Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Севастопольский государственный университет»

ИССЛЕДОВАНИЕ КОЛЛЕКЦИЙ И ИТЕРАТОРОВ В ЯЗЫКЕ JAVA

Методические указания
к выполнению лабораторной работы
для студентов, обучающихся по направлению
09.03.02 “Информационные системы и технологии”
очной и заочной форм обучения

Севастополь
2015

УДК 004.42 (075.8)

Исследование коллекций и итераторов в языке Java: методические указания к лабораторной работе №3 по дисциплине “Платформа Java” для студентов направления 09.03.02 “Информационные системы и технологии”/ Сост. **С.А. Кузнецов, А.Л. Овчинников —** Севастополь: Изд-во СевГУ, 2015. — 19 с.

Цель указаний: оказание помощи студентам направления 09.03.02 “Информационные системы и технологии” при выполнении лабораторной работы №3 по дисциплине “Платформа Java”.

Методические указания составлены в соответствии с требованиями программы дисциплины «Платформа Java» для студентов дневной и заочной формы обучения направления 09.03.02 “Информационные системы и технологии” и утверждены на заседании кафедры «Информационные системы» протоколом № 1 от 31 августа 2015 года.

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Рецензент: Кожаев Е.А., канд. техн. наук, доцент кафедры кибернетики и вычислительной техники.

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

В ходе выполнения данной лабораторной работы необходимо ознакомиться с организацией коллекций объектов на языке Java, приобрести практические навыки использования списков, очередей, хеш-таблиц при создании Java программ.

2. ПОСТАНОВКА ЗАДАЧИ

2.1. В соответствии с вариантом задания(см. таблицу 4.1) реализовать класс для представления требуемой информации.

2.2. Реализовать коллекцию типа T1(см. таблицу 4.1) объектов разработанного в п. 2.1. класса с возможностью ввода элементов из файла, вывода на консоль, проверки членства по введенному с консоли значению поля 1. Имя файла вводить параметром командной строки –i.

2.3. Реализовать коллекцию типа LinkedList объектов разработанного в п. 2.1. с возможностью: упорядочивания по полю 1 (использовать Collections.sort(list)); с возможностью упорядочивания по полю P(см. таблицу 4.1) в направлении U класса (использовать Collections.sort(list, myComp), где myComp – экземпляр разработанного класса, реализующего интерфейс Comparator); с возможностью ввода элементов из файла, вывода на консоль и сохранения в файл. Имена файлов вводить параметрами командной строки –i и –o.

2.4. Реализовать коллекцию типа T2(см. таблицу 4.1) объектов разработанного в п. 2.1. класса с ключом по значению поля 1, с возможностью ввода элементов из файла, вывода на консоль в виде «Ключ -> Значения» (значения остальных полей), вывода значения полей по введенному с консоли значению поля 1. Имя файла вводить параметром командной строки –i.

2.5. Реализовать класс Lab3Java, в методе main которого реализовать работу с объектами классов из п. 2.1-2.4:

1. Ввести записи из файла заданного параметром командной строки –i в коллекцию T1.

2. Отобразить записи в консоли.

3. Предложить пользователю ввести значение поля 1.

4. Отобразить в консоли результат проверки наличия записи по введенному значению поля 1.

5. Ввести записи из файла заданного параметром командной строки –i в коллекцию LinkedList.

6. Отобразить записи в консоли. Отсортировать по полю 1. Отобразить записи в консоли. Отсортировать по полю P в направлении U. Отобразить записи в консоли.7. Вывести записи в файл, заданный параметром командной строки –o.

8. Ввести записи из файла заданного параметром командной строки –i в коллекцию T2.

9. Отобразить записи в консоли.

10. Предложить пользователю ввести значение поля 1.

11. Отобразить в консоли значения остальных полей по введенному значению поля 1.

3. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

3.1 Коллекции в Java

Коллекции объектов в Java реализованы различными классами пакета java.util. Коллекции обладают одним важным свойством — в отличии от массивов их размер не ограничен. Выделение необходимых для коллекции ресурсов реализовано внутри соответствующего класса.

Интерфейсы коллекций

В Java коллекции объектов разбиты на (см. рисунок 1): Set (множество), List (список) Queue (очередь) и Deque(дек). Интерфейсы Map (отображение) и SortedMap (упорядоченное отображение) не относятся непосредственно к коллекциям, но входят в Java Collections Framework:

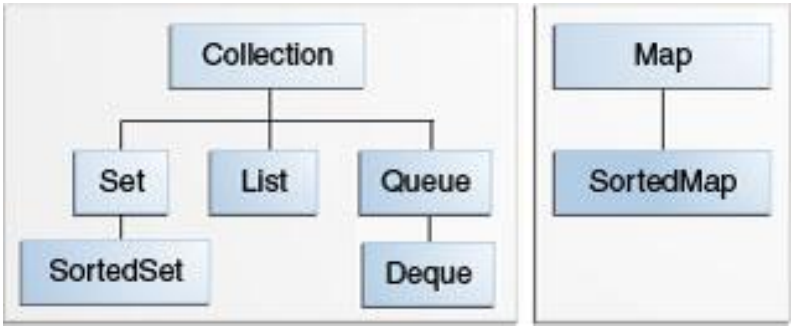

Рисунок 1 - Ключевые интерфейсы коллекций

- Интерфейс Collection – корневой элемент иерархии коллекций. Коллекция представляет группу объектов, называемых элементами коллекции.
- Интерфейс Set описывает множество. Элементы множества не упорядочены, множество не может содержать двух одинаковых элементов.
- Интерфейс List описывает упорядоченный список. Элементы списка пронумерованы, начиная с нуля, и к конкретному элементу можно обратиться с использованием целочисленного индекса.
- Интерфейс Queue описывает очередь. Элементы могут добавляться в очередь только с одного конца, а извлекаться с другого.
- Интерфейс Deque описывает двухстороннюю очередь - линейную структуру данных, которая поддерживает вставку и удаление элементов на обоих концах. Интерфейс Deque позволяет реализовывать и стеки и очереди.
- Интерфейс Map - отображение или ассоциативный массив - абстрактный тип данных, позволяющий хранить пары вида «(ключ, значение)» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу. Добавление пары с уже существующим в Map ключом

приводит к замене, а не к добавлению. Из отображения (Map) можно получить множество (Set) ключей и список (List) значений.

- SortedSet – это упорядоченное множество объектов.
- SortedMap – ассоциативный массив, с упорядоченными ключами.

В интерфейсе Collection<E> определены методы, которые работают для всех коллекций (<E> обозначает тип сохраняемых объектов):

boolean add(E obj) – добавляет obj к вызывающей коллекции и возвращает true, если объект добавлен, и false, если obj уже элемент коллекции;

boolean addAll(Collection<? extends E> c) – добавляет все элементы коллекции к вызывающей коллекции;

void clear() – удаляет все элементы из коллекции;

boolean contains(Object obj) – возвращает true, если вызывающая коллекция содержит элемент obj;

boolean equals(Object obj) – возвращает true, если коллекции эквивалентны;

boolean isEmpty() – возвращает true, если коллекция пуста;

Iterator<E> iterator() – извлекает итератор(см. ниже);

boolean remove(Object obj) – удаляет obj из коллекции;

int size() – возвращает количество элементов в коллекции;

Object[] toArray() – копирует элементы коллекции в массив объектов;

<T> T[] toArray(T a[]) – копирует элементы коллекции в массив объектов определенного типа.

Интерфейс Set не содержит новых методов, но предполагает реализацию метода add() таким образом, чтобы не допустить добавление в множество элемента, который в нем уже содержится.

Методы интерфейса List<E> позволяют вставлять и удалять элементы из позиций, указываемых через отсчитываемый от нуля индекс:

void add(int index, E element) – вставляет element в позицию, указанную в index;

void addAll(int index, Collection<? extends E> c) – вставляет в вызывающий список все элементы коллекции c, начиная с позиции index;

E get(int index) – возвращает элемент в виде объекта из позиции index;

int indexOf(Object ob)– возвращает индекс указанного объекта;

E remove(int index) – удаляет объект из позиции index;

E set(int index, E element) – заменяет объект в позиции index, возвращает при этом удаляемый элемент;

List<E> subList(int fromIndex, int toIndex) – извлекает часть коллекции в указанных границах.

Интерфейс Queue<E> дополняет родителя методами:

E poll() - возвращает первый элемент и удаляет его из очереди. Если очередь пуста, возвращает null;

E peek() - возвращает первый элемент очереди, не удаляя его. Если очередь пуста, возвращает null;

boolean offer(E e) - добавляет в конец очереди новый элемент и возвращает true, если вставка удалась.

E element() - возвращает первый элемент очереди, не удаляя его. В отличии от *peek*, если очередь пуста генерирует NoSuchElementException;

Специфические методы интерфейса Deque представлены в таблице 1

Таблица 1 – Методы интерфейса Deque

Тип операции	**Первый элемент**	**Последний элемент**
Вставка	addFirst(e) offerFirst(e)	addLast(e) offerLast(e)
Получение, удаление	removeFirst() pollFirst()	removeLast() pollLast()
Получение	getFirst() peekFirst()	getLast() peekLast()

Интерфейс Map<K,V> содержит следующие методы, работающие с ключами и значениями:

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

boolean containsValue (Object value) - проверяет наличие значения value;

public Set<Map.Entry<K,V>> entrySet() – представляет коллекцию в виде множества, каждый элемент которого – пара из данного отображения, с которой можно работать методами вложенного интерфейса Map.Entry(см. ниже);

V get(Object key) – возвращает значение, отвечающее ключу key; S et key S et() – представляет ключи коллекции в виде множества;

V put(K key, V 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<? extends K, ? extends V> m) – добавляет к коллекции все пары из отображения m;

public Collection<V> values() – представляет все значения в виде коллекции.

Как и интерфейс Collection интерфейс Map также содержит аналогичные методы size и isEmpty.

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

Интерфейс SortedSet<E> определяется следующим образом:

```
public interface SortedSet<E> extends Set<E> {
   // Range-view
   SortedSet<E> subSet(E fromElement, E toElement);
```

```
        SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);

    // Endpoints
    E first();
    E last();

    // Comparator access
    Comparator<? super E> comparator();
}
```

Как видно, интерфейс предусматривает возможность выборки подмножеств, а также содержит методы доступа к первому (*first()*) и последнему (*last()*) элементам множества, и предоставляет возможность определения отношение порядка между объектами при помощи специального класса, наследующего интерфейс Comparator.

Аналогично определен и интерфейс SortedMap:

```
public interface SortedMap<K, V> extends Map<K, V>{
    Comparator<? super K> comparator();
    SortedMap<K, V> subMap(K fromKey, K toKey);
    SortedMap<K, V> headMap(K toKey);
    SortedMap<K, V> tailMap(K fromKey);
    K firstKey();
    K lastKey();
}
```

Интерфейс Comparable

Интерфейс Comparable — это обобщенный интерфейс, объявленный следующим образом:

interface Comparable<T>

Интерфейс Comparable объявляет один метод, который используется для определения того, что в языке Java называется натуральным порядком экземпляров класса. Сигнатура метода показана ниже:

int compareTo(T объект)

Этот метод сравнивает вызывающий объект с указанным параметром объект. Возвращает значение 0, если значения эквивалентны. Отрицательное значение возвращается, если вызывающий объект имеет меньшее значение. В противном случае возвращается положительное значение.

Этот интерфейс реализован в классах Byte, Character, Double, Float, Long, Short, String и Integer. Все перечисленные классы определяют метод compareTo().

Так, например, классы, реализующие интерфейс Comparable самостоятельно, имеют возможность влиять на порядок их сортировки при их использовании в качестве элементов коллекций, основанных на интерфейсах *SortedSet* и *SortedMap*(например, TreeSet и TreeMap – см. ниже), а также при выполнении сортировки списков и массивов при вызове метода sort класса Collections.

Классы, реализующие коллекции

Основные классы, реализующие интерфейсы коллекций представлены в таблице 2:

Таблица 2 – Основные классы, реализующие интерфейсы коллекций

Интерфейсы	**Реализации отображений**	**Реализации расширяемых массивов**	**Реализации деревьев**	**Реализации связных списков**	**Реализации отображений + связных списков**
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Как видно из таблицы, существует две универсальные реализации интерфейса List – это классы ArrayList и LinkedList.

Класс ArrayList расширяет AbstractList и реализует интерфейс List. Основное назначение – создание расширяемых (динамических) массивов. В следующем примере выполняется создание расширяемого массива объектов:

```
import java.util.*;
public class DemoGeneric {
public static void main(String args[]) {
ArrayList<String> list = new ArrayList<String>();
list.add("Java");
list.add("ArrayList ");
String res = list.get(0);/* компилятор “знает” тип значения- приведение не требуется */
// list.add(new StringBuilder("C#")); // ошибка компиляции
// компилятор не позволит добавить “посторонний” тип
System.out.print(list);// будет выведено: [Java, ArrayList]
}
}
```

Класс ArrayList может быть использован и без указания типа объектов, в этом случае будет позволено добавлять объекты всех типов, но в этом случае может понадобиться приведение типов при работе с элементами коллекции:

```
import java.util.*;
public class UncheckCheck {
        public static void main(String args[]) {
                ArrayList list = new ArrayList();
                list.add(71);
```

```
        list.add(new Boolean("TruE"));
            list.add("Java 1.6.0");
            // требуется приведение типов
            int i = (Integer)list.get(0);
            boolean b = (Boolean)list.get(1);
            String str = (String)list.get(2);
            ArrayList<Integer> s = new ArrayList<Integer>();
            s.add(71);
            s.add(92);
            // s.add("101");// ошибка компиляции: s параметризован
    }
}
```

Класс LinkedList расширяет AbstractSequentialList и реализует интерфейсы List, Deqeue (и Queue). Он представляет структуру данных связного списка.

В следующей программе иллюстрируется использование класса LinkedList.

```
// Демонстрация применения LinkedList.
import java.util.*;
class LinkedListDemo {
    public static void main(String args[]) {
        // Создать связный список.
        LinkedList<String> ll = new LinkedList<String>();
        // Добавить элементы в связный список.
        ll.add("F");
        ll.add("B");
        ll.add("D");
        ll.add("E");
        ll.add("C");
        ll.addLast("Z");
        ll.addFirst("A");
        ll.add(1, "A2");
        System.out.println("Исходное содержимое ll: " + ll);// [A, A2 , F, B, D , E, C, Z]
        // Удалить элементы из связного списка.
        ll.remove("F ");
        ll.remove(2);
        System.out.println("Содержимое ll после удаления: " + ll); //[A, A2 , D, E, C,
Z]
        // Удалить первый и последний элементы.
        ll.removeFirst();
        ll.removeLast();
        System.out.println("ll после удаления первого и последнего: " +ll ) ;// [A2 ,
D, E, C]
        // Получить и присвоить значение.
        String val = ll.get(2);
        ll.set(2, val + " Изменен");
        System.out.println("ll после изменения: " + ll);// [A2,D, E Изменен, C]
```

```
    }
}
```

Класс HashSet расширяет AbstractSet и реализует интерфейс Set. Он создает коллекцию, которая использует для хранения *хэш-таблицу*. Хеш-таблица хранит информацию, используя так называемый механизм *хеширования*, в котором содержимое ключа используется для определения уникального значения, называемого хеш-кодом. Этот хеш-код затем применяется в качестве индекса, с которым ассоциируются данные, доступные по этому ключу. Преобразование ключа в хеш-код выполняется автоматически. Выгода от хеширования состоит в том, что оно обеспечивает константное время выполнения методов add(), contains(), remove() и size (), даже для больших наборов данных.

Важно отметить, что класс HashSet не гарантирует упорядоченности элементов, поскольку процесс хеширования сам по себе обычно не порождает сортированных наборов:

```
// Demonstrate HashSet.
import java.util.*;
class HashSetDemo {
  public static void main(String args[]) {
    // Create a hash set.
    HashSet<String> hs = new HashSet<String>();

    // Add elements to the hash set.
    hs.add("B");
    hs.add("A");
    hs.add("D");
    hs.add("E");
    hs.add("C");
    hs.add("F");
    System.out.println(hs);// [D, E , F , A, B, C] – порядок хранения не соответствует порядку добавления
  }
}
```

Класс LinkedHashSet расширяет класс HashSet, не добавляя никаких новых методов. Класс LinkedHashSet поддерживает связный список элементов набора в том порядке, в котором они вставлялись. Это позволяет организовать упорядоченную итерацию вставки в набор. То есть, когда идет перебор объекта класса LinkedHashSet с применением итератора(см. ниже), элементы извлекаются в том же порядке, в каком они были вставлены. Это также тот порядок, в котором они будут возвращены методом toString() объекта класса LinkedHashSet. Таким образом, если в предыдущем примере использовать LinkedHashSet вместо класса HashSet, вывод программы будет выглядеть так: [B, A, D , E, C, F].

Класс TreeSet расширяет класс AbstractSet и реализует интерфейс NavigableSet. Он создает коллекцию, которая для хранения элементов применяет дерево. Объекты сохраняются в отсортированном порядке по возрастанию.

Время доступа и извлечения элементов достаточно мало, что делает класс TreeSet отличным выбором для хранения больших объемов отсортированной информации, с быстрым доступом:

```
import java.util.*;
class TreeSetDemo {
        public static void main(String args[]) {
                // Создать TreeSet.
                TreeSet<String> ts = new TreeSet<String>();
                // Добавить элементы в TreeSet.
                ts.add("C") ;
                ts.add("A");
                ts.add("B");
                ts.add("E");
                ts.add("F ");
                ts.add("D");
                System.out.println(ts);// [A, B , C, D , E, F]
        }
}
```

Класс HashMap расширяет AbstractMap и реализует интерфейс Map. Он использует хеш-таблицу для хранения отображения (ассоциативного массива). Это позволяет обеспечить константное время выполнения методов get() и put() даже при больших наборах.

Как и класс HashSet класс HashMap не гарантирует порядка элементов. Таким образом, порядок, в котором элементы добавляются к хеш-карте, не обязательно соответствует порядку, в котором они читаются итератором.

В следующей программе иллюстрируется применение класса HashMap. Она соотносит имена вкладчиков с балансовыми счетами:

```
import java.util.*;
class HashMapDemo {
  public static void main(String args[]) {
    // Создание hash map.
    HashMap<String, Double> hm = new HashMap<String, Double>();
    // Добавление элементов
    hm.put("John Doe", new Double(3434.34));
    hm.put("Tom Smith", new Double(123.22));
    hm.put("Jane Baker", new Double(1378.00));
    hm.put("Tod Hall", new Double(99.22));
    hm.put("Ralph Smith", new Double(-19.08));
    // Получение множества значений.
    Set<Map.Entry<String, Double>> set = hm.entrySet();
```

```
    // Вывод элементов множества.
    for(Map.Entry<String, Double> me : set) {
      System.out.print(me.getKey() + ": ");
      System.out.println(me.getValue());
    }
    System.out.println();

    // Deposit 1000 into John Doe's account.
    double balance = hm.get("John Doe");
    hm.put("John Doe", balance + 1000);
    System.out.println("John Doe's new balance: " + hm.get("John Doe"));
  }
}
```

Класс TreeMap расширяет класс AbstractMap и реализует интерфейс NavigableMap. Он создает отображение, размещенное в древовидной структуре. Класс TreeMap предлагает эффективный способ хранения пар “ключ-значение” в отсортированном порядке и обеспечивает быстрое извлечение. Следует отметить, что, в отличие от HashMap класс TreeMap гарантирует, что его элементы будут отсортированы в порядке возрастания ключей.

Класс LinkedHashMap расширяет класс HashMap. Он создает связный список элементов отображения, расположенных в том порядке, в котором они вставлялись. Это позволяет организовать перебор элементов в порядке вставки. То есть, когда происходит итерация по коллекционному представлению объекта класса LinkedHashMap, элементы будут возвращаться в том порядке, в котором они вставлялись. Также возможно создать объект класса LinkedHashMap, возвращающий свои элементы в том порядке, в котором к ним в последний раз осуществлялся доступ (для этого используется параметр конструктора accessOrder).

Компараторы

Как было отмечено выше, классы TreeSet и TreeMap сохраняют элементы в отсортированном порядке. Однако понятие “порядок сортировки” точно определяет применяемый ими *компаратор*.

По умолчанию эти классы сохраняют элементы, используя то, что в Java называется “естественным порядком”(A перед B, 1 перед 2 и т.д.).

Если необходимо упорядочить элементы этих колллекций иным образом, то требуется указать объект интерфейса Comparator при создании набора или отображения. Это позволяет управлять порядком следования элементов в отсортированных коллекциях.

Интерфейс Comparator — это обобщенный интерфейс со следующим объявлением:

interface Comparator<T>

Интерфейс Comparator определяет два метода — compare () и equals ().

Метод compare (), представленный ниже, сравнивает два элемента по порядку:

int compare(T объект1, T объект2)

Здесь *объект1* и *объект2* — это объекты, которые нужно сравнить. Обычно этот метод возвращает значение нуль, если объекты эквивалентны; положительное значение, если *объект1* больше, чем *объект2*; в противном случае возвращается отрицательное значение.

Метод equals() проверяет объект на эквивалентность вызывающему компаратору. Метод возвращает значение true, если объект и вызывающий объект представляют собой объекты интерфейса Comparator и используют одинаковый способ упорядочения. В противном случае он возвращает значение false. Переопределение метода equals() не требуется, и большинство простых компараторов в этом не нуждается.

Реализуя метод compare(), можно изменить порядок объектов. Например, чтобы сортировать в обратном порядке, можно создать компаратор, который возвращает обратные значения при сравнении:

```
// Использование настраиваемого компаратора,
import java.util.*;
// Обратный компаратор для строк,
class MyComp implements Comparator<String> {
    public int compare(String a, String b) {
        String aStr, bStr;
        aStr = a;
        bStr = b;
        // Обратное сравнение,
        return bStr.compareTo(aStr);
    }
        // Нет необходимости переопределять equals().
}

class CompDemo {
    public static void main(String args[]) {
        // Создать TreeSet.
        TreeSet<String> ts = new TreeSet<String>(new MyComp());
        // Добавить элементы в TreeSet.
        ts.add("C");
        ts.add("A");
        ts.add("B");
        ts.add("E");
        ts.add("F");
        ts.add("D");
        // Отобразить элементы,
        for(String element : ts)
            System.out.print(element + " ");
        System.out.println();
    } // выведет F E D C B A
}
```

Компаратор также может быть использован для определения порядка сортировки элементов списков и массивов при вызове статического метода sort класса Collections:

```
//Обратный компаратор для строк,
class MyComp implements Comparator<String> {
    public int compare(String a, String b) {
        String aStr, bStr;
        aStr = a;
        bStr = b;
        //Обратное сравнение,
        return bStr.compareTo(aStr);
    }
    //Нет необходимости переопределять equals().
}

class ComporatorTest {
    public static void main(String args[]) {
        //Создать LinkedList
        LinkedList<String> ts = new LinkedList<String>();
        //Добавить элементы в TreeSet.
        ts.add("C");
        ts.add("A");
        ts.add("B");
        ts.add("E");
        ts.add("F");
        ts.add("D");
        //Отобразить элементы,
        for(String element : ts)
            System.out.print(element + " ");
        //выведет C A B E F D
        System.out.println();

        //Сортировать с использованием компаратора MyComp
        Collections.sort(ts, new MyComp());
        //Отобразить элементы,
        for(String element : ts)
            System.out.print(element + " ");
        System.out.println();
        // выведет F E D C B A
    }
}
```

3.2 Обход коллекции

Существует два способа обойти коллекцию: используя конструкцию for-each и используя итераторы (Iterators).

Конструкция for-each

Конструкция for-each позволяет обойти коллекцию или массив, используя специально предназначенный для этого вариант цикла for. Следующий фрагмент кода печатает каждый элемент коллекции:

```
for (Object o : collection)
   System.out.println(o);
```

Итераторы

Iterator – это объект, который позволяет обходить коллекцию и если необходимо удалять элементы из коллекции. Получить объект, реализующий интерфейс Iterator можно с помощью метода iterator() на объекте Collection. Интерфейс Iterator реализован следующим образом.

```
public interface Iterator<E> {
   boolean hasNext();
   E next();
   void remove(); //optional
}
```

Каждый класс коллекций предлагает метод iterator(), который возвращает итератор на начало коллекции. Используя объект итератора, возможно получить доступ к каждому элементу коллекции — одному за другим. В общем случае, применение итератора для перебора содержимого коллекции сводится к выполнению следующих действий:

1. Установить итератор на начало коллекции, получив его от метода iterator() коллекции.

2. Организовать цикл, вызывающий метод hasNext (). Выполнять перебор дотех пор, пока метод hasNext () возвращает значение true.

3. Внутри цикла получать каждый элемент, вызывая метод next().

Использование итераторов предпочтительнее в случае когда:

- необходимо удалить текущий элемент коллекции. Конструкция for-each скрывает использование итератора, поэтому метод remove недоступен. Поэтому for-each не подходит для задач фильтрации коллекции.
- Параллельная итерация нескольких коллекций.

Следующий фрагмент показывает, как можно отфильтровать коллекцию по некоторому условию.

```
static void filter(Collection<?> c) {
   for (Iterator<?> it = c.iterator(); it.hasNext(); )
      if (!cond(it.next()))
         it.remove();
}
```

Для коллекций, реализующих интерфейс List, также можно получить итератор, вызывая метод listIterator(). Итератор списка (реализует интерфейс ListIterator) обеспечивает доступ к элементам коллекции, как в прямом, так и обратном направлении, а также позволяет модифицировать элементы. Во всем остальном интерфейс ListIterator применяется так же, как интерфейс Iterator.

В следующем примере выполняются все перечисленные действия с демонстрацией обоих интерфейсов — Iterator и Listlterator. Здесь используется объект класса ArrayList, но общие принципы применимы к коллекциям любого типа.

Конечно, интерфейс Listlterator доступен только тем коллекциям, которые реализуют интерфейс List.

```
// Демонстрация применения итераторов,
import java.util.*;
class IteratorDemo {
    public static void main(String args[]) {
    // Создать массив-список.
    ArrayList<String> al = new ArrayList<String>();
    // Добавить элементы в массив-список.
    al.add("C");
    al.add("A");
    al.add("E");
    al.add("B");
    al.add("D");
    al.add("F");
    // Использовать итераторы для отображения содержимого al.
    System.out.print("Исходное содержимое al: ");
    Iterator<String> itr = al.iterator();
    while(itr.hasNext()) {
        String element = itr.nextO;
        System.out.print(element + " ");
    }
    System.out.println() ;
    // Модифицировать текущий объект итерации.
    ListIterator<String> litr = al.listIterator();
    while(litr.hasNext()) {
        String element = litr.nextO;
        litr.set(element + “+”);
    }
    System.out.print("Модифицированное содержимое al: ");
    itr = al.iterator();
    while(itr.hasNext()) {
        String element = itr.next();
        System.out.print(element + " ");
    }
    System.out.println();
```

```
        // Теперь отображаем список в обратном порядке.
        System.out.print("Модифицированный список в обратном порядке: ");
        while(litr.hasPrevious()) {
            String element = litr.previous();
            System.out.print(element + " ");
        }
        System.out.println();
        }
}
```

4. ВАРИАНТЫ ЗАДАНИЙ

Таблица 4.1 Варианты заданий

№	Тип информации (см. ниже)	Поле для сортировки (P)	Направление (U)	Тип коллекции (T1)	Тип коллекции (T2)
1	A	1	Убывание	HashSet	HashMap
2	B	2	Возрастание	TreeSet	TreeMap
3	C	3	Убывание	LinkedHashSet	LinkedHashMap
4	D	4	Возрастание	HashSet	HashMap
5	E	1	Убывание	TreeSet	TreeMap
6	A	2	Возрастание	LinkedHashSet	LinkedHashMap
7	B	3	Убывание	HashSet	HashMap
8	C	4	Возрастание	TreeSet	TreeMap
9	D	1	Убывание	LinkedHashSet	LinkedHashMap
10	E	2	Возрастание	HashSet	HashMap
11	A	3	Убывание	TreeSet	TreeMap
12	B	4	Возрастание	LinkedHashSet	LinkedHashMap
13	C	1	Убывание	HashSet	HashMap
14	D	2	Возрастание	TreeSet	TreeMap
15	E	3	Убывание	LinkedHashSet	LinkedHashMap

Тип информации:

A: Книга(Автор, Год издания, Количество страниц, Издательство);

B: Автомобиль(Марка, Год выпуска, Объем двигателя, Максимальная скорость);

C: Компакт диск(Название альбома, Исполнитель , Количество треков, Длительность звучания);

D: Ноутбук(Идентификатор модели, Производитель процессора, Тактовая частота процессора, Объем ОЗУ);

E: Смартфон(Модель, Размер экрана, Тип экрана, Объем встроенной флэш-памяти).

5. СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать:

Титульный лист, цель работы, постановку задачи, вариант задания, текст программы с комментариями, скриншоты выполнения и описание тестовых примеров, выводы по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие интерфейсы для реализации коллекций объектов существуют в языке Java?
2. В чем особенность интерфейса Deque?
3. Какие методы определены в интерфейсе Collection?
4. Для чего используется интерфейс Set?
5. Какие методы определены в интерфейсе List?
6. Какими методами дополняет родителя интерфейс Queue?
7. Для чего используется интерфейс Map?
8. Что позволяет реализовать интерфейс Comparable?
9. Перечислите классы, реализующие интерфейсы коллекций.
10. Приведите пример использования класса TreeSet?
11. Каким образом возможно выполнить перебор элементов коллекции**?**

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ноутон, П. Java™ 2 [Текст] : пер. с англ. / П. Ноутон, Г. Шилдт. - СПб. : БХВ – Петербург, 2007. - 1050 с.

2. Шилдт, Г. Искусство программирования на Java [Текст] : пер. с англ. / Г. Шилдт, Д. Холмс. - М. ; СПб. ; К. : Вильямс, 2005. - 334 с

3. Хабибуллин, И. Ш. Java 2 [Текст] : самоучитель / И. Ш. Хабибуллин. - СПб. : БХВ - Петербург, 2005. - 720 с.

Заказ № _____ от «_____» _____2015г. Тираж _____экз.
Изд-во СевГУ