ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САМАРСКИЙ ГОСУДАРСТВЕНЫЙ УНИВЕРСИТЕТ»

Программирование на языке JAVA

Задания к лабораторным работам

САМАРА 2015

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРТСВЕННЫЙ УНИВЕРСИТЕТ»

Программирование на языке JAVA

Методические указания

САМАРА

2015

УДК 510.2

Составители ***Т.П.Рубцова***

Рецензент к.ф.-м.н. Ширяева Л. К.

Java –технологии: метод. указания / сост. *Т.П. Рубцова* – Самара, 2015. –40 с.: ил.

Предназначено для студентов специальностей «Математическое обеспечение и администрирование информационных систем», «Прикладная математика», «Фундаментальная математика и механика», изучающих данную тему по курсам «Программирование», «Java – технологии» и «Практикум на ЭВМ».

Подготовлено на кафедре информатики и вычислительной математики.

УДК 510.2

СОДЕРЖАНИЕ

Введение

История происхождения языка Java достаточно интересна. Первые идеи появились у сотрудника Sun Microsystems Патрика Нотона (Patrick Naughton) и в некотором смысле были связаны с его недовольством наличием и необходимостью поддержания достаточно большого количества используемых в компании интерфейсов программ. На основании его письма руководству, содержавшего перечень увиденных недостатков и проблем в архитектуре программного обеспечения (ПО), которое на тот момент разрабатывали в компании, было принято решение о необходимости создания команды ведущих разработчиков, целью которых стало бы осуществление новых подходов к программированию ПО. Одной из целей было разработать такой язык, который позволял бы разрабатывать ПО для использования на различных процессорах и под различными операционными системами.

Результатом работы этой команды стал объектно-ориентированный язык программирования Oak, который оказался мощным инструментом разработки программ, работающих в сетевом окружении. Более того, его объекты могли работать на любом устройстве, а не только на персональном компьютере. Этот язык был ориентирован на распределенную архитектуру и имел механизмы безопасности, шифрования, процедур аутентификации, причем встроенные и, следовательно, незаметные и удобные для пользователя [4].

Затем была выпущена первая версия графического браузера Mosaic, основанного на языке HTML (HyperText Markup Language), что привело к бурному развитию Internet. До этого момента Глобальная Сеть использовалась лишь в академической и государственной среде по причине недостаточного удобства применения.

Патрик Нотон предложил использовать язык Oak для разработки Internet-приложений. Были разработаны соответствующие компилятор и браузер. А в 1995 году продукт был переименован в Java.

На начальном этапе web-страницы были статическими, но затем стала реализовываться возможность двунаправленной передачи данных (например, для целей электронной торговли). Кроме того, в некоторых случаях часть обработки данных может производиться на клиентской машине, однако сервер не обязательно «знает», каков тип компьютера у клиента. В языке Java присутствуют средства для решения такого рода задач [2].

Особенности языка программирования Java:

a) *Кроссплатформенность*. Довольно частой является ситуация, когда заказчикам требуется одинаковая функциональность на различных платформах при использовании процессоров с разной архитектурой, что приводит к необходимости переноса приложений. Эта задача не нова, однако далеко не во всех случаях получается осуществить перенос корректно, поскольку поддержка многих возможностей может существенно различаться. Для решения этой проблемы применяется механизм виртуальной машины. В данном случае для исполнения приложений, написанных с использованием Java, применяется среда Java Virtual Machine (JVM) [6]. Она же определяет многие свойства Java. Средства создания Java-приложений разработаны для различных платформ: Linux, Solaris, Windows, MacOS и т. д.

b) *Объектная ориентированность*. Этот язык изначально позиционировался как объектно-ориентированный, здесь практически все реализовано в виде объектов, включая потоки выполнения, потоки данных, работу с сетью, изображениями и пользовательским интерфейсом, обработку ошибок. Фактически приложение на Java представляет собой набор классов, описывающих новые типы объектов. Все принципы объектно-ориентированного программирования (ООП) используются здесь в полном объеме.

Особенностями объектной модели Java являются:

–Отказ от множественного наследования, которое могло излишне усложнить и запутать программу; здесь используется специальный тип «интерфейс».

–Наличие строгой типизации приводит к тому, что любые переменные и выражения имеют известный на момент компиляции тип, а это, в свою очередь,

упрощает выявление проблем. Однако поиск исключительных ситуаций (о них будет рассказано далее) во время исполнения программы требует сложного тестирования, то есть в данном случае повышение надежности кода будет достигаться за счет дополнительных усилий при его написании [4].

–Сходство с C/C++, поскольку на момент создания Java C++ считался основным инструментом разработки и незнакомый синтаксис мог осложнить внедрение, однако модель и идеология Java были построены в соответствии с поставленными целями (что в ряде случаев позволяет говорить о том, что Java – упрощенный вариант C++).

–Легкость в освоении и разработке, например за счет работы с памятью: в Java изначально был предусмотрен механизм автоматической сборки мусора. Кроме того, начиная с самой первой версии, в Java присутствует возможность создания многопоточных приложений. За счет использования многопоточности в интерактивном графическом приложении удается достичь высокой производительности, что особенно актуально в распределенных приложениях, когда процессы сетевого обмена могут идти одновременно и асинхронно. При этом программа продолжает реагировать на ввод информации пользователем без неприятных задержек. Многопоточность поддерживается на уровне языка, а системные библиотеки могут быть использованы в такого рода приложениях. Приложение легко сопровождается и модифицируется. Система обеспечивает динамическую сборку программы, т. е. классы подгружаются по мере необходимости с любой точки сети, что позволяет сделать внесение изменений в приложения прозрачным для пользователя.

c) Сокращение цикла разработки приложений. Сокращение происходит за счет того, что система построена на основе интерпретатора. Исходный код программы на Java представляет собой текстовые файлы, создаваемые в текстовом редакторе или специализированном средстве разработки и имеющие расширение .java. Они подаются на вход Java-компилятора, транслирующего их в байт-код. Этот набор инструкций поддерживается JVM и является неотъемлемой частью платформы Java. Результат работы компилятора сохраняется в бинарных файлах с

расширением .class. Java-приложение, состоящее из таких файлов, подается на вход виртуальной машины для исполнения. Первоначально байт-код каждый раз интерпретировался, что значительно замедляло работу приложений. Затем была предложена схема, называемая JIT-компиляцией (Just-InTime), при применении которой для инструкции или набора инструкций в первый раз происходит компиляция соответствующего байт-кода с сохранением скомпилированного кода в буфере, а затем содержимое буфера используется при повторном вызове. Несколько отличается механизм работы оптимизирующих JIT-компиляторов: так как компиляция инструкции обычно идет гораздо дольше ее интерпретации, время ее выполнения в первый раз при наличии JIT-компиляции может быть заметно больше, чем при чистой интерпретации, а, значит, выгоднее сначала запустить процесс интерпретации, а параллельно в фоновом режиме компилировать инструкцию. После окончания компиляции при последующих вызовах инструкции будет исполняться ее скомпилированный код, а до этого момента все вызовы будут интерпретироваться. Разработанная Sun виртуальная машина HotSpot осуществляет JIT-компиляцию только тех участков байт-кода, которые критичны к времени выполнения программы. При этом по ходу работы программы происходит оптимизация [7].

d) *Безопасность*. Вопрос безопасности, пожалуй, является наиболее важным. Поэтому при работе виртуальной машины применяется комплекс мер, связанных с работой с памятью и отсечением опасного кода на каждом этапе работы. При этом анализируется, подчиняются ли class-файлы общим правилам безопасности Java или были созданы злоумышленниками с помощью какихто других средств, что позволяет интерпретатору в дальнейшем проверить каждое действие на допустимость. Следует отметить, что возможности классов, загруженных с локального диска или по сети, существенно различаются. Кроме этого механизм подписания апплетов и других приложений, загружаемых по сети, позволяет клиенту либо отказаться от работы с приложениями ненадежных производителей, либо сразу увидеть, что в программу внесены неавторизованные

изменения, а специальный сертификат должен гарантировать, что пользователь получил код именно в том виде, в каком его выпустил производитель.

Лабораторная работа № 1. Первая программа

Поскольку язык Java изначально является объектно-ориентированным, то любое написанное с его помощью приложение (даже самое простое) будет содержать класс. Для разработки приложений на языке Java требуется компилятор и виртуальная машина, а для удобства разработки можно использовать интегрированную среду разработки (IDE), но это не является обязательным.

Целью данной лабораторной работы будет создание простейшей программы, с написания которой начинается изучение практически любого языка программирования, а именно программы, выводящей на экран фразу «Hello, World!». В данном случае код будет выглядеть следующим образом:

```
public class Hello {
    public static void main(String args[]) {
      System.out.println(«Hello, World!»);
    }
}
```

Эта программа, как и любая другая на Java, представляет собой набор классов. В рассматриваемом случае это единственный класс Hello. Для того чтобы программу можно было запустить, в ней должен присутствовать класс, который содержит открытую (public) функцию main, имеющую прототип, как в примере. Эта функция должна быть статической, что нужно для ее вызова без создания экземпляра класса. Для вывода на экран в текстовом режиме (в консоль) используется фунция System.out.println(), которая позволяет выводить различные объекты: строки, символы, числа и др.

Код необходимо сохранить в файле Hello.java. Применение именно такого имени файла является необходимым в соответствии с требованиями языка. Заметьте, что имя файла должно строго соответствовать имени объявленного в нем класса.

Для компиляции должен быть установленным JDK – (Java Development Kit) и/или одна из известных сред разработки (например, Eclipse).

Компиляция программы производится командой:

$ javac Hello.java

После компиляции будет создан файл Hello.class. А запуск программы производится командой:

$ java Hello

Рассмотрим пример преобразовывания градусы Фаренгейта в градусы Цельсия используя следующую формулу:

FahrenheitToCelsius.java

Источник | История

```
1  import java.util.Scanner;
2
3  public class FahrenheitToCelsius {
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6
7          System.out.print("Введите градусы в фаренгейтах: ");
8          double fahrenheit = input.nextDouble();
9
10         // Конвертация фаренгейтов в Цельсии
11         double celsius = (5.0 / 9) * (fahrenheit - 32);
12         System.out.println("Фаренгейт " + fahrenheit + " это "
13                 + celsius + " в градусах Цельсия");
14     }
15 }
```

Рис. 1.Листинг программы

Результат работы программы представлен на рисунке 2.

Вывод - FahrenheitToCelsius (run)

```
run:
Введите градусы в фаренгейтах: 100
Фаренгейт 100.0 это 37.77777777777778 в градусах Цельсия
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 2 секунды)
```

Рис. 2.Результат работы программы

Задания для самостоятельного выполнения

Составить программу вычисления значений функции y=f(x) при заданном значении x, которое вводится с клавиатуры.

Таблица 1

№	y=f(x)	Исходные данные	№	y=f(x)	Исходные данные
1	$y=\frac{\sqrt{x+62{,}7e^{cx}}}{ax^2+7x+b\ln x}$	$a=6{,}2$ $b=14{,}3$ $c=13{,}4$ $x=5{,}6$	9	$y=\frac{ax+3{,}8tgx}{\sqrt{cx^3+cb}}$	$a=1{,}53$ $b=5{,}14$ $c=3{,}97$ $x=7{,}1$
2	$y=\left(\frac{a}{bx^2+c}+b\sin^2 x\right)^2$	$a=2{,}27$ $b=1{,}18$ $c=3{,}92$ $x=0{,}78$	10	$y=\sqrt{\frac{ax^3+arctgx}{cx+b\lvert\ln x\rvert}}$	$a=2{,}71$ $b=1{,}63$ $c=0{,}81$ $x=0{,}51$
3	$y=\left(a\sqrt{4{,}19x^3-c}-\sqrt{b\ln x+c}\right)^{-1}$	$a=9{,}2$ $b=3{,}5$ $c=12{,}3$ $x=3{,}2$	11	$y=\frac{ax}{\sqrt{b^2+2e^x-bx}}$	$a=6{,}32$ $b=3{,}704$ $x=7{,}15$
4	$y=\ln\lvert a\sin x+b\cos(c*x^2)\rvert$	$a=1{,}2$ $b=2{,}3$ $x=5{,}6$	12	$y=\cos(ax)+b$	$a=7{,}1$ $b=1{,}8$ $x=0{,}9$
5	$y=\frac{\sqrt{e^{ax}+x^2}\cdot\ln(x^2+bx+10)}{\sin(cx)+4{,}2}$	$a=5{,}7$ $b=6{,}4$ $c=3{,}1$ $x=2{,}8$	13	$y=\sin(ax)+b$	$a=7{,}1$ $b=1{,}8$ $x=0{,}9$
6	$y=\frac{\sqrt{e^{2x+b}}-1{,}7\cos(cx)}{\ln(x^2+a)}+x^3$	$a=2{,}1$ $b=5{,}3$ $c=1{,}4$ $x=1{,}2$	14	$y=\frac{(cx)^2-e^{bx}}{\sqrt{x}+\cos(ax)}$	$a=4{,}5$ $b=2{,}2$ $c=1{,}67$ $x=2{,}36$
7	$y=\frac{\sin\sqrt{e^x+ax^2+b\ln x}}{ax^2+cx+13{,}7}$	$a=3{,}7$ $b=4{,}9$ $c=2{,}5$ $x=1{,}3$	15	$y=\frac{\sin(x^2+a^2)\cdot e^{b+x}}{\sqrt{ax^3+c}}$	$a=4{,}26$ $b=1{,}71$ $c=3{,}86$ $x=2{,}73$
8	$y=\sqrt{\frac{a}{1+bx^2}}+b\,ctg\,x+e^{cx}$	$a=4{,}5$ $b=2{,}2$ $c=-1{,}5$ $x=0{,}85$		$y=\frac{\ln\sqrt{x^2+b}+cx^3}{e^x+a}$	$a=4{,}7$ $b=7{,}21$ $c=1{,}72$ $x=0{,}91$

Лабораторная работа № 2. Использование условных операторов

Одновариантная инструкция ***if*** выполняет действие, если условие является ***true*** (истина). Синтаксис одновариантных выражений **if** следующий:

```
1   if (булево-выражение) {
2       инструкция(ии);
3   }
```

Следующая программа просит пользователя ввести целое число. Если число кратно 5, то программа печатает Дай Пять! Если число кратно 2, то программа показывает Дайте Две!

```
1  import java.util.Scanner;
2
3  public class SimpleIfDemo {
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6          System.out.println("Введите целое число: ");
7          int number = input.nextInt();
8
9          if (number % 5 == 0) {
10             System.out.println("Дай Пять!");
11         }
12         if (number % 2 == 0) {
13             System.out.println("Дайте Две!");
14         }
15     }
16 }
```

Рис. 3.Листинг программы

```
Введите целое число:
4
Дайте Две!
```

```
Введите целое число:
30
Дай Пять!
Дайте Две!
```

Рис. 4.Результат работы программы

Задания для самостоятельного выполнения

Составить программу вычисления значений функции y = f(x) при произвольных значениях x. Получить результат работы программы для двух заданных значений **x**. Варианты заданий в таблице 2.

Таблица 2.

№.	Y=f(x)	№	
1	$y=\begin{cases} b+2\ln\lvert x\rvert & при\ x\le 3,\\ \dfrac{x^2}{x^2+a} & при\ x>3 \end{cases}$	9	$y=\begin{cases} \sqrt{1+x\sqrt{ax}} & при\ x\ge 2,\\ a*\sin(bx)+3 & при\ x<2 \end{cases}$
2	$y=\begin{cases} a+\dfrac{1}{2}e^{-x} & при\ x>0,\\ \cos(bx+1) & при\ x\le 0 \end{cases}$	10	$y=\begin{cases} \sqrt{e^{2x-b}}-1 & при\ x\le 0,\\ \dfrac{1}{x^2+a} & при\ x>0 \end{cases}$
3	$y=\begin{cases} \dfrac{1}{a^2+x^2} & при\ x\le 1,\\ b\cdot\ln\lvert x\rvert & при\ x>1 \end{cases}$	11	$y=\begin{cases} \sqrt{a+\lvert\sin x\rvert} & при\ x>4,\\ tg(bx) & при\ x\le 4 \end{cases}$
4	$y=\begin{cases} \dfrac{a+x^2}{b+\ln(\lvert x\rvert+1)} & при\ x\le 2,\\ e^x+x^2 & при\ x>2 \end{cases}$	12	$y=\begin{cases} 2x^2+a\cos(bx) & при\ x\le 1,\\ e^x+tg\ x^3 & при\ x>1 \end{cases}$
5	$y=\begin{cases} a\sin^2 x+\sqrt{x} & при\ x\le 1,\\ be^{x^2} & при\ x>1 \end{cases}$	13	$y=\begin{cases} \ln(a+x^2) & при\ x\ge 2,\\ e^{\sin x}+2b & при\ x<2 \end{cases}$
6	$y=\begin{cases} a\cdot tg(x^2) & при\ x\le -1,\\ b+\dfrac{x^2}{x^2+a} & при\ x>-1 \end{cases}$	14	$y=\begin{cases} 0{,}2x^3+a & при\ x>-1,\\ bx^2+\ln\lvert x+3\rvert & при\ x\le -1 \end{cases}$
7	$y=\begin{cases} (a+x)arctg(ax) & при\ x>3,\\ \cos^2(b+x^3) & при\ x\le 3 \end{cases}$	15	$y=\begin{cases} \sin(x+a^2) & при\ x<2,\\ \ln(x^2+2x+b) & при\ x\ge 2 \end{cases}$
8	$y=\begin{cases} \sin^3(a+x) & при\ x<5,\\ \ln\sqrt{\lvert b-x\rvert} & при\ x\ge 5 \end{cases}$	16	$y=\begin{cases} \cos(x+a^2) & при\ x<2,\\ \ln(x^2+2x+b) & при\ x\ge 2 \end{cases}$

Лабораторная работа № 3. Работа с циклами

Конструкция оператора while:

```
while (Условие выполнения) {
    Тело цикла;
}
```

Рассмотрим пример выводящий все цифры с использованием конструкции while.

```
public class DoWhileLoop {
  public static void main (String[] args) {
    int i = 0;
    do {
      System.out.print(i);
      i++;
    } while (i < 10);
    System.out.println(); //Это можно использовать для простого перевода строки
    do {
        System.out.print("Do...while is cool;");
    } while (2 == 3); //Можно было написать и просто false
  }
}
```

Конструкция оператора for:

```
for (Начальное значение переменной; Логическое выражение с переменной (условие выполнения цикла); Действие над переменной, после выполнения тела цикла) {
    Операторы, которые будут выполнять цикл при условии, что логическое выражение - true;
}
```

Рассмотрим пример выводящий все цифры с использованием конструкции for.

```
class ForLoop {
  public static void main (String[] args ) {
    for (int i = 0; i < 10; i++) {
      System.out.print ("Ku-Ku ");
    }
  }
```

```
}
```

Конструкция оператора `do while`:

```
do {
    Тело цикла;
} while (условие выполения);
```

Отличие данного оператора от `while` только в том, что он является оператором **постусловия (сначала выполнит, потом проверит)**.

То есть, даже если условие не выполняется никогда, всё равно действие будет выполнено один раз.

```
public class DoWhileLoop {
  public static void main (String[] args) {
    int i = 0;
    do {
      System.out.print(i);
      i++;
    } while (i < 10);
    System.out.println(); //Это можно использовать для простого перевода строки
}
}
```

Задания для самостоятельного выполнения

Составить программу вычисления значений функции. Выведите количество итераций. Варианты заданий в таблице 3.

Таблица 3

№	Задание	№	Задание
1	$S=\sum_{n=1}^{+\infty}a_n,\ \ a_n=(-1)^n\frac{2n\cdot x^{2n+1}}{(2n+1)!}$	9	$S=2\sum_{n=1}^{+\infty}a_n,\ \ a_n=\left(1+\frac{1}{2}+...+\frac{1}{n}\right)\frac{x^{n+1}}{n+1},\ \ -1\le x<1$
2	$S=\sum_{n=0}^{+\infty}a_n,\ \ a_n=\frac{x^{2n+1}}{2n+1},\ \ -1<x<1$	10	$S=\sum_{n=1}^{+\infty}a_n,\ \ a_n=(-1)^{n-1}\left(1+\frac{1}{2}+...+\frac{1}{n}\right)x^n,\ \ -1<x<1$
3	$S=\sum_{n=0}^{+\infty}a_n,\ \ a_n=\frac{x^{4n+1}}{4n+1},\ \ -1<x<1$	11	$S=\sum_{n=0}^{+\infty}a_n,\ \ a_n=\frac{(2n+1)!!x^{2n}}{(2n)!!},\ \ -1<x<1$
4	$S=\sum_{n=0}^{+\infty}a_n,\ \ a_n=\frac{2^{\frac{n}{2}}\cos\left(\frac{pn}{4}\right)x^n}{n!}$	12	$S=\operatorname{arctg}2+\sum_{n=1}^{+\infty}a_n,\ \ a_n=\frac{(-1)^n2^{2n-1}x^{2n-1}}{2n-1},\ \ -\frac{1}{4}<x\le\frac{1}{2}$
5	$S=\sum_{n=0}^{+\infty}a_n,\ \ a_n=\frac{2^{\frac{n}{2}}\sin\left(\frac{pn}{4}\right)x^n}{n!}$	13	$S=\sum_{n=1}^{+\infty}a_n,\ \ a_n=(-1)^{n+1}\frac{x^{2n}}{2n(2n-1)},\ \ -1\le x\le1$
6	$S=1+\sum_{n=2}^{+\infty}a_n,\ \ a_n=\frac{(-1)^{n+1}(n-1)x^n}{n!}$	14	$S=\sum_{n=0}^{+\infty}a_n,\ \ a_n=(-1)^n\frac{x^{2n+1}(1+x^2)}{2n+1},\ \ -1\le x\le1$
7	$S=\sum_{n=1}^{+\infty}a_n,\ \ a_n=(-1)^n\frac{2n\cdot x^{2n+1}}{(2n+1)!}$	15	$S=\sum_{n=1}^{+\infty}a_n,\ \ a_n=(-1)^{n-1}\frac{x^n(1+x)}{n},\ \ -1\le x\le1$
8	$S=\sum_{n=1}^{+\infty}a_n,\ \ a_n=(-1)^n\frac{2n\cdot x^{2n+1}}{(2n+1)!}$	16	$S=x+2\sum_{n=1}^{+\infty}a_n,\ \ a_n=\frac{(-1)^{n+1}x^{2n+1}}{4n^2-1},\ \ -1\le x\le1$

Лабораторная работа № 4. Потоки ввода/вывода и строки в Java

Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан.

```
import java.util.Scanner; // импортируем класс
public class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in); // создаём объект класса Scanner
      int i = 2;
      System.out.print("Введите целое число: ");
      if(sc.hasNextInt()) { // возвращает истинну если с потока ввода можно считать целое число
        i = sc.nextInt(); // считывает целое число с потока ввода и сохраняем в переменную
        System.out.println(i*2);
      } else {
        System.out.println("Вы ввели не целое число");
      }
   }
}
```

Метод hasNextDouble(), применённый объекту класса Scanner, проверяет, можно ли считать с потока ввода вещественное число типа double, а метод nextDouble() — считывает его. Если попытаться считать значение без предварительной проверки, то во время исполнения программы можно получить ошибку (отладчик заранее такую ошибку не обнаружит). Например, попробуйте в представленной далее программе ввести какое-то вещественное число:

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

```
        double  i = sc.nextDouble(); // если ввести букву s, то случится ошибка во время исполнения
        System.out.println(i/3);
    }
}
```

Имеется также метод nextLine(), позволяющий считывать целую последовательность символов, т.е. строку, а, значит, полученное через этот метод значение нужно сохранять в объекте класса String. В следующем примере создаётся два таких объекта, потом в них поочерёдно записывается ввод пользователя, а далее на экран выводится одна строка, полученная объединением введённых последовательностей символов.

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s1, s2;
        s1 = sc.nextLine();
        s2 = sc.nextLine();
        System.out.println(s1 + s2);
    }
}
```

Существует и метод hasNext(), проверяющий остались ли в потоке ввода какие-то символы.

В классе String существует масса полезных методов, которые можно применять к строкам (перед именем метода будем указывать тип того значения, которое он возвращает):

1. int length() — возвращает длину строки (количество символов в ней);
2. boolean isEmpty() — проверяет, пустая ли строка;
3. String replace(a, b) — возвращает строку, где символ a (литерал или переменная типа char) заменён на символ b;

4. String toLowerCase() — возвращает строку, где все символы исходной строки преобразованы к строчным;
5. String toUpperCase() — возвращает строку, где все символы исходной строки преобразованы к прописным;
6. boolean equals(s) — возвращает истинну, если строка к которой применён метод, совпадает со строкой s указанной в аргументе метода (с помощью оператора == строки сравнивать нельзя, как и любые другие объекты);
7. int indexOf(ch) — возвращает индекс символа ch в строке (индекс это порядковый номер символа, но нумероваться символы начинают с нуля). Если символ совсем не будет найден, то возвратит -1. Если символ встречается в строке нескольо раз, то вовзвратит индекс его первого вхождения.
8. int lastIndexOf(ch) — аналогичен предыдущему методу, но возвращает индекс последнего вхождения, если смивол встретился в строке несколько раз.
9. int indexOf(ch,n) — возвращает индекс символа ch в строке, но начинает проверку с индекса n (индекс это порядковый номер символа, но нумероваться символы начинают с нуля).
10. char charAt(n) — возвращает код символа, находящегося в строке под индексом n (индекс это порядковый номер символа, но нумероваться символы начинают с нуля).

```
public class Main {
    public static void main(String[] args) {
        String s1 = "firefox";
        System.out.println(s1.toUpperCase()); // выведет «FIREFOX»
        String s2 = s1.replace('o', 'a');
        System.out.println(s2); // выведет «firefax»
        System.out.println(s2.charAt(1)); // выведет «i»
        int i;
        i = s1.length();
```

```
        System.out.println(i); // выведет 7
        i = s1.indexOf('f');
        System.out.println(i); // выведет 0
        i = s1.indexOf('r');
        System.out.println(i); // выведет 2
        i = s1.lastIndexOf('f');
        System.out.println(i); // выведет 4
        i = s1.indexOf('t');
        System.out.println(i); // выведет -1
        i = s1.indexOf('r',3);
        System.out.println(i); // выведет -1
    }
}
```

Пример программы, которая выведет на экран индексы всех пробелов в строке, введенное пользователем с клавиатуры:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        for(int i=0; i < s.length(); i++) {
            if(s.charAt(i) == ' ') {
                System.out.println(i);
            }
        }
    }
}
```

Задания для самостоятельного выполнения

Вариант 1. Написать программу, проверяющую, является ли строка S допустимым *идентификатором*, то есть непустой строкой, которая содержит только латинские буквы, цифры и символ подчеркивания «_» и не начинается с цифры. Если S является допустимым идентификатором, то функция возвращает 0. Если S является пустой строкой, то возвращается –1, если S начинается с цифры, то возвращается –2. Если S содержит недопустимые символы, то возвращается номер первого недопустимого символа.

Вариант 2. Написать программу, возвращающую строку длины N, заполненную повторяющимися копиями строки-*шаблона* S (последняя копия строки-шаблона может входить в результирующую строку частично).

Вариант 3. Написать программу,, преобразующую все строчные русские буквы строки S в прописные (остальные символы строки S не изменяются). Строка S является входным и выходным параметром.

Вариант 4. Написать программу, преобразующую все прописные русские буквы строки S в строчные (остальные символы строки S не изменяются). Строка S является входным и выходным параметром.

Вариант 5. Написать программу, удаляющую в строке S начальные символы, совпадающие с символом C. Строка S является входным и выходным параметром. Дан символ C и пять строк.

Вариант 6. Написать программу, удаляющую в строке S конечные символы, совпадающие с символом C. Строка S является входным и выходным параметром. Дан символ C и пять строк.

Вариант 7. Написать программу, возвращающую инвертированную подстроку строки S, содержащую в обратном порядке N символов строки S, начиная с ее K-го символа. Если K превосходит длину строки S, то возвращается пустая строка; если длина строки меньше K + N, то инвертируются все символы строки, начиная с ее K-го символа.

Вариант 8. Написать программу, возвращающую номер позиции, начиная с которой в строке S содержится первое вхождение строки S_0, причем анализируются только N символов строки S, начиная с ее K-го символа (таким образом, PosSub обеспечивает поиск в подстроке). Если K превосходит длину строки S, то возвращается 0, если длина строки меньше K + N, то анализируются все символы строки, начиная с ее K-го символа. Если в требуемой подстроке строки S вхождения S_0 отсутствуют, то функция возвращает 0.

Вариант 9. Написать программу, возвращающую номер позиции, начиная с которой в строке S содержится последнее вхождение подстроки S_0. Считать, что перекрывающихся вхождений подстрок S_0 строка S не содержит.

Вариант 10. Написать программу, возвращающую номер позиции, начиная с которой в строке *S* содержится *K*-е вхождение подстроки S_0 (*K* > 0). Если количество вхождений S_0 в строке *S* меньше *K*, то функция возвращает 0. Считать, что перекрывающихся вхождений подстрок S_0 строка *S* не содержит.

Вариант 11. Написать программу, возвращающую K-е слово строки S (словом считается набор символов, не содержащий пробелов и ограниченный пробелами или началом/концом строки).

Вариант 12. Написать программу, которая формирует по данной строке S массив W слов, входящих в S (массив W и его размер N являются выходными параметрами). Словом считается набор символов, не содержащий пробелов и ограниченный пробелами или началом/концом строки; предполагается, что строка S содержит не более 10 слов.

Вариант 13. Написать программу, выполняющую сжатие строки S по следующему правилу: каждая подстрока строки S, состоящая из более чем четырех одинаковых символов C, заменяется текстом вида «C{K}», где K — количество символов C (предполагается, что строка S не содержит фигурных скобок «{» и «}»). Например, для строки S = «bbbccccccе» программа вернет строку «bbbc{5}e».

Вариант 14. Написать программу, возвращающую строковое представление целого неотрицательного числа N в 16-ричной системе счисления. Результирующая строка состоит из символов «0»–«9», «A»–«F» и не содержит ведущих нулей (за

исключением представления числа 0). Используя эту программу, получить 16-ричные представления пяти данных чисел.

Вариант 15. Написать программу, определяющую целое неотрицательное число по его строковому представлению S в 16-ричной системе счисления. Параметр S имеет строковый тип, состоит из символов «0»–«9», «A»–«F» и не содержит ведущих нулей (за исключением значения «0»). Используя эту программу, вывести пять чисел, для которых даны их 16-ричные представления.

Лабораторная работа № 5. Массивы

Массив — это конечная последовательность упорядоченных элементов одного типа, доступ к каждому элементу в которой осуществляется по его индексу.

Размер или длина массива — это общее количество элементов в массиве. Размер массива задаётся при создании массива и не может быть изменён в дальнейшем, т. е. нельзя убрать элементы из массива или добавить их туда, но можно в существующие элементы присвоить новые значения.

Индекс начального элемента — 0, следующего за ним — 1 и т. д. Индекс последнего элемента в массиве — на единицу меньше, чем размер массива.

В Java массивы являются объектами. Это значит, что имя, которое даётся каждому массиву, лишь указывает на адрес какого-то фрагмента данных в памяти. Кроме адреса в этой переменной ничего не хранится. Индекс массива, фактически, указывает на то, насколько надо отступить от начального элемента массива в памяти, чтоб добраться до нужного элемента.

Чтобы создать массив надо объявить для него подходящее имя, а затем с этим именем связать нужный фрагмент памяти, где и будут друг за другом храниться значения элементов массива.

Рассмотрим программу поиска суммы всех положительных элементов массива.

```
1      public static void main(String args[]) {
2          Scanner s = new Scanner(System.in);
3          int numbers = Integer.valueOf(s.nextLine()).intValue();
4          String line = s.nextLine();
5          s.close();
6          String [] all = line.split("\\s");
7          int elements[] = new int [numbers];
8          for(int i = 0; i<numbers; i++)
9              elements[i] = Integer.valueOf(all[i]).intValue();
10
11         int count = 0;
12         for(int i = 0; i<numbers; i++)
13             if(elements[i] > 0)
14                 count++;
15         System.out.println(count);
16     }
```

Задания для самостоятельного выполнения

Вариант 1º. Написать программу, находящую минимальный элемент целочисленного массива A размера N. С помощью этой программы найти минимальные элементы массивов A, B, C размера N_A, N_B, N_C соответственно.

Вариант 2. Написать программу, находящую номер максимального элемента вещественного массива A размера N. С помощью этой программы найти номера максимальных элементов массивов A, B, C размера N_A, N_B, N_C соответственно.

Вариант 3. Написать программу, находящую номера минимального и максимального элемента вещественного массива A размера N. Выходные параметры целого типа: $NMin$ (номер минимального элемента) и $NMax$ (номер максимального элемента). С помощью этой программы найти номера минимальных и максимальных элементов массивов A, B, C размера N_A, N_B, N_C соответственно.

Вариант 4. Написать программу, меняющую порядок следования элементов вещественного массива A размера N на противоположный (*инвертирование* массива). Массив A является входным и выходным параметром. С помощью этой программы инвертировать массивы A, B, C размера N_A, N_B, N_C соответственно.

Вариант 5. Написать программу, выполняющую *сглаживание* вещественного массива A размера N следующим образом: элемент A_K заменяется на среднее арифметическое первых K исходных элементов массива A. Массив A является входным и выходным параметром.

Вариант 6. Написать программу, выполняющую *сглаживание* вещественного массива A размера N следующим образом: элемент A_1 не изменяется, элемент A_K (K = 2, …, N) заменяется на полусумму исходных элементов A_{K-1} и A_K. Массив A является входным и выходным параметром. С помощью этой программы выполнить пятикратное сглаживание данного массива A размера N, выводя результаты каждого сглаживания.

Вариант 7. Написать программу, выполняющую *сглаживание* вещественного массива A размера N следующим образом: каждый элемент массива заменяется на его среднее арифметическое с соседними элементами (при вычислении среднего

арифметического используются *исходные* значения соседних элементов). Массив A является входным и выходным параметром. С помощью этой программы выполнить пятикратное сглаживание данного массива A размера N, выводя результаты каждого сглаживания.

Вариант 8. Написать программу, удаляющую из целочисленного массива A размера N элементы, равные целому числу X. Массив A и число N являются входными и выходными параметрами. С помощью этой программы удалить числа X_A, X_B, X_C из массивов A, B, C размера N_A, N_B, N_C соответственно и вывести размер и содержимое полученных массивов.

Вариант 9. Написать программу, удаляющую из вещественного массива A размера N «лишние» элементы так, чтобы оставшиеся элементы оказались упорядоченными по возрастанию: первый элемент не удаляется, второй элемент удаляется, если он меньше первого, третий — если он меньше предыдущего элемента, оставленного в массиве, и т. д. Например, массив 5.5, 2.5, 4.6, 7.2, 5.8, 9.4 должен быть преобразован к виду 5.5, 7.2, 9.4. Массив A и число N являются входными и выходными параметрами. С помощью этой программы преобразовать массивы A, B, C размера N_A, N_B, N_C соответственно и вывести размер и содержимое полученных массивов.

Вариант 10. Написать программу, дублирующую в целочисленном массиве A размера N элементы, равные целому числу X. Массив A и число N являются входными и выходными параметрами. С помощью этой программы продублировать числа X_A, X_B, X_C в массивах A, B, C размера N_A, N_B, N_C соответственно и вывести размер и содержимое полученных массивов.

Вариант 11. Написать программу, выполняющую сортировку по возрастанию вещественного массива A размера N. Массив A является входным и выходным параметром. С помощью этой программы отсортировать массивы A, B, C размера N_A, N_B, N_C соответственно.

Вариант 12. Написать программу, формирующую для вещественного массива A размера N *индексный массив* I — массив целых чисел того же размера, содержащий номера элементов массива A в том порядке, который соответствует

возрастанию элементов массива A (сам массив A при этом не изменяется). Индексный массив I является выходным параметром. С помощью этой программы создать индексные массивы для массивов A, B, C размера N_A, N_B, N_C соответственно.

Вариант 13. Написать программу, меняющую порядок элементов вещественного массива A размера N на следующий: наименьший элемент массива располагается на первом месте, наименьший из оставшихся элементов — на последнем, следующий по величине располагается на втором месте, следующий — на предпоследнем и т. д. (в результате график значений элементов будет напоминать *колокол*). Массив A является входным и выходным параметром. С помощью этой программы преобразовать массивы A, B, C размера N_A, N_B, N_C соответственно.

Вариант 14. Написать программу, формирующую по вещественному массиву A размера N_A два вещественных массива B и C размера N_B и N_C соответственно; при этом массив B содержит все элементы массива A с нечетными порядковыми номерами (1, 3, …), а массив C — все элементы массива A с четными номерами (2, 4, …). Массивы B и C и числа N_B и N_C являются выходными параметрами. Применить эту программу к данному массиву A размера N_A и вывести размер и содержимое полученных массивов B и C.

Вариант 15. Написать программу, формирующую по целочисленному массиву A размера N_A два целочисленных массива B и C размера N_B и N_C соответственно; при этом массив B содержит все четные числа из массива A, а массив C — все нечетные числа (в том же порядке). Массивы B и C и числа N_B и N_C являются выходными параметрами. Применить эту программу к данному массиву A размера N_A и вывести размер и содержимое полученных массивов B и C.

Лабораторная работа № 6. Сортировка

Сортировка пузырьком

Расположим массив сверху вниз, от нулевого элемента - к последнему.

Идея метода: шаг сортировки состоит в проходе снизу вверх по массиву. По пути просматриваются пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами.

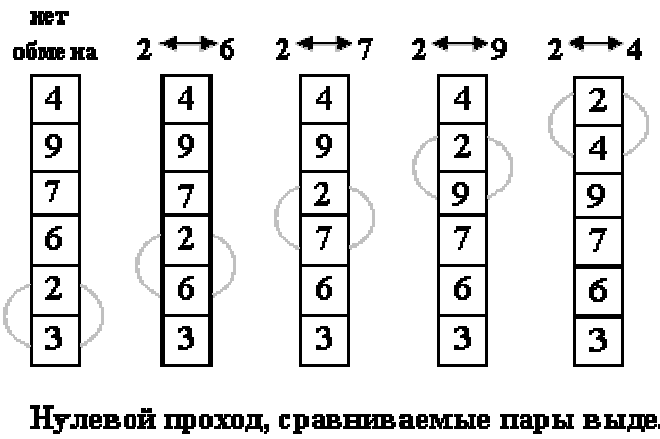

Рис. 5 Подробная схема прохода сортировки пузырьком

После нулевого прохода по массиву "вверху" оказывается самый "легкий" элемент - отсюда аналогия с пузырьком. Следующий проход делается до второго сверху элемента, таким образом второй по величине элемент поднимается на правильную позицию...

Делаем проходы по все уменьшающейся нижней части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, так как последовательность упорядочена по возрастанию.

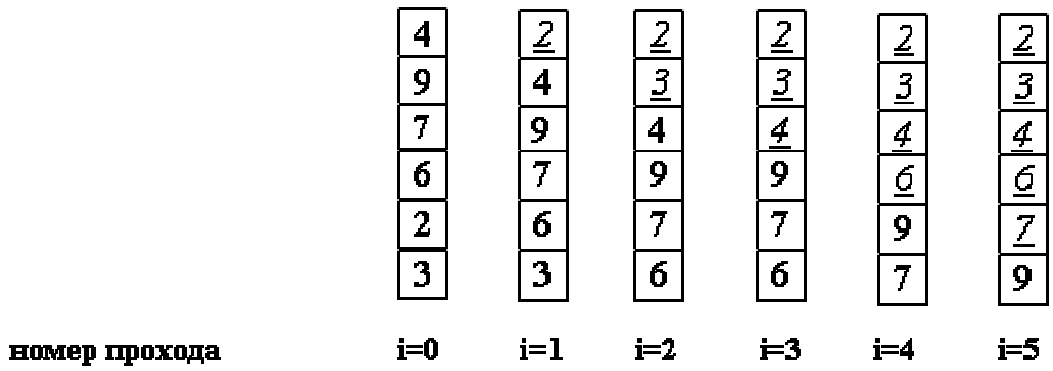

Рис. 6 Итоговые массивы на каждом проходе сортировки пузырьком

Среднее число сравнений и обменов имеют квадратичный порядок роста: $O(n^2)$, отсюда можно заключить, что алгоритм пузырька очень медленен и малоэффективен. Дополнительная память, очевидно, не требуется. Сортировка пузырьком устойчива.

Сортировка выбором

Идея метода состоит в том, чтобы создавать отсортированную последовательность путем присоединения к ней одного элемента за другим в правильном порядке.

Будем строить готовую последовательность, начиная с левого конца массива. Алгоритм состоит из *n* последовательных шагов, начиная от нулевого и заканчивая *(n-1)*-м.

На *i*-м шаге выбираем наименьший из элементов *a[i] ... a[n]* и меняем его местами с *a[i]*. Последовательность шагов при *n=5* изображена на рис. 3.3.

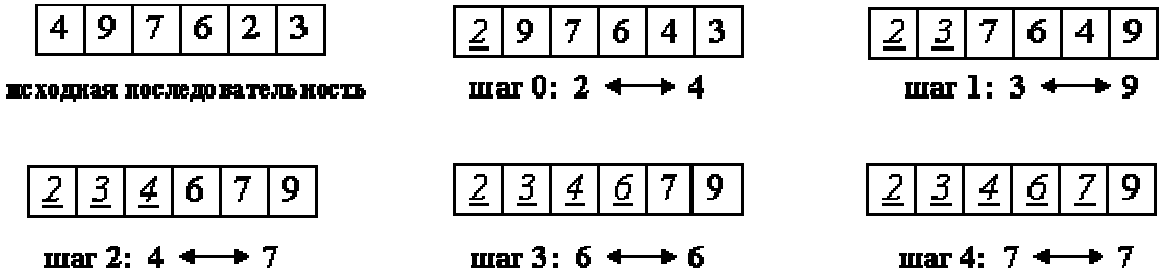

Рис. 7 Итоговые массивы на каждом проходе сортировки выбором

Вне зависимости от номера текущего шага *i*, последовательность *a[0]...a[i]* (выделена курсивом) является упорядоченной. Таким образом, на *(n-1)*-м шаге вся последовательность, кроме *a[n]* оказывается отсортированной, а *a[n]* стоит на последнем месте по праву: все меньшие элементы уже ушли влево.

Для нахождения наименьшего элемента из *n+1* рассматримаемых алгоритм совершает n сравнений. С учетом того, что количество рассматриваемых на очередном шаге элементов уменьшается на единицу, общее количество операций $O(n^2)$. Таким образом, так как число обменов всегда будет меньше числа сравнений, время сортировки растет квадратично относительно количества элементов.

Алгоритм не использует дополнительной памяти: все операции происходят "на месте".

Сортировка вставками

Сортировка простыми вставками в чем-то похожа на вышеизложенные методы.

Аналогичным образом делаются проходы по части массива, и аналогичным же образом в его начале "вырастает" отсортированная последовательность...

Однако в сортировке пузырьком или выбором можно было четко заявить, что на i-м шаге элементы *a[0]...a[i]* стоят на правильных местах и никуда более не переместятся. Здесь же подобное утверждение будет более слабым: последовательность *a[0]...a[i]* упорядочена. При этом по ходу алгоритма в нее будут вставляться (см. название метода) все новые элементы.

Будем разбирать алгоритм, рассматривая его действия на *i*-м шаге. Как говорилось выше, последовательность к этому моменту разделена на две части: готовую *a[0]...a[i]* и неупорядоченную *a[i+1]...a[n]*.

На следующем, *(i+1)*-м каждом шаге алгоритма берем *a[i+1]* и вставляем на нужное место в готовую часть массива. Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним. В зависимости от результата сравнения элемент либо остается на текущем месте(вставка завершена), либо они меняются местами и процесс повторяется.

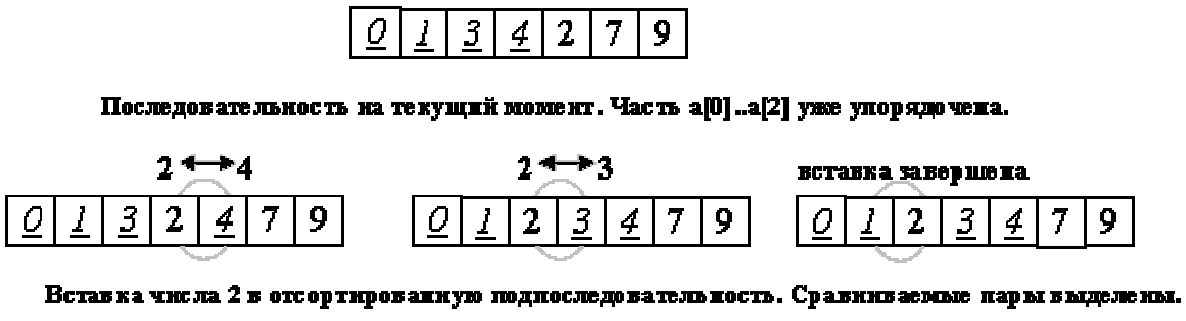

Рис. 8 Подробная схема прохода сортировки вставками

Таким образом, в процессе вставки мы "просеиваем" элемент *x* к началу массива, останавливаясь в случае, когда найден элемент, меньший *x* или достигнуто начало последовательности.

Аналогично сортировке выбором, среднее, а также худшее число сравнений и пересылок оцениваются как $O(n^2)$, дополнительная память при этом не используется.

Сортировка Шелла

Сортировка Шелла является довольно интересной модификацией алгоритма сортировки простыми вставками.

Рассмотрим следующий алгоритм сортировки массива *a[0].. a[15]*.

12	8	14	6	4	9	1	8	13	5	11	3	18	3	10	9

Рис. 9 Исходный массив для сортировки Шелла

1. Вначале сортируем простыми вставками каждые 8 групп из 2-х элементов (*a[0], a[8[), (a[1], a[9]), ... , (a[7], a[15]*).

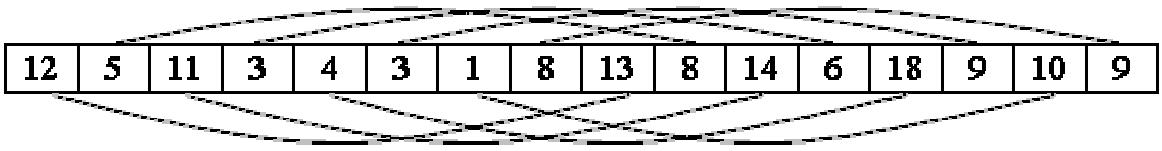

Рис. 10 Схема первого прохода сортировки Шелла

2. Потом сортируем каждую из четырех групп по 4 элемента (*a[0], a[4], a[8], a[12]), ..., (a[3], a[7], a[11], a[15]*).

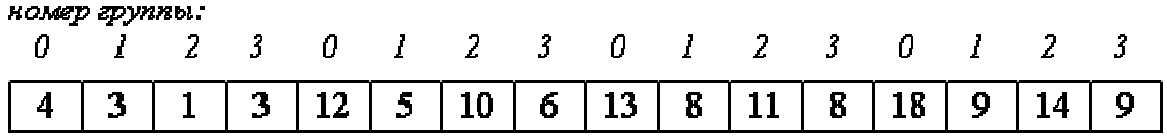

Рис. 11 Схема второго прохода сортировки Шелла

В нулевой группе будут элементы 4, 12, 13, 18, в первой - 3, 5, 8, 9 и т.п.

3. Далее сортируем 2 группы по 8 элементов, начиная с (*a[0], a[2], a[4], a[6], a[8], a[10], a[12], a[14]*).

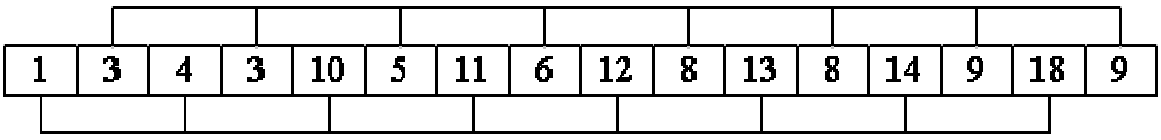

Рис. 12- Схема третьего прохода сортировки Шелла

4. В конце сортируем вставками все 16 элементов.

Рис. 13. Результат последнего прохода сортировки Шелла

Очевидно, лишь последняя сортировка необходима, чтобы расположить все элементы по своим местам. Так зачем нужны остальные?

На самом деле они продвигают элементы максимально близко к соответствующим позициям, так что в последней стадии число перемещений будет весьма невелико. Последовательность и так почти отсортирована. Ускорение подтверждено многочисленными исследованиями и на практике оказывается довольно существенным.

Единственной характеристикой сортировки Шелла является приращение - расстояние между сортируемыми элементами, в зависимости от прохода. В конце приращение всегда равно единице - метод завершается обычной сортировкой вставками, но именно последовательность приращений определяет рост эффективности.

Использованный в примере набор ..., 8, 4, 2, 1 - неплохой выбор, особенно, когда количество элементов - степень двойки. Однако возможно использовать любую убывающую с некоторым шагом последовательность. Общее количество операций $O(nk)$, где k - количество сравнений.

Сортировка слиянием

Сортировка слиянием также построена на принципе "разделяй-и-властвуй", однако реализует его несколько по-другому, нежели быстрая сортировка. А именно, вместо разделения по опорному элементу массив просто делится пополам.

Рекурсивный алгоритм обходит получившееся дерево слияния в прямом порядке. Каждый уровень представляет собой проход сортировки слияния - операцию, полностью переписывающую массив.

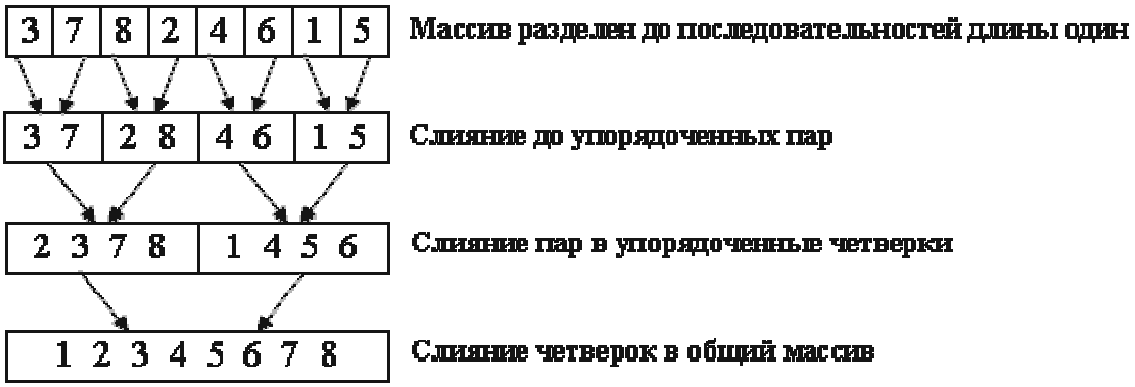

Рис. 14 Пример проходов для сортировки слиянием

Обратим внимание, что деление происходит до массива из единственного элемента. Такой массив можно считать упорядоченным, а значит, задача сводится к написанию функции слияния merge.

Один из способов состоит в слиянии двух упорядоченных последовательностей при помощи вспомогательного буфера, равного по размеру общему количеству имеющихся в них элементов. Элементы последовательностей будут перемещаться в этот буфер по одному за шаг. Пример работы на последовательностях 2 3 6 7 и 1 4 5 приведен на рис. 3.15.

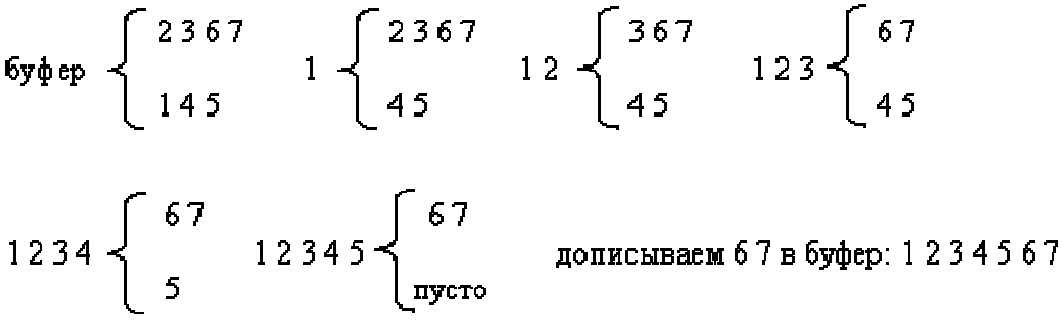

Рис. 15 Пример слияния двух массивов

Результатом является упорядоченная последовательность, находящаяся в буфере. Каждая операция слияния требует n пересылок и n сравнений, где *n* - общее число элементов, так что время слияния: *O(n)*.

Оценим быстродействие алгоритма: время работы определяется рекурсивной формулой $O(n) = 2O(n/2) + O'(n)$. Ее решение: $O(n) = n \log n$ - результат весьма неплох, учитывая отсутствие "худшего случая". Однако, несмотря на хорошее общее быстродействие, у сортировки слиянием есть и серьезный минус: она требует $O(n)$ памяти.

Хорошо запрограммированная внутренняя сортировка слиянием работает немного быстрее пирамидальной, но медленнее быстрой, при этом требуя много памяти под буфер. Поэтому сортировку слиянием используют для упорядочения массивов, лишь если требуется устойчивость метода (которой нет ни у быстрой, ни у пирамидальной сортировок).

Сортировка слиянием является одним из наиболее эффективных методов для односвязных списков и файлов, когда есть лишь последовательный доступ к элементам.

Ознакомиться с алгоритмами сортировки так же можно по следующей ссылке: www.sorting-algorithms.com.

Задания для самостоятельного выполнения

Реализовать консольное приложение, предназначенное для сортировки целочисленного вектора стандартными средствами и собственной функцией, реализующей алгоритм сортировки согласно варианту.

Входные данные: количество элементов, тип входного вектора, вектор.

Выходные данные:. отсортированный массив, время работы библиотечной и реализованной функций сортировок, ускорение библиотечной сортировки.

Особенности:

- должна присутствовать возможность выбора исходного (сортируемого) вектора: вводимый с клавиатуры или генерируемый согласно варианту (два возможных для выбора типа генерируемых данных);

- в случае, если число элементов вектора менее некоторого разумного значения, исходный и отсортированные двумя методами векторы должны выводиться на экран для сравнения и проверки правильности результатов сортировки;

- в случае, если число элементов вектора более некоторого разумного значения, должны выводиться на экран для сравнения и проверки правильности результатов сортировки некоторое количество элементов исходного и отсортированных двумя методами векторов.

Вариант 1. Алгоритм сортировки бинарными вставками. Просматриваются последовательно $a_2, \ldots, a_n$, и каждый новый элемент a_i вставляется на подходящее место в уже упорядоченную совокупность $a_1, \ldots, a_{i-1}$. Место, на которое надо вставить a_i, определяется алгоритмом деления пополам.

Оцените число сравнений и перемещений элементов в этом алгоритме.

Вариант 2. Алгоритм фон Неймана (сортировка слияниями). Вначале весь массив рассматривается как совокупность упорядоченных групп по одному элементу в каждой. Слиянием соседних групп получаем упорядоченные группы, каждая из которых содержит два элемента (кроме, может быть, последней группы, для которой не нашлось парной). Далее, упорядоченные группы укрупняются тем же способом.

Вариант 3. Написать программу, реализующую следующий алгоритм (относящийся к классу сортировки обменами). Последовательным просмотром чисел $a_1, \ldots, a_n$ найти наименьшее i такое, что $a_i > a_{i+1}$. Поменять a_i и a_{i+1} местами и возобновить просмотр с начала массива. Когда не удастся найти такое i, массив будет упорядочен нужным образом.

Вариант 4. Написать программу, реализующую следующий алгоритм (сортировка посредством подсчета). В окончательно упорядоченной последовательности элемент № j превышает по значению ровно j–1 остальных элементов. Иначе говоря, если некоторый элемент превышает по значению ровно 7 других элементов и если никакие два элемента не равны, то после сортировки этот элемент должен занять 8 место.

Вариант 5. (Сортировка Шелла). Идея, лежащая в его основе, состоит в следующем. На первом проходе отдельно группируются и сортируются все элементы, отстоящие друг от друга на h_t (например, 8) позиций. Затем (на втором проходе) группируются и сортируются все элементы, отстоящие друг от друга на h_{t-1} (например, 4) позиций. На третьем и дальнейших проходах процедура повторяется, изменяется только шаг группировки – в меньшую сторону. Важно, что на последнем проходе шаг группировки должен быть равен 1.

Замечание: метод был предложен Д.Л. Шеллом в 1959 г. Однако до сих пор неизвестно, какая последовательность шагов группировки наилучшая. В оригинале предлагалось использовать смещения N/2, N/4, N/8 и т.д. Однако выяснилось, что лучшие результаты получаются, если h_i не кратны друг другу.

Указание. Сортировка внутри групп элементов выполняется каким-либо простым методом, например, методом простых вставок.

Вариант 6. Напишите улучшенную программу сортировки обменами. Напомним алгоритм сортировки обменами (метод пузырька). При каждом проходе по массиву сравниваются два стоящих рядом элемента, и если они (локально) не упорядочены, осуществляется обмен их местами. Если контролировать, как происходят перестановки, можно добиться улучшения программы. Во-первых, если после очередного прохода не было перестановок, то последовательность элементов

уже упорядочена, и дальнейшие проходы не имеют смысла. Во-вторых, можно следить за тем, с какого элемента следует начинать очередной проход.

Вариант 7. Напишите программу шейкер-сортировки (одна из разновидностей сортировки обменами). Напомним алгоритм сортировки обменами (метод пузырька). При каждом проходе по массиву сравниваются два стоящих рядом элемента, и если они (локально) не упорядочены, осуществляется обмен их местами. Шейкер-сортировка предполагает, что элементы массива просматриваются попеременно в обоих направлениях.

Замечание: если j – такой индекс, что элементы R_j и R_{j+1} не меняются местами в двух последовательных проходах, то их можно исключить из последующих сравнений.

Вариант 8. Напишите программу пирамидальной сортировки. Будем называть массив $k_1, k_2, \ldots, k_n$ пирамидой, если $k_{ëj/2û} >= k_j$ при $1 <= ëj/2û < j < n$. В этом случае $k_1 >= k_2$, $k_1 >= k_3$, $k_2 >= k_4$ и т.д. Из этого следует, в частности, что наибольший ключ оказывается «на вершине пирамиды»: $k_1 = max(k_1, k_2, \ldots, k_n)$. Если как-нибудь преобразовать исходный массив в пирамиду, то для получения эффективного алгоритма сортировки можно многократно исключать вершину пирамиды и записывать ее на окончательное место.

Указание. Если уже имеется пирамида, и необходимо добавить в нее элемент, то действуют обычно следующим образом. Новый элемент помещается в вершину дерева, а затем просеивается по пути, на котором находятся меньшие по сравнению с ним элементы, которые одновременно поднимаются вверх. (Уильямс, Флойд).

Вариант 9. Сортировка методом двухпутевого слияния. Сначала опишем алгоритм фон Неймана (сортировка слияниями). Вначале весь массив рассматривается как совокупность упорядоченных групп по одному элементу в каждой. Слиянием соседних групп получаем упорядоченные группы, каждая из которых содержит два элемента (кроме, может быть, последней группы, для которой не нашлось парной). Далее, упорядоченные группы укрупняются тем же способом.
Измените этот алгоритм так, чтобы «свеча сгорала с двух концов»: массив анализируется и слева и справа.

Вариант 10. Сортировка методом распределения. Опишем идею алгоритма. Пусть имеется колода из 52 игральных карт, которую необходимо упорядочить по старшинству карт в масти, а также по масти. Одна из карт предшествует другой, если она либо младше по масти, либо масти одинаковы, но она младше по достоинству. Естественно рассортировать сначала карты по масти – в четыре стопки, а затем упорядочить каждую стопку по достоинству. Однако есть более быстрый способ. Сначала разложим карты в 13 стопок – по их достоинству. После этого соберем все стопки вместе: снизу – тузы, затем – двойки, тройки и т.д. Затем снова разложим колоду, на этот раз – в четыре стопки по масти. Сложив вместе полученные стопки так, чтобы внизу были карты младшей масти, а вверху – старшей, получим упорядоченную колоду.

Указание. Роль достоинства будет играть младшая цифра в системе счисления, роль масти – цифра десятков, и т.д. Считайте, что в числах не более 5 разрядов.

Вариант 11. Метод пузырька (обменная сортировка) выполняется за время порядка N^2. Чтобы уменьшить время работы алгоритма, нужно сравнивать несоседние элементы (так называемая параллельная сортировка Бэтчера; идейно этот подход похож на сортировку Шелла). Например, сначала выполнить (независимо) сортировку подмассивов элементов, стоящих на нечетных и четных местах соответственно, а затем объединить отсортированные последовательности. Разработайте усовершенствованный таким образом алгоритм обменной сортировки.

Вариант 12. Модифицируем алгоритм быстрой сортировки следующим образом. Вместо того, чтобы разбивать массив на две части (и сортировать каждую из них в отдельности), разобьем его на три (примерно равные) части и отсортируем: сначала первые две трети, затем последние две трети и, наконец, снова первые две трети. (Кстати, докажите, что этот алгоритм действительно сортирует массив).

Вариант 13. Метод сортировки простыми вставками можно обобщить так, чтобы работать не с одним списком, а с несколькими. Чтобы пояснить это, приведем пример. Допустим, Вам поручили расставить на полке несколько десятков книг по фамилиям авторов. Все книги, которые нужно расставить, лежат на столе в произвольном порядке. Естественно, что, ставя книгу на полку, Вы постараетесь

прикинуть, где она, в конечном счете, будет стоять (чтобы сократить количество перемещений). Эффективность процесса повышается, если на полке немного больше места, чем требуется. Упорядочение в каждом списке проводится при помещении в него очередного элемента.

Указание. Подходящей структурой данных для хранения каждого списка книг будет линейный список.

Вариант 14. Сортировка вычерпыванием близка по своей идее к поразрядной сортировке. Предположим, что все числа, которые нужно отсортировать, лежат в промежутке [0, 1). Тогда этот промежуток можно разделить на n равных частей и сопоставить каждой части «ящик-черпак». Затем следует разложить подлежащие сортировке числа по ящикам (каждому ящику в соответствие ставится список). В предположении, что сортируемые числа приблизительно равномерно распределены в заданном промежутке, содержимое каждого ящика будет сопоставимо по объему. Теперь следует отсортировать числа в каждом ящике, а затем собрать списки воедино.

Указание. В зависимости от количества чисел в ящике к ним можно применить разные сортировки. Если их немного, подойдет сортировка простыми вставками, если же количество их велико, можно вновь воспользоваться сортировкой вычерпыванием.

Вариант 15. Придумайте алгоритм сортировки массива из 5 элементов с помощью не более, чем 7 сравнений. Обобщите придуманный алгоритм на случай n элементов.

Требования по выполнению лабораторных работ

В ходе выполнения лабораторных работ студент должен написать программный код на языке программирования Java, выполняющий основное задание лабораторной работы, оформить отчет по и отчитаться преподавателю.

В ходе выполнения лабораторных работ для получения зачета студент обязан написать отчет о выполненной работе. Шаблон отчета предоставляется преподавателем.

Структура отчета приведена ниже:

1. Титульный лист. На титульном листе указываются номер ЛР, фамилия и инициалы (ФИО) студента, номер группы, ФИО преподавателя.
2. Задание. Задание лабораторной работы копируется из методических указаний согласно варианту.
3. Ход выполнения лабораторной работы. В данной части подробно описывается процесс выполнения лабораторных работ, приводятся скрин-шоты работы программы, листинг и интерпретация результатов.
4. Заключение. В заключении описывается основной полученный результат, формируются выводы по проделанной работе.
5. Список литературы. Список литературы должен содержать минимум две ссылки.
6. Приложение. Здесь и только здесь приводится полный код программ(ы).

К отчету предъявляются следующие требования:

1. структура отчета должна быть строго соблюдена;
2. отчет должен быть отформатирован;
3. каждая структурная часть должна начинаться с новой страницы;
4. все страницы в отчете кроме титульной должны быть пронумерованы;
5. в отчете должен содержаться листинг работы программы на всех ее функциональных возможностях;
6. отчет должен содержать описание всех особенностей реализации кода.

В ходе выполнения лабораторных работ работы для получения зачета студент обязан в срок защитить свою работу перед преподавателем. В случае, если студент не защитит в срок все ЛР, то он будет не допущен к сдаче экзамена.

Защита может состояться только при:

1) Наличие работающего программного кода, выполняющего основное задания лабораторной работы;

2) Наличие отчета.

В ходе защиты студент обязан продемонстрировать владение материалом, ответить на дополнительные вопросы преподавателя.

Список литературы

1. Королев В.А. Практика программирования на языке Java : учеб. пособие / В.А. Королев, И.В. Зайцев. – СПб. : С.-Петерб. торгово-эконом. ин-т, 2009. – 66 с. – (http://conf.spbtei.ru/java/posobie.pdf).

2. Эккель Б. Философия Java / Б. Эккель. – СПб. : Питер, 2009. – 638 с.

3. Казарин С.А. Среда разработки Java-приложений Eclipse: (ПО для объектно-ориентированного программирования и разработки приложений на языке Java) : учеб. пособие / С.А. Казарин, А.П. Клишин. – М., 2008. –
77 с.

4. Вязовик Н. Программирование на JAVA / Н. Вязовик, Е. Жилин. – М. : Центр Sun технологий МФТИ, ЦОС и ВТ МФТИ, 2003.

5. Материалы сайта http://ru.sun.com/

6. Монахов В. Язык программирования Java и среда NetBeans / В. Монахов. – 3-е изд. – СПб., 2011. – 704 с.

7. Шилдт Г. Полный справочник по Java. – (http://www.realcoding.net/ /articles/glava-6-klassy-kollektsii.html).

8. Картузов А.В. Программирование на языке Java. – (http://www. proklondike.com/books/java/kartuzov_programming.html).

9. Flanagan D. Java Examles in a Nutshell, 3rd edition. – O'Relly, 2004. (электронное издание).

10. Абрамян М Э. 1000 задач по программированию. Часть **I:** Скалярные типы данных, управляющие операторы, процедуры и функции. — Ростов н/Д: УПЛ РГУ, 2004. — 43 с.

11. Абрамян М. Э. 1000 задач по программированию. Часть **II:** Минимумы и максимумы, одномерные и двумерные массивы, символы и строки, двоичные файлы. — Ростов н/Д.: УПЛ РГУ, 2004 —42 с.

Методические материалы

Рубцова Татьяна Павловна

Программирование на языке JAVA

Методические указания