Лабораторная работа 4. Использование коллекций в Java

# 1. Основные теоретические сведения

## 1.1. Компоненты коллекций

Коллекция, иногда называемая контейнером, – это объект, который объединяет несколько элементов в один объект. Коллекции используются для хранения данных, доступа и манипуляций с данными, а также для передачи данных от одного метода к другому. Коллекция обычно представляет данные, которые представляют естественную группу, например телефонный справочник (коллекция соответствий между именем и телефонным номером). Массив также можно рассматривать как коллекцию, объединяющую данные одного типа, элементы которой расположены последовательно, в порядке возрастания индекса.

Все интерфейсы и классы, относящиеся к коллекциям, находятся в пакете **java.util**.

Схема коллекций (collections framework) – это унифицированная архитектура для представления и манипулирования коллекциями. Все схемы коллекций содержат следующие три компоненты:

* **Интерфейсы** – абстрактные типы данных, представляющие коллекции. Интерфейсы позволяют манипулировать коллекциями независимо от деталей их представления. В Java, как и в других объектно-ориентированных языках, эти интерфейсы обычно образуют иерархию.
* **Реализации** – конкретные реализации интерфейсов коллекции. По своей сути они являются повторно используемыми структурами данных.
* **Алгоритмы** – методы, выполняющие некоторые полезные действия, например, поиск или сортировку, над объектами, реализующими интерфейсы коллекций. Эти методы называют полиморфными, так как один и тот же метод может использоваться во многих различных реализациях соответствующего интерфейса коллекций. По сути алгоритмы обеспечивают повторно используемую функциональность.

## 1.2. Интерфейсы коллекций

Ключевые интерфейсы коллекций введены, начиная с JDK 1.2, и используются для манипулирования коллекциями, а также для передачи их от одного метода к другому. Главной целью этих интерфейсов является возможность манипулирования коллекциями независимо от деталей их реализации.

### 1.2.1. Интерфейс **Collection**

Интерфейс **Collection** представляет группу объектов, известных как его элементы. Некоторые реализации коллекций разрешают дублирование элементов, другие – нет. Пакет SDK не обеспечивает прямой реализации этого интерфейса и он используется для передачи коллекций и манипулирования ими, когда требуется максимальное обобщение.

Интерфейс **Collection** объявляет методы, выполняющие следующие операции:

* определение количества элементов в коллекции (метод int **size()**);
* проверка, содержит ли коллекция элементы (метод boolean **isEmpty**());
* проверка, находится ли данный элемент в коллекции (метод boolean **contains**(Object element));
* сравнение заданного объекта с данной коллекцией на равенство (метод public boolean **equals**(Object o));
* добавления элемента в коллекцию (метод boolean **add**(Object element)) и удаления элемента из коллекции (метод boolean **remove**(Object element)), а также очистки коллекции (метод public void clear());
* обеспечение итерационных операций над коллекцией (метод Iterator **iterator**());
* обеспечение моста между коллекциями и старыми интерфейсами прикладных программ, которые предполагали передачу им параметров объекта в виде массива (методы Object[] **toArray**() и Object[] **toArray**(Object a[])).

Кроме этого, для коллекций определены следующие групповые операции

* boolean **containsAll**(Collection c)
* boolean **addAll**(Collection c)
* boolean **removeAll**(Collection c)
* boolean **retainAll**(Collection c) ,

которые позволяют проверить, содержатся ли все элементы коллекции c в данной коллекции; добавить все элементы коллекции c к данной коллекции; удалить из данной коллекции все элементы, содержащиеся в коллекции c или оставить в данной коллекции только те элементы, которые содержатся в коллекции c.

Метод **iterator** возвращает объект интерфейса **Iterator**. Интерфейс **Iterator** объявляет следующие методы для коллекций:

* boolean **hasNext**(), проверяющий, имеется ли следующий элемент,
* Object **next**(), возвращающий следующий элемент и
* void **remove**(), удаляющий данный элемент.

Ниже приводится пример, как объект **Iterator** используется для фильтрации коллекции, то есть удаления из коллекции элементов, отвечающих определенным условиям:

static void filter(Collection c)

{

for (Iterator i = c.iterator(); i.hasNext(); )

if (!cond(i.next()))

i.remove();

}

Данный код является полиморфным, то есть будет работать для любой коллекции, поддерживающей удаление элементов, независимо от реализации.

### 1.2.2. Интерфейс Set

Интерфейс **Set** является коллекцией, которая не должна содержать повторяющихся элементов. Интерфейс расширяет интерфейс **Collection** и содержит те же методы, что и родительский интерфейс. Однако методы в этом интерфейсе имеют более конкретный математический смысл. Так, например:

выражение s1.containsAll(s2) возвращает true, если s2 является подмножеством s1;

выражение s1.addAll(s2) преобразует s1 в объединение s1 и s2;

выражение s1.retainAll(s2) преобразует s1 в пересечение s1 и s2 (пересечение двух множеств содержит элементы, общие для обоих множеств);

выражение s1.removeAll(s2) удаляет из s1 все элементы, содержащиеся в s2.

### 1.2.3. Интерфейс List

Интерфейс **List** является упорядоченной коллекцией (иногда называемой последовательностью). Коллекция **List** может содержать упорядоченные элементы. В дополнение к операциям, унаследованным от коллекции, интерфейс объявляет следующие операции:

* получение значения элемента с заданным индексом (метод public Object **get**(int index)) и установка значения элемента с заданным индексом (метод public Object **set**(int index, Object element));
* поиск элемента в списке и возврат значения его индекса (методы public int **indexOf**(Object o) и public int **lastIndexOf**(Object o));
* добавление (метод public void **add**(int index, Object element)) или удаление (метод public Object remove(int index)) элемента по заданному индексу, а также очистка списка (метод public void **clear**());
* операции над частью списка (метод public List **subList**(int fromIndex, int toIndex));
* итерационные операции в списке (метод public ListIterator **listIterator**() и public ListIterator **listIterator**(int index)).

Интерфейс **ListIterator** расширяет интерфейс **Iterator** и содержит следующие методы просмотра списка:

* Object **next**() – возвращает следующий элемент (если следующего элемента нет, то выбрасывается исключение типа *NoSuchElementException*);
* Object **previous**() – возвращает предыдущий элемент (если предыдущего элемента нет, то выбрасывается исключение типа *NoSuchElementException*);
* boolean **hasNext**() – возвращает true, если существует следующий элемент, иначе возвращает false;
* boolean **hasPrevious**() – возвращает true, если существует предыдущий элемент, иначе возвращает false;
* int **nextIndex**() – возвращает индекс следующего элемента (если следующего элемента нет, возвращает размер списка);
* int **previousIndex**() – возвращает индекс предыдущего элемента (если предыдущего элемента нет, возвращает -1);
* void **add**(Object obj) – вставляет obj в список перед элементом, который будет возвращен следующим вызовом next();
* void **remove**() – удаляет текущий элемент из списка (выбрасывается исключение типа IllegalStateException, если метод **remove**() вызывается, прежде чем вызван метод **next**() или **previous**());
* void **set**(Object obj) – назначает obj на текущий элемент (это последний элемент, возвращенный вызовом метода **next**() или **previous**()).

Объекты в программах Java сортируются с помощью метода public int **compareTo**(Object obj), объявленного в интерфейсе **Comparable** (это единственный метод данного интерфейса). Метод должен возвращать значение 0, если сравниваемые объекты равны друг другу; значение, меньшее нуля, если принимающий объект меньше obj; значение, большее нуля, если принимающий объект больше obj. Конкретные реализации этого метода в объектных расширениях числовых классов (например, Integer или Float) позволяют сравнивать числа по значению, для строк – сравнивать строки в лексикографическом порядке, для дат – в хронологическом порядке. Сортировка, основанная на таком сравнении, называется *естественным* порядком сравнения.

Но часто необходимо отсортировать элементы коллекции в порядке, отличном от естественного или отсортировать элементы без реализации интерфейса **Comparable**. В этом случае необходимо реализовать интерфейс **Comparator** и обеспечить конкретную реализацию его методов public int **compare**(Object o1, Object o2) и public boolean **equals** (Object obj).

Первый метод возвращает отрицательное значение, если объект o1 меньше объекта o2, значение 0 в случае равенства объектов и положительное значение, если объект o1 больше объекта o2. Второй метод переопределяет соответствующий метод класса Object и проверяет равенства данного объекта интерфейса **Comparator** с объектом obj.

### 1.2.4. Интерфейс SortedSet

Интерфейс **SortedSet** является расширением интерфейса **Set**, элементы которого отсортированы в естественном порядке или согласно порядку, задаваемому методом интерфейса **Comparator**.

Интерфейс реализует следующие операции (дополнительно к операциям интерфейса **Set**):

* операции над частью набора (метод public SortedSet **subSet**(Object fromElement, Object toElement)), головной частью набора (метод public SortedSet **headSet**(Object toElement)) и хвостовой частью набора (метод public SortedSet **tailSet**(Object fromElement));
* возврат первого (метод public Object **first**()) и последнего (метод public Object **last**()) элемента отсортированного набора;
* доступ к интерфейсу Comparator (метод public Comparator **comparator**()).

### 1.2.5. Интерфейс Map

Интерфейс **Map** является объектом, который ставит в соответствие ключам значения. Ключи должны быть уникальными и им должно соответствовать единственное значение. Такую коллекцию называют отображением (map), словарем (dictionary) или ассоциативным массивом (associative array).

В интерфейсе **Map** определены следующие основные операции:

* получение размера отображения (метод public int **size**());
* проверка отображения на наличие элементов (метод public boolean **isEmpty**());
* помещение элемента в отображение (метод public Object **put**(Object key, Object value)) и получение значения элемента с заданным ключом (метод public Object **get**(Object key));
* сравнение отображения с заданным объектом на равенство (метод public boolean **equals**(Object o));
* представление ключей коллекции в виде множества (метод public Set **keySet**());
* удаление элемента с заданным ключом (метод public Object **remove**(Object key)) и очистка отображения (метод public void **clear**());
* проверка наличия элемента с заданным ключом (метод public boolean **containsKey**(Object key)) или значением (метод public boolean **containsValue**(Object value));
* добавление к данному отображению всех пар из заданного отображения (метод public void **putAll**(Map t));
* представление всех значений данного отображения в виде коллекции (метод public Collection **values**());
* представление коллекции в виде множества, каждый элемент которого – пара из данного отображения, с которой можно работать методами вложенного интерфейса **Map.Entry** (метод public Set **entrySet**()).

Интерфейс **Map.Entry** описывает методы работы с парами «ключ-значение», полученными методом **entrySet**():

* получение ключа (метод public Object **getKey**()) и значения (метод public Object **getValue**());
* изменение значения (метод public Object **setValue**(Object value));
* сравнение заданного объекта с данной парой «ключ-значение» на равенство (метод public boolean **equals**(Object o)).

Возможно также использование объектов **Map**, в которых ключам соответствует несколько значений. Это достигает обычно, если в качестве значений задать объекты **List**.

### 1.2.6. Интерфейс SortedMap

Интерфейс **SortedMap** является расширением **Map**. Элементы для этого интерфейса отсортированы в естественном порядке или согласно порядку, задаваемому методами интерфейса **Comparator**.

Интерфейс реализует следующие операции (дополнительно к операциям интерфейса Map):

* выделение части отображения (метод public SortedMap **subMap**(Object fromKey, Object toKey)), головной части отображения (метод public SortedMap **subMap**(Object fromKey, Object toKey)) или хвостовой части отображения (метод public SortedMap **tailMap**(Object fromKey));
* получение первого (метод public Object **firstKey**()) и последнего (метод public Object **lastKey**()) элемента отображения;
* доступ к интерфейсу Comparator (метод public Comparator **comparator**()).

В целом, интерфейс **SortedMap** является аналогом интерфейса **SortedSet**.

## 1.3. Реализации коллекций и алгоритмы

Реализации являются действительными объектами данных, которые воплощают в жизнь ключевые интерфейсы коллекций.

Общецелевые реализации и их связь с интерфейсами и структурами данных представлены в таблице:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Интерфейс | Реализации | | | |
|  | Хеш-таблица | Изменяемый массив | Сбалансирован-ное дерево | Связанный список |
| **Set** | **HashSet** |  | **TreeSet** | **LinkedHashSet** |
| **List** |  | **ArrayList** |  | **LinkedList** |
| **Map** | **HashMap**  **WeekHashMap** |  | **TreeMap** | **LinkedHashMap** |

Все реализации имеют не только схожие имена, но и схожее поведение. Все они реализуют каждую из операций, определенных в своем интерфейсе. Все позволяют использовать элементы, ключи и значения с константой null.

Основным при реализации структур данных является выбор интерфейса. В большинстве случаев выбор реализации влияет только на производительность программы. Поэтому предпочтительным стилем программирования является: сначала создание коллекции, затем выбор реализации и присваивание новой коллекции переменной соответствующего интерфейсного типа (или передача коллекции методу, ожидающему аргумент интерфейсного типа). Таким образом, программа становится независимой от любых добавляемых новых методов в данной реализации, оставляя за программистом право изменения реализации, если это необходимо для эффективности программы.

### 1.3.1. Класс HashSet

Для интерфейса **Set** двумя общецелевыми реализациями являются **HashSet** и **TreeSet**.

Класс **HashSet**  реализует интерфейс **Set** c поддержкой хеш-таблицы (обычно являющейся реализацией **HashMap**) и имеет следующие конструкторы:

* **HashSet**() – создает новый, пустой набор с емкостью по умолчанию и фактором загрузки, равным 0.75;
* **HashSet**(Collection c) – создает новый набор, содержащий элементы заданной коллекции;
* **HashSet**(int initialCapacity) – создает новый пустой набор с заданной емкостью и фактором загрузки, равным 0.75;
* **HashSet**(int initialCapacity, float loadFactor) – создает новый пустой набор с заданной емкостью по умолчанию и заданным фактором загрузки.

В классе **HashSet** реализованы следующие методы интерфейса Set: **add**(), **clear**(), **contains**(), **isEmpty**(), **iterator**(), **remove**() и **size**(). Метод public Object **clone**() позволяет получить пустую копию объекта **HashSet** (без элементов).

### 1.3.2. Класс ArrayList

Класс **ArrayList** обеспечивает реализацию интерфейса **Set** для массивов с изменяемыми размерами. Так же, как объекты класса **StringBuffer**, объекты **ArrayList** имеют две характеристики – емкость и размер массива. Если размер списка превысит емкость, емкость автоматически увеличивается на некоторое количество элементов.

Класс **ArrayList** имеет следующие конструкторы:

* **ArrayList**() – создает пустой список;
* **ArrayList**(Collection c) – создает список из заданной коллекции в порядке, заданном итератором данной коллекции;
* **ArrayList**(int initialCapacity) – создает пустой список с заданной емкостью.

Класс **ArrayList** реализует следующие методы интерфейса **List**: два метода **add**(), два метода **addAll**(), методы **clear**(), **contains**(), **get**(), **indexOf**(), **isEmpty**(), **lastIndexOf**(), **remove**(), **set**(), **size**() и два метода **toArray**().

Кроме того, класс содержит следующие собственные методы:

* public Object **clone**()– получение пустой копии объекта **ArrayList** (без элементов);
* public void **ensureCapacity**(int minCapacity) – увеличивает емкость экземпляра **ArrayList**, если это необходимо, для того, чтобы он мог содержать число элементов, заданное в minCapacity;
* protected void **removeRange**(int fromIndex, int toIndex) – удаляет из элементы в заданном диапазоне индексов;
* public void **trimToSize**() – уменьшает емкость объекта **ArrayList** до его предыдущего значения.

### 1.3.3. Класс LinkedList

Класс **LinkedList**, в добавление к реализации методов интерфейса **List** обеспечивает методы для получения, удаления и вставки элементов в список, что позволяет использовать связанные списки как стеки, очереди или очереди с двумя концами.

Класс **LinkedList** имеет два конструктора:

* **LinkedList**() – создает пустой связанный список;
* **LinkedList**(Collection c) – создает связанный список из заданной коллекции в порядке, заданном итератором данной коллекции;

В дополнение к двум метода **add**(), двум методам **addAll**(), методам **clear**(), **contains**(), **get**(), **indexOf**(), **isEmpty**(), **lastIndexOf**(), **listIterator**(), двум методам **remove**(), методам **set**(), **size**() и двум методам **toArray**() интерфейса **List**, класс **LinkedList** обеспечивает следующие методы:

* public void **addFirst**(Object o) и public void **addLast**(Object o) – добавление элемента в начало или конец связанного списка;
* public Object **getFirst**() и public Object **getLast**() – получение первого или последнего элемента связанного списка;
* public Object **removeFirst**() и public Object **removeLast**() – удаление первого или последнего элемента связанного списка;
* public Object **clone**()– получение пустой копии объекта LinkedList (без элементов).

Класс **ArrayList** является предпочтительным перед **LinkedList** в случае, если необходима высокая производительность. Однако, если часто приходится добавлять элементы в начало списка или удалять элементы из его середины, более предпочтительным является класс **LinkedList**. Этот класс полностью реализует все возможности класса **Stack**.

### 1.3.4. Класс HashMap

Класс **HashMap** является реализацией интерфейса **Map** с использованием хеш-таблиц и по своим возможностям соответствует классу **HashTable**.

Класс **HashMap** имеет следующие конструкторы:

* **HashMap**() – создает новое, пустое отображение с емкостью по умолчанию 16 и фактором загрузки, равным 0.75;
* **HashMap** (Map m) – создает новое отображение, содержащее элементы заданного отображения;
* **HashMap** (int initialCapacity) – создает новое, пустое отображение с заданной емкостью и фактором загрузки, равным 0.75;
* **HashMap** (int initialCapacity, float loadFactor) – создает новое, пустое отображение с заданной емкостью и заданным фактором загрузки.

Класс **HashMap** реализует следующие методы интерфейса **Map**: **clear**(), **containsKey**(), **containsValue**(), **entrySet**(), **get**(), **isEmpty**(), **keySet**(), **put**(), **putAll**(), **remove**(), **size**() и **values**() .

### 1.3.5. Класс Collections

Реализации-оболочки делегируют всю свою работу коллекциям, но добавляют в коллекции некоторую функциональность. Эти реализации являются анонимными, то есть они не обеспечивают общедоступный класс, а реализуются с помощью статических методов в классе **Collections**.

Класс **Collections** оперирует с коллекциями или возвращает коллекции. Класс не содержит конструктора, а только следующие поля с модификаторами public static final:

* List EMPTY\_LIST для пустых списков;
* Map EMPTY\_MAP для пустых отображений
* Set EMPTY\_SET для пустых множеств,

а также следующие public static методы, в основном реализующие полиморфные алгоритмы:

* int **binarySearch**(List list, Object key) – двоичный поиск в заданном списке заданного объекта;
* int **binarySearch**(List list, Object key, Comparator c) – двоичный поиск в заданном списке заданного объекта с использованием компаратора;
* void **copy**(List dest, List src) – копирование списка-источника в список назначения;
* void **fill**(List dest, Object o) – заполнение списка заданным объектом;
* Object **max**(Collection c) и Object **min**(Collection c) – возвращение максимального или минимального элемента коллекции в соответствии с естественным порядком сравнения;
* Object **max**(Collection c, Comparator com) и Object **min**(Collection c, Comparator com) – возвращение максимального или минимального элемента коллекции в соответствии с порядком, заданным с помощью компаратора;
* void **reverse**(List l) – переставляет элементы списка в обратном порядке;
* void **shuffle**(List l) – перемешивает элементы списка с использованием генератора случайных чисел по умолчанию;
* void **shuffle**(List l, Random rnd) – перемешивает элементы списка с использованием заданного генератора случайных чисел;
* List **singletonList**(Object o) – возвращает список, содержащий только данный объект;
* Map **singletonMap**(Object key, Object value) – возвращает отображение, содержащее только данный ключ и данное значение;
* void **sort**(List l) и void **sort**(List l, Comparator c) – сортирует список по возрастанию в соответствии с естественным порядком или с использованием компаратора;
* Collection **synchronizedCollection**(Collection c), List **synchronizedList**(List l), Map **synchronizedMap**(Map m), Set **synchronizedSet**(Set s), SortedSet **synchronizedSortedSet**(SortedSet ss) и SortedMap **synchronizedSortedMap**(SortedMap sm) – создают синхронизированные (не зависящие от потоков) экземпляры соответствующих типов коллекций;
* Collection **unmodifiableCollection**(Collection c), List **unmodifiableList**(List l), Map **unmodifiableMap**(Map m), Set **unmodifiableSet**(Set s), SortedSet **unmodifiableSortedSet**(SortedSet ss) и SortedMap **unmodifiableSortedMap**(SortedMap sm) – создают неизменяемые экземпляры соответствующих типов коллекций.

# 2. Варианты заданий

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Задачи** | **№** | **Задачи** | **№** | **Задачи** | **№** | **Задачи** |
| **1** | 1,5,16,40 | **7** | 8,13,18,22 | 13 | 2,14,28,38 | 19 | 8,15,22,34 |
| **2** | 2,13,17,41 | **8** | 9,14,23,43 | 14 | 3,5,29,39 | 20 | 9,13,21,35 |
| **3** | 3,15,18,42 | **9** | 5,10,24,44 | 15 | 4,15,26,30 | 21 | 10,14,20,36 |
| **4** | 4,14,19,43 | **10** | 11,15,25,35 | 16 | 5,13,25,31 | 22 | 5,11,19,37 |
| **5** | 5,6,20,44 | **11** | 12,14,26,36 | 17 | 5,7,24,32 | 23 | 12,15,18,38 |
| **6** | 7,15,17,21 | **12** | 1,13,27,37 | 18 | 14,6,23,33 | 24 | 1,13,17,45 |

1. Вывести таблицу преобразований целых десятичных чисел в интервале от 10 до 100 с шагом 20 в 16-ное представление.
2. Вывести таблицу преобразований целых десятичных чисел в интервале от min до max с шагом step в 16-ном представлении. Параметры задачи задаются как параметры командной строки.
3. Вывести таблицу преобразований целых десятичных чисел в интервале от min до max с шагом step в 16-ном представлении. Параметры задачи вводятся пользователем в ходе диалога с программой за один раз.
4. Обеспечьте вывод команд меню в виде нумерованного списка и запрос на ввод номера, что приводит к выводу сообщения о выполнении команды.
5. Создайте программу для шифрования\расшифровки текста методом Цезаря. В нем ключом является целое число, а шифрование\расшифровка заключается в суммировании\ вычитании кодов символов открытого текста\криптотекста с ключом.
6. Создайте программу для определения кода введенного символа в стандарте Юникод. Обеспечьте цикличность выполнения программы до ввода знака равенства (что приводит к выходу из программы).
7. Создайте программу для определения кода введенного символа в 16-ном представлены в формате '\ uXXXX ". Обеспечьте цикличность выполнения программы до ввода символа пробел. Проверьте правильность работы программы, воспользовавшись ею для получения кодов символов в любом слове (например, "cat") и выведя на экран слово из полученных кодов.
8. Создайте программу, которая принимает произвольный текстовую строку, а возвращает другой, в котором символы располагаются в обратном порядке.
9. Создайте программу для проверки текста на палиндром. Регистр символов и пробелы не учитываются.
10. Создайте программу для определения количества вхождений указанного символа в заданном тексте. Работа программы прекращается после введения символа "-".
11. Создайте программу для определения позиций вхождений указанного символа в заданном тексте. Работа программы прекращается после введения символа "-".
12. Создайте программу, которая принимает текстовую строку и выдает другой, в котором буквы переставлены в случайном порядке.
13. Создайте программу, которая осуществляет шифрование / расшифровка методом простой замены. В нем каждый символ незашифрованного текста из исходного алфавита заменяется другим из алфавита, символы в каком представлены в соответствии с ключом шифрования.
14. Создайте программу, которая осуществляет шифрование / расшифровка методом гаммирования. В нем ключом является текстовая строка такой же длины, как и открытый текст, а шифрование / расшифровка заключается в суммировании / вычитании кодов символов открытого текста / криптотексте с кодами символов ключа.
15. Создайте программу, которая осуществляет шифрование / расшифровка модифицированным методом Цезаря. В нем ключом является лозунг, который многократно повторяется до тех пор, чтобы достичь длины открытого текста, а шифрование / расшифровка заключается в суммировании / вычитании кодов символов открытого текста / криптотексте с соответствующими кодами символов лозунга.
16. Создайте приложение для добавления абонента и просмотра списка абонентов телефонной сети. Записи в списке (5 записей) создаются в программе и являются объектами класса HashMap, где ключом является номер телефона (типа Integer), а значением – объект Abonent, содержащий четыре значения типа String: фамилию, имя, отчество и адрес. Предусмотреть возможность сортировки элементов коллекции по 2-3 полям, для чего использовать класс ArrayList.
17. Создайте приложение для поиска абонента телефонной сети. Список абонентов (5 записей) создается в программе и является объектом класса TreeMap. Ключом записи является номер телефона (типа String), а значением – объект Abonent, содержащий четыре значения типа String: фамилию, имя, отчество и адрес. Предусмотреть возможность сортировки элементов коллекции по 2-3 полям, для чего использовать класс ArrayList.
18. Создайте приложение для просмотра списка элементов – целых чисел (типа Integer) и изменения элементов списка в заданном диапазоне индексов. Список (10 чисел) является объектом класса ArrayList. Предусмотреть возможность расчета количества повторов элементов, для чего использовать HashMap.
19. Создайте приложение для просмотра списка элементов и добавления элементов в список. Список (5 элементов типа Integer) создается в программе и является объектом класса ArrayList. Предусмотреть возможность проверки повторов элементов, для чего использовать TreeSet.
20. Создайте приложение для просмотра списка книг и удаления книг в библиотечном каталоге. Записи в списке (5 записей) являются объектами класса HashMap, где ключом является индекс ISBN книги (типа Integer), а значением – объект Book, содержащий наименование книги, фамилию, имя и отчество (ФИО) автора, издательство (все поля типа String), год издания (типа int) и цену книги (типа float). Предусмотреть возможность сортировки книг по 2-3 полям, для чего использовать ArrayList.
21. Создайте приложение для добавления книг и просмотра списка книг в библиотечном каталоге. Список книг (5 записей) создается в программе и является объектом класса HashMap. Ключом записи является индекс ISBN книги (типа Integer), а значением – объект Book, содержащий наименование книги, фамилию, имя и отчество (ФИО) автора, издательство (все записи типа String), год издания (типа int) и цену книги (типа float). Предусмотреть возможность получения списка авторов книг, где авторы не повторяются для чего использовать TreeSet.
22. Создайте приложение для просмотра списка очередников. Список (5 записей) создается в программе и является объектом класса LinkedList. Запись списка является объектом QueuePerson, содержащим поля фамилии, имени и отчества очередника (типа String), поле типа AddressValue для адреса очередника и приоритет очередника (типа int). В свою очередь, поле типа AddressValue содержит три поля типа String: наименование города, наименование улицы и номер дома, а также поле типа int – номер квартиры (если номер квартиры равен 0, дом, в котором проживает субъект, не имеет квартир). Записи в очереди сортируются в соответствии с приоритетом, и очередник добавляется последним в очереди своего приоритета. Предусмотреть возможность получения чтоб в очереди не было повторения по ФИО очередника, для чего использовать TreeSet.
23. Создайте приложение для просмотра списка очередников и изменения приоритета очередников в очереди. Список (5 записей) создается в программе и является объектом класса LinkedList. Запись списка является объектом QueuePerson, содержащим поля фамилии, имени и отчества очередника (типа String), поле типа AddressValue для адреса очередника и приоритет очередника (типа int). В свою очередь, поле типа AddressValue содержит три поля типа String: наименование города, наименование улицы и номер дома, а также поле типа int – номер квартиры (если номер квартиры равен 0, дом, в котором проживает очередник, не имеет квартир). Записи в очереди сортируются в соответствии с приоритетом, и очередник добавляется последним в очереди своего приоритета.
24. Создайте приложение для просмотра списка изображений и удаления изображения из списка изображений. Список (5 изображений) создается в программе и является объектом класса HashMap, где ключом является наименование изображения (типа String), а значением – изображение (объект класса Image). Предусмотреть возможность сортировки изображений с помощью ArrayList.
25. Создайте приложение для поиска изображения в списке изображений. Список (5 изображений) создается в программе и является объектом класса HashMap, где ключом является наименование изображения (типа String), а значением – изображение (объект класса Image). Предусмотреть возможность чтобы изображения добавлялись с уникальными именами с помощью TreeSet.
26. Создайте приложение для изменения значения элементов и просмотра списка элементов во множестве целых чисел типа Integer. Множество (10 элементов) создается в программе и является объектом класса HashSet (элементы множества не должны совпадать). Предусмотреть возможность сортировки с помощью ArrayList.
27. Создайте приложение для добавления элементов и просмотра списка элементов во множестве целых чисел типа Integer. Множество (10 элементов) создается в программе и является объектом класса HashSet (элементы множества не должны совпадать). Предусмотреть возможность сортировки с помощью TreeSet.
28. Создайте приложение для изменения абонента телефонной сети. Список абонентов (5 записей) создается в программе и является объектом класса HashMap. Ключом записи является номер телефона (типа Integer), а значением – объект Abonent, содержащий четыре значения типа String: фамилию, имя, отчество и адрес. Предусмотреть возможность сортировки по фамилии с помощью ArrayList.
29. Создайте приложение для удаления абонента и просмотра списка абонентов телефонной сети. Список абонентов (5 записей) создается в программе и является объектом класса HashMap. Ключом записи является номер телефона (типа Integer), а значением – объект Abonent, содержащий четыре значения типа String: фамилию, имя, отчество и адрес. Предусмотреть возможность сортировки по фамилии с помощью TreeMap.
30. Создайте приложение для покупки товаров в электронном магазине. Список товаров (5 записей) создается в программе и является объектом класса HashMap. Ключом записи является артикул товара marking (типа Integer), а значением – объект Article, содержащий наименование товара (типа String) и цену товара (типа float). Предусмотреть возможность проверки уникальности наименования товара с помощью TreeSet.
31. Создайте приложение для просмотра списка товаров и изменения цены товара в электронном магазине. Список товаров (5 записей) создается в программе и является объектом класса TreeMap. Ключом записи является артикул товара marking (типа Integer), а значением – объект Article, содержащий наименование товара (типа String) и цену товара (типа float). Предусмотреть возможность проверки уникальности наименования товара с помощью HashSet.
32. Создайте приложение для поиска книг в библиотечном каталоге по заданному критерию. Список книг (5 записей) создается в программе и является объектом класса HashMap. Ключом записи является индекс ISBN книги (типа Integer), а значением – объект Book, содержащий наименование книги, фамилию, имя и отчество (ФИО) автора, издательство (все записи типа String), год издания (типа int) и цену книги (типа float). Предусмотреть возможность сортировки каталога по ФИО автора или году издания, для чего использовать ArrayList. Предусмотреть возможность проверки уникальности книги с помощью TreeSet.
33. Создайте приложение для просмотра списка книг в библиотечном каталоге и изменения цены книги. Список книг (5 записей) создается в программе и является объектом класса HashMap. Ключом записи является индекс ISBN книги (типа Integer), а значением – объект Book, содержащий наименование книги, ФИО автора, издательство (все записи типа String), год издания (типа int) и цену книги (типа float). Предусмотреть возможность сортировки книг по издательству или цене, используя ArrayList.
34. Создайте приложение для добавления очередника в очередь и просмотра очереди. Список (5 записей) создается в программе и является объектом класса LinkedList. Запись списка является объектом QueuePerson, содержащим поля фамилии, имени и отчества очередника (все поля типа String), поле типа AddressValue для адреса очередника и приоритет очередника (типа int). В свою очередь, поле типа AddressValue содержит три поля типа String: наименование города, наименование улицы и номер дома, а также поле типа int – номер квартиры (если номер квартиры равен 0, дом, в котором проживает субъект, не имеет квартир). Записи в очереди сортируются в соответствии с приоритетом, и очередник добавляется последним в очереди своего приоритета. Предусмотреть возможность расчета количества очередников каждого приоритета, используя HashMap.
35. Создайте приложение для выборки очередника из очереди. Список (5 записей) создается в программе и является объектом класса LinkedList. Запись списка является объектом QueuePerson, содержащим поля фамилии, имени и отчества очередника (все поля типа String), поле типа AddressValue для адреса очередника и приоритет очередника (типа int). В свою очередь, поле типа AddressValue содержит три поля типа String: наименование города, наименование улицы и номер дома, а также поле типа int – номер квартиры (если номер квартиры равен 0, дом, в котором проживает субъект, не имеет квартир). Записи в очереди сортируются в соответствии с приоритетом, и очередник добавляется последним в очереди своего приоритета.
36. Создайте приложение для просмотра списка элементов в массиве целых чисел и удаления элементов списка. Список (5 элементов типа Integer) создается в программе и является объектом класса ArrayList.
37. Создайте приложение для просмотра списка элементов и изменения значений элементов списка. Список (5 элементов типа Integer) создается в программе и является объектом класса ArrayList.
38. Создайте приложение для добавления элемента в стек с приоритетами и просмотра стека. Стек (5 записей) создается в программе и является объектом класса LinkedList. Запись в стеке является объектом StackMember, содержащим наименование программы (типа String), объем памяти для программы в мегабайтах (типа int) и приоритет программы (типа int). Записи в стеке сортируются в соответствии с приоритетом, и очередник добавляется первым в очереди своего приоритета.
39. Создайте приложение для выборки элемента из стека с приоритетами. Стек (5 записей) создается в программе и является объектом класса LinkedList. Запись в стеке является объектом StackMember, содержащим наименование программы (типа String), объем памяти для программы в мегабайтах (типа int) и приоритет программы (типа int). Записи в стеке сортируются в соответствии с приоритетом, и очередник добавляется первым в очереди своего приоритета.
40. Создайте приложение для просмотра списка товаров и удаления товара в электронном магазине. Список товаров (5 записей) создается в программе и является объектом класса HashMap. Ключом записи является артикул товара marking (типа Integer), а значением – объект Article, содержащий наименование товара (типа String) и цену товара (типа float).
41. Создайте приложение для добавления товара и просмотра списка товаров электронного магазина. Список товаров (5 записей) создается в программе и является объектом класса HashMap. Ключом записи является артикул товара marking (типа Integer), а значением – объект Article, содержащий наименование товара (типа String) и цену товара (типа float).
42. Создайте приложение для просмотра списка студентов и изменения значений элементов списка. Список (5 элементов) создается в программе и является объектом класса ArrayList. Каждый элемент списка содержит фамилию, имя и отчество (ФИО) студента (типа String), имя группы (типа String), дату рождения (типа BirthDate) и среднюю оценку за прошедшую сессию (типа float). Объект класса BirthDate в свою очередь содержит три элемента типа int (год, номер месяца и день рождения).
43. Создайте приложение для просмотра, добавления и удаления элементов списка студентов. Список (5 элементов) создается в программе и является объектом класса ArrayList. Каждый элемент списка содержит фамилию, имя и отчество (ФИО) студента (типа String), имя группы (типа String), дату рождения (типа BirthDate) и среднюю оценку за прошедшую сессию (типа float). Объект класса BirthDate в свою очередь содержит три элемента типа int (год, номер месяца и день рождения).
44. Создайте приложение для просмотра списка файлов и добавления файла в список. Список (для 5 текстовых файлов с расширением .txt) создается в программе и является объектом класса HashMap, где ключом является имя файла (типа String), а значением – объект типа TextFile, содержащий два элемента String – абсолютный путь к файлу (без имени файла) и краткое описание содержимого файла.
45. Создайте приложение для поиска в списке файлов и удаления файла из списка. Список (для 5 текстовых файлов с расширением .txt) создается в программе и является объектом класса HashMap, где ключом является имя файла (типа String), а значением – объект типа TextFile, содержащий два элемента String – абсолютный путь к файлу (без имени файла) и краткое описание содержимого файла.

# 3. Ход выполнения работы

1. В IntelliJ IDEA создайте проект **Maven**.
2. Создайте классы программ в соответствии с вариантом индивидуального задания. При этом в заданиях №1-15 не использовать массивы (если возможно), а воспользоваться подходящими классами коллекций!
3. Создайте unit-тесты для тестирования созданной программы.
4. Выполните тестирование программ.
5. Оформите отчет о проделанной работе.