2. Java_MIDDLE (Коллекции)

- Работа с коллекциями

- Создание обьектов (maps, lists, sets,queue/deque, stack)

- Конструктивные особенности

- Обзор основных методов

- Пример реализации динамического списка на базе LinkedList.

Class Lists:

```
public class Lists {
    public static void main(String[] args) {
        // Динамический список
        List<String> arrayList = new ArrayList<>();

        // Связный список
        List<String> linkedList = new LinkedList<>();

        // Аналогичен связному списку, но все методы синхронизированы
        List<String> vector = new Vector<>();

        // Last in - First out
        List<String> stack = new Stack<>();

        // Синхронизированный список, все операции будут атомарными
        List<String> syncList = Collections.synchronizedList(new ArrayList<>());


        // Реализация своего динамического массива на основе Linked list
        MyLinkedList myLinkedList = new MyLinkedList();
        myLinkedList.add(1);
        myLinkedList.add(2);
        myLinkedList.add(3);
        myLinkedList.add(4);
        myLinkedList.add(10);
        myLinkedList.add(20);
        myLinkedList.add(100);

        System.out.println(myLinkedList);
        System.out.println(myLinkedList.get(4));
        System.out.println(myLinkedList.get(5));
        myLinkedList.remove(2);
        System.out.println(myLinkedList);
        myLinkedList.remove(0);
        System.out.println(myLinkedList);
    }
}
```

Class Maps:

```
public class Maps {
    public static void main(String[] args) {
        // не гарантирует порядок
        Map<Integer, String> hashMap = new HashMap<>();
```

```
        // соблюдается порядок добавления (элементы head, tail)
        Map<Integer, String> linkedHashMap = new LinkedHashMap<>();

        // сортирует по ключу (концепция red-black tree)
        Map<Integer, String> treeMap = new TreeMap<>();

        // hashMap с синхронизированными методами
        Map<String, String> hashTable = new Hashtable<>();

        // synchronizedMap, в параметрах можно указать объект синхронизации
        Map<String, String> synchMap = Collections.synchronizedMap(new
HashMap<>());

        // использование navigableMap
        TreeMap treeMap1 = new TreeMap();
        treeMap1.put(1,"a");
        treeMap1.put(2,"b");
        treeMap1.put(3,"c");
        System.out.println(treeMap1.subMap(1,3).firstKey());
        System.out.println(treeMap1.headMap(2));
        System.out.println(treeMap1.tailMap(3));
        System.out.println(treeMap1.headMap(3).lastKey());


        testMap(hashMap);
        System.out.println("-----------------");
        testMap(linkedHashMap);
        System.out.println("-----------------");
        testMap(treeMap);

    }

    public static void testMap(Map<Integer, String> map) {
        map.put(39, "Bob");
        map.put(32, "Ann");
        map.put(356, "Tom");
        map.put(3572, "Kate");
        map.put(0, "Tim");

        for (Map.Entry<Integer, String> entry : map.entrySet()) {
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }
    }
}
```

Class Queues:

```
public class Queues {
    public static void main(String[] args) {
        Person p1 = new Person(1);
        Person p2 = new Person(2);
        Person p3 = new Person(3);
        Person p4 = new Person(4);

        // Обычная очередь (First in - First out)
        Queue <String> linkedList = new LinkedList<>();

        // Сортирует эелементы
        Queue <String> priorityQueue = new PriorityQueue<>();

        // Двунаправленные очереди
        Deque <String> linkedDeque = new LinkedList<>();
        Deque <String> arrayDeque = new ArrayDeque<>();
```

```
        // Больший функционал по сравнению с обычным queue
        System.out.println(linkedDeque.offer("111"));
        System.out.println(linkedDeque.offerFirst("222"));
        System.out.println(linkedDeque.getLast());
        System.out.println(linkedDeque.pollFirst());

        // Очередь с блокировкой
        Queue<Person> personQueue = new ArrayBlockingQueue<Person>(3);

        System.out.println(personQueue.offer(p3));
        System.out.println(personQueue.offer(p2));
        System.out.println(personQueue.offer(p4));
        System.out.println(personQueue.offer(p1));

        for (Person person : personQueue) {
            System.out.println(person);
            System.out.println(personQueue.remove());
            System.out.println(personQueue.peek());

        }
    }
}
```

Class Sets:

```
public class Sets {
    public static void main(String[] args) {
        Set<String> hashSet = new HashSet<>(); // без порядка
        Set<String> treeSet = new TreeSet<>(); // отсортировано
        Set<String> linkedHashSet = new LinkedHashSet<>(); // сохраняет порядок
добавления

        hashSet.add("Mike");
        hashSet.add("Tom");
        hashSet.add("Tim");
        hashSet.add("Bob");
        hashSet.add("Ann");
        hashSet.add("Tom");
        hashSet.add("Tom");

        for (String name : hashSet) {
            System.out.println(name);
            System.out.println(hashSet.contains("Tom"));
            System.out.println(hashSet.isEmpty());
        }

        System.out.println(hashSet);
        Set<Integer> set1 = new HashSet<>();
        Set<Integer> set2 = new HashSet<>();

        set1.add(0);
        set1.add(1);
        set1.add(2);
        set1.add(3);
        set1.add(4);
        set1.add(5);

        set2.add(2);
        set2.add(3);
        set2.add(4);
        set2.add(5);
        set2.add(6);
        set2.add(7);

        //обьединение
        Set<Integer> union = new HashSet<>(set1); // содержит все элементы set1
```

```
        union.addAll(set2);
        System.out.println(union);

        // пересечение
        Set<Integer> intersection = new HashSet<>(set1);
        intersection.retainAll(set2);
        System.out.println(intersection);

        // разность
        Set<Integer> difference = new HashSet<>(set1);
        difference.removeAll(set2);
        System.out.println(difference);
    }
}
```

Class Stacks:

```
public class Stacks {
    public static void main(String[] args) {
        // Last in - First out
        Stack<Integer> stack = new Stack<>();
        stack.push(5);
        stack.push(3);
        stack.push(1);

        System.out.println(stack.peek());
        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println(stack.search("3"));

        while (!stack.empty()) {
            System.out.println(stack.pop());
        }
    }
}
```

Реализация динамического списка на базе LinkedList:

```
class MyLinkedList {
    Node head;// <---  Узел по умолчанию Null
    int size;

    public void add(int value) {
        Node temp = head;
        if (head == null) {
            this.head = new Node(value);
        } else {                                    // [1] > [2] > [3] > [4] > null
            while (temp.getNext() != null) {
                temp = temp.getNext();
            }
            temp.setNext(new Node(value));
        }
        size++;
    }

    public String toString() {
        int currInt = 0;
        Node temp = head;
        int[] arr = new int[size];
        while (temp != null) {
            arr[currInt++] = temp.getValue();
            temp = temp.getNext();
        }
```

```
        return Arrays.toString(arr);
    }

    public int get(int index) {
        int curIndx = 0;
        Node temp = head;                               // temp
        while (temp != null) {                          // [1]    [2]    [3]    [4]
[5]    [6]    [7]
            if (curIndx == index) {                     // ci=0                  i=3
                return temp.getValue();
            } else {
                temp = temp.getNext();
                curIndx++;
            }
        }
        throw new IllegalArgumentException();
    }

    public void remove(int Index) {
        int curInd = 0;
        Node temp = head;
        if (Index == 0) {
            head = head.getNext();
            size--;
            return;
        }
        while (temp != null) {
            if (curInd + 1 == Index) {
                temp.setNext(temp.getNext().getNext());
                size--;
                return;// [1] [2] [3] [4] [5] [6]
            } else {
                temp = temp.getNext();
                curInd++;
            }
        }
    }

    private static class Node {
        private int value;
        private Node next;

        public Node(int value) {
            this.value = value;
        }

        public int getValue() {
            return value;
        }

        public void setValue(int value) {
            this.value = value;
        }

        public Node getNext() {
            return next;
        }

        public void setNext(Node next) {
            this.next = next;
        }
    }
}
```