Лабораторная работа №9

РАБОТА С КОЛЛЕКЦИЯМИ

Понятие коллекции

Для хранения большого количества однотипных данных могут использоваться массивы, но они не всегда являются идеальным решением. Во-первых, длина массива задается заранее и в случае, если количество элементов заранее неизвестно, придется либо выделять память «с запасом», либо предпринимать сложные действия по переопределению массива. Во-вторых, элементы массива имеют жестко заданное размещение в его ячейках, поэтому, например, удаление элемента из массива не является простой операцией.

В программировании давно и эффективно используются такие структуры данных как стек, очередь, список, множество и т.д., объединенные общим названием коллекция. *Коллекция* — это группа элементов с операциями добавления, извлечения и поиска элемента. Механизм работы операций существенно различается в зависимости от типа коллекции. Например, элементы *стека* упорядочены в последовательность, добавление нового элемента может происходить только в конец этой последовательности, и получить можно только элемент, находящийся в конце (то есть, добавленный последним). *Очередь*, напротив, позволяет получить лишь первый элемент (элементы добавляются в один конец последовательности, а «забираются» с другого). Другие коллекции (например, *список*) позволяют получить элемент из любого места последовательности, а *множество* вообще не упорядочивает элементы и позволяет (помимо добавления и удаления) только узнать, содержится ли в нем данный элемент.

Язык Java предоставляет библиотеку стандартных коллекций, которые собраны в пакете java.util, поэтому нет необходимости программировать их самостоятельно.

При работе с коллекциями главное — не пользоваться наиболее универсальной коллекцией вместо той, которая необходима для решения задачи — например, списком вместо стека. Если логика работы программы такова, что данные должны храниться в стеке (появляться и обрабатываться в обратной последовательности), следует использовать именно стек. В этом случае вы не сможете нарушить логику обработки данных, обратившись напрямую к середине последовательности, а значит, шанс появления трудно обнаруживаемых ошибок резко уменьшается.

Чтобы выбрать коллекцию, которая лучше всего подходит условию задачи, необходимо знать особенности каждой из них.

Интерфейсы и классы коллекций

Коллекции в библиотеке java.util представлены набором классов и интерфейсов.

Каждый класс реализует некоторую коллекцию со специфичным для нее набором операций доступа к элементам. Чтобы использовать коллекцию в своей программе, нужно создать объект соответствующего класса.

Элементы большинства коллекций имеют тип Object. Это значит, что (в отличие от обычного массива) вы не должны заранее указывать тип элементов, которые будете помещать в коллекцию. Вы можете добавлять в нее объекты любого класса, поскольку все классы являются наследниками класса Object, более того — в одной коллекции могут храниться объекты разных классов.

Некоторые коллекции в пакете java.utils не представлены самостоятельными классами. Например, очередь и список. Но для этих полезных структур данных определены соответствующие интерфейсы, то есть можно пользоваться любым классом, реализующим такой интерфейс.

Интерфейс Collection

Интерфейс Collection содержит набор общих методов, которые используются в большинстве коллекций.

add(Object item) — добавляет в коллекцию новый элемент. Метод возвращает **true**, если добавление прошло удачно и **false** — если нет. Если элементы коллекции каким-то образом упорядочены, новый элемент добавляется в конец коллекции.

clear() — удаляет все элементы коллекции.

contains(Object obj) — возвращает **true**, если объект obj содержится в коллекции и **false**, если нет.

isEmpty() — проверяет, пуста ли коллекция.

remove(Object obj) — удаляет из коллекции элемент obj. Возвращает **false**, если такого элемента в коллекции не нашлось.

size() — возвращает количество элементов коллекции.

Существуют разновидности перечисленных методов, которые в качестве параметра принимают любую другую коллекцию. Например, метод addAll(Collection coll) добавляет все элементы другой коллекции coll в конец данной, метод removeAll(Collection coll) удаляет из коллекции все элементы, которые присутствуют также в коллекции coll, а метод retainAll(Collection coll) поступает наоборот, удаляя все элементы, кроме содержащихся в coll.

Метод toArray() возвращает все элементы коллекции в виде массива.

Интерфейс List

Интерфейс List описывает упорядоченный список. Элементы списка пронумерованы, начиная с нуля и к конкретному элементу можно обратиться по целочисленному индексу. Имеются следующие реализации этого интерфейса: ArrayList, LinkedList, Vector, Stack.

Интерфейс List является наследником интерфейса Collection, поэтому содержит все его методы и добавляет к ним несколько своих:

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

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

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

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

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

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

Интерфейс Set

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

Интерфейс Queue

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

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

о методах интерфейса Queue

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

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

Класс Vector

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

Особенность вектора в том, что помимо текущего размера, определяемого количеством элементов в нем и возвращаемого методом size(), он имеет емкость, возвращаемую методом capacity() — размер выделенной под элементы памяти, которая может быть больше текущего размера. Ее можно увеличить методом ensureCapacity(int minCapacity) или сравнять с размером вектора методом trimToSize(). Каждый раз, когда нужно добавить в полностью «заполненный» вектор новый элемент, емкость увеличивается с запасом.

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

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.

Методы класса Vector:

add (Object element) — позволяет добавить элемент в конец вектора:

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

addAll (Collection coll) — позволяет добавить в конец вектора все элементы коллекции coll;

addAll(int index, Collection coll) — вставляет в позицию index все элементы коллекции coll.

set (int index, object element) — заменяет элемент, стоявший в векторе в позиции index, на элемент element;

Количество элементов в векторе всегда можно узнать методом **size().**

capacity()— возвращает емкость вектора.

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

Обратиться к первому элементу вектора можно методом **firstEiement()**, к последнему — методом **lastEiement()**, к любому элементу — методом **get(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()**.

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

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

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.

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.

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 coll возможно методом **removeAll(Collection coll).**

Удалить последние элементы можно, просто урезав вектор методом **setSize(int 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 **removeAllElements()** или обнулив размер вектора методом **setSize(0).**

Объекты можно либо записывать в конец объекта Vector с помощью метода addElement, либо вставлять в указанную индексом позицию методом **insertElementAt**. Вы можете также записать в Vector массив объектов, для этого нужно воспользоваться методом **copyInto**. После того, как в Vector записана коллекция объектов, можно найти в ней индивидуальные элементы с помощью методов **Contains, indexOf** и **lastIndexOf**. Кроме того методы **elementAt, firstElement и lastElement** позволяют извлекать объекты из нужного положения в объекте Vector.

Класс ArrayList

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

Класс Stack

Класс Stack расширяет класс Vector. Стек (stack) реализует порядок работы с элементами по принципу, который 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.

Ниже приведен пример программы, которая создает стек, заносит в него несколько объектов типа Integer, а затем извлекает их.

```
import java.util.Stack;
import java.util.EmptyStackException;
class StackDemo {
static void showpush(Stack st, int a) {
st.push(new Integer(a));
System.out.println("push(" + a + ")");
System.out.println("stack: " + st);
```

```
}
static void showpop(Stack st) {
System.out.print("pop -> ");
Integer a = (Integer) st.pop();
System.out.println(a);
System.out.println("stack: " + st);
}
public static void main(String args[]) {
Stack st = new Stack();
System.out.println("stack: " + st);
showpush(st, 42);
showpush(st, 66);
showpush(st, 99);
showpop(st);
showpop(st);
showpop(st);
try {
showpop(st);
}
catch (EmptyStackException e) {
System.out.println("empty stack");
} }
}
```

Ниже приведен результат, полученный при запуске этой программы. Обратите внимание на то, что обработчик исключений реагирует на попытку извлечь данные из пустого стека. Благодаря этому мы можем аккуратно обрабатывать ошибки такого рода.

```
stack: []
push(42)
stack: [42]
push(66)
stack: [42, 66]
push(99)
stack: [42, 66, 99]
pop -> 99
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: []
pop -> empty stack
```

Класс Dictionary

Dictionary (словарь) — абстрактный класс, представляющий собой хранилище информации типа “ключ-значение”. Ключ — это имя, по которому осуществляется доступ к значению. Имея ключ и значение, вы можете записать их в словарь методом put(key, value). Для получения значения по заданному ключу служит метод get(key). И ключи, и значения можно получить в форме перечисления (объект Enumeration) методами keys и elements. Метод size возвращает количество пар “ключ-значение”, записанных в словаре, метод isEmpty возвращает true, если словарь пуст. Для удаления ключа и связанного с ним значения предусмотрен метод remove(key).

Класс HashTable

HashTable — это подкласс Dictionary, являющийся конкретной реализацией словаря. Представителя класса HashTable можно использовать для хранения произвольных объектов, причем для индексации в этой коллекции также годятся любые объекты. Наиболее часто HashTable используется для хранения значений объектов, ключами которых служат строки (то есть объекты типа String).

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.

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.

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 ().

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

containsKey(Object key) — возвращает true, если в таблице есть ключ key;

containsValue(Object value) — возвращает true, если в таблице есть ключи со значением value;

isEmpty() — возвращает true, если в таблице нет элементов;

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

keySet()— предоставляет все ключи key таблицы в виде интерфейса Set. Все изменения в объекте Set корректируют таблицу, и наоборот;

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

toString () — возвращает строку, содержащую все пары;

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

clear() — удаляет все элементы, очищая таблицу.

В очередном примере в HashTable хранится информация об этой книге.

```
import java.util.Dictionary;
```

```
import java.util.Hashtable;
class HTDemo {
public static void main(String args[]) {
Hashtable ht = new Hashtable();
ht.put("title", "The Java Handbook");
ht.put("author", "Patrick Naugnton");
ht.put("email", "naughton@starwave.com");
ht.put("age", new Integer(30));
show(ht);
}
static void show(Dictionary d) {
System.out.println("Title: " + d.get("title"));
System.out.println("Author: " + d.get("author"));
System.out.println("Email: " + d.get("email"));
System.out.println("Age: " + d.get("age"));
} }
```

Результат работы этого примера иллюстрирует тот факт, что метод show, параметром которого является абстрактный тип Dictionary, может извлечь все значения, которые мы занесли в ht внутри метода main.

Title: The Java Handbook
Author: Patrick Naughton
Email: naughton@starwave.com
Age: 30

Класс Properties

Properties — подкласс HashTable, в который для удобства использования добавлено несколько методов, позволяющих получать значения, которые, возможно, не определены в таблице. В методе getProperty вместе с именем можно указывать значение по умолчанию:

```
getProperty("имя","значение_по_умолчанию");
```

При этом, если в таблице свойство “имя” отсутствует, метод вернет “значение_по_умолчанию”. Кроме того, при создании нового объекта этого класса конструктору в качестве параметра можно передать другой объект Properties, при этом его содержимое будет использоваться в качестве значений по умолчанию для свойств нового объекта. Объект Properties в любой момент можно записать либо считать из потока — объекта Stream.

В классе Properties два конструктора:

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

Кроме унаследованных от класса Hashtable 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.
setProperty(String key, String value) — добавляет новую пару, если ключа key нет, и меняет значение, если ключ key есть;
load(InputStream in) — загружает свойства из входного потока in;

Ниже приведен пример, в котором создаются и впоследствии считываются некоторые свойства:

```
import java.util.Properties;
class PropDemo {
static Properties prop = new Properties();
public static void main(String args[]) {
prop.put("Title", "put title here");
prop.put("Author", "put name here");
prop.put("isbn", "isbn not set");
Properties book = new Properties(prop);
book.put("Title", "The Java Handbook");
book.put("Author", "Patrick Naughton");
System.out.println("Title: " +
book.getProperty("Title"));
System.out.println("Author: " +
book.getProperty("Author"));
System.out.println("isbn: " +
book.getProperty("isbn"));
System.out.println("ean: " +
book.getProperty("ean", "???"));
} }
```

Здесь мы создали объект prop класса Properties, содержащий три значения по умолчанию для полей Title, Author и isbn. После этого мы создали еще один объект Properties с именем book, в который мы поместили реальные значения для полей Title и Author. В следующих трех строках примера мы вывели результат, возвращенный методом getProperty для всех трех имеющихся ключей. В четвертом вызове getProperty стоял несуществующий ключ “ean”. Поскольку этот ключ отсутствовал в объекте book и в объекте по умолчанию prop, метод getProperty выдал нам указанное в его вызове значение по умолчанию, то есть “???”:

```
Title: The Java Handbook
Author: Patrick Naughton
isbn: isbn not set
ean: ???
```

Итераторы

В коллекциях широко используются *итераторы* . В Java итератор — это вспомогательный объект, используемый для прохода по коллекции объектов. Как и сами коллекции, итераторы базируются на интерфейсе. Это интерфейс *Iterator*, определенный в пакете java.util. Любой итератор, как бы он не был устроен, имеет следующие три метода:

- boolean hasNext() — проверяет есть ли еще элементы в коллекции

- Object next() — выдает очередной элемент коллекции
- void remove() — удаляет последний выбранный элемент из коллекции.

Ниже показан пример создания списка ***ArrayList*** и использование итератора в обычном цикле со счетчиком, а также цикла for-each для вывода элементов списка на экран.

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;

public class Collect {

      static void testForLoop() throws IOException {
          //создание списка
          ArrayList listA = new ArrayList();

          listA.add("element 1");
          listA.add("element 2");
          listA.add("element 3");
          //вставка первого элемента списка с помощью индекса
          listA.add(0, "element 0");

          //вывод элементов списка с помощью Iterator
            for (Iterator i = listA.iterator(); i.hasNext(); ) {
              Object listElement = i.next();
                 System.out.println(listElement.toString());
                    }
          //вывод элементов списка с помощью цикла for-each
            for (Object b : listA)
                 System.out.println(b);

          }

     public static void main(String[] args) {
          try{
          testForLoop();
          }
          catch (IOException e)
          {
                System.out.println("My exception");
          }
     }
}
```

Задание 1.
Создать вектор, ввести в него пять целых чисел, вывести содержимое стека на экран. Добавить в вектор два числа между вторым и третьим, удалить предпоследнее число и снова вывести содержимое стека на экран в цикле.

Задание 2.
Создать стек, ввести в него четыре любых слова, вывести содержимое стека на экран. Удалить два последних введенных слова и снова вывести содержимое стека на экран.

Задание 3.
Используя класс Hashtable создать телефонный справочник для трех человек по шаблону: **фамилия - номер телефона**. Найти номер телефона по указанной фамилии и вывести его на экран.