ЛАБОРАТОРНАЯ РАБОТА № 23
Разработка консольного приложения с использованием коллекций.

1. Цель работы

Научиться разрабатывать программы на языке Java с использованием коллекций.

2. Методические указания

Лабораторная работа направлена на приобретение навыков использования коллекций на языке Java.

Требования к результатам выполнения лабораторного практикума:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- по завершении выполнения задания составить отчет о проделанной работе.

3. Теоретический материал

Коллекции – это хранилища или контейнеры, поддерживающие различные способы накопления и упорядочения объектов с целью обеспечения возможностей эффективного доступа к ним. Они представляют собой реализацию абстрактных структур данных, поддерживающих три основные операции:

• добавление нового элемента в коллекцию;
• удаление элемента из коллекции;
• изменение элемента в коллекции.

В качестве других операций могут быть реализованы следующие: заменить, просмотреть элементы, подсчитать их количество и др.

При программировании на *Java* операций над группой однотипных объектов важно выбирать наиболее эффективную структуру данных (класс) для хранения этих объектов. В языке *java* определены специальные классы для хранения однотипных объектов, которые называются ***коллекциями***, определяющими такие структуры как список, множество, очередь.

Выбор определенного класса для работы с коллекциями определяет набор методов, которые будут доступны для объекта этого класса. Например, если используется список (который определяет интерфейс *List*), то существует выбор для его реализации: *ArrayList, LinkedList, Vector, Stack.* Конкретный выбор реализации списка сказывается на эффективности манипуляций с объектами списка. Так, *ArrayList* хранит элементы в виде массива, а значит, доступ и замена будет выполняться относительно быстро. В то же время *LinkedList* хранит элементы в виде связного списка, что влечет за собой относительно медленный поиск элементов и быструю операцию добавления/удаления. Рекомендуется ознакомиться с описаниями следующих коллекций по *JSDK* -документации:

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

Ниже приведен краткий разбор наиболее важных классов при работе с коллекциями.

Интерфейс Collection

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

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

clear() – удаляет все элементы коллекции;

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

isEmpty() – проверяет, пуста ли коллекция;

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

size() – возвращает количество элементов коллекции.

Интерфейс List

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

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

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

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

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

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

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

Интерфейс Set

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

Интерфейс Queue

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

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

Методы интерфейса *Queue*:

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

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

Класс Vector

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

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

Класс ArrayList

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

Класс Stack

Stack – коллекция, объединяющая элементы в стек. Стек работает по принципу *LIFO* (последним пришел – первым ушел). Элементы кладутся в стек «друг на друга», причем взять можно только «верхний» элемент, т.е. тот, который был положен в стек последним. Для стека характерны операции, реализованные в следующих методах класса *Stack*:

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

pop() – извлекает из стека верхний элемент;

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

empty() – проверяет, не пуст ли стек;

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

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

Интерфейс Iterator

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

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

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

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

Интерфейс *Iterator* имеет всего три метода:

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

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

remove() – удаляет текущий элемент.

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

```
Iterator iter = coll.iterator(); // coll - коллекция
while (iter.hasNext()) {
// обрабатываем объект, возвращаемый методом iter.next()
}
```

Для коллекций, элементы которых проиндексированы, определен более функциональный итератор, позволяющий двигаться как в прямом, так и в обратном направлении, а также добавлять в коллекцию элементы. Такой итератор имеет интрефейс

ListIterator, унаследованный от интерфейса Iterator и дополняющий его следующими методами:

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

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

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

set(Object item) – заменяет текущий элемент;

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

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

Интерфейс Map

Интерфейс *Map* из пакета *java.util* описывает коллекцию, состоящую из пар «ключ – значение», которые широко используются для хранения настроек в файлах конфигурации (см., например, /etc/services). У каждого ключа только одно значение, что соответствует математическому понятию однозначной функции или отображения (*Map*). Интерфейс *Map* содержит следующие методы, работающие с ключами и значениями:

boolean containsKey (Object key) – проверяет наличие ключа *key*;

boolean containsValue (Object value) – проверяет наличие значения *value*;

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

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

Set key Set() – представляет ключи коллекции в виде множества;

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

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

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

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

Вложенный интерфейс Map.Entry

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

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

Ход работы. Рассмотрим пример: вычислить сколько раз каждая буква встречается в тексте**.**

```
package by.bsac.lab3;

import java.util.HashMap;
import java.util.Map.Entry;

public class Main {
   public static void main(String[] args) {
      String txt = " лабораторная работа ";
      HashMap<Character, Integer> map = new HashMap<Character, Integer>(40);

      for (int i = 0; i < txt.length(); ++i) {
```

```
            char c = txt.charAt(i);
            //проверяем является ли символ буквой
            if (Character.isLetter(c)) {
                if (map.containsKey(c)) {
                    map.put(c, map.get(c) + 1);
                } else {
                    map.put(c, 1);
                }
            }
        }

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

Результат:
буква: т кол - во: 2
буква: я кол - во: 1
буква: а кол - во: 5
буква: б кол - во: 2
буква: л кол - во: 1
буква: н кол - во: 1
буква: о кол - во: 3

Рассмотрим другой пример работы с коллекциями:

```
package by.bsac.lab3;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedHashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.SortedMap;
import java.util.SortedSet;
import java.util.TreeMap;
import java.util.TreeSet;

public class CollectionFrameworkTest {
```

```
public static int arrayLength = 0;

public static void main(String[] args) {
    String[] testArray = null;
    try {
        testArray = initStringArrayFromConsole();
    } catch (IOException | NumberFormatException e) {
        System.out.println("Ошибка при инициализации массива:" + e);
    }
    List<String> list = Arrays.asList(testArray);
    printCollection(list);

    String text = "Пример использования коллекций";
    List<String> yodaStyleList = new ArrayList<String>(Arrays.asList(text.split(" ")));
    Collections.reverse(yodaStyleList);
    printCollection(yodaStyleList);

    List lnkLst = new LinkedList();
    lnkLst.add("element1");
    lnkLst.add("element2");
    lnkLst.add("element3");
    lnkLst.add("element4");
    printCollection(lnkLst);

    List aryLst = new ArrayList();
    aryLst.add("x");
    aryLst.add("y");
    aryLst.add("z");
    aryLst.add("w");
    printCollection(aryLst);

    Set hashSet = new HashSet();
    hashSet.add("set1");
    hashSet.add("set2");
    hashSet.add("set3");
    hashSet.add("set4");
    printCollection(hashSet);

    SortedSet treeSet = new TreeSet();
    treeSet.add("1");
    treeSet.add("2");
    treeSet.add("3");
    treeSet.add("4");
    printCollection(treeSet);

    LinkedHashSet lnkHashset = new LinkedHashSet();
    lnkHashset.add("one");
    lnkHashset.add("two");
    lnkHashset.add("three");
    lnkHashset.add("four");
    printCollection(lnkHashset);
```

```
        Map map1 = new HashMap();
        map1.put("key1", "J");
        map1.put("key2", "K");
        map1.put("key3", "L");
        map1.put("key4", "M");
        printCollection(map1.keySet());
        printCollection(map1.values());

        SortedMap map2 = new TreeMap();
        map2.put("key1", "JJ");
        map2.put("key2", "KK");
        map2.put("key3", "LL");
        map2.put("key4", "MM");
        printCollection(map2.keySet());
        printCollection(map2.values());

        LinkedHashMap map3 = new LinkedHashMap();
        map3.put("key1", "JJJ");
        map3.put("key2", "KKK");
        map3.put("key3", "LLL");
        map3.put("key4", "MMM");
        printCollection(map3.keySet());
        printCollection(map3.values());

        // создаем ArrayList с дублирующимися элементами
        List<String> strArrayList = new ArrayList<String>();
        strArrayList.add("Ваня");
        strArrayList.add("Петя");
        strArrayList.add("Вася");
        strArrayList.add("Ваня"); // дубликат
        strArrayList.add("Аня");
        strArrayList.add("Дима");
        // распечатаем содержимое списка
        System.out.println("распечатаем содержимое списка ");
        printCollection(strArrayList);
        // Теперь удалим дубликаты, сохраняя порядок вставки в список
        // LinkedHashSet гаранитирует сохранение порядка вставки, так как
        // является множеством(set)
        // повторяющиеся элементы будут автоматически удалены
        Set<String> stringCollectionWithoutDuplicates = new
LinkedHashSet<String>(strArrayList);
        // теперь очистим ArrayList для того, чтобы скопировать LinkedHashSet
        strArrayList.clear();
        // копируем элементы, но без дубликатов
        strArrayList.addAll(stringCollectionWithoutDuplicates);
        System.out.println("распечатаем содержимое списка без дубликатов");
        printCollection(strArrayList);

    }

    public static String[] initStringArrayFromConsole() throws IOException,
NumberFormatException {
```

```
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Количество элементов в массиве:");
        int n = Integer.parseInt(in.readLine());
        String[] stingArray = new String[n];
        for (int i = 0; i < n; i++) {
            stingArray[i] = in.readLine();
        }

        return stingArray;
    }

    public static void printCollection(Collection collection) throws IllegalArgumentException {
        if (collection != null) {
            Iterator itr = (Iterator) collection.iterator();
            while (itr.hasNext()) {
                String str = (String) itr.next();
                System.out.print(str + " ");
            }
            System.out.println();
        } else {
            throw new IllegalArgumentException("аргумент не может быть null");
        }
    }

}
```

Результат:

```
Количество элементов в массиве:1
12
12
коллекций использования Пример
element1 element2 element3 element4
x y z w
set3 set2 set4 set1
1 2 3 4
one two three four
key1 key2 key3 key4
J K L M
key1 key2 key3 key4
JJ KK LL MM
key1 key2 key3 key4
JJJ KKK LLL MMM
распечатаем содержимое списка
Ваня Петя Вася Ваня Аня Дима
распечатаем содержимое списка без дубликатов
Ваня Петя Вася Аня Дима
```

4. Порядок выполнения работы

- изучить теоретический материал;
- напишите программы на языке *Java* в соответствии с вашим вариантом.

5. Индивидуальные задания

1. В кругу стоят N человек, пронумерованных от 1 до N. При ведении счета по кругу вычеркивается каждый второй человек, пока не останется один. Составить две программы, моделирующие процесс. Одна из программ должна использовать класс ArrayList, а вторая – LinkedList. Какая из двух программ работает быстрее? Почему?
2. Задан список целых чисел и число X. Не используя вспомогательных объектов и не изменяя размера списка, переставить элементы списка так, чтобы сначала шли числа, не превосходящие X, а затем числа, больше X.
3. Написать программу, осуществляющую сжатие английского текста. Построить для каждого слова в тексте оптимальный префиксный код по алгоритму Хаффмена. Использовать класс PriorityQueue.
4. Реализовать класс Graph, представляющий собой неориентированный граф. В конструкторе класса передается количество вершин в графе. Методы должны поддерживать быстрое добавление и удаление ребер.
5. На базе коллекций реализовать структуру хранения чисел с поддержкой следующих операций:
• добавление/удаление числа;
• поиск числа, наиболее близкого к заданному (т. е. модуль разницы минимален).
6. Реализовать класс, моделирующий работу N-местной автостоянки. Машина подъезжает к определенному месту и едет вправо, пока не встретится свободное место. Класс должен поддерживать методы, обслуживающие приезд и отъезд машины.
7. Во входном файле хранятся две разреженные матрицы – A и B. Построить циклически связанные списки CA и CB, содержащие ненулевые элементы соответственно матриц A и B. Просматривая списки, вычислить: а) сумму S = A + B; б) произведение P = A × B.
8. Во входном файле хранятся наименования некоторых объектов. Построить список C1, элементы которого содержат наименования и шифры данных объектов, причем элементы списка должны быть упорядочены по возрастанию шифров. Затем «сжать» список C1, удаляя дублирующие наименования объектов.
9. Во входном файле расположены два набора положительных чисел; между наборами стоит отрицательное число. Построить два списка C1 и C2, элементы которых содержат соответственно числа 1-го и 2-го набора таким образом, чтобы внутри одного списка числа были упорядочены по возрастанию. Затем объединить списки C1 и C2 в один упорядоченный список, изменяя только значения полей ссылочного типа.
10. Во входном файле хранится информация о системе главных автодорог, связывающих г. Минск с другими городами Беларуси. Используя эту информацию, построить дерево, отображающее систему дорог республики, а затем, продвигаясь по дереву, определить минимальный по длине путь из г. Минска в другой заданный город. Предусмотреть возможность сохранения дерева в виртуальной памяти.
11. Один из способов шифрования данных, называемый «двойным шифрованием», заключается в том, что исходные данные при помощи некоторого преобразования последовательно шифруются на некоторые два ключа – K1 и K2. Разработать и реализовать эффективный алгоритм, позволяющий находить ключи K1 и K2 по исходной строке и ее зашифрованному варианту. Проверить, оказался ли разработанный способ действительно эффективным, протестировав программу для случая, когда оба ключа являются 20-битными (время ее работы не должно превосходить одной минуты).
12. На плоскости задано N точек. Вывести в файл описания всех прямых, которые проходят более чем через одну точку из заданных. Для каждой прямой указать, через сколько точек она проходит. Использовать класс HashMap.
13. На клетчатой бумаге нарисован круг. Вывести в файл описания всех клеток, целиком лежащих внутри круга, в порядке возрастания расстояния от клетки до центра круга. Использовать класс PriorityQueue.
14. На плоскости задано N отрезков. Найти точку пересечения двух отрезков, имеющую минимальную абсциссу. Использовать класс TreeMap.

15. На клетчатом листе бумаги закрашена часть клеток. Выделить все различные фигуры, которые образовались при этом. Фигурой считается набор закрашенных клеток, достижимых друг из друга при движении в четырех направлениях. Две фигуры являются различными, если их нельзя совместить поворотом на угол, кратный 90 градусам, и параллельным переносом. Используйте класс HashSet.
16. Дана матрица из целых чисел. Найти в ней прямоугольную подматрицу, состоящую из максимального количества одинаковых элементов. Использовать класс Stack.
17. Реализовать структуру «черный ящик», хранящую множество чисел и имеющую внутренний счетчик K, изначально равный нулю. Структура должна поддерживать операции добавления числа в множество и возвращение K-го по минимальности числа из множества.
18. На прямой гоночной трассе стоит N автомобилей, для каждого из которых известны начальное положение и скорость. Определить, сколько произойдет обгонов.
19. На прямой гоночной трассе стоит N автомобилей, для каждого из которых известны начальное положение и скорость. Вывести первые K обгонов.
20. Ввести строки из файла, записать в список ArrayList. Выполнить сортировку строк, используя метод sort() из класса Collections.
21. Задана строка, состоящая из символов «(», «)», «[», «]», «{», «}». Проверить правильность расстановки скобок. Использовать стек.
22. Задан файл с текстом на английском языке. Выделить все различные слова. Слова, отличающиеся только регистром букв, считать одинаковыми. Использовать класс HashSet.
23. Задан файл с текстом на английском языке. Выделить все различные слова. Для каждого слова подсчитать частоту его встречаемости. Слова, отличающиеся регистром букв, считать различными. Использовать класс HashMap.
24. Ввести строки из файла, записать в список. Вывести строки в файл в обратном порядке.
25. Сложить два многочлена заданной степени, если коэффициенты многочленов хранятся в объекте HashMap.
26. С использованием множества выполнить попарное суммирование произвольного конечного ряда чисел по следующим правилам: на первом этапе суммируются попарно рядом стоящие числа, на втором этапе суммируются результаты первого этапа и т. д. до тех пор, пока не останется одно число.
27. Не используя вспомогательных объектов, переставить отрицательные элементы данного списка в конец, а положительные – в начало списка.
28. Списки (стеки, очереди) I(1..n) и U(1..n) содержат результаты n-измерений тока и напряжения на неизвестном сопротивлении R. Найти приближенное число R методом наименьших квадратов.
29. Создать стек из номеров записи. Организовать прямой доступ к элементам записи.
30. Задать два стека, поменять информацию местами.

6. Контрольные вопросы

1. Что такое коллекции? Перечислите типы коллекции.
2. Чем отличается ArrayList от LinkedList? Приведите достоинства и недостатки каждой из них.
3. Как устроена HashMap?
4. Роль equals и hashCode?