Билет 1

Парадигма ООП. Основные принципы ООП и их реализация в Java & C++

Основные принципы: инкапсуляция (атрибуты и поведение объекта объединяются в одном классе, внутренняя реализация объекта скрывается от пользователя, а для работы с объектом предоставляется открытый интерфейс), полиморфизм(код работает с разными типами данных), наследование (позволяет объектам наследовать данные и функциональность у уже существующих типов).

	C++	JAVA
Ключевые слова для создания объектов:	Class, structure	Class, interface
Модификаторы доступа:	private, protected и public	private, protected, public, default
Множественное наследование	Да, есть абстрактные классы и виртуальные функции, подразумевающие наследование и переопределение	Нет, но класс может реализовывать сколько угодно интерфейсов

Понятие структуры данных список. Линейный список. Виды списков и их реализация на Java. Доступ к элементу структуры данных список. Использование списков. Трудоемкость операции со списками.

Список – это динамическая линейная структура данных, в которой каждый элемент ссылается либо только на предыдущий – однонаправленный линейный список, либо на предыдущий и следующий за ним – двунаправленный линейный список.

ArrayList линейный список — реализация изменяемого массива интерфейса List, часть Collection Framework, который отвечает за список (или

динамический массив), расположенный в пакете java.utils. Этот класс реализует все необязательные операции со списком и предоставляет методы управления размером массива, который используется для хранения динамических наборов данных. В основе ArrayList лежит идея динамического массива. А именно, возможность добавлять и удалять элементы, при этом будет увеличиваться или уменьшаться по мере необходимости.

Виды списков ArrayList LinkedList (Класс LinkedList реализует одновременно List и Deque. Это список, в котором у каждого элемента есть ссылка на предыдущий и следующий элементы). В сумме сложность вставки в середину у ArrayList и у LinkedList получается одинаковая — O(n).

Получение информации о типе. Создание экземпляров классов. Вызов методов класса

Билет 2

Получение информации о типе. Создание экземпляров классов. Вызов методов класса

Задачи

Билет 1

```
import java.util.ArrayList;
import java.util.Arrays;

public class MergeArrays {
    public static ArrayList<Integer> alternate(ArrayList<Integer> a, ArrayList<Integer> b) {
        ArrayList<Integer> result = new ArrayList<>();
        int size = a.size() + b.size();
        int j = 0;
        for (int i = 0; i < size; i++) {
            if (j < a.size()) {
                result.add(a.get(j));
            }
            if (j < b.size()) {
                result.add(b.get(j));
            }
            j++;
        }
        return result;
    }

    public static void main(String[] args) {
        ArrayList<Integer> a = new ArrayList<>(Arrays.asList(1, 2, 3, 4));
        ArrayList<Integer> b = new ArrayList<>(Arrays.asList(5, 6, 7, 8, 9, 10));
        System.out.println(alternate(a, b));
    }
}
```

Билет 2

```
/*
 * Метод reverse принимает на вход Map от целых чисел к строкам и возвращает Map, в которой ключи и значения поменяны местами.
 */

import java.util.HashMap;
import java.util.Map;

public class ReversMap {
    public static void main(String[] args) {
        Map<Integer, String> map = new HashMap<>();
        map.put(1, "One");
        map.put(2, "Two");
        map.put(3, "Three");
        map.put(4, "Four");
        map.put(5, "Five");
        map.put(6, "Six");
        map.put(7, "Seven");
        map.put(8, "Eight");
        map.put(9, "Nine");
        System.out.println(map);
        System.out.println(reverse(map));
    }

    public static Map<String, Integer> reverse(Map<Integer, String> map) {
        Map<String, Integer> result = new HashMap<>();
        for (Map.Entry<Integer, String> entry : map.entrySet()) {
            result.put(entry.getValue(), entry.getKey());
        }
        return result;
    }

}
```

Билет 3

```
import java.util.*;

public class ForIsUnique {
    public static boolean isUnique(Map<String, String> map) {
        for (String key : map.keySet()) {
            for (String key2 : map.keySet()) {
                if (!Objects.equals(key, key2) && map.get(key).equals(map.get(key2))) {
                    return false;
                }
            }
        }
        return true;
    }

    public static boolean isUnique2(Map<String, String> map) {
        Set<String> tmp = new HashSet<>(map.values());
        return (tmp.size() == map.size());
    }

    public static void main(String[] args) {
        Map<String, String> map = new HashMap<>();
        map.put("Marty", "Stepp");
        map.put("Stuart", "Reges");
        map.put("Jessica", "Miller");
        map.put("Amanda", "Camp");
        map.put("Hal", "Perkins");

        System.out.println(isUnique(map));
        System.out.println(isUnique2(map));
        map.put("Kendrick", "Perkins");
        System.out.println(isUnique(map));
        System.out.println(isUnique2(map));
    }
}
```

Билет 4

```
/*
 * Метод hasOdd принимает множество целых чисел и возвращает true, если в множестве есть нечетное число.
 * Если в множестве нет нечетных чисел, то метод возвращает false.
 */

import java.util.HashSet;
import java.util.Set;

public class HasOddSet {
    /**
     * Метод hasOdd принимает множество целых чисел и возвращает true, если в множестве есть нечетное число
     * @param set
     * @return true, если в множестве есть нечетное число
     */
    public static boolean hasOdd(Set<Integer> set) {
        for (Integer i : set) {
            if (i % 2 != 0) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        Set<Integer> set1 = new HashSet<>();
        set1.add(2);
        set1.add(4);
        set1.add(6);

        Set<Integer> set2 = new HashSet<>();
        set2.add(1);
        set2.add(3);
        set2.add(5);

        System.out.println(hasOdd(set1));
        System.out.println(hasOdd(set2));
    }
}
```

Билет 5

```
/*
 * Напишите метод rarest, который принимает Map, ключи которого являются строками, а значения являются целыми числами.
 * Метод возвращает челочисленное значение, которое встречаеся в словаре наименьшее количество раз.
 * Если словарь пуст, скгенерируется исключение IllegalArgumentException.
 */

import java.util.HashMap;
import java.util.Map;

public class ForRarest {
    public static int rarest(Map<String, Integer> map) {
        if (map.isEmpty()) { // Проверка на пустоту
            throw new IllegalArgumentException();
        }
        int min = Integer.MAX_VALUE;
        int result = 0;

        // Перебираем все значения в словаре
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            int count = 0;
            // Считаем количество вхождений каждого значения
            for (Map.Entry<String, Integer> entry2 : map.entrySet()) {
                if (entry.getValue().equals(entry2.getValue())) {
                    count++;
                }
            }

            // Если количество вхождений меньше минимального, то обновляем минимальное значение
            if (count < min) {
                min = count;
                result = entry.getValue();
            }
```

```
        }
        return result;
    }

    public static void main(String[] args) {

        Map<String, Integer> map = new HashMap<>();
        map.put("Alissia", 22);
        map.put("Char", 25);
        map.put("Dan", 25);
        map.put("Jeff", 20);
        map.put("Keasy", 20);
        map.put("Kim", 20);
        map.put("Morgan", 20);
        map.put("Rayn", 25);
        map.put("Stef", 22);

        System.out.println(rarest(map));
    }

}
```

Билет 6

```
/*
 * Написать метод guavaSort, который принимает на вход список строк и возвращает отсортированный список строк в порядке возрастания.
 * Алгоритм должен использовать FJC для MultiMap для реализации варианта алгоритма блочной соортировки, который будет
 * работать со строками. Использовать коллекцию FJC дя подсчёта вхождений строк, аналогично тому, как это происходит в
 * блочной сортировке.
 */

import java.util.Arrays;
import java.util.TreeMap;
import java.util.TreeSet;

public class ToGuavaSort {


    public static void main(String[] args) {
        String[] arr = {"Farm", "Zoo", "Car", "Apple", "Bee", "Dog", "Golf", "Zoo", "Zoo", "Bee", "Apple"};

        // Тестирование функции
        System.out.println(Arrays.toString(guavaSort(arr)));

    }

    /**
     * Метод guavaSort принимает на вход список строк и возвращает отсортированный список строк в порядке возрастания.
     */
    public static String[] guavaSort(String[] arr) {
        TreeMap<String, Integer> stringCountMap = new TreeMap<String, Integer>(); // Создаём TreeMap для подсчёта вхождений строк
        for (String s : arr) { // Проходим по массиву строк
            if (stringCountMap.containsKey(s)) { // Если строка уже есть в TreeMap, то увеличиваем счётчик на 1
                stringCountMap.put(s, stringCountMap.get(s) + 1);
            } else {
                stringCountMap.put(s, 1); // Если нет, то добавляем строку в TreeMap и устанавливаем счётчик в 1
            }
        }

        TreeSet<String> sortedSet = new TreeSet<String>(stringCountMap.keySet()); // Создаём TreeSet из ключей TreeMap
        String[] sortedArr = new String[arr.length]; // Создаём массив для отсортированных строк
        int i = 0;
        for (String s : sortedSet) {
            for (int j = 0; j < stringCountMap.get(s); j++) {
                sortedArr[i] = s;
                i++;
            }
        }

        return sortedArr;
    }
}
```

Билет 7

```
/**
 * LinkedIntList - класс, который представляет собой односвязный список целых чисел.
 * Класс содержит внутренний класс ListNode, который представляет собой узел списка.
 * Написать метод removeAll, который удаляет из списка все элементы, которые содержаться в другом списке.
 * Методо должен эффетикто удалять за время lon N, где N - количество элементов в списке.
 */

public class LinkedIntList {
    public ListNode front;

    public LinkedIntList() {
        front = null;
    }

    public void add(int value) {
        if (front == null) {
            front = new ListNode(value);
        } else {
            ListNode current = front;
            while (current.next != null) {
                current = current.next;
            }
            current.next = new ListNode(value);
        }
    }

    public void removeAll(LinkedIntList list) {
        ListNode current = front;
        ListNode prev = null;
        ListNode current2 = list.front;

        while (current != null && current2 != null) {
            if (current.data == current2.data) { // Если элементы равны
                if (prev == null) { // Если это первый элемент
                    front = front.next; // Первый элемент становится вторым
                    current = front; // Текущий элемент становится первым
                } else { // Если это не первый элемент
                    prev.next = current.next; // Предыдущий элемент становится следующим за текущим
                    current = current.next; // Текущий элемент становится следующим
                }
            } else if (current.data < current2.data) { // Если текущий элемент меньше элемента из второго списка
                prev = current; // Предыдущий элемент становится текущим
                current = current.next; // Текущий элемент становится следующим
            } else { // Если текущий элемент больше элемента из второго списка
                current2 = current2.next; // Текущий элемент из второго списка становится следующим
            }
        }
    }


    public static class ListNode {
        public int data;
        public ListNode next;

        public ListNode(int data) {
            this.data = data;
        }
    }

    public void print() {
        ListNode current = front;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        LinkedIntList list1 = new LinkedIntList();
        // 1-3-5-7
//        list1.add(1);
//        list1.add(3);
```

```
//        list1.add(5);
//        list1.add(7);

        //1-2-3-4-5-6
        list1.add(1);
        list1.add(2);
        list1.add(3);
        list1.add(4);
        list1.add(5);
        list1.add(6);


        LinkedIntList list2 = new LinkedIntList();
        // 1-2-3-4-5
//        list2.add(1);
//        list2.add(2);
//        list2.add(3);
//        list2.add(4);
//        list2.add(5);

        // 1-3
        list2.add(1);
        list2.add(3);

        list1.print();
        list2.print();
        list1.removeAll(list2);
        list1.print();
//        list2.print();
    }
}
```

Билет 8

```
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

public class Mystery {
    public static void mystery(Map<String, String> map) {
        Map<String, String> result = new TreeMap<String, String>();
        for (String key : map.keySet()) {
            if (key.compareTo(map.get(key)) < 0) {
                result.put(key, map.get(key));
            } else {
                result.put(map.get(key), key);
            }
        }
        System.out.println(result);
    }

    public static void main(String[] args) {
        Map<String, String> map = new HashMap<>();
        map.put("two", "deux");
        map.put("five", "cinq");
        map.put("one", "un");
        map.put("three", "trois");
        map.put("four", "quatre");
        mystery(map);
    }
}
```

Билет 9

```
/*
 * Реализуйте класс LinkedIntList, который представляет собой односвязный список целых чисел.
 * Реализовать метод removeDuplicates, который удаляет все дубликаты из списка.
 *
 * Нельзя использовать дополнительные структуры данных. Нужно решать задачу поменяв местами ссылки.
 */

public class LinkedIntList {
    private ListNode head;
```

```
    private int size;

    public LinkedIntList() {
        head = null;
        size = 0;
    }

    public static class ListNode {
        public int data;
        public ListNode next;

        public ListNode(int data) {
            this.data = data;
        }
    }

    public void add(int value) {
        if (head == null) {
            head = new ListNode(value);
            return;
        }
        ListNode current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = new ListNode(value);
    }

    public void removeDuplicates() {
        ListNode current = head; // Текущий элемент
        while (current != null) { // Пока не дошли до конца списка
            ListNode next = current.next; // Следующий элемент
            ListNode prev = current; // Предыдущий элемент
            while (next != null) {
                if (current.data == next.data) { // Если текущий элемент равен следующему
                    prev.next = next.next; // То пропускаем следующий элемент
                } else {
                    prev = next; // Иначе переходим к следующему элементу
                }
                next = next.next; // Переходим к следующему элементу
            }
            current = current.next;
        }
    }

    public void print() {
        ListNode current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        LinkedIntList list = new LinkedIntList();
        list.add(1);
        list.add(4);
        list.add(4);
        list.add(1);
        list.add(2);
        list.add(2);
        list.add(3);
        list.add(3);
        list.add(4);
        list.add(4);
        list.add(5);
        list.add(5);
        list.print();
        list.removeDuplicates();
        list.print();
    }
}
```

Билет 10

Билет 11

```
/*
 * Реализуйте класс LinkedIntList, который представляет собой односвязный список целых чисел.
 * Реализовать метод firstLast, который перемещвает первый элемент списка в конец.
 */

public class LinkedIntList {
    /**
     * Класс ListNode представляет узел односвязного списка
     */
    public static class ListNode {
        public int data;
        public ListNode next;

        public ListNode(int data) {
            this.data = data;
        }
    }


    private ListNode head; // Головной/первый элемент

    /**
     * Метод добавляет элемент в конец односвязного списка
     *
     * @param value
     */
    public void add(int value) {

        // Если в списке нет элементов
        if (head == null) {
            head = new ListNode(value);
            return;
        }

        // Если в списке есть элементы, то нужно найти последний элемент
        ListNode current = head;
        while (current.next != null) {
            current = current.next;
        }
        // Когда мы дошли до последнего элемента, current.next == null и мы можем добавить новый элемент
        current.next = new ListNode(value);
    }


    /**
     * Метод перемещает первый элемент списка в конец
     */
    public void firstLast() {
        // Если в списке нет элементов или в списке только один элемент, то ничего делать не нужно
        if (head == null || head.next == null) {
            return;
        }

        // Находим последний элемент
        ListNode last = head;
        while (last.next != null) {
            last = last.next;
        }

        // Перемещаем первый элемент в конец
        last.next = head; // Последний элемент теперь ссылается на первый элемент
        head = head.next; // Теперь новый первый элемент - второй элемент
        last.next.next = null;
    }

    /**
     * Метод выводит элементы односвязного списка в консоль
     */
```

```
    public void print() {
        ListNode current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        LinkedIntList list = new LinkedIntList();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);
        list.print();
        list.firstLast();
        list.print();
    }


}
```

Билет 12

```
public class LinkedIntList {

    ListNode head;

    public void add(int data) {
        if (head == null) {
            head = new ListNode(data);
        } else {
            ListNode current = head;
            while (current.next != null) current = current.next;
            current.next = new ListNode(data);
        }
    }

    public void insert(int index, int data) {
        if (head == null) {
            throw new IndexOutOfBoundsException();
        }
        int i = 0;
        ListNode current = head;
        while (current.next != null && i != index) {
            current = current.next;
            i++;
        }
        if (i == index) {
            current.next = new ListNode(data, current.next);
        } else {
            throw new IndexOutOfBoundsException();
        }

    }

    public void print() {
        ListNode current = head;
        while (current != null) {
            System.out.printf(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static class ListNode {
        public int data;
        public ListNode next;

        public ListNode() {
        }

        public ListNode(int data) {
```

```
        this.data = data;
    }

    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}

public static void main(String[] args) {
    LinkedIntList list = new LinkedIntList();
    list.add(1);
    list.add(2);
    list.add(3);
    list.add(4);
    list.add(5);
    list.print();

    list.insert(2, 9);
    list.print();

}
}
```

Билет 13

```
```

Билет 14

```
/*
 * Метод equals(), который принимамет 2 стека и возращает true, если они равны.
 */

import java.util.Objects;
import java.util.Stack;

public class ToEquals {
    public static boolean equals(Stack<Integer> stack1, Stack<Integer> stack2) {
        Stack<Integer> tmp = new Stack<>();
        if (stack1.size() == stack2.size()) {
            while (!stack1.isEmpty()) {
                if (Objects.equals(stack1.peek(), stack2.peek())) {
                    tmp.push(stack1.pop());
                    stack2.pop();
                } else {
                    while (!tmp.isEmpty()) {
                        stack1.push(tmp.peek());
                        stack2.push(tmp.pop());
                    }
                    return false;
                }
            }
            while (!tmp.isEmpty()) {
                stack1.push(tmp.peek());
                stack2.push(tmp.pop());
            }
            return true;
        } else {
            return false;
        }
    }

    public static void main(String[] args) {
        Stack<Integer> stack1 = new Stack<>();
        Stack<Integer> stack2 = new Stack<>();
        stack1.push(1);
        stack1.push(2);
        stack1.push(3);
        stack2.push(1);
        stack2.push(2);
        stack2.push(3);
```

```
        System.out.println(equals(stack1, stack2));
    }
}
```

Билет 15

```
/*
 * Метод splitStack, который принимает стек и перемещвет элементы так, что бы в стеке сначала шли отрицательные
 * элементы, а потом положительные. Порядок элементов внутри групп не важен. Можно использовать только одну
 * дополнительную очередь.
 */

import java.util.ArrayDeque;
import java.util.Queue;
import java.util.Stack;

public class SplitStack {
    public static void splitStack(Stack<Integer> stack) {
        ArrayDeque<Integer> queue = new ArrayDeque<>();
        while (!stack.isEmpty()) {
            Integer i = stack.pop();
            if (i < 0) {
                queue.addFirst(i);
            } else {
                queue.addLast(i);
            }
        }
        while (!queue.isEmpty()) {
            stack.push(queue.pop());
        }
    }

    public static void splitStack2(Stack<Integer> stack) {
        Queue<Integer> queue = new ArrayDeque<>();
        int size = stack.size();
        while (!stack.isEmpty()) queue.add(stack.pop());
        for (int i = 0; i < size; i++) {
            if (queue.peek() < 0) {
                stack.push(queue.poll());
            } else {
                queue.add(queue.poll());
            }
        }
        while (!queue.isEmpty()) {
            stack.push(queue.poll());
        }
    }

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(-2);
        stack.push(3);
        stack.push(-4);
        stack.push(5);
        stack.push(-6);
        stack.push(7);
        stack.push(-8);
        stack.push(9);
        stack.push(-10);
        System.out.println(stack);
        splitStack(stack);
        System.out.println(stack);
    }
}
```

Билет 16

```
/*
 * Написать метод, который принимает на вход два стека и возвращает стек, в котором содержатся элементы из обоих стеков.
 * В качестве вспомогательной структуры данных можно использовать только одну очередь
 */
```

```
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Queue;
import java.util.Stack;

public class StackCopier {
    public static Stack<Integer> copyStack(Stack<Integer> stack) {
        Stack<Integer> result = new Stack<>();
        ArrayDeque<Integer> queue = new ArrayDeque<>();
        while (!stack.isEmpty()) {
            queue.addFirst(stack.pop());
        }
        while (!queue.isEmpty()) {
            int value = queue.remove();
            result.push(value);
            stack.push(value);
        }
        return result;
    }

    public static Stack<Integer> copyStack2(Stack<Integer> stack) {
        Stack<Integer> result = new Stack<>();
        Queue<Integer> queue = new ArrayDeque<>();
        while (!stack.isEmpty()) queue.add(stack.pop());
        while (!queue.isEmpty()) stack.push(queue.poll());
        while (!stack.isEmpty()) queue.add(stack.pop());
        while (!queue.isEmpty()) {
            int value = queue.poll();
            result.push(value);
            stack.push(value);
        }
        return result;
    }


    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        stack.push(6);
        stack.push(7);
        stack.push(8);
        stack.push(9);
        stack.push(10);
        System.out.println(stack);
        Stack<Integer> copy = copyStack(stack);
        System.out.println(stack);
        System.out.println(copy);

        Stack<Integer> copy2 = copyStack2(stack);
        System.out.println(stack);
        System.out.println(copy);
    }
}
```

Билет 17

```

```

Билет 18

```
public class LongestSortedSequence {
    private int [] elementData;
    private int size;

    public LongestSortedSequence(int[] elementData) {
        this.elementData = elementData;
        this.size = elementData.length;
```

```
    }

    public int getLongestSortedSequence() {
        int max = 1; // Максимальная длина последовательности
        int current = 1; // Текущая длина последовательности
        for (int i = 1; i < size; i++) { // Проходим по всем элементам массива
            if (elementData[i] > elementData[i - 1]) { // Если текущий элемент больше предыдущего
                current++; // Увеличиваем текущую длину последовательности
            } else {
                max = Math.max(max, current); // Сохраняем максимальную длину последовательности
                current = 1;
            }
        }
        if (current > max) {
            max = current;
        }
        return max;
    }

    public void print() {
        for (int i = 0; i < size; i++) {
            System.out.print(elementData[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        // Рандомный массив
        int[] array = new int[10];
        for (int i = 0; i < array.length; i++) {
            array[i] = (int) (Math.random() * 10);
        }
        LongestSortedSequence longestSortedSequence = new LongestSortedSequence(array);

        // Выводим массив
        longestSortedSequence.print();
        System.out.println(longestSortedSequence.getLongestSortedSequence());

    }
}
```

Билет 19

```
import java.util.ArrayDeque;
import java.util.Queue;
import java.util.Stack;

public class ForReverseHalf {
    public static Queue<Integer> reverseHalf(Queue<Integer> queue) {
        Stack<Integer> stack = new Stack<>();
        int size = queue.size();

        // Добавляем в стек элементы с нечётными индексами
        for (int i = 0; i < size; i++) {
            if (i % 2 != 0) {
                stack.push(queue.remove());
            } else {
                queue.add(queue.remove());
            }
        }

        // Добавляем в очередь элементы из стека и из очереди
        for (int i = 0; i < size; i++) {
            if (i % 2 != 0) {
                queue.add(stack.pop());
            } else {
                queue.add(queue.remove());
            }
        }

        return queue;
    }

    public static void main(String[] args) {
        Queue<Integer> queue = new ArrayDeque<>();
        queue.add(1);
```

```
        queue.add(2);
        queue.add(3);
        queue.add(4);
        queue.add(5);
        queue.add(6);
        queue.add(7);
        queue.add(8);

        System.out.println(queue);
        System.out.println(reverseHalf(queue));
    }
}
```

Билет 20

```
```

Билет 21

```
import org.jetbrains.annotations.NotNull;

import java.util.Objects;

public class Cat implements Comparable<Cat> {

    private final String name;

    public Cat(String name) {
        this.name = name;
    }

    @Override
    public int compareTo(@NotNull Cat o) {
        return name.compareTo(o.name);
    }
```

```
public class Searcher {
    public static int search(Comparable[] array, Object obj) {
        for (int i = 0; i < array.length; i++) {
            if (array[i].compareTo(obj) == 0) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        Comparable[] array = new Comparable[3];
        array[0] = new Cat("Murzik");
        array[1] = new Cat("Barsik");
        array[2] = new Cat("Vaska");

        System.out.println(search(array, new Cat("Murzik")));
        System.out.println(search(array, new Cat("Barsik")));
        System.out.println(search(array, new Cat("Vaska")));
        System.out.println(search(array, new Cat("Tom")));
    }
}
```

Билет 22

```
import org.jetbrains.annotations.NotNull;

public class Cat implements Comparable<Cat> {
    private String name;
    private int age;

    public Cat(String name, int age) {
        this.name = name;
        this.age = age;
```

```
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void print() {
        System.out.println("Cat: " + name + ", " + age);
    }

    @Override
    public String toString() {
        return "Cat{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public int compareTo(@NotNull Cat o) {
        return name.compareTo(o.name);
    }
}
```

```
public class Sorter {
    public static void sort(Comparable [] array) {
        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < array.length - 1; j++) {
                if (array[j].compareTo(array[j + 1]) > 0) {
                    Comparable temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        Comparable[] array = new Comparable[3];
        array[0] = new Cat("Murzik", 3);
        array[1] = new Cat("Barsik", 2);
        array[2] = new Cat("Vaska", 5);

        Sorter.sort(array);

        for (Comparable sortable : array) {
            System.out.println(sortable);
        }
    }
}
```

Билет 23

Билет №23
-----ЗАДАНИЕ 1------
Расширение классов.
Порядок создания экземпеляра дочернего класса.

Расширение классов

С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого. Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника

ключевое слово extends, после которого идет имя базового класса.

С ключевым словом implements связано чуть больше хитростей. Слово «имплементировать» можно понимать, как «реализовывать», а в тот самый момент, когда возникает слово «реализовывать», где-то недалеко появляются интерфейсы. Так вот конструкция public class Door implements Openable означает, что класс дверь реализует интерфейс «открывающийся». Следовательно класс должен переопределить все методы интерфейса. Главная фишка в том, что можно реализовывать сколь угодно много интерфейсов.

Порядок создания дочернего Экземпляра дочернего класса java

1)Первое что произойдет — проинициализируются статические переменные класса.
2)После инициализации статических переменных класса-предка инициализируются статические переменные класса-потомка
3) Третьими по счету будут инициализированы нестатические переменные класса-предка
4) Конструктор базового класса.
5) инициализация нестатических полей класса-потомка
6) Вызывается конструктор дочернего класса

-----ЗАДАНИЕ 2------
Понятие серилизации и её использование в ООП программах.

Сериализация
Сериализация (Serialization) — это процесс, который переводит объект в последовательность байтов или строку, по которой затем его можно полностью восстановить. Зачем это нужно? Дело в том, при обычном выполнении программы максимальный срок жизни любого объекта известен — от запуска программы до ее окончания. Сериализация позволяет расширить эти рамки и «дать жизнь» объекту также между запусками программы

```
public class Circle implements Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }
}
```

```
public class Rectangle implements Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    public double getArea() {
        return width * height;
    }

    @Override
    public double getPerimeter() {
        return 2 * (width + height);
    }
}
```

```
public interface Shape {
    double getArea();
    double getPerimeter();
}
```

```
public class ShapeFactory {
    public static Shape createShape(Shapes shape, double... args) {
        switch (shape) {
            case CIRCLE:
                return new Circle(args[0]);
            case RECTANGLE:
                return new Rectangle(args[0], args[1]);
            default:
                throw new IllegalArgumentException("Unknown shape");
        }
    }
```

```
    public static void main(String[] args) {
        Shape circle = ShapeFactory.createShape(Shapes.CIRCLE, 5);
        System.out.println("Circle area: " + circle.getArea());
        System.out.println("Circle perimeter: " + circle.getPerimeter());

        Shape rectangle = ShapeFactory.createShape(Shapes.RECTANGLE, 5, 10);
        System.out.println("Rectangle area: " + rectangle.getArea());
        System.out.println("Rectangle perimeter: " + rectangle.getPerimeter());
    }
}
```

```
public enum Shapes {
    CIRCLE,
    RECTANGLE,
}
```

Билет 24

```
public class INum extends Num {
    private int iNum;

    public INum(int num, int iNum) {
        super(num);
        this.iNum = iNum;
    }

    public int getINum() {
        return iNum;
    }

    public void setINum(int iNum) {
        this.iNum = iNum;
    }

    @Override
    public void print() {
        System.out.println("Num: " + getNum() + ", iNum: " + iNum);
    }
}
```

```
public class Num {
    private int num;

    public Num(int num) {
        this.num = num;
    }

    public int getNum() {
        return num;
    }

    public void setNum(int num) {
        this.num = num;
    }

    public void print() {
        System.out.println("Num: " + num);
    }
}
```

```
public class NumFabric {
    public static Num getNum(int num, int iNum) {
        return new INum(num, iNum);
    }

    public static Num getNum(int num) {
        return new Num(num);
```

```
    }

    public static void main(String[] args) {
        Num num = getNum(1);
        num.print();
        num = getNum(2, 3);
        num.print();
    }
}
```

Билет 25

```
import java.util.ArrayList;
import java.util.List;

public class StackOnList<T> {

    private final List<T> list;

    public StackOnList() {
        this.list = new ArrayList<>();
    }

    public void push(T element) {
        list.add(element);
    }

    public T pop() {
        return list.remove(list.size() - 1);
    }

    public T peek() {
        return list.get(list.size() - 1);
    }

    public boolean isEmpty() {
        return list.isEmpty();
    }

    public int size() {
        return list.size();
    }

    public void clear() {
        list.clear();
    }

    public static void main(String[] args) {
        StackOnList<Integer> stack = new StackOnList<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println(stack.pop());
    }
}
```

Билет 26

```
import java.util.List;

public class ArrayListOnList<T> {

    private T [] array;

    public ArrayListOnList() {
        this.array = null;
    }
```

```
    public void add(T element) {
        if (array == null) {
            array = (T[]) new Object[1];
            array[0] = element;
        } else {
            T[] newArray = (T[]) new Object[array.length + 1];
            System.arraycopy(array, 0, newArray, 0, array.length);
            newArray[array.length] = element;
            array = newArray;
        }
    }

    public T get(int index) {
        return array[index];
    }

    public void remove(int index) {
        T[] newArray = (T[]) new Object[array.length - 1];
        System.arraycopy(array, 0, newArray, 0, index);
        System.arraycopy(array, index + 1, newArray, index, array.length - index - 1);
        array = newArray;
    }

    public int size() {
        return array.length;
    }

    public static void main(String[] args) {

    }



}
```

Билет 27

```
public class ArrayListOnArray<T> {
    private Object[] array; // Массив для хранения элементов
    private int size; // Количество элементов в списке

    public ArrayListOnArray() {
        array = new Object[10]; // Создаем массив на 10 элементов
    }

    public void add(T element) {
        if (size == array.length) { // Если массив заполнен
            Object[] newArray = new Object[array.length * 2]; // Создаем новый массив в 2 раза больше
            System.arraycopy(array, 0, newArray, 0, array.length); // Копируем в него все элементы
            array = newArray; // Присваиваем ссылку на новый массив
        }
        array[size++] = element; // Добавляем элемент в массив
    }

    public T get(int index) {
        if (index < 0 || index >= size) { // Если индекс выходит за границы массива
            throw new IndexOutOfBoundsException(); // Выбрасываем исключение
        }
        return (T) array[index]; // Возвращаем элемент
    }

    public int size() {
        return size;
    }

    public static void main(String[] args) {
        ArrayListOnArray<String> list = new ArrayListOnArray<>(); // Создаем список
        list.add("Hello"); // Добавляем в него элементы
        list.add("World");
        list.add("!");

        // Выводим все элементы списка
        for (int i = 0; i < list.size(); i++) {
            System.out.println(list.get(i));
        }
```

```
    }

}
```

Билет 28

```
public class SumOfNum {
    public static int sumOfNum(int n) {
        if (n == 0) return 0;
        return n % 10 + sumOfNum(n / 10);
    }

    public static void main(String[] args) {
        System.out.println(sumOfNum(1234));
        System.out.println(sumOfNum(4321));
        System.out.println(sumOfNum(100000));
    }
}
```

Билет 29

```
import java.util.LinkedList;

public class StackOnArray<T> {
    private Object[] array; // Массив для хранения элементов
    private int size; // Количество элементов в стеке

    public StackOnArray() {
        array = new Object[10]; // Создаем массив на 10 элементов
    }

    public void push(T element) {
        if (size == array.length) { // Если массив заполнен
            Object[] newArray = new Object[array.length * 2]; // Создаем новый массив в 2 раза больше
            System.arraycopy(array, 0, newArray, 0, array.length); // Копируем в него все элементы
            array = newArray; // Присваиваем ссылку на новый массив
        }
        array[size++] = element; // Добавляем элемент в массив
    }

    public T pop() {
        if (size == 0) { // Если стек пуст
            throw new IndexOutOfBoundsException(); // Выбрасываем исключение
        }
        return (T) array[--size]; // Возвращаем элемент
    }

    public int size() {
        return size;
    }

    public static void main(String[] args) {
        StackOnArray<String> stack = new StackOnArray<>(); // Создаем стек
        stack.push("Hello"); // Добавляем в него элементы
        stack.push("World");
        stack.push("!");

        // Выводим все элементы стека
        for (int i = 0; i < stack.size(); i++) {
            System.out.println(stack.pop());
        }
    }

}
```

Билет 30

```
public class PowTwo {
```

```
    public static void isPow(int n) {
        if (n != 0 && (n & (n - 1)) == 0) {
            System.out.println("YES");
        } else {
            System.out.println("NO");
        }
    }

    public static void main(String[] args) {
        isPow(0);
        isPow(2);
        isPow(7);
        isPow((int) Math.pow(2, 30));
    }
}
```