**Лабораторная работа № 7.**

**Коллекции языка Java.**

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

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

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

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

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

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

**Интерфейс Collection**

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

* **add(Object item)** –добавляет в коллекцию новый элемент,если элементыколлекции каким-то образом упорядочены, новый элемент добавляется в конец коллекции;
* **clear() –** удаляет все элементы коллекции;
* **contains(Object obj)** –возвращаетtrue,если объектobjсодержится в кол-лекции и false, если нет;
* **isEmpty()** –проверяет,пуста ли коллекция;
* **remove(Object obj)** –удаляет из коллекции элементobj,возвращаетfalse,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 элементitemв позициюindex,приэтом список раздвигается (все элементы, начиная с позиции index, увеличивают свой индекс на 1);
* **get(int index)** –возвращает объект,находящийся в позицииindex; **indexOf(Object obj)** –возвращает индекс первого появления элементаobjв списке;
* **lastIndexOf(Object obj)** –возвращает индекс последнего появления элемен-та obj в списке;
* **add(int index, Object item)** –заменяет элемент,находящийся в позицииindex объектом item;
* **subList(int from, int to)** –возвращает новый список,представляющий собойчасть 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 оче-редь только с одного конца, а извлекаться с другого (аналогично очереди в ма-газине). Интерфейс Queue так же унаследован от интерфейса Collection. Спе-цифические для очереди методы:

* **poll()** –возвращает первый элемент и удаляет его из очереди;
* **peek()** –возвращает первый элемент очереди,не удаляя его;
* **offer(Object obj)** –добавляет в конец очереди новый элемент и возвращаетtrue, если вставка удалась.

**Класс Vector**

**Vector (вектор)** –набор упорядоченных элементов,к каждому из которыхможно 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 элементимеет позицию 1, находящийся под ним – 2 и т.д. Если объекта в стеке нет, возвращает –1.

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

**Интерфейс Iterator**

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

for (int i = 1; i < array.length; i++){

обрабатываем элемент array[i]

}

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

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

**Интерфейс Iterator** имеет всего три метода:

* **next()** –возвращает очередной элемент коллекции,к которой«привязан»итератор (и делает его текущим). Порядок перебора определяет сам итератор;
* **hasNext()** –возвращаетtrue,если перебор элементов еще не закончен;
* **remove()** –удаляет текущий элемент.

**Интерфейс Collection** помимо рассмотренных ранее методов,имеет методiterator(), 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() –** возвращаетtrue,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 следующегои предыдущего элементов соответственно.

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

**Интерфейс Map**

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

* **boolean containsKey (Object key)** — проверяет наличие ключа key;
* **boolean containsValue (Object value)** — проверяет наличие значения value;
* **Set entrySet()** – представляет коллекцию в виде множества, каждый элемент которого – пара из данного отображения, с которой можно работать методами вложенного интерфейса Map.Entry;
* **Оbject 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 та-кой 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 Mар 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 (Оbject value) меняет значение в данной паре.

**Пример решения одного из заданий на лабораторную работу**

Задание: Вычислить сколько раз каждая буква встречается в тексте.

Решение:

import java.util.HashMap;

import java.util.\*;

public class Main {

public static void main(String[] args) {

String txt = "лабораторная работа";

HashMap<Character, Integer> map = new HashMap<Character, Integer >(40);

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

char c = txt.charAt(i);

//проверяем является ли символ буквой

if (Character.isLetter(c)) {

if (map.containsKey(c)) {

map.put(c, map.get(c) + 1);

} else {

map.put(c, 1);

}

}

}

//вывод на экран букв и с частотой их появления

for (Entry<Character, Integer> entry : map.entrySet()){

System.out.println("буква: "+entry.getKey()+" кол-во:"+entry.getValue());

}

}

}

**Варианты заданий**

1. Ввести строки из файла, записать их в стек. Вывести строки в файл в обратном порядке.
2. Ввести число, занести его цифры в стек. Вывести в число, у которого цифры идут в обратном порядке.
3. Сложить два многочлена заданной степени, если коэффициенты многочленов хранятся в объекте HashMap.
4. Создать стек из элементов каталога.
5. Не используя вспомогательных объектов, переставить отрицательные элементы данного списка в конец, а положительные - в начало этого списка.
6. Организовать вычисления в виде стека.
7. Выполнить попарное суммирование произвольного конечного ряда чисел следующим образом: на первом этапе суммируются попарно рядом стоящие числа, на втором этапе суммируются результаты первого этапа и т.д. до тех пор, пока не останется одно число.
8. Задать два стека, поменять информацию местами.
9. Определить класс Stack. Объявить объект класса. Ввести последовательность символов и вывести ее в обратном порядке.
10. Умножить два многочлена заданной степени, если коэффициенты многочленов хранятся в списках.
11. Определить класс Set на основе множества целых чисел, n = размер. Создать методы для определения пересечения и объединения множеств.
12. Программа получает N параметров вызова (аргументы командной строки). Эти параметры – элементы вектора. Строится массив типа double, а на базе этого массива – объект класса DoubleVector. Далее программа выводит в консоль значения элементов вектора в виде: Вектор: 2.3 5.0 7.3.
13. Списки (стеки) I(1..N) и U(1..N) содержат результаты N измерений тока и напряжения на неизвестном сопротивлении R. Найти приближённое число R методом наименьших квадратов.