Лабораторная работа № 4: Контейнеры

1. Создайте файл `ListExample.java`, демонстрирующий работу с контейнерами типа `List`:

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
public class ListExample {
    /**
     *  Returns  elements  for  the  list  of  Strings  that  have
certain length
     *
     * Method is executed in {@link #main(String[])}
     *
     * @param len - maximum length of String element
     * @param list - List of values
     * @return list of elements, that have length less than len
     *
     */
    public    static    List    getListElementsUnder(int    len,
List<String> list) {
        List<String> result = new ArrayList<>();
        for (int i = 0; i < list.size(); i++) {
            if(list.get(i).length()<len)result.add(list.get(i));
        }
        return result;
    }
    public static void main(String[] args){
        List<String> arrayListStr = new ArrayList<>();
        arrayListStr.add("Hello");
        arrayListStr.add("Student");
        List<String>result=getListElementsUnder(6,arrayListStr);
        System.out.println(result);
        List<String> linkedListStr = new LinkedList<>();
        linkedListStr.add("Math");
        linkedListStr.add("IT");
        linkedListStr.add("Physics");
        linkedListStr.add("PE");
        List<String>res = getListElementsUnder(6,linkedListStr);
        System.out.println(res);
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
[Hello]
[Math, IT, PE]
```

Список (`List`)

`List` представляет собой интерфейс контейнера с индексируемым доступом к его элементам. В отличие от обычных массивов, он является динамическим. Основными методами контейнеров типа `List` являются:

- `add(Object obj)` – осуществляет добавление объекта `obj`, переданного в качестве аргумента, в конец контейнера;
- `add(int index, Object obj)` – осуществляет добавление объекта `obj`, переданного в качестве аргумента, в контейнер на позицию `index`;
- `addAll(Collection collection)` – осуществляет добавление элементов коллекции `collection` в конец списка;
- `addAll(int index, Collection collection)` – осуществляет добавление элементов коллекции `collection` в список, начиная с позиции `index`;
- `get(index)` – получение элемента с позиции `index`;
- `size()` – получение числа элементов контейнера;
- `remove(Object obj)` – удаление объекта `obj` из контейнера;
- `remove(int index)` – удаление элемента с позиции `index`;
- `removeAll(Collection collection)` – удаление всех элементов коллекции `collection` из контейнера;
- `contains(Object obj)` – проверка наличия элемента `obj` в контейнере.

Вызов методов происходит напрямую от объекта-списка с указанием имени метода через точку после имени объекта. С остальными методами можно ознакомиться в документации по ссылке:

`https://docs.oracle.com/javase/7/docs/api/java/util/List.html`

К основным контейнерам типа `List` относятся:

- `ArrayList` – данный тип контейнеров отличается быстротой произвольного доступа к элементам, однако добавление и удаление элементов из середины списка происходит с относительно небольшой скоростью.
- `LinkedList` – данный тип контейнеров характеризуется высокой скоростью вставки и удаления элементов, в том числе и из середины списка, однако скорость произвольного доступа к элементам в `LinkedList` невысокая. Может использоваться для реализации стека.

Объявление объектов типов `List`, `ArrayList` и `LinkedList` может происходить следующим образом:

```
List list = new ArrayList(); // может содержать элементы любых типов
List<String> list1 = new ArrayList<>(); // содержит строки
LinkedList<Integer> list2 = new LinkedList<>(); // содержит целые числа
```

Типизированные контейнеры

Типизированные контейнеры (Generics) – контейнеры некоторого неопределенного типа <T>, позволяющие применять обозначенные в них методы к различным типам данных. При таком подходе к разработке для обеспечения безопасности рекомендуется указывать тип значений, которые могут храниться в контейнере. Объявление таких контейнеров происходит следующим образом:

```
ArrayList<String> list1 = new ArrayList<>();
```

Тип значений контейнера указывается в треугольных скобках <> после типа контейнера.

Документация Java (JavaDoc)

Большинство сред разработки поддерживают автоматическое генерирование HTML-документации на основе оформленной документации к вашему Java-коду. Документации к коду, написанному на языке Java носит специальное название – JavaDoc – и оформляется как комментарии перед именем метода, класса и т.д. следующим образом:

```
    /**
     *  Returns elements for the list of Strings that have
certain length
     *
     * Method is executed in {@link #main(String[])}
     *
     * @param len - maximum length of String element
     * @param list - List of values
     * @return list of elements, that have length less than len
     *
     */
     public    static    List    getListElementsUnder(int    len,
List<String> list) { . . . }
```

В отличие от обычных комментариев, комментарии, содержащие документацию, начинаются с символов /** . . . */. Среда разработки автоматически проставляет для таких комментариев символы * слева от каждой строки. Первые несколько строк документации метода (класса, …) несут в себе общую информацию о предназначении данного метода (класса, …). Далее перечисляются основные характеристики методы в соответствии с шаблоном ключевых аннотаций (тегов), указанных в таблице:

Тег и формат	Описание	Применение
`@author <имя>`	Описание автора	Class, Interface, Enum
`@version <версия>`	Информация о версии ПО. Должен встречаться один раз на класс, интерфейс и т.д.	Class, Interface, Enum
`@since <дата>`	Дата первого внедрения функционала	Class, Interface, Enum, Field, Method
`@see <ссылка>`	Ссылка на другой элемент документации	Class, Interface, Enum, Field, Method
`@param <имя>`	Параметр некого метода с указанием	Method

<описание>	его имени и краткого описания	
@return <описание>	Возвращаемое значение	Method
@exception <имя класса> <описание> @throws <имя класса> <описание>	Исключение, которое может возникнуть в некотором методе	Method
@deprecated <описание>	Использование данного функционала запрещено	Class, Interface, Enum, Field, Method
{@inheritDoc}	Копия описания из метода родительского класса	Overriding Method
{@link <ссылка>}	Ссылка	Class, Interface, Enum, Field, Method
{@value #<static-поле>}	Значение переменной	Static Field
{@code <литерал>}	Форматирует литерал в HTML-тег <code> {@literal} </code>	Class, Interface, Enum, Field, Method
{@literal <литерал>}	Буквенный текст, не содержащий HTML-элементов	Class, Interface, Enum, Field, Method

После генерирования подготовленной документации для классов, методов и т.д., будет получена веб-страница с документацией.

2. Создайте класс `Device` с полями, хранящими значение идентификатора и емкости устройства. Добавьте метод `getInfo()`, возвращающий информацию о полях объекта в виде строки.

3. Создайте класс `ListCheck`, в котором определите методы:
   - `printList(..)`, принимающий в качестве аргумента контейнер `List<Device>` и осуществляющий вывод информации о содержащихся в контейнере объектах;
   - `checkElement(..)`, принимающий в качестве аргумента контейнер `List<Device>`, элемент типа `Device` и некоторое целое число `number` и осуществляющий проверку кратности идентификатора элемента `Device` переданному числу `number`. Перед проверкой на кратность необходимо удостоверится, что элемент `Device` присутствует в контейнере, если элемент отсутствует, на экран необходимо вывести сообщение об ошибке.

Добавьте в класс запускаемый метод, в котором заполните 50 значениями два контейнера типов `ArrayList` и `LinkedList` соответственно. Осуществите проверку реализованных методов на созданных контенейрах.

4. Дополните класс `ListCheck`, добавив документацию в формате JavaDoc для методов `printList()` и `checkElement()`, указав краткое описание метода, его аргументы и возвращаемое значение.

5. Создайте файл IteratorExample.java, демонстрирующий работу с итераторами:

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
public class IteratorExample {
    public static void main(String[] args){
        List<String> linkedListStr = new LinkedList<>();
        linkedListStr.add("Math");
        linkedListStr.add("IT");
        linkedListStr.add("Physics");
        // пример итерации с использованием итератора
        Iterator iterator = linkedListStr.iterator();
        System.out.println("Before: " + linkedListStr);
        while(iterator.hasNext()){
            String elem = iterator.next().toString();
            if (elem.length() >= 6) {
                iterator.remove(); // удаление элемента
            }
        }
        System.out.println("After: " + linkedListStr);
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Before: [Math, IT, Physics]
After: [Math, IT]
```

Итератор

Для итерируемых контейнеров (коллекций) предусмотрены различные варианты итераций по объектам контейнера:

1. Для индексируемых контейнеров проход по контейнеру с использованием индексом элементов.
2. С использованием цикла с итерацией по объектам контейнера без индексирования.
3. С помощью итераторов.

Итератор – это специальный объект, позволяющий пройти по контейнеру, получить значение элемента, удалить элемент и т.д. Для использования данного способа итерации необходимо определить объект-итератор для конкретного контейнера, для которого станут доступны методы, позволяющие получить текущее значение контейнера, до которого доходит итератор:

- `hasNext()` – проверка наличия следующего элемента в контейнере (конца длины контейнера);
- `next()` – получение следующего значения итератора (контейнера);
- `remove()` – удаляет значение контейнера, полученное при последнем вызове метода `next()`.

Вызов данных методов осуществляется напрямую от объекта-итератора с указанием имени метода через точку после имени объекта. Получение итератора для контейнера осуществляется с помощью метода `iterator()`, вызываемого от объекта-контейнера.

Пример программы с использованием итератора:

```
. . .
Iterator iterator = list.iterator();
System.out.println("Before: " + linkedListStr);
while(iterator.hasNext()){ // проверка наличия значений
    String elem = iterator.next().toString(); // получение
значения
    if (elem.length() >= 6) {
        iterator.remove(); // удаление элемента
    }
}
```

6. Дополните класс `ListCheck`, определив в нем метод `printListWithIterator(..)`, принимающий в качестве аргумента контейнер `List<Device>` и выводящий информацию о каждом элементе контейнера на экран с использованием итератора. Осуществите проверку работы метода на созданных ранее контейнерах в главном методе класса.

7. Создайте файл `SetExample.java`, демонстрирующий работу с контейнерами типа `Set`:

```
import java.util.*; // импорт всех библиотек пакета java.util
public class SetExample {
    public static void main(String[] args) {
        Set<Integer> hashSet = new HashSet<>();
        Set<Integer> treeSet = new TreeSet<>();
        Set<Integer> linkedHashSet = new LinkedHashSet<>();
        Random random = new Random();
        for (int i=0; i< 20; i++) {
            int val = random.nextInt(50);
            hashSet.add(val);
            treeSet.add(val);
            linkedHashSet.add(val);
        }
        // вывод контейнеров
        System.out.println(hashSet);
        System.out.println(treeSet);
        System.out.println(linkedHashSet);
        // проверка наличия элемента
        System.out.println("Is 5 presented? " +
                hashSet.contains(5));
        // получение значений контейнера
        System.out.println("Let's compare first elements: " +
                        hashSet.iterator().next() + " | " +
                        treeSet.iterator().next() + " | " +
                        linkedHashSet.iterator().next());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
[32, 0, 33, 2, 34, 35, 37, 5, 38, 8, 9, 11, 45, 14, 17, 23, 24, 27, 29]
[0, 2, 5, 8, 9, 11, 14, 17, 23, 24, 27, 29, 32, 33, 34, 35, 37, 38, 45]
[27, 35, 2, 33, 9, 29, 14, 34, 37, 32, 23, 45, 38, 24, 17, 11, 5, 8, 0]
Is 5 presented? true
Let's compare first elements: 32 | 0 | 27
```

Множества (`Set`)

Множества (`Set`) представляют собой контейнеры, в которых каждый элемент встречается ровно один раз. Основными методами контейнеров типа `Set` являются:

- `add(Object obj)` – осуществляет добавление объекта `obj`, переданного в качестве аргумента, в конец контейнера, если он отсутствует в контейнере;
- `addAll(Collection collection)` – осуществляет добавление элементов коллекции `collection` в конец контейнера, если они отсутствуют в контейнере;
- `retainAll(Collection collection)` – удаляет из контейнера элементы, не содержащиеся в коллекции `collection`;
- `remove(Object obj)` – удаление объекта `obj` из контейнера;
- `removeAll(Collection collection)` – удаление всех элементов коллекции `collection` из контейнера;
- `size()` – получение числа элементов контейнера;
- `contains(Object obj)` – проверка наличия элемента `obj` в контейнере.

Тип `Collection` представляет группу элементов, например, список, множество и т.д.
Вызов методов происходит напрямую от объекта-множества с указанием имени метода через точку после имени объекта. С остальными методами можно ознакомиться в документации по ссылке:

`https://docs.oracle.com/javase/7/docs/api/java/util/Set.html`

Получение значений элементов множества возможно с использованием итератором для созданного контейнера.
К основным контейнерам типа `Set` относятся:

- `HashSet` – использует хеширование (преобразование входных данных в выходную строку в бинарном виде) для ускорения выборки элементов, вследствие чего последовательность вывода элементов при выборке незакономерна;
- `TreeSet` – осуществляет быструю сортировку элементов при выборке;
- `LinkedHashSet` – расширяет класс `HashSet`, позволяя получить значения контейнера в том порядке, в котором они добавлялись в контейнер.

8. Создайте класс `SetCheck`, определив в нем следующие методы:

- `setFromString(..)` – принимает на вход строку, из которой необходимо составить контейнер, выделив в ней отдельные слова, и контейнер типа `Set`, который будет заполнен словами;
- `printSet(..)`, принимающий в качестве аргумента контейнер `Set<String>` и осуществляющий поэлементный вывод информации об объектах контейнера с использованием итератора;
- `union(..)` – принимает на вход два множества (контейнера типа `Set`), осуществляет операцию объединения множеств с использованием стандартного метода контейнера `Set` и возвращает новое множество, полученное за счет объединения множеств;
- `intersection(..)` – принимает на вход два множества (контейнера типа `Set`), осуществляет операцию пересечения множеств с использованием стандартного метода контейнера `Set` и возвращает новое множество, полученное за счет пересечения множеств;
- `subtraction(..)` – принимает на вход два множества (контейнера типа `Set`), осуществляет операцию разности множеств с использованием стандартного метода контейнера `Set` и возвращает новое множество, полученное за счет разности множеств.

Для реализации операций `union`, `intersection` и `subtraction` используйте стандартные методы контейнера `Set`. Создайте для класса главный запускаемый метод, в котором осуществите проверку всех методов на контейнерах типов `HashSet<String>`, `TreeSet<String>` и `LinkedHashSet<String>`, созданных из указанной строки:

```
A computer's memory can be viewed as a list of cells into which numbers can be placed or read. Each cell has a numbered address and can store a single number. The computer can be instructed to put the number 123 into the cell numbered 1357 or to add the number that is in cell 1357 to the number that is in cell 2468 and put the answer into cell 1595.
```

9. Создайте файл `MapExample.java`, демонстрирующий работу с контейнерами типа `Map`:

```
import java.util.*;
public class MapExample {
    public static void main(String[] args) {
        Map<String,Integer> hashMap = new HashMap<>();
        Map<String,Integer> treeMap = new TreeMap<>();
        Map<String,Integer>linkedHashMap=new LinkedHashMap<>();
        Random random = new Random();
        for (int i=0; i<20; i++) {
            int val = random.nextInt(50);
            hashMap.put("k" + i,val);
            treeMap.put("k" + i,val);
            linkedHashMap.put("k" + i,val);
        }
        // вывод контейнеров
        System.out.println(hashMap);
        System.out.println(treeMap);
        System.out.println(linkedHashMap);
        // получение значений по ключу
        System.out.println("Let's compare first elements: " +
```

```
                hashMap.get("k0") + " | " +
                treeMap.get("k0") + " | " +
                linkedHashMap.get("k0"));
        // проверка наличия ключа
        System.out.println("Is key k30 presented? " +
                           hashMap.containsKey("k30"));
        // проверка наличия значения
        System.out.println("Is value 23 presented? " +
                           treeMap.containsValue(23));
        // получить множество ключей контейнера
        System.out.println("Keys: " + linkedHashMap.keySet());
        // получить множество значений контейнера
        System.out.println("Values: " + hashMap.values());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
{k0=9, k1=14, k2=44, k3=20, k4=36, k5=28, k11=24, k6=4, k10=27,
k7=6, k13=46, k8=27, k12=7, k9=46, k15=29, k14=32, k17=12,
k16=23, k19=27, k18=42}
{k0=9, k1=14, k10=27, k11=24, k12=7, k13=46, k14=32, k15=29,
k16=23, k17=12, k18=42, k19=27, k2=44, k3=20, k4=36, k5=28,
k6=4, k7=6, k8=27, k9=46}
{k0=9, k1=14, k2=44, k3=20, k4=36, k5=28, k6=4, k7=6, k8=27,
k9=46, k10=27, k11=24, k12=7, k13=46, k14=32, k15=29, k16=23,
k17=12, k18=42, k19=27}
Let's compare first elements: 9 | 9 | 9
Is key k30 presented? false
Is value 23 presented? true
Keys: [k0, k1, k2, k3, k4, k5, k6, k7, k8, k9, k10, k11, k12,
k13, k14, k15, k16, k17, k18, k19]
Values: [9, 14, 44, 20, 36, 28, 24, 4, 27, 6, 46, 27, 7, 46, 29,
32, 12, 23, 27, 42]
```

Карта (`Map`)

Карты (`Map`) представляют собой контейнеры с парами ключ-значения, также называемыми ассоциативными массивами. Основными методами контейнеров типа `Map` являются:

- `put(K key, V value)` – вставка пары ключ-значение (`key-value`) в контейнер;
- `entrySet()` – множество пар ключ-значение (тип `Map.Entry`) для контейнера;
- `get(Object key)` – получение значения по ключу `key`;
- `containsKey(Object key)` – проверка наличия ключа `key` в контейнере;
- `containsValue(Object value)` – проверка наличия значения `value` в контейнере;
- `remove(Object key)` – удаление пары ключ-значение по ключу `key`;
- `keySet()` – получение множества ключей контейнера;

- `values()` – получение коллекции значений контейнера;
- `size()` – получение числа элементов списка.

Вызов методов происходит напрямую от объекта-множества с указанием имени метода через точку после имени объекта. С остальными методами можно ознакомиться в документации по ссылке:

`https://docs.oracle.com/javase/7/docs/api/java/util/Map.html`

К основным контейнерам типа `Map` относятся:

- `HashMap` – использует хеширование для ускорения выборки элементов, вследствие чего последовательность вывода элементов при выборке незакономерна;
- `TreeMap` – осуществляет быструю сортировку пар ключ-значение по ключам при выборке;
- `LinkedHashMap` – расширяет класс `HashMap`, позволяя получить значения контейнера в том порядке, в котором они добавлялись в контейнер.

10. Создайте класс `Person`, со следующими полями (все имеют модификатор доступа `private`):
    - `String lastname` – фамилия;
    - `String firstname` – имя;
    - `String middlename` – отчество;
    - `Calendar birthday` – дата рождения.

Со спецификацией типа `Calendar` можно ознакомиться по ссылке:
`http://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html`
Добавьте классу геттеры и сеттеры для каждого поля, а также метод `getFIO()`, возвращающий строковое значение информации о человеке в формате "`Фамилия И.О.`".

11. Создайте класс `MapCheck`, определив в нем следующие методы:
    - `setMap(..)` – принимает на вход контейнер типа `List<Person>` и контейнер типа `Map` и осуществляет заполнение контейнера типа `Map` по следующему принципу: в качестве ключей выступают года рождения людей, в качестве значений – списки людей (`ArrayList<Person>`) с годом рождения, указанным в качестве ключа; метод возвращает заполненный контейнер типа `Map`;
    - `compareMaps(..)` – принимает на вход три контейнера типов `HashMap<Integer, ArrayList<Person>>`, `TreeMap<Integer, ArrayList<Person>>` и `LinkedHashMap<Integer,ArrayList<Person>>` и осуществляет поэлементный вывод информации о парах ключ-значение, где значение выводится с помощью метода `getFIO()` класса `Person` в следующем виде, используя разделители, располагающиеся друг под другом (в виде таблицы):

```
HashMap  | TreeMap  | LinkedHashMap
Fam I.O. | Fam I.O. | Fam I.O.
```

Создайте для класса главный запускаемый метод, в котором осуществите проверку всех методов на контейнерах типов `HashMap<Integer, ArrayList<Person>>`, `TreeMap<Integer, ArrayList<Person>>` и `LinkedHashMap<Integer,`

ArrayList<Person>>, созданных с использованием метода setMap() для списка из 9 разных объектов класса Person (по три объекта на каждый год рождения).

12. Создайте класс Research, в котором реализуйте проведение анализа эффективности работы методов по нижеуказанной программе исследований для различного количества элементов в контейнерах, равного **10**, **100**, **1000** и **10000**. Показателем эффективности является время выполнения метода. По каждому из указанных методов для каждого из контейнеров (8 представленных в программе исследований) необходимо выводить информацию в следующем виде:
<количество_элементов> elements (<тип_контейнера>): <метод_и_его_параметры>: <время_в_миллисекундах> мс

Группируйте вывод по количеству элементов, оставляя пустую строку между группами.
Все контейнеры содержат внутри себя объекты следующего типа:

```
class Sample .. {
     private int id;
    /** Переопределяет метод Object.toString()
     * Возвращает информацию об объекте в формате id:..,
     * где вместо многоточия подставляется уникальный id
     * текущего объекта, в строковом виде.
     */
     public String toString(){
          . . .
     }
}
```

Для корректной работы сортировки в контейнерах типов TreeSet и TreeMap необходимо классом Sample реализовать интерфейс Comparable и переопределить метод compareTo(..) с целью проведения сравнения при сортировке по идентификатору объекта.

При заполнении контейнеров ключ для Map – "k..", где вместо многоточия указывается номер текущего элемента (пары ключ-значение), например k20.

Программа исследования:
Для контейнеров типов ArrayList, LinkedList:

- вставка элемента в конец списка;
- вставка элемента в середину списка;
- получение элемента с позиции в начале списка;
- получение элемента с позиции в середине списка;
- получение элемента с позиции в конце списка;
- проверка наличия элемента в списке;
- вывода содержимого контейнера на экран;
- удаление элемента (по значению) из списка;
- удаление коллекции элементов из списка (на примере использования Set);
- удаление элемента с позиции в начале списка;
- удаление элемента с позиции в середине списка;
- удаление элемента с позиции в конце списка.

Для контейнеров типов HashSet, TreeSet, LinkedHashSet:

- добавление элемента в множество;

- добавление значений коллекции в множество (на примере включения `List` в `Set`);
- проверка наличия элемента в множестве;
- удаление элемента (по значению) из множества;
- удаление коллекции элементов из списка (на примере использования `List`);
- вывода содержимого контейнера на экран.

Для контейнеров типов `HashMap`, `TreeMap`, `LinkedHashMap`:

- вставка пары ключ-значение;
- получение множества пар ключ-значение;
- получение значения по ключу;
- проверка наличия ключа в контейнере;
- проверка наличия значения в контейнере;
- удаление пары ключ-значение по ключу;
- получение множества ключей контейнера;
- получение множества значений контейнера.

13. Создайте пакет `collections`, содержащий запускаемый класс `StructureService`, который на основе указанных требований формирует сложные структуры данных, содержащие информацию о различных системах и их компонентах, в соответствии с вашим вариантом. Система и ее компоненты должны быть описаны отдельными классами в соответствии с предметной областью. В ходе проектирования соблюдайте логическую вложенность компонентов. Необходимо определить подходящую структуру для перечня различных объектов системы, а также структуры, хранящие информацию о перечнях компонентов системы.
Для каждой структуры доступны следующие действия:

- добавление отдельного элемента
- изменение отдельного элемента
- удаление отдельного элемента
- вывод информации (с учетом вложенности коллекций)

Для выполнения указанных действий используйте стандартные методы работы с коллекциями. Для добавления, изменения или удаления элемента отдельной структуры данных в запускаемом методе класса `StructureService` с клавиатуры считывается код исполняемой команды, идентификационные данные добавляемого, изменяемого или удаляемого объекта, а после выполнения операции выводится следующая информация:

- тип выполненной операции (из указанных выше)
- время выполнения операции (в миллисекундах)
- результат выполнения операции («Был добавлен/изменен/удален элемент … »)
- текущее состояние коллекции (вывод информации)

Вариант	**Описание систем**
1	Система: Метрополитен Компоненты и требования к структурам: Линии (быстрая сортировка, уникальность элементов), Станции (важна последовательность, быстрая выборка, уникальность элементов)

2	Система: Аэропорт Компоненты и требования к структурам: Терминалы (важна последовательность, быстрый произвольный доступ), Рейсы (важна последовательность, быстрая выборка, уникальность элементов)
3	Система: Университет Компоненты и требования к структурам: Подразделения (важна последовательность, уникальность элементов), Сотрудники (быстрая сортировка, уникальность элементов)
4	Система: Спортивная арена Компоненты и требования к структурам: Матчи (ключ – дата/время мероприятия, важна последовательность), Концерты (ключ – дата/время мероприятия, важна последовательность)
5	Система: Серверная Компоненты и требования к структурам: Стойки (важна последовательность, быстрый произвольный доступ), Серверы (последовательность не важна, уникальность элементов, быстрая выборка)
6	Система: Образовательная программа Компоненты и требования к структурам: Дисциплины (важна последовательность, уникальность элементов), Преподаватели (быстрая сортировка, уникальность элементов)
7	Система: Кинотеатр Компоненты и требования к структурам: Залы (важна последовательность, быстрый произвольный доступ), Места (ключ – номер места/номер ряда, последовательность не важна)
8	Система: Конференция Компоненты и требования к структурам: Секции (ключ – идентификатор секции, важна последовательность), Участники (быстрая сортировка, уникальность элементов)
9	Система: Завод Компоненты и требования к структурам: Помещения (важна последовательность, уникальность элементов), Оборудование (важна последовательность, быстрый произвольный доступ)
10	Система: Книжный магазин Компоненты и требования к структурам: Книжные полки (важна последовательность, уникальность элементов), Книги (важна последовательность, быстрый произвольный доступ)
11	Система: Железнодорожный вокзал Компоненты и требования к структурам: Пути (важна последовательность, быстрый произвольный доступ), Рейсы (важна последовательность, быстрая выборка, уникальность элементов)
12	Система: Компьютер Компоненты и требования к структурам: Устройства ввода(важна последовательность, быстрый произвольный доступ), Устройства вывода (важна последовательность, быстрая вставка/удаление)
13	Система: Банк Компоненты и требования к структурам: Офисы (важна последовательность, уникальность элементов), Банкоматы (важна последовательность, быстрая вставка/удаление)
14	Система: БРИКС Компоненты и требования к структурам: Наблюдатели (быстрая сортировка, уникальность элементов), Саммиты (важна последовательность, уникальность элементов)

15	Система: Строительная компания Компоненты и требования к структурам: Проекты (ключ – идентификатор проекта, важна последовательность), Сотрудники (быстрая сортировка, уникальность элементов)
16	Система: Школа Компоненты и требования к структурам: Кабинеты (важна последовательность, быстрый произвольный доступ), Предметы (важна последовательность, уникальность элементов)
17	Система: Ресторан Компоненты и требования к структурам: Блюда (важна последовательность, быстрая вставка/удаление), Ингредиенты (важна последовательность, быстрый произвольный доступ)
18	Система: Велопрокат Компоненты и требования к структурам: Станции (важна последовательность, уникальность элементов), Велосипеды (важна последовательность, быстрый произвольный доступ)
19	Система: Ванная комната Компоненты и требования к структурам: Мебель (важна последовательность, быстрый произвольный доступ), Сантехника (важна последовательность, быстрая вставка/удаление)
20	Система: Морской порт Компоненты и требования к структурам: Причалы (важна последовательность, быстрый произвольный доступ), Рейсы (важна последовательность, быстрая выборка, уникальность элементов)
21	Система: Компания Компоненты и требования к структурам: Офисы (ключ – адрес офиса, быстрая сортировка), Сотрудники (быстрая сортировка, уникальность элементов)
22	Система: Кухня Компоненты и требования к структурам: Мебель (важна последовательность, быстрый произвольный доступ), Оборудование (важна последовательность, быстрая вставка/удаление)
23	Система: Словарь Компоненты и требования к структурам: Слова (важна последовательность, быстрый произвольный доступ), Определения (важна последовательность, уникальность элементов)
24	Система: Таксопарк Компоненты и требования к структурам: Автомобили (важна последовательность, быстрая вставка/удаление), Водители (быстрая сортировка, уникальность элементов)
25	Система: Магазин электроники Компоненты и требования к структурам: Категории (ключ – название категории, последовательность не важна), Оборудование (важна последовательность, быстрая вставка/удаление)
26	Система: Рюкзак Компоненты и требования к структурам: Отделения (важна последовательность, быстрый произвольный доступ), Вещи (важна последовательность, быстрая вставка/удаление)
27	Система: Торговый центр Компоненты и требования к структурам: Этажи (ключ – номер этажа, важна последовательлность), Магазины (важна последовательность, быстрый произвольный доступ)

28	Система: Олимпиада Компоненты и требования к структурам: Места проведения (ключ – адрес места проведения, важна последовательность), Виды спорта (быстрая сортировка, уникальность элементов)
29	Система: Садово-парковый ансамбль Компоненты и требования к структурам: Зоны (важна последовательность, быстрый произвольный доступ), Фонтаны (ключ – название фонтана, быстрая сортировка)
30	Система: Жилой квартал Компоненты и требования к структурам: Корпуса (важна последовательность, уникальность элементов), Инфраструктура (ключ – наименование объекта инфраструктуры, важна последовательность)
31	Система: Формула 1 Компоненты и требования к структурам: Трассы (быстрая сортировка, уникальность элементов), Команды (важна последовательность, уникальность элементов)
32	Система: Аквапарк Компоненты и требования к структурам: Бассейны (важна последовательность, быстрый произвольный доступ), Сауны (быстрая сортировка, уникальность элементов)
33	Система: Лицензия Компоненты и требования к структурам: Виды деятельности (ключ – наименование вида деятельности, последовательность не важна), Лицензиаты (быстрая сортировка, уникальность элементов)
34	Система: Форум Компоненты и требования к структурам: Темы (важна последовательность, быстрый произвольный доступ), Сообщения (важна последовательность, быстрая вставка/удаление)
35	Система: Факультет Компоненты и требования к структурам: Образовательные программы (важна последовательность, быстрая вставка/удаление), Руководители программ (быстрая сортировка, уникальность элементов)
36	Система: Министерство Компоненты и требования к структурам: Департаменты (ключ – название департамента, быстрая сортировка), Подведомственные организации (быстрая сортировка, уникальность элементов)
37	Система: Платформа онлайн-обучения Компоненты и требования к структурам: Направления подготовки (ключ – код направления, последовательность не важна), Онлайн-курсы (важна последовательность, уникальность элементов)
38	Система: Футбол Компоненты и требования к структурам: Лиги (важна последовательность, уникальность элементов), Клубы (быстрая сортировка, уникальность элементов)
39	Система: Автосервис Компоненты и требования к структурам: Места (последовательность не важна, уникальность элементов), Ремонтное оборудование (важна последовательность, быстрый произвольный доступ)
40	Система: Шкаф Компоненты и требования к структурам: Полки (последовательность не важна, уникальность элементов), Вещи (важна последовательность, быстрый произвольный доступ)

41	Система: Энциклопедия Компоненты и требования к структурам: Категории (ключ – название категории, последовательность не важна), Статьи (последовательность не важна, уникальность элементов)
42	Система: Хранилище данных Компоненты и требования к структурам: Каталоги (ключ – путь каталога, быстрая сортировка), Файлы (важна последовательность, быстрый произвольный доступ)
43	Система: Музей Компоненты и требования к структурам: Залы (важна последовательность, уникальность элементов), Экспонаты (важна последовательность, быстрый произвольный доступ)
44	Система: Фильм Компоненты и требования к структурам: Персонал съемочной площадки (ключ – идентификатора сотрудника, последовательность не важна), Актеры (быстрая сортировка, уникальность элементов)
45	Система: Спортзал Компоненты и требования к структурам: Тренеры (ключ – идентификатора сотрудника, последовательность не важна), Тренажеры (важна последовательность, быстрый произвольный доступ)
46	Система: Европейский союз Компоненты и требования к структурам: Политические центры (важна последовательность, уникальность элементов), Члены (быстрая сортировка, уникальность элементов)
47	Система: Сервис электронной почты Компоненты и требования к структурам: Почтовые ящики (быстрая сортировка, уникальность элементов), Письма (важна последовательность, быстрый произвольный доступ)
48	Система: Медиатека Компоненты и требования к структурам: Музыкальные композиции (важна последовательность, быстрая вставка/удаление), Фильмы (важна последовательность, быстрый произвольный доступ)
49	Система: Автосалон Компоненты и требования к структурам: Автомобили (важна последовательность, быстрая вставка/удаление), Сотрудники (быстрая сортировка, уникальность элементов)
50	Система: Научная лаборатория Компоненты и требования к структурам: Помещения (последовательность не важна, уникальность элементов), Оборудование (важна последовательность, быстрый произвольный доступ)