) {
			System.out.println("ошибка доступа ->" + e);
		}
		System.out.println("Мой IP ->" + myIP);
	}
```

Следующая программа демонстрирует, как получить IP-адрес из имени домена с помощью сервера имен доменов (DNS), к которому обращается метод getByName().

```
import java.net.*;

public class IPfromDNS {
	public static void main(String[] args) {
		InetAddress omgtu = null;
		try {
			omgtu = InetAddress.getByName("omgtu.ru");
		} catch (UnknownHostException e) {
			System.out.println("ошибка доступа ->" + e);
		}
		System.out.println("IP-адрес ->" + omgtu);
	}
}
```

Будет выведено: IP-адрес –>omgtu.ru/195.69.204.35

Сокеты – это сетевые разъемы, через которые осуществляются двунаправленные поточные соединения между компьютерами. Сокет определяется номером порта и IP-адресом. При этом IP-адрес используется для идентификации компьютера, номер порта – для идентификации процесса, работающего на компьютере. Когда одно при-

ложение знает сокет другого, создается сокетное соединение. Клиент пытается соединиться с сервером, инициализируя сокетное соединение. Сервер ждет, пока клиент не свяжется с ним. Первое сообщение, посылаемое клиентом на сервер, содержит сокет клиента. Сервер в свою очередь создает сокет, который будет использоваться для связи с клиентом, и посылает его клиенту с первым сообщением. После этого устанавливается коммуникационное соединение.

Сокетное соединение с сервером создается с помощью объекта класса Socket. При этом указывается IP-адрес сервера и номер порта (80 для HTTP). Если указано имя домена, то Java преобразует его с помощью DNS-сервера к IP-адресу:

```
try {
      Socket socket = new Socket("localhost", 8030);
       } catch (IOException e) {
      System.out.println("ошибка: " + e);
   }.
```

Сервер ожидает сообщения клиента и должен быть запущен с указанием определенного порта. Объект класса ServerSocket создается с указанием конструктору номера порта и ожидает сообщения клиента с помощью метода accept(), который возвращает сокет клиента:

```
       Socket socket = null;
       try {
             ServerSocket server = new ServerSocket(8030);
             socket = server.accept();
              } catch (IOException e) {
            System.out.println("ошибка: " + e);
        }.
```

Клиент и сервер после установления сокетного соединения могут получать данные из потока ввода и записывать данные в поток вывода с помощью методов getInputStream() и getOutputStream() или к PrintStream для того, чтобы программа могла трактовать поток как выходные файлы.

В следующем примере для посылки клиенту строки "привет!" сервер вызывает метод getOutputStream() класса Socket. Клиент получает данные от сервера с помощью метода getInputStream(). Для разъединения клиента и сервера после завершения работы сокет закрывается с помощью метода close() класса Socket. В данном примере сервер посылает клиенту строку "привет!", после чего разрывает связь.

```
// передача клиенту строки : MyServerSocket.java
import java.io.*;
import java.net.*;

public class MyServerSocket {
      public static void main(String[] args) throws Exception {
            Socket s = null;
            try {// посылка строки клиенту
                  ServerSocket server = new ServerSocket(8030);
                  s = server.accept();
                  PrintStream ps = new PrintStream(s.getOutputStream());
                  ps.println("привет!");
                  ps.flush();
                  s.close(); // разрыв соединения
            } catch (IOException e) {
                  System.out.println("ошибка: " + e);
```

```
            }
      }
}

/* получение клиентом строки : MyClientSocket.java */
import java.io.*;
import java.net.*;

public class MyClientSocket {
      public static void main(String[] args) {
            Socket socket = null;
            try {// получение строки клиентом
                  socket = new Socket("имя_компьютера", 8030);
                  BufferedReader dis = new BufferedReader(new InputStreamReader(
                              socket.getInputStream()));
                  String msg = dis.readLine();
                  System.out.println(msg);
            } catch (IOException e) {
                  System.out.println("ошибка: " + e);
            }
      }
   }
```

Аналогично клиент может послать данные серверу через поток вывода с помощью метода getOutputStream(), а сервер может получать данные с помощью метода getInputStream().

Если необходимо протестировать подобный пример на одном компьютере, можно выступать одновременно в роли клиента и сервера, используя статические методы getLocalHost() класса InetAddress для получения динамического IP-адреса компьютера, который выделяется при входе в Internet.

Примеры индивидуальных заданий для выполнения лабораторной работы №5

Создать на основе сокетов клиент/серверное визуальное приложение

1. Клиент посылает через сервер сообщение другому клиенту.
2. Клиент посылает через сервер сообщение другому клиенту, выбранному из списка.
3. Чат. Клиент посылает через сервер сообщение, которое получают все клиенты. Список клиентов хранится на сервере в файле.
4. Клиент при обращении к серверу получает случайно выбранный сонет Шекспира из файла.
5. Сервер рассылает сообщения выбранным из списка клиентам. Список хранится в файле.
6. Сервер рассылает сообщения в определенное время определенным клиентам.
7. Сервер рассылает сообщения только тем клиентам, которые в настоящий момент находятся в on-line.
8. Чат. Сервер рассылает всем клиентам информацию о клиентах вошедших в чат и покинувших его.
9. Клиент выбирает изображение из списка и пересылает его другому клиенту через сервер.
10. Игра по сети в «Морской бой».

11. Игра по сети в «21».

12. Игра по сети в классические «Крестики-нолики».

13. Игра по сети в «Го» («крестики-нолики» на безразмерном (большом) поле. Для победы необходимо выстроить пять в один ряд).

Лабораторная работа №6

СОЗДАНИЕ ВЕБ-ПРИЛОЖЕНИЙ

В лабораторной работе №6 необходимо создать веб-приложение (Java 2 Enterprise Edition), разместив его на веб-сервере Tomcat. На настоящий момент более 90% корпоративных систем поддерживают Java 2 Enterprise Edition. Эта платформа позволяет быстро и без особых издержек объединить возможности сети Интернет и корпоративных информационных систем. Сервлеты – это компоненты приложений Java 2 Platform Enterprise Edition (J2EE), выполняющиеся на стороне сервера, способные обрабатывать клиентские запросы и динамически генерировать ответы на них. Наибольшее распространение получили сервлеты, обрабатывающие клиентские запросы по протоколу HTTP. Сервлет может применяться, например, для создания серверного приложения, получающего от клиента запрос, анализирующего его и делающего выборку данных из базы, а также пересылающего клиенту страницу HTML, сгенерированную с помощью JSP на основе полученных данных.

Все сервлеты реализуют общий интерфейс **Servlet**. Для обработки HTTP-запросов можно воспользоваться в качестве базового класса абстрактным классом **HttpServlet**. Базовая часть классов JSDK помещена в пакет **javax.servlet**. Однако класс **HttpServlet** и все, что с ним связано, располагаются на один уровень ниже в пакете **javax.servlet.http**.

Жизненный цикл сервлета начинается с его загрузки в память контейнером сервлетов при старте либо в ответ на первый запрос. Далее происходят инициализация, обслуживание запросов и завершение существования.

Первым вызывается метод **init()**. Он дает сервлету возможность инициализировать данные и подготовиться для обработки запросов. Чаще всего в этом методе программисты помещают код, кэширующий данные фазы инициализации.

После этого сервлет можно считать запущенным, он находится в ожидании запросов от клиентов. Появившийся запрос обслуживается методом **service()** сервлета, а все параметры запроса упаковываются в объект **ServletRequest**, который передается в качестве первого параметра методу **service()**. Второй параметр метода – объект **ServletResponse**. В этот объект упаковываются выходные данные в процессе формирования ответа клиенту. Каждый новый запрос приводит к новому вызову метода **service()**. В соответствии со спецификацией JSDK, метод **service()** должен уметь обрабатывать сразу несколько запросов, т.е. быть синхронизирован для выполнения в многопоточных средах. Если же нужно избежать множественных запросов, сервлет должен реализовать интерфейс **SingleThreadModel**, который не содержит ни одного метода и только указывает серверу об однопоточной природе сервлета. При обращении к такому сервлету каждый новый запрос будет ожидать в очереди, пока не завершится обработка предыдущего запроса.

После завершения выполнения сервлета контейнер сервлетов вызывает метод **destroy()**, в теле которого следует помещать код освобождения занятых сервлетом ресурсов.

Интерфейсом **Servlet** предусмотрена реализация еще двух методов: **getServletConfig()** и **getServletInfo()**. Первый возвращает объект типа **ServletConfig**, содержащий параметры конфигурации сервлета, а второй – строку, описывающую назначение сервлета.

При разработке сервлетов в качестве базового класса в большинстве случаев используют не интерфейс **Servlet**, а класс **HttpServlet**, отвечающий за обработку запросов HTTP.

Класс **HttpServlet** имеет реализованный метод **service()**, служащий диспетчером для других методов, каждый из которых обрабатывает методы доступа к ресурсам. В спецификации HTTP определены следующие методы: **GET**, **HEAD**, **POST**, **PUT**, **DELETE**, **OPTIONS** и **TRACE**. Наиболее часто употребляются методы **GET** и **POST**, с помощью которых на сервер передаются запросы, а также параметры для их выполнения.

При использовании метода GET (по умолчанию) параметры передаются как часть URL, значения могут выбираться из полей формы или передаваться непосредственно через URL. При этом запросы кэшируются и имеют ограничения на размер. При использовании метода POST (method=POST) параметры (поля формы) передаются в содержимом HTTP-запроса и упакованы согласно полю заголовка Content-Type. По умолчанию в формате: <имя>=<значение>&<имя>=<значение>&...

Однако форматы упаковки параметров могут быть самые разные, например: в случае передачи файлов с использованием формы

`enctype="multipart/form-data"`.

В задачу метода **service()** класса **HttpServlet** входит анализ полученного через запрос метода доступа к ресурсам и вызов метода, имя которого сходно с названием метода доступа к ресурсам, но перед именем добавляется префикс **do**: **doGet()** или **doPost()**. Кроме этих методов могут использоваться методы: **doHead()**, **doPut()**, **doDelete()**, **doOptions()** и **doTrace()**. Разработчик должен переопределить нужный метод, разместив в нем функциональную логику.

В следующем примере приведен готовый к выполнению «шаблон» сервлета:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class MyServlet extends HttpServlet {
/**  Constructor of the object.    */
     public MyServlet() {
          super();
     }
/** Destruction of the servlet. */
     public void destroy() {
super.destroy(); // Just puts "destroy" string in log
          // Put your code here
     }
/** The doDelete method of the servlet.
```

```
 * This method is called when a HTTP delete request is received.
 * @param request the request send by the client to the erver
 * @param response the response send by the server to the client
 * @throws ServletException if an error occurred
 * @throws IOException if an error occurred */
     public void doDelete(HttpServletRequest request,
                HttpServletResponse response)
          throws ServletException, IOException {
          // Put your code here
     }
/** The doGet method of the servlet.
* This method is called when a form has its tag value method equals to get.
 * @param request the request send by the client to the server
 * @param response the response send by the server to the client
 * @throws ServletException or @throws IOException if an error occurred  */
     public void doGet(HttpServletRequest request, HttpServletRe-sponse response)
      throws ServletException, IOException {
          response.setContentType("text/html");
          this.preventCaching(request, response);
          PrintWriter out = response.getWriter();
          out.println( "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01
Transitional//EN\">");
          out.println("<HTML><HEAD>");
     out.println("<TITLE>A Servlet in GET</TITLE>");
          out.println("</HEAD><BODY>");
          out.print("    This is <B>");
          out.print(this.getClass().getName());
out.println("</B>, using the <B>GET</B> method<BR>");
          out.println("</BODY></HTML>");
          out.flush();
          out.close();
     }
/** The doPost method of the servlet.
 * This method is called when a form has its tag value method equals to post.
 * @param request the request send by the client to the server
 * @param response the response send by the server to the client
 * @throws ServletException or @throws IOException if an error occurred */
     public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
          response.setContentType("text/html");
          this.preventCaching(request, response);
          PrintWriter out = response.getWriter();
          out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01
Transitional//EN\">");
          out.println("<HTML><HEAD>");
     out.println("<TITLE>A Servlet in POST</TITLE>");
          out.println("</HEAD><BODY>");
          out.print("This is <B>");
          out.print(this.getClass().getName());
out.println("</B>, using the <B>POST</B> method");
          out.println("</BODY></HTML>");
          out.flush();
          out.close();
     }
/** The doPut method of the servlet.
 * This method is called when a HTTP put request is received.
 * @param request the request send by the client to the server
 * @param response the response send by the server to the client
 * @throws ServletException or @throws IOException if an error occurred */
     public void doPut(HttpServletRequest request,            HttpServletResponse re-
sponse)
          throws ServletException, IOException {
          // Put your code here
     }
```

```
/** Returns information about the servlet, such as
 * author, version, and copyright.
 * @return String information about this servlet */
     public String getServletInfo() {
     return "This is my default servlet created by Eclipse";
     }
/* Initilaisation of the servlet.
 * @throws ServletException if an error occure */
     public void init() throws ServletException {
          // Put your code here
     }
/** Prevents navigator from caching data.
 * @param request the request send by the client to the server
 * @param response the response send by the server to the client
 */
     protected void preventCaching(HttpServletRequest request,HttpServletResponse re-
sponse) {
/* see http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html */
          String protocol = request.getProtocol();
if ("HTTP/1.0".equalsIgnoreCase(protocol)) {
          response.setHeader("Pragma", "no-cache");
} else if ("HTTP/1.1".equalsIgnoreCase(protocol)) {
     response.setHeader("Cache-Control", "no-cache");
               }
          response.setDateHeader("Expires", 0);
     }
}
```

Примеры индивидуальных заданий для выполнения лабораторной работы №6

Создать сервлет и взаимодействующие с ним пакеты Java-классов и HTML-документов, выполняющие следующие действия:

1. Генерация таблиц по переданным параметрам: заголовок, количество строк и столбцов, цвет фона.
2. Вычисление тригонометрических функций в градусах и радианах с указанной точностью. Выбор функций должен осуществляться через выпадающий список.
3. Поиск слова, введенного пользователем. Поиск и определение частоты встречаемости осуществляется в текстовом файле, расположенном на сервере.
4. Вычисление объемов тел (параллелепипед, куб, сфера, тетраэдр, тор, шар, эллипсоид и т.д.) с точностью и параметрами, указываемыми пользователем.
5. Поиск и (или) замена информации в коллекции по ключу (значению).
6. Выбор текстового файла из архива файлов по разделам (поэзия, проза, фантастика и т.д.) и его отображение.
7. Выбор изображения по тематике (природа, автомобили, дети и т.д.) и его отображение.
8. Информация о среднесуточной температуре воздуха за месяц задана в виде списка, хранящегося в файле. Определить: а) среднемесячную температуру воздуха; б) количество дней, когда температура была выше среднемесячной; в) количество дней, когда температура опускалась ниже ; г) три самых теплых дня.
9. Реализация адаптивного теста из цепочки в 3 – 4 вопроса.
10. Вывод фрагментов текстов шрифтами различного размера. Размер шрифта и количество строк задается на стороне клиента.

11. Информация о точках на плоскости хранится в файле. Выбрать все точки, наиболее приближенные к заданной прямой. Параметры прямой и максимальное расстояние от точки до прямой вводятся на стороне клиента.

12. Осуществить сортировку введенного пользователем массива целых чисел. Числа вводятся через запятую.

13. Осуществить форматирование выбранного пользователем текстового файла, так чтобы все абзацы имели отступ ровно 3 пробела, а длина каждой строки была ровно 80 символов и не имела начальными и конечными символами пробел.