Классы-коллекции

Как было рассмотрено в предыдущей лабораторной работе, класс StringTokenizer позволяет решить задачу синтаксический разбор строки. Задача разбора введенного текста — *парсинг* (parsing) — вечная задача программирования, наряду с сортировкой и поиском. Написана масса программ-парсеров (parser), разбирающих текст по различным признакам. Есть даже программы, генерирующие парсеры по заданным правилам разбора: YACC, LEX и др.

В пакет java.utii входит простой класс StringTokenizer , облегчающий разбор строк. Класс StringTokenizer из пакета java.utii небольшой, в нем три конструктора и шесть методов.

Первый конструктор StringTokenizer (String str) создает объект, готовый разбить строку str на слова, разделенные пробелами, символами табуляций '\t', перевода строки '\n' и возврата каретки '\r' . Разделители не включаются в число слов.

Второй конструктор StringTokenizer (String str. String delimeters) задает разделители вторым параметром deiimeters , например:

```
StringTokenizer("Казнить,нельзя:пробелов-нет", " \t\n\r,:-");
```

Здесь первый разделитель — пробел. Потом идут символ табуляции, символ перевода строки, символ возврата каретки, запятая, двоеточие, дефис. Порядок расположения разделителей в строке deiimeters не имеет значения. Разделители не включаются в число слов.

Третий конструктор позволяет включить разделители в число слов:

```
StringTokenizer(String str, String deiimeters, boolean flag);
```

Если параметр flag равен true , то разделители включаются в число слов, если false — нет. Например:

```
StringTokenizer("a - (Ь + с) / Ь * с", " \t\n\r+*-/(), true);
```

В разборе строки на слова активно участвуют два метода:

метод nextToken () возвращает в виде строки следующее слово;

логический метод hasMoreTokens () возвращает true , если в строке еще есть слова, и false , если слов больше нет.

Третий метод countTokens () возвращает число оставшихся слов.

Четвертый метод nextToken(string newDeiimeters) позволяет "на ходу" менять разделители. Следующее слово будет выделено по новым разделителям newDeiimeters; новые разделители действуют далее вместо старых разделителей, определенных в конструкторе или предыдущем методе nextToken () .

Оставшиеся два метода nextEiement () и hasMoreEiements () реализуют интерфейс Enumeration . Они просто обращаются к методам nextToken () и hasMoreTokens().

Схема очень проста:

Разбиение строки на слова :

```
import java.util.*;
class token {
 public static void  main(String arg[]) {
   String s = "Строка, которую мы    хотим   разобрать на слова";
   StringTokenizer st = new StringTokenizer(s, " \t\n\r,.");
   while(st.hasMoreTokens()){
     // Получаем слово и что-нибудь делаем с ним, например,
     // просто выводим на экран
     System.out.println(st.nextToken() ) ;
   };
 };
}
```

В листинге мы разобрали строку на слова. Как их сохранить для дальнейшей обработки?

Полученные слова обычно заносятся в какой-нибудь класс-коллекцию: Vector, Stack или другой, наиболее подходящий для дальнейшей обработки текста контейнер.

До сих пор мы пользовались массивами. Они удобны, если необходимо быстро обработать однотипные элементы, например, просуммировать числа, найти наибольшее и наименьшее значение, отсортировать элементы. Но уже для поиска нужных сведений в большом объеме информации массивы неудобны. Для этого лучше использовать бинарные деревья поиска.

Кроме того, массивы всегда имеют постоянную, предварительно заданную, длину, в массивы неудобно добавлять элементы. При удалении элемента из массива оставшиеся элементы следует перенумеровывать.

При решении задач, в которых количество элементов заранее неизвестно, элементы надо часто удалять и добавлять, надо искать другие способы хранения.

В языке Java с самых первых версий есть класс vector , предназначенный для хранения переменного числа элементов самого общего типа object .

Класс Vector

В классе vector из пакета java.util.* хранятся элементы типа object, а значит, любого типа. Количество элементов может быть любым и наперед не определяться. Элементы получают индексы 0, 1, 2, ... К каждому элементу вектора можно обратиться по индексу, как и к элементу массива.

Кроме количества элементов, называемого *размером* (size) вектора, есть еще размер буфера — *емкость* (capacity) вектора. Обычно емкость совпадает с размером вектора, но можно ее увеличить методом ensureCapacity(int minCapacity) или сравнять с размером вектора методом trimToSize().

В Java 2 класс vector переработан, чтобы включить его в иерархию классов-коллекций. Поэтому многие действия можно совершать старыми и новыми методами. Рекомендуется использовать новые методы, поскольку старые могут быть исключены *из* следующих версий Java.

Как создать вектор

В классе четыре конструктора:

vector () — создает пустой объект нулевой длины;

Vector (int capacity) — создает пустой объект указанной емкости capacity ;

vector (int capacity, int increment) — создает пустой объект указанной емкости capacity и задает число increment , на которое увеличивается емкость при необходимости;

vector (Collection c) — вектор создается по указанной коллекции. Если capacity отрицательно, создается исключительная ситуация. После создания вектора его можно заполнять элементами.

Как добавить элемент в вектор

Метод add(Object element) позволяет добавить элемент в конец вектора

(то же делает старый метод addElement (Object element).

Методом add(int index, Object element) или старым методом insertElementAt (Object element, int index) можно вставить элемент в указанное место index . Элемент, находившийся на этом месте, и все последующие элементы сдвигаются, их индексы увеличиваются на единицу.

Метод addAll(Collection coll) позволяет добавить в конец вектора все элементы коллекции coll .

Методом addAll(int index, Collection coll) возможно вставить в позицию index все элементы коллекции coll.

Как заменить элемент

Метод set(int index, object element) заменяет элемент, стоявший в векторе в позиции index , на элемент element (то же позволяет выполнить старый метод setElementAt (Object element, int index))

Как узнать размер вектора

Количество элементов в векторе всегда можно узнать методом size(). Метод capacity() возвращает емкость вектора.

Логический метод isEmpty() возвращает true , если в векторе нет ни одного элемента

Как обратиться к элементу вектора

Обратиться к первому элементу вектора можно методом firstEiement() , к последнему — методом lastEiement() , к любому элементу — методом get(int index) или старым методом elementAt (int index).

Эти методы возвращают объект класса object. Перед использованием его следует привести к нужному типу.

Получить все элементы вектора в виде массива типа object[] можно методами toArray() и toArray(Object [] a) . Второй метод заносит все элементы вектора в массив а, если в нем достаточно места.

Как узнать, есть ли элемент в векторе

Логический метод contains(object element) возвращает true, если элемент element находится в векторе.

Логический метод containsAii(Collection c) возвращает true , если вектор содержит все элементы указанной коллекции.

Как узнать индекс элемента

Четыре метода позволяют отыскать позицию указанного элемента element:

indexof (Object element) — возвращает индекс первого появления элемента в векторе;

indexof (Object element, int begin) — ведет поиск, начиная с индекса begin включительно;

lastindexof (object element) — возвращает индекс последнего появления элемента в векторе;

lastindexof (Object element, int start) — ведет поиск от индекса start включительно к началу вектора.

Если элемент не найден, возвращается —1.

Как удалить элементы

Логический метод remove (Object element) удаляет из вектора первое вхождение указанного элемента element . Метод возвращает true, если элемент найден и удаление произведено.

Метод remove(int index) удаляет элемент из позиции index и возвращает его в качестве своего результата типа object .

Аналогичные действия позволяют выполнить старые методы типа void: removeElement (Object element) и removeElementAt(int index), не возвращающие результата.

Удалить диапазон элементов можно методом removeRange(int begin, int end), не возвращающим результата. Удаляются элементы от позиции begin включительно до позиции end исключительно.

Удалить из данного вектора все элементы коллекции coil возможно логическим Методом removeAll(Collection coll).

Удалить последние элементы можно, просто урезав вектор методом setSizefin(newSize).

Удалить все элементы, кроме входящих в указанную коллекцию coil, разрешает логический метод retainAll(Collection coll).

Удалить все элементы вектора можно методом clear() или старым методом removeAHElements() или обнулив размер вектора методом setSize(0).

Листинг 1 расширяет предыдущую программу, обрабатывая выделенные из строки слова с помощью вектора.

Листинг 1. Работа с вектором

```
Vector v = new Vector();
String s = "Строка, которую мы хотим разобрать на слова.";
 StringTokenizer st = new StringTokenizer(s, " \t\n\r,.");
 while (st.hasMoreTokens()){
   // Получаем слово и заносим в вектор
  v.add(st.nextToken());        // Добавляем в конец вектора
}


System.out.println(v.firstElement());       // Первый элемент
System.out.println(v.lastElement());        // Последний элемент
 v.setSize(4);                              // Уменьшаем число элементов
```

```
v.add("собрать.");                                  // Добавляем в конец
                                                    // укороченного вектора
v.set(3, "опять");                                  // Ставим в позицию 3
for (int i = 0; i < v.size(); i++)                 // Перебираем весь вектор
   System.out.print(v.get(i) + " ");
System.out.println();
```

Класс vector является примером того, как можно объекты класса object, а значит, любые объекты, объединить в коллекцию. Этот тип коллекции упорядочивает и даже нумерует элементы. В векторе есть первый элемент, есть последний элемент. К каждому элементу обращаются непосредственно по индексу. При добавлении и удалении элементов оставшиеся элементы автоматически перенумеровываются.

Второй пример коллекции — класс stack — расширяет класс vector .

Класс Stack

Класс stack из пакета java.utii.* объединяет элементы в стек.

Стек (stack) реализует порядок работы с элементами подобно магазину винтовки— первым выстрелит патрон, положенный в магазин последним,— или подобно железнодорожному тупику — первым из тупика выйдет вагон, загнанный туда последним. Такой порядок обработки называется LIFO (Last In — First Out).

Перед работой создается пустой стек конструктором stack().

Затем на стек кладутся и снимаются элементы, причем доступен только "верхний" элемент, тот, что положен на стек последним.

Дополнительно к методам класса vector класс stack содержит пять методов, позволяющих работать с коллекцией как со стеком:

push(Object item) —помещает элемент item в стек;

pop() — извлекает верхний элемент из стека;

peek() — читает верхний элемент, не извлекая его из стека;

empty() — проверяет, не пуст ли стек;

search(object item) — находит позицию элемента item в стеке. Верхний элемент имеет позицию 1, под ним элемент 2 и т. д. Если элемент не найден, возвращается — 1.

Листинг 2 показывает, как можно использовать стек для проверки парности символов.

Листинг 2. Проверка парности скобок

```
import java.utii.*;
class StackTest
static boolean checkParity(String expression,
                    String open, String close){
   Stack stack = new Stack ();
   StringTokenizer st = new StringTokenizer(expression,
                                  "\t\n\r+*/-(){}", true);
   while (st.hasMoreTokens ()) {
     String tmp = st.nextToken();
     if ( tmp.equals(open) )  stack.push(open);
           if (tmp.equals(close) ) stack.pop();
   }
   if (stack.isEmpty () ) return true else return false;
}
public static void main(String[] args){
  System.out.println(
          checkParity("a - (b - (c - a) / (b + c) - 2)" , "(", ")" );
 }
}
```

Как видите, коллекции значительно облегчают обработку наборов данных.

Еще один пример коллекции совсем другого рода — таблицы — предоставляет класс Hashtable.

Класс Hashtable

Класс Hashtable расширяет абстрактный класс Dictionary . В объектах этого класса хранятся пары "ключ — значение".

Из таких пар "Фамилия И. О. — номер" состоит, например, телефонный справочник.

Еще один пример — анкета. Ее можно представить как совокупность пар "Фамилия — Иванов", "Имя — Петр", "Отчество — Сидорович", "Год рождения — 1975" и т. д.

Подобных примеров можно привести множество.

Каждый объект класса Hashtable кроме *размера* (size) — количества пар, имеет еще две характеристики: *емкость* (capacity) — размер буфера, и *показатель загруженности* (load factor) — процент заполненности буфера, по достижении которого увеличивается его размер.

Как создать таблицу

Для создания объектов класс Hashtable предоставляет четыре конструктора:

Hashtable() — создает пустой объект с начальной емкостью в 101 элемент и показателем загруженности 0,75;

Hashtable(int capacity) — создает пустой объект с начальной емкостью capacity и показателем загруженности 0,75;

Hashtable(int capacity, float loadFactor) — создает пустой Объект с начальной емкостью capacity и показателем загруженности loadFactor;

Hashtable(Map f) — создает объект класса Hashtable , содержащий все элементы отображения f, с емкостью, равной удвоенному числу элементов отображения f , но не менее 11, и показателем загруженности 0,75.

Как заполнить таблицу

Для заполнения объекта класса Hashtable используются два метода:

Object put(Object key, Object value) — добавляет пару " key— value ", если ключа key не было в таблице, и меняет значение value ключа key, если он уже есть в таблице. Возвращает старое значение ключа или null, если его не было. Если хотя бы один параметр равен null , возникает исключительная ситуация;

void putAll(Map f) — добавляет все элементы отображения f . В объектах-ключах key должны быть реализованы методы hashCode() и equals ().

Как получить значение по ключу

Метод get(Object key) возвращает значение элемента с ключом key в виде объекта класса object . Для дальнейшей работы его следует преобразовать к конкретному типу.

Как узнать наличие ключа или значения

Логический метод containsKey(object key) возвращает true, если в таблице есть ключ key .

Логический метод containsvalue(Object value) или старый метод contains(object value) возвращают true, если в таблице есть ключи со значением value .

Логический метод isEmpty() возвращает true , если в таблице нет элементов.

Как получить все элементы таблицы

Метод values() представляет все значения value таблицы в виде интерфейса Collection . Все модификации в объекте collection изменяют таблицу, и наоборот.

Метод keyset() предоставляет все ключи key таблицы в виде интерфейса set . Все изменения в объекте set корректируют таблицу, и наоборот.

Метод entrySet() представляет все пары "key— value" таблицы в виде интерфейса Set . Все модификации в объекте set изменяют таблицу, и наоборот.

Метод tostring () возвращает строку, содержащую все пары.

Старые методы elements() и keys() возвращают значения и ключи в виде интерфейса Enumeration .

Как удалить элементы

Метод remove(Object key) удаляет пару с ключом key , возвращая значение этого ключа, если оно есть, и null , если пара с ключом key не найдена.

Метод clear() удаляет все элементы, очищая таблицу.

В листинге 3 показано, как можно использовать класс Hashtabie для создания телефонного справочника, а на рис. 1 — вывод этой программы.

Листинг 3. Телефонный справочник

```
import java.util.*;
class PhoneBook{
public static void main(String[] args){
  Hashtable yp = new Hashtable();
  String name = null;
  yp.put("John", "123-45-67");
  yp.put ("Lemon", "567-34-12");
  yp.put("Bill", "342-65-87");
  yp.put("Gates", "423-83-49");
  yp.put("Batman", "532-25-08");
  try{
      name = args[0];
  }
  catch(Exception e){
       System.out.println("Usage: Java PhoneBook Name");
       return;
  }
  if (yp.containsKey(name))
      System.out.println(name + "'s phone = " + yp.get(name));
  else
    System.out.println("Sorry, no such name");
 };
}
```

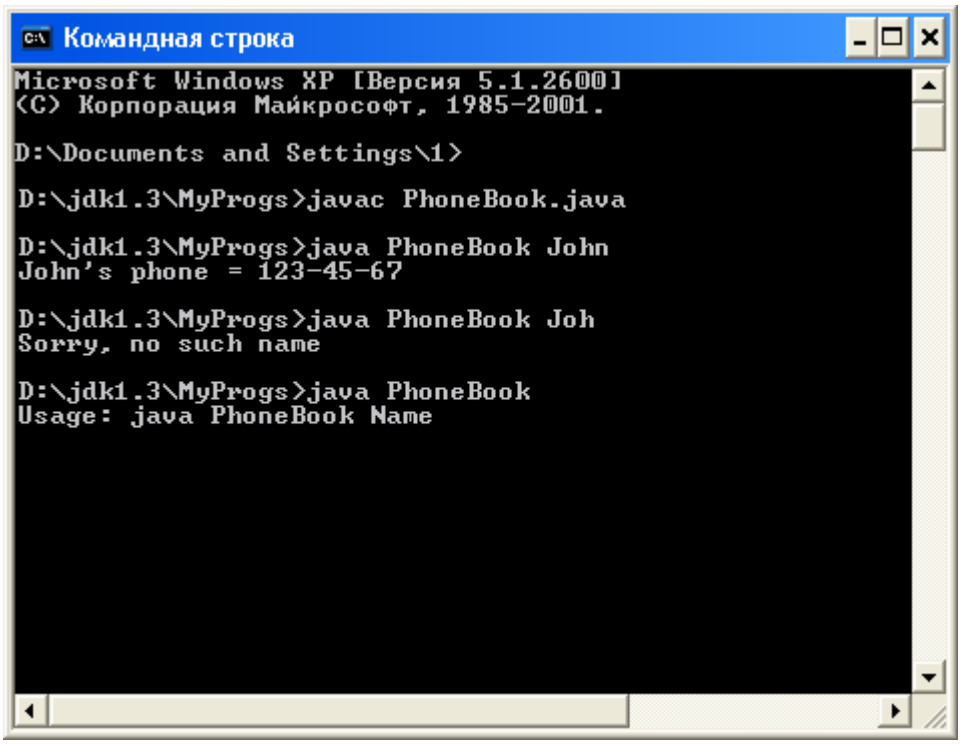

Рис. 1. Работа с телефонной книгой

Класс Properties

Класс Properties расширяет класс Hashtabie . Он предназначен в основном для ввода и вывода пар свойств системы и их значений. Пары хранятся в виде строк типа String . В классе Properties два конструктора:

Properties () — создает пустой объект;

Properties (Properties default) — создает объект с заданными парами свойств default .

Кроме унаследованных от класса Hashtabie методов в классе Properties есть еще следующие методы.

Два метода, возвращающих значение ключа-строки в виде строки:

- string getProperty(string key) — возвращает значение по ключу key ;
- String getProperty(String key, String defaultValue) — возвращает значение по ключу key ; если такого ключа нет, возвращается defaultValue .

Метод setProperty(String key, String value) добавляет новую пару, если ключа key нет, и меняет значение, если ключ key есть.

Метод load(Inputstream in) загружает свойства из входного потока in .

Методы list(PrintStream out) и list(PrintWriter out) выводят свойства в выходной поток out.

Метод store(OutputStream out, String header) выводит свойства в выходной поток out с заголовком header .

Очень простой листинг 4 и рис. 2 демонстрируют вывод всех системных свойств Java.

Листинг 4. Вывод системных свойств

```
class Prop{
  public static void main(String[] args){
  System.getProperties().list(System.out);
 }
}
```

Примеры классов Vector, Stack, Hashtabie, Properties показывают удобство классов-коллекций. Поэтому в Java 2 разработана целая иерархия коллекций. Она показана на рис. 3. Курсивом записаны имена интерфейсов. Пунктирные линии указывают классы, реализующие эти интерфейсы. Все коллекции разбиты; на три группы, описанные в интерфейсах List, Set и Map.

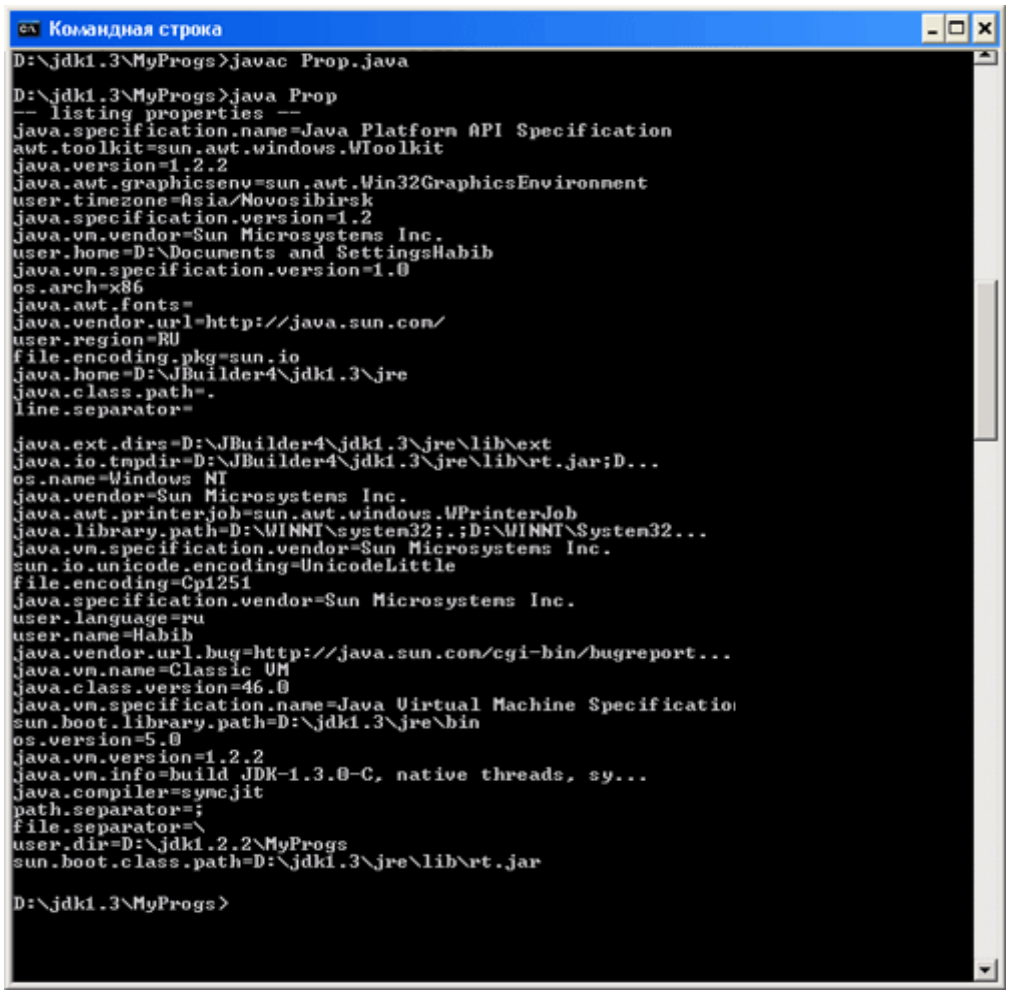

Рис. 6.2. Системные свойства

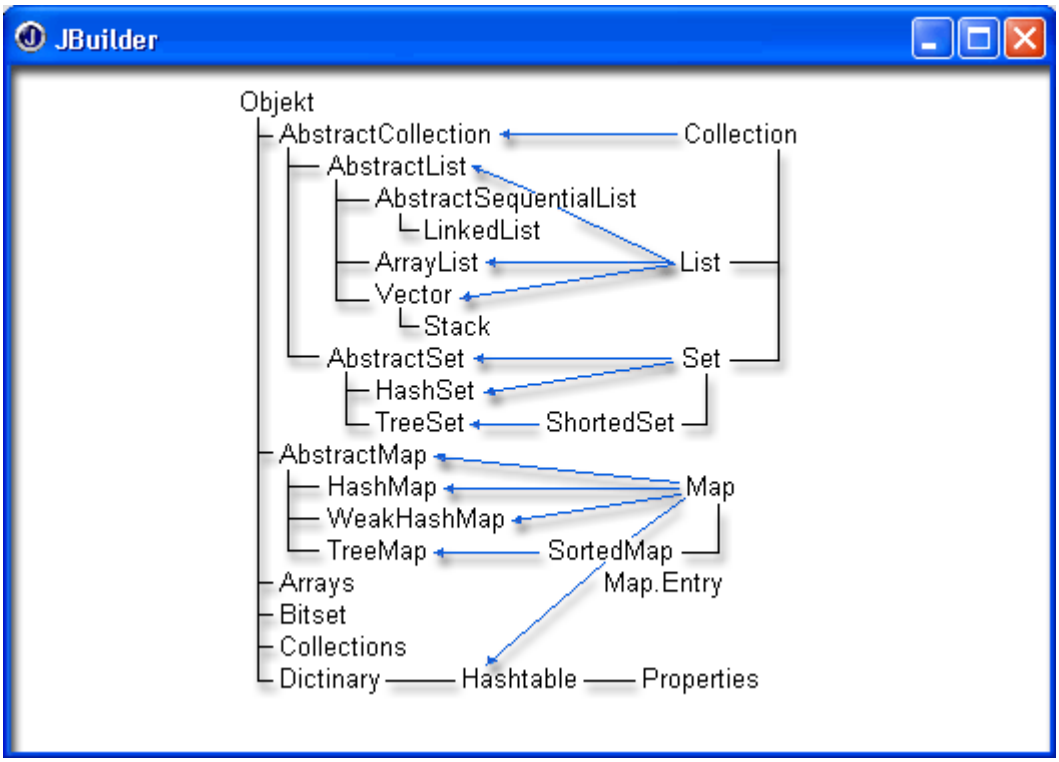

Рис. 3. Иерархия классов и интерфейсов-коллекций

Примером реализации интерфейса List может служить класс Vector , примером реализации интерфейса map — класс Hashtable .

Коллекции List и set имеют много общего, поэтому их общие методы объединены и вынесены в суперинтерфейс Collection .

Посмотрим, что, по мнению разработчиков Java API, должно содержаться в этих коллекциях.

Интерфейс Collection

Интерфейс collection из пакета java.util описывает общие свойства коллекций List и set . Он содержит методы добавления и удаления элементов, проверки и преобразования элементов:

boolean add(Object obj) — добавляет элемент obj в конец коллекции; возвращает false , если такой элемент в коллекции уже есть, а коллекция не допускает повторяющиеся элементы; возвращает true , если добавление прошло удачно;

boolean addAll(Collection coll) — добавляет все элементы коллекции coll в конец данной коллекции;

void clear( ) — удаляет все элементы коллекции;

boolean contains (Object obj) — проверяет наличие элемента obj в коллекции;

boolean containsAii (Collection coll) — проверяет наличие всех элементов коллекции coll в данной коллекции;

boolean isEmpty() — проверяет, пуста ли коллекция;

iterator iterator () — возвращает итератор данной коллекции;

boolean remove(object obj) — удаляет указанный элемент из коллекции; возвращает false , если элемент не найден, true , если удаление прошло успешно;

boolean removeAll(Collection coil) — удаляет элементы указанной коллекции, лежащие в данной коллекции;

boolean retainAll(Collection coll) — удаляет все элементы данной коллекции, кроме элементов коллекции coll ;

int size () — возвращает количество элементов в коллекции;

object [] toArray () — возвращает все элементы коллекции в виде массива;

Objectn toArray(object[] a) — записывает все элементы коллекции в массив a, если в нем достаточно места.

Интерфейс List

Интерфейс List из пакета java.utii, расширяющий интерфейс collection , описывает методы работы с упорядоченными коллекциями. Иногда их называют *последовательностями* (sequence). Элементы такой коллекции пронумерованы, начиная от нуля, к ним можно обратиться по индексу. В отличие от коллекции Set элементы коллекции List могут повторяться.

Класс vector — одна из реализаций интерфейса List .

Интерфейс List добавляет к методам интерфейса Collection методы, использующие индекс index элемента:

void add(int index, object obj) — вставляет элемент obj в позицию index ; старые элементы, начиная с позиции index , сдвигаются, их индексы увеличиваются на единицу;

boolean addAll(int index, Collection coll) — вставляет все элементы коллекции coil ;

object get(int index) — возвращает элемент, находящийся в позиции index ;

int indexOf(Object obj) — возвращает индекс первого появления элемента obj в коллекции;

int lastindexOf(object obj) — возвращает индекс последнего появления элемента obj в коллекции;

Listiterator listiterator () — возвращает итератор коллекции;

Listiterator listiterator (int index) — возвращает итератор конца коллекции от позиции index ;

object set (int index, object obj) — заменяет элемент, находящийся в позиции index , элементом obj ;

List subListUnt from, int to) — возвращает часть коллекции от позиции from включительно до позиции to исключительно.

Интерфейс Set

Интерфейс set из пакета java.utii, расширяющий интерфейс Collection , описывает неупорядоченную коллекцию, не содержащую повторяющихся элементов. Это соответствует математическому понятию *множества* (set) . Такие коллекции удобны для проверки наличия или отсутствия у элемента свойства, определяющего множество. Новые методы в интерфейс Set не добавлены, просто метод add () не станет добавлять еще одну копию элемента, если такой элемент уже есть в множестве.

Этот интерфейс расширен интерфейсом sortedset

Интерфейс SortedSet

Интерфейс sortedset из пакета java.utii, расширяющий интерфейс Set, описывает упорядоченное множество, отсортированное по естественному порядку возрастания его элементов или по порядку, заданному реализацией интерфейса comparator.

Элементы не нумеруются, но есть понятие первого, последнего, большего и меньшего элемента.

Дополнительные методы интерфейса отражают эти понятия:

comparator comparator() — возвращает способ упорядочения коллекции; object first ()— возвращает первый, меньший элемент коллекции;

SortedSet headset(Object toEiement) — возвращает начальные, меньшие элементы до элемента toEiement исключительно;

object last() — возвращает последний, больший элемент коллекции;

SortedSetsubset(Object fromElement, Object toEiement) — Возвращает подмножество коллекции от элемента fromElement включительно до элемента toEiement исключительно;

SortedSettallSet (Object fromElement) — возвращает последние, большие элементы коллекции от элемента fromElement включительно.

Интерфейс Map

Интерфейс Map из пакета java.utii описывает коллекцию, состоящую из пар "ключ — значение". У каждого ключа только одно значение, что соответствует математическому понятию однозначной функции или *отображения* (map).

Такую коллекцию часто называют еще *словарем* (dictionary) или *ассоциативным массивом* (associative array).

Обычный массив — простейший пример словаря с заранее заданным числом элементов. Это отображение множества первых неотрицательных целых чисел на множество элементов массива, множество пар "индекс массива ^-элемент массива".

Класс HashTable — одна из реализаций интерфейса map.

Интерфейс Map содержит 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 keyset () — 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():

методы getKey() и getvaiue() позволяют получить ключ и значение пары; метод setvaiue (object value) меняет значение в данной паре.

Интерфейс SortedMap

Интерфейс SortedMap , расширяющий интерфейс Map , описывает упорядоченную по ключам коллекцию map. Сортировка производится либо в естественном порядке возрастания ключей, либо, в порядке, описываемом в интерфейсе Comparator .

Элементы не нумеруются, но есть понятия большего и меньшего из двух элементов, первого, самого маленького, и последнего, самого большого элемента коллекции. Эти понятия описываются следующими методами:

comparator comparator () — возвращает способ упорядочения коллекции;

object firstKey() — возвращает первый, меньший элемент коллекции;

SortedMap headMap(Object toKey) — возвращает начало коллекции до элемента с ключом toKey исключительно;

object lastKey() — возвращает последний, больший ключ коллекции;

SprtedMap subMap (Object fromKey, Object toKey) — возвращает часть коллекции от элемента с ключом fromKey включительно до элемента с ключом toKey исключительно;

SortedMap taiiMap (object fromKey) — возвращает остаток коллекции от элемента fromKey включительно.

Вы можете создать свои коллекции, реализовав рассмотренные интерфейсы. Это дело трудное, поскольку в интерфейсах много методов. Чтобы облегчить эту задачу, в Java API введены частичные реализации интерфейсов — абстрактные классы-коллекции.

Абстрактные классы-коллекции

Эти классы лежат в пакете java.util,

Абстрактный класс AbstractGollection реализует интерфейс Collection, но оставляет нереализованными методы iterator (), size ().

Абстрактный класс AbstractList реализует интерфейс List , но оставляет нереализованным метод get(mt) и унаследованный метод size() Этот класс позволяет реализовать коллекцию с прямым доступом к элементам, подобно массиву.

Абстрактный класе AbsttaatSequantaaiList реализует интерфейс List, но оставляет нереализованным метод listiterator(int index) и унаследованный метод size() . Данный класс позволяет реализовать коллекции с последовательным доступом к элементам с помощью итератора Listiterator

Абстрактный класс Abstractset реализует интерфейс Set, но оставляет нереализованными методы, унаследованные от Absjractcollection

Абстрактный класс AbstractMap реализует интерфейс Map , но оставляет нереализованным метод entrySet (),

Наконец, в составе Java API есть полностью реализованные классы-коллекции помимо уже рассмотренных классов Vectdr, Stack, Hashtable и Properties , Это классы ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap, WeakHashMap ,

Для работы с этими классами разработаны интерфейсы iterator ,

Listiterator, Comparator И классы *Arrays* И Collections.

Перед тем Как рассмотреть использование данных классов, обсудим понятие итератора..

Интерфейс Iterator

В 70—80-х годах прошлого столетия, после того как была осознана важность правильной организации данных в определенную структуру, большое внимание уделялось изучению' и' Построению различных структур данных: связанных списков, очередей, деков, стеков, деревьев, сетей

Вместе с развитием структур данных развивались и алгоритмы работы с ними: сортировка, поиск, обход, хэширование.

Этим вопросам посвящена Обширная литература, посмотрите, например, книгу [11]. '

В 90-х годах было решено заносить данные в определенную коллекцию, скрыв ее внутреннюю структуру, а для работы с данными использовать методы этой коллекции.

В частности, задачу обхода возложили на саму коллекцию. В Java API введен интерфейс iterator , описывающий способ обхода всех элементов коллекции. В каждой коллекции есть метод iterator (), возвращающий реализацию интерфейса iterator для указанной коллекции. Получив эту реализацию, можно обходить коллекцию в некотором порядке, определенном данным итератором, с помощью методов, описанных в интерфейсе iterator и реализованных в этом итераторе. Подобная техника использована в классе

StringTokenizer.

В интерфейсе iterator описаны всего три метода:

- логический метод hasNext () возвращает true , если обход еще не завершен;
- метод next () делает текущим следующий элемент коллекции и возвращает его в виде объекта класса object ;
- метод remove() удаляет текущий элемент коллекции.

Можно представить себе дело так, что итератор — это указатель на элемент коллекции. При создании итератора указатель устанавливается перед первым элементом, метод next () перемещает указатель на первый элемент и показывает его. Следующее применение метода next () перемещает указатель на второй элемент коллекции и показывает его. Последнее применение метода next () выводит указатель за последний элемент коллекции.

Метод remove (), пожалуй, излишен, он уже не относится к задаче обхода коллекции, но позволяет при просмотре коллекции удалять из нее ненужные элементы.

В листинге 5 к тексту листинга 1 добавлена работа с итератором.

Листинг 5. Использование итератора вектора

```
Vector v = new Vector();
String s = "Строка, которую мы хотим разобрать на слова.";
StringTokenizer st = new StringTokenizer(s, " \t\n\r,.");
while (st.hasMoreTokens()){
     // Получаем слово и заносим в вектор.
     v.add(st.nextToken());                          // Добавляем в конец вектора
                          }
System.out.println(v.firstElement());                      // Первый элемент
System.out.println(v.lastElement() );                      // Последний элемент
v.setSize(4);                              // Уменьшаем число элементов
  v.add("собрать.");                       // Добавляем в конец укороченного вектора
v.set(3, "опять");                         // Ставим в позицию 3
for (int i = 0; i < v.size(); i++)           // Перебираем весь вектор
System.out.print(v.get(i) + ".");
System.out.println();
Iterator it = v.iterator ();                          // Получаем итератор вектора
```

```
try{
    while(it.hasNext())                          // Пока в векторе есть элементы,
        System.out.println(it.next());           // выводим текущий элемент
}catch(Exception e){}
```

Интерфейс ListIterator

Интерфейс Listiterator расширяет интерфейс iterator , обеспечивая перемещение по коллекции как в прямом, так и в обратном направлении. Он может быть реализован только в тех коллекциях, в которых есть понятия следующего и предыдущего элемента и где элементы пронумерованы.

В интерфейс Listiterator добавлены следующие методы:

void add (Object element) — добавляет элемент element перед текущим элементом;

boolean hasPrevious () — возвращает true , если в коллекции есть элементы, стоящие перед текущим элементом;

int nextindex() — возвращает индекс текущего элемента; если текущим является последний элемент коллекции, возвращает размер коллекции;

Object previous () — возвращает предыдущий элемент и делает его текущим;

int previous index () — возвращает индекс предыдущего элемента;

void set (Object element) — заменяет текущий элемент элементом element;

выполняется сразу после next () или previous ().

Как видите, итераторы могут изменять коллекцию, в которой они работают, добавляя, удаляя и заменяя элементы. Чтобы это не приводило к конфликтам, предусмотрена исключительная ситуация, возникающая при попытке использования итераторов параллельно "родным" методам коллекции. Именно поэтому в листинге 5 действия с итератором заключены в блок try {} catch(){} .

Изменим окончание листинга 5 с использованием итератора Listiterator .

```
// Текст листинга 1...
// ...
Listiterator lit = v.listlterator();    // Получаем итератор вектора
                 // Указатель сейчас находится перед началом вектора
try{
   while(lit.hasNext())              // Пока в векторе есть элементы
      System.out.println(lit.next());          // Переходим к следующему
                                               // элементу и выводим его
         // Теперь указатель за концом вектора. Пройдем к началу
while (lit.hasPrevious ())
       System.out.println( lit.previblis()); :
}catch (Exception e) ()
```

Интересно, что повторное применение методов next () и previous () друг за другом будет выдавать один и тот же текущий элемент. Посмотрим теперь, какие возможности предоставляют классы-коллекции Java 2.

Классы, создающие списки

Класс ArrayList полностью реализует интерфейс List и итератор типа iterator . Класс ArrayList очень похож на класс Vector,имеет тот же набор методов и может использоваться в тех же ситуациях.

В классе ArrayList три конструктора;

ArrayList ()—создает пустой объект;

ArrayList (Collection coil) — создает объект, содержащий все элементы коллекции coll ;

ArrayList (int initCapacity) — создает пустой Объект емкости initCapacity .

Единственное отличие класса ArrayList от класса vector заключается в том, что класс ArrayList не синхронизован. Это означает что одновременное изменение экземпляра этого класса несколькими подпроцессами приведет к непредсказуемым результатам.

Двунаправленный список

Класс LinkedList полностью реализует интерфейс List и содержит дополнительные методы, превращающие его в двунаправленный список. Он реализует итераторы типа iterator и bistiterator .

Этот класс можно использовать для обработки элементов в стеке, деке или двунаправленном списке.

В классе LinkedList два конструктора: .

LinkedList - создает пустойобъект

LinkedList (Collection coil) — создает объект, содержащий все элементы коллекции coll.

Классы, создающие отображения

Класс например полностью реализует интерфейс Map , а также итератор типа iterator . Класс HashMap очень похож на класс Hashtabie и может использоваться в тех же ситуациях. Он имеет тот же набор функций и такие же конструкторы:

HashMap () — создает пустой объект с показателем загруженности 0,75;

HashMap(int .capacity) - создает пустой объект с начальной емкостью capacity и показателем загруженности 0,75;

HashMap (int capacity, float loadFactor) — создает пустой объект С начальной емкостью capacity и показателем загруженности loadFactor ;

HashMap(Map f) — создает объект класса HashMap , содержащий все элементы отображения f, с емкостью, равной удвоенному числу элементов отображения f, но не менее 11, и показателем загруженности 0,75.

Класс WeakHashMap отличается от класса HashMap только тем, что в его объектах неиспользуемые элементы, на которые никто не ссылается, автоматически исключаются из объекта.

Упорядоченные отображения

Класс TreeMap полностью реализует интерфейс sortedMap . Он реализован как бинарное дерево поиска, значит его элементы хранятся в упорядоченном виде. Это значительно ускоряет поиск нужного элемента.

Порядок задается либо естественным следованием элементов, либо объектом, реализующим интерфейс сравнения Comparator .

В этом классе четыре конструктора:

TreeMap () — создает пустой объект с естественным порядком элементов;

TreeMap (Comparator c) — создает пустой объект, в котором порядок задается объектом сравнения с ;

TreeMap (Map f) — создает объект, содержащий все элементы отображения f, с естественным порядком 'его элементов;

TreeMap (SortedMap sf) — создает объект, содержащий все элементы отображения sf , в том же порядке.

Здесь надо пояснить, каким образом можно задать упорядоченность элементов коллекции

Сравнение элементов коллекций

Интерфейс Comparator описывает два метода сравнения:

int compare (Object obji, object obj2) — возвращает отрицательное число, если objl в каком-то смысле меньше obj2 ; нуль, если они считаются равными; положительное число, если objl больше obj2 . Для читателей, знакомых с теорией множеств, скажем, что этот метод сравнения обладает свойствами тождества, антисимметричности и транзитивности;

boolean equals (Object obj) — сравнивает данный объект с объектом obj , возвращая true , если объекты совпадают в каком-либо смысле, заданном этим методом.

Для каждой коллекции можно реализовать эти два метода, задав конкретный способ сравнения элементов, и определить объект класса SortedMap вторым конструктором. Элементы коллекции будут автоматически отсортированы в заданном порядке.

Листинг 6 показывает один из возможных способов упорядочения комплексных чисел — объектов класса complex из листинга 4. Здесь описывается класс ComplexCompare , реализующий интерфейс Comparator , В листинге 7 он применяется для упорядоченного хранения множества комплексных чисел.

Листинг 6. Сравнение комплексных чисел

```
import java.util.*;
class ComplexCompare implements Comparator{
public int compare(Object objl, Object obj2){
Complex zl = (Complex)objl, z2 = (Complex)obj2;
```

```
double rel = zl.getRe(), iml = zl.getlm();
double re2 = z2.getRe(), im2 = z2.getlm();
if (rel != re2) return (int)(rel - re2);
else if (iml != im2) return (int)(iml — im2);
else return 0;
}
public boolean equals(Object z) {
return compare(this, z) == 0;
 }
}
```

Классы, создающие множества

Класс HashSet полностью реализует интерфейс set и итератор типа iterator . Класс Hashset используется в тех случаях, когда надо хранить только одну копию каждого элемента.

В классе HashSet четыре конструктора:

Hashset () — создает пустой объект с показателем загруженности 0,75;

HashSet (int capacity) — создает пустой объект с начальной емкостью capacity и показателем загруженности 0,75;

HashSet (int capacity, float loadFactor) — создает пустой объект с начальной емкостью capacity и показателем загруженности loadFactor ;

HashSet (Collection coll) — создает объект, содержащий все элементы коллекции coll , с емкостью, равной удвоенному числу элементов коллекции coll , но не менее 11, и показателем загруженности 0,75.

Упорядоченные множества

Класс TreeSet полностью реализует интерфейс sortedset и итератор типа iterator . Класс TreeSet реализован как бинарное дерево поиска, значит, его элементы хранятся в упорядоченном виде. Это значительно ускоряет поиск нужного элемента.

Порядок задается либо естественным следованием элементов, либо объектом, реализующим интерфейс сравнения Comparator .

Этот класс удобен при поиске элемента во множестве, например, для проверки, обладает ли какой-либо элемент свойством, определяющим множество.

В классе TreeSet четыре конструктора:

TreeSet () — создает пустой объект с естественным порядком элементов;

TreeSet (Comparator c) — создает пустой объект, в котором порядок задается объектом сравнения с;

TreeSet (Collection coll) — создает объект, содержащий все элементы коллекции coll , с естественным порядком ее элементов;

TreeSet (SortedMap sf) — создает объект, содержащий все элементы отображения sf , в том же порядке.

В листинге 7 показано, как можно хранить комплексные числа в упорядоченном виде. Порядок задается объектом класса ComplexCompare , определенного в листинге 6.

Листинг 7. Хранение комплексных чисел в упорядоченном виде

```
TreeSet ts = new TreeSet (new ComptexCompare());
ts.add(new Complex(1.2, 3.4));
ts. add (new Complex (-1.25, 33.4»;
ts.add(new Complex(1.23, -3.45));
ts.add(new Complex(16.2, 23.4));
Iterator it = ts.iterator();
while(it.hasNext()) , ((Complex)it.next()).pr();
```

Действия с коллекциями

Коллекции предназначены для хранения элементов в удобном для дальнейшей обработки виде. Очень часто обработка заключается в сортировке элементов и поиске нужного элемента. Эти и другие методы обработки собраны В Класс Collections .

Методы класса Collections

Все методы класса collections статические, ими можно пользоваться, не создавая экземпляры классу Collections

Как обычно в статических методах, коллекция, с которой работает метод, задается его аргументом.

Сортировка может быть сделана только в упорядочиваемой коллекции, реализующей интерфейс List . Для сортировки в классе collections есть два метода:

static void sort (List coll) — сортирует в естественном порядке возрастания коллекцию coll, реализующую интерфейс List;

static void sort (List coll, Comparator c) — сортирует коллекцию coll

в порядке, заданном объектом с. После сортировки можно осуществить бинарный поиск в коллекции:

static int binarySearch(List coll, Object element) — отыскивает элемент element в отсортированной в естественном порядке возрастания коллекции coll и возвращает индекс элемента или отрицательное число, если элемент не найден; отрицательное число показывает индекс, с которым элемент element был бы вставлен в коллекцию, с обратным знаком;

static int binarySearchfList coll, Object element, Comparator c) — ТО же, но коллекция отсортирована в порядке , определенном объектом с .

Четыре метода находят наибольший и наименьший элементы в упорядочиваемой коллекции:

static object max (Collection coll) — возвращает наибольший в естественном порядке элемент коллекции coll;

static Object max (Collection coll, Comparator c) — ТО же В порядке , заданном объектом с ;

static object mm (Collection coll) — возвращает наименьший в естественном порядке элемент коллекции сои;

static Object min(Collection coll, Comparator c) — ТО же В порядке , заданном объектом с .

Два метода "перемешивают" элементы коллекции в случайном порядке:

static void shuffle (List coll) — случайные числа задаются по умолчанию;

static void shuffle (List coll, Random r) — случайные числа определяются объектом г .

Метод reverse (List coll) меняет порядок расположения элементов на обратный.

Метод copy (List from, List to) копирует коллекцию from в коллекцию to .

Метод fill (List coll, object element) заменяет все элементы существующей коллекции coll элементом element .

С остальными методами познакомимся по мере надобности.

Заключение

Итак, мы выяснили, что язык Java предоставляет множество средств для работы с большими объемами информации. В большинстве случаев достаточно добавить в программу три-пять операторов, чтобы можно было проделать нетривиальную обработку информации.