Билет 1

Напишите метод под названием alternate, который принимает два списка целых чисел в качестве параметров и возвращает новый список, содержащий чередующиеся элементы из двух списков, в следующем порядке:

- **Первый элемент из первого списка**
- **Первый элемент из второго списка**
- **Второй элемент из первого списка**
- **Второй элемент из второго списка**
- **Третий элемент из первого списка**
- **Третий элемент из второго списка**

Если списки не содержат одинаковое количество элементов, оставшиеся элементы из более длинного списка должны быть расположены последовательно в конце. Например, для первого списка (1, 2, 3, 4, 5) и второго списка (6, 7, 8, 9, 10, 11, 12) вызов alternate (list1, list2) должен вернуть список, содержащий (1, 6, 2, 7, 3, 8, 4, 9, 5, 10, 11, 12). Не изменяйте передаваемые списки параметров.

```
public static List<Integer> alternate(List<Integer> firstList, List<Integer> secondList) {
    List<Integer> finishList = new ArrayList<>();

    for (int i = 0; i < Math.max(firstList.size(), secondList.size()); i++) {
       if (firstList.size() >= i + 1) {
          finishList.add(firstList.get(i));
       }

       if (secondList.size() >= i + 1) {
          finishList.add(secondList.get(i));
       }
    }

    return finishList;
  }
```

Билет 2

Напишите метод reverse, который принимает Map от целых чисел к строкам в качестве параметра и возвращает новый Map из строк к целым числам, который является «зеркальным отображением» оригинальной структуры. Риверс исходного Map определяется здесь как новый Map, который использует значения из оригинала в качестве своих ключей и ключи от оригинала в качестве своих значений. Поскольку значения Map не обязательно должны быть уникальными, а его ключи обязательно должны быть уникальными, допустимо иметь любой из исходных ключей в результате в качестве значения. Другими словами, если исходный словарь имеет пары (k1, v) и (k2, v), то новый словарь должен содержать либо пару (v, k1), либо (v, k2).

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

public class Num_2 {
```

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

    public static void main(String[] args) {
        Map<Integer, String> firstMap = new HashMap<>();

        firstMap.put(1, "Akhmed");
        firstMap.put(2, "Mansur");
        firstMap.put(3, "Magomed");
        firstMap.put(4, "Rashid");

        System.out.println(firstMap);
        System.out.println(reverse(firstMap));
    }
}
```

Билет 3

Напишите метод isUnique, который принимает Map <string, string> в качестве параметра и возвращает true, если никакие два ключа не отображаются на одно и то же значение (и false, если любые два или более ключа соответствуют одному и тому же значению). Например, вызов вашего метода на следующем словаре вернет true:
{Marty=Stepp, Stuart=Reges, Jessica=Miller, Amanda=Camp, Hal=Perkins}
Вызов его на следующем словаре вернул бы false из-за двух отображений для Perkins и Reges:
{Kendrick=Perkins, Stuart=Reges, Jessica=Miller, Bruce=Reges, Hal=Perkins}
Пустой словарь считается уникальным, поэтому ваш метод должен возвращать true, если передастся пустой словарь

```
public static boolean isUnique(Map<String, String> map) {
      return (new TreeSet<>(map.values()).size() == map.values().size());
}
```

Билет 4

Напишите метод hasOdd, который принимает множество (Set) целых чисел в качестве параметра и возвращает true, если набор содержит хотя бы одно нечетное целое число, и false в противном случае. Если передано пустое множество, ваш метод должен вернуть false

```
import java.util.Set;
import java.util.TreeSet;

public class Main {
    public static void main(String[] args) {
        Set<Integer> set = new TreeSet<>();
        System.out.println(hasOdd(set));
```

```
    }
    public static boolean hasOdd(Set<Integer> set) {
        for (Integer val : set) {
            if (val % 2 != 0) {
                return true;
            }
        }
        return false;
    }
}
```

Билет 5

Напишите метод rarest, который принимает map, ключи которого являются строками, а значения являются целыми числами, в качестве параметра. метод возвращает целочисленное значение, которое встречается словаре наименьшее количество раз. если словарь не пуст, то верните меньшее целочисленное значение. Если словарь пуст, сгенерируйте исключение. например, предположим, что словарь содержит сопоставления студентов(строки) и их возраста (целые числа). ваш метод вернет наименее часто встречающийся возраст. рассмотрим переменную словаря m, содержащую следующие пары ключ / значение: {alyssa=22, char=25, dan=25, jeff=20, kasey=20, kim=20, mogran=25, ryan=25, stef= 22}

три человека имеют возраст 20 лет (джефф. кейси и ким), и два человека - 22 года (алисса стеф), и четыре человека - 25 лет (чар, дэн, могран и райан). таким образом, метод rarest (м) возвращает 22, потому что только два человека имеют этот возраст.

```
import java.util.*;

public class Main {
    public static void main(String[] args) throws Exception {
            Map<String, Integer> map1 = new HashMap<String, Integer>();
            map1.put("Alyssa",22);
            map1.put("Char",25);
            map1.put("Den",25);
            map1.put("Jeff",20);
            map1.put("Kasey",20);
            map1.put("Kim",20);
            map1.put("Morgan",25);
            map1.put("Ryan",25);
            map1.put("Stef",22);
            System.out.println(rarest(map1));
    }

    public static Integer rarest(Map<String, Integer> map) throws Exception {
            if (map.isEmpty())
                throw new Exception();
            HashMap<Integer, Integer> numOfValues = new HashMap<>(); // значение : количество
вхождений
            for (Integer i : map.values()) {
                if (!numOfValues.containsKey(i))
```

```
                numOfValues.put(i, 1);
            else
                numOfValues.put(i, numOfValues.get(i) + 1);
        }
        Integer res = Collections.min(numOfValues.entrySet(), (o1, o2) -> o1.getValue() -
o2.getValue()).getKey();
        return res;
    }
}
```

Билет 6

Напишите метод с именем guavaSort, который принимает массив строк в качестве параметра и упорядочивает строки в массиве в отсортированном порядке возрастания. В частности, ваш алгоритм сортировки должен использовать FJC для Multiset или Multimap для реализации варианта алгоритма блочной сортировки, который будет работать со строками. Используйте коллекцию FJC для подсчета вхождений строк, аналогично тому, как это делается в блочной сортировке, а затем поместите эти строки обратно в массив в отсортированном порядке. Например, предположим, что вашему методу передается следующий массив:
[Farm, Zoo, Car, Apple, Bee, Golf, Bee, Dog, Golf, Zoo, Zoo, Bee, Bee, Apple]
Ваша коллекция должна хранить следующие вхождения строк:
[Apple x 2, Bee x 4, Car, Dog, Farm, Golf x 2, Zoo x 3]
Что вы должны использовать, чтобы поместить строки обратно в массив в отсортированном порядке:
[Apple, Apple, Bee, Bee, Bee, Bee, Car, Dog, Farm, Golf, Golf, Zoo, Zoo, Zoo]
Ваш код должен выполняться за время O (N log N) и использовать память O (N), где N – количество элементов в массиве. Вы можете предположить, что переданный массив и все строки в нем не равны нулю. Не используйте никаких других вспомогательных коллекций, кроме одного Multiset или Multimap.

```
import com.google.common.collect.Multiset;
import com.google.common.collect.TreeMultiset;

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

public class Main {
    public static void main(String[] args) {
        String[] zoo =
{"Farm","Zoo","Car","Apple","Bee","Golf","Bee","Dog","Golf","Zoo","Zoo","Bee","Bee","Apple"};
        System.out.println(Arrays.asList(zoo));
        guavaSort(zoo);
        System.out.println(Arrays.asList(zoo));
    }

    public static void guavaSort(String[] arr) {
        Multiset<String> multiset = TreeMultiset.create();
        multiset.addAll(Arrays.asList(arr));
        int i = 0;
        for (String item : multiset) {
            arr[i]= item;
```

```
        i++;
    }

  }
}
```

Билет 7

Напишите метод removeAll, который можно добавить в класс LinkedIntList. Метод должен эффективно удалить из отсортированного списка целых чисел все значения, появляющиеся во втором отсортированном списке целых чисел. Например, предположим, что переменные LinkedIntList list1 и list2 ссылаются на следующие списки:
list1: [1, 3, 5, 7]
list2: [1, 2, 3, 4, 5]
Если была вызвана listl.removeAll(list2), то списки должны хранить следующие значения после вызова:
list1: [7]
list2: [1, 2, 3, 4, 5]
Обратите внимание, что все значения из list1, которые появляются в list2, были удалены, а list2 не изменился. Если бы вместо этого был вызов list2.removeAll(list1), списки будут иметь следующие значения:
list1: [1, 3, 5, 7]
list2: [2, 4]
Оба списка гарантированно находятся в отсортированном (неубывающем) порядке, хотя в любом списке могут быть дубликаты. Поскольку списки отсортированы, вы можете решить эту проблему очень эффективно за один проход данных. Ваше решение должно выполняться за время O (M + N), где M и N - длины двух списков. Предположим, что мы добавляем этот метод в класс LinkedIntList, как показано ниже. Вы не можете вызывать какие-либо другие методы класса для решения этой задачи, вы не можете создавать новые узлы и не можете использовать какие-либо вспомогательные структуры данных для решения этой проблемы (например, массив, ArrayList, Queue, String и т.д.). Вы также не можете изменять какие-либо поля данных узлов. Вы должны решить эту задачу, переставив ссылки на списки.

```
public class LinkedIntList {
private ListNode front;
...
}
public class ListNode {
public int data;
public ListNode next;
...
}
```

```
package com.company;

import java.util.LinkedList;

public class Main {
```

```
    public static void main(String[] args) {
        // write your code here
        LinkedIntList list1 = new LinkedIntList();
        for (int i = 1; i < 9; i += 2) {
            list1.push(i);
        }
        LinkedIntList list2 = new LinkedIntList();
        for (int i = 1; i < 6; i++) {
            list2.push(i);
        }
        list1.print();
        list2.print();
        list1.removeAll(list2);
        list1.print();
        list2.print();


    }
}

class ListNode {
    public int data;
    public ListNode next = null;

    ListNode() {
        data = 0;
        next = null;
    }

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

class LinkedIntList {
    ListNode front;
    ListNode last;

    LinkedIntList() {
        front = new ListNode();
    }

    /* public void firstLast(){
         last.next = front;
         front = front.next;
         last.next.next = null;
     }*/
    public void removeAll(LinkedIntList list) {
        if (front == null || list.front == null) return;
         ListNode current2 =list.front;
        ListNode current1 = front;
        ListNode prev = null;
        while (current1 != null) {
            if (current1.data == current2.data) {
                if (current1 == front) {
```

```
                front = current1.next;
                current1 = current1.next;
//current2 = current2. next;
                if (current1 == null || current2 == null) return;
            } else {
                prev.next = current1.next;
// current2 = current2. next;
                current1 = current1.next;
                if (current2 == null) return;
            }
        } else if (current1.data < current2.data) {
            prev = current1;
            current1 = current1.next;
            if (current1 == null) return;
        } else {
            current2 = current2.next;
            if (current2 == null) return;
        }
    }
  }

  public void push(int data) {
    ListNode node = new ListNode();
    node.data = data;
    if (last == null) {
      front = node;
      last = node;
    } else {
      last.next = node;
      last = last.next;
    }
  }

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

```java
public void removeAll(LinkedIntList other) {
    ListNode current = this.head;
    ListNode previous = null;
    ListNode otherCurrent = other.head;
    while (current != null && otherCurrent != null) {
        if (current.data == otherCurrent.data) {
            if (previous == null) {
                this.head = current.next;
            } else {
                previous.next = current.next;
            }
            current = current.next;
            this.size--;
        } else if (current.data < otherCurrent.data) {
            previous = current;
            current = current.next;
        } else {
            otherCurrent = otherCurrent.next;
        }
    }
}
```

Билет 8

Запишите выходные данные, которые выводятся, когда указанному ниже методу передается каждый из следующих отображений в качестве параметра. Напомним, что содержимое печатается в формате ключ-значение. Ваш ответ должен отображать правильные ключи и значения в правильном порядке.

```
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);
}
```

{two=deux, five=cinq, three=trois, four=quatre}

{skate=board, drive=car, program=computer, play=computer}

{siskel=ebert, girl=boy, heads=tails, ready=begin, first=last, begin=end}

{cotton=shirt, tree=violin, seed=tree, light=tree, rain=cotton}

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

public class task8 {
   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("three", "trois");
//      map.put("four", "quatre");
      // --------------------
//      map.put("skate", "board");
//      map.put("drive", "drive");
//      map.put("program", "computer");
//      map.put("play", "computer");
      // --------------------
//      map.put("siskel", "ebert");
//      map.put("girl", "boy");
//      map.put("heads", "tails");
//      map.put("ready", "begin");
//      map.put("first", "last");
//      map.put("begin", "end");
      // --------------------
      map.put("cotton", "shirt");
      map.put("tree", "violin");
      map.put("seed", "tree");
      map.put("light", "tree");
      map.put("rain", "cotton");
      mystery(map);
   }
}
```

Билет 9

Напишите метод removeDuplicates, который можно добавить в класс LinkedIntList. Метод должен удалить любые дубликаты из связанного списка целых чисел. Результирующий список должен иметь значения в том же относительном порядке, что и их первое вхождение в исходном списке. Другими словами, значение i должно появляться перед значением j в окончательном списке тогда и только тогда, когда первое вхождение i появилось до первого появления j в исходном списке. Например, если переменная с именем list хранит следующий список: [14, 8, 14, 12, 1, 14, 11, 8, 8, 10, 4, 9, 1, 2, 5, 2, 4, 12, 12]
После вызова list.removeDuplicates(); список должен хранить эти значения в таком виде: [14, 8, 12, 1, 11, 10, 4, 9, 2, 5]
Предположим, что мы добавляем этот метод в класс LinkedIntList, как показано ниже. Вы не можете вызывать какие-либо другие методы класса для решения этой задачи, вы не можете создавать новые узлы и не можете использовать какие-либо вспомогательные структуры данных для решения этой проблемы (например, массив, ArrayList, Queue, String и т. д.). Вы также не можете изменять какие-либо поля данных узлов. Вы должны решить эту задачу, переставив ссылки в списке.

```
public class LinkedIntList {
private ListNode front;
...
}
public class ListNode {
```

public int data;
public ListNode next;
...
}

```
package com.company;

import java.util.LinkedList;

public class Main {

    public static void main(String[] args) {
        // write your code here
        LinkedIntList list1 = new LinkedIntList();
        for (int i = 1; i <14; i += 2) {
            list1.push(i);
            list1.push(10);
        }
        list1.push(13);
        list1.push(5);
        list1.push(5);
        list1.print();
        list1.removeDuplicates();
        list1.print();
    }
}

class ListNode {
    public int data;
    public ListNode next = null;

    ListNode() {
        data = 0;
        next = null;
    }

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

class LinkedIntList {
    ListNode front;
    ListNode last;

    LinkedIntList() {
        front = new ListNode();
    }

    /* public void firstLast(){
         last.next = front;
         front = front.next;
         last.next.next = null;
     }*/
    public void removeDuplicates() {
        if (front == null) return;
```

```
        ListNode current = front;
        while (current != null) {
            ListNode runner = current;
            while (runner.next != null) {
                if (runner.next.data == current.data) {
                    runner.next = runner.next.next;
                } else {
                    runner = runner.next;
                }
            }
            current = current.next;
        }
    }

    public void push(int data) {
        ListNode node = new ListNode();
        node.data = data;
        if (last == null) {
            front = node;
            last = node;
        } else {
            last.next = node;
            last = last.next;
        }
    }

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

Билет 11

Напишите метод firstLast, который можно добавить в класс LinkedIntList, который перемещает первый элемент списка в конец списка. Предположим, что переменная LinkedIntList с именем list хранит следующие элементы спереди (слева) и сзади (справа):

[18, 4, 27, 9, 54, 5, 63]

Если вы сделали вызов list.firstLast(), список будет хранить элементы в следующем порядке^

[4, 27, 9, 54, 5, 63, 18]

Если список пуст или содержит только один элемент, его содержимое не должно изменяться.

Соблюдайте следующие ограничения в вашем решении:

- **Не вызывайте никакие другие методы объекта LinkedintList, такие как add, remove или size.**
- **Не создавайте новые объекты ListNode (хотя у вас может быть столько переменных ListNode, сколько вам нужно). Не используйте другие структуры данных, такие как массивы, списки, очереди и т.д.**
- **Не изменяйте данные любого существующего узла; изменять список только путем изменения ссылок между узлами.**

Ваше решение должно выполняться за время O(N), где N - количество элементов в связанном списке.
Предположим, что вы добавляете этот метод в класс LinkedIntList (который использует класс ListNode) как показано ниже:

```
public class LinkedIntList {
...
}
public class ListNode {
public int data;
public ListNode next;
...
}
```

```
package com.company;

import java.util.LinkedList;

public class Main {

    public static void main(String[] args) {
            // write your code here
        LinkedIntList linkedIntList = new LinkedIntList();
        for (int i = 0; i < 5; i++) {
            linkedIntList.push(i);
        }
        linkedIntList.print();
        linkedIntList.firstLast();
        linkedIntList.print();

    }
}
class LinkedIntList{
    ListNode front;
    ListNode last;
    LinkedIntList(){
        front=new ListNode();
    }
    public void firstLast(){
        last.next = front;
        front = front.next;
        last.next.next = null;
    }
    public void push(int data){
        ListNode node = new ListNode();
        node.data = data;
        if(last==null){
            front = node;
            last = node;
        }
        else {
            last.next = node;
            last = last.next;
        }
    }
    public void print(){
        for(ListNode node = front;node!=null;node=node.next)
```

```
            System.out.print(node.data + " ");
        System.out.println();
    }

}
class ListNode{
    public int data;
    public ListNode next = null;
    ListNode(){
        data = 0;
        next = null;
    }
    ListNode(int data,ListNode next){
        this.data = data;
        this.next = next;
    }
}
```

Билет 12

Напишите код, необходимый для преобразования следующей последовательности объектов ListNode list -> [1] -> [2] / В следующую последовательность объектов ListNode list -> [1] -> [2] -> [3] /

public class ListNode {

public int data; // data stored in this node

public ListNode next; // a link to the next in the list

public ListNode() {…}

public ListNode(int data) {…}

public ListNode(int data, ListNode next) {…

```
class ListNode12 {
    public int data;
    public ListNode12 next;

    public ListNode12() {
    }

    public ListNode12(int data) {
        this.data = data;
    }


    public ListNode12(int data, ListNode12 next) {
        this.data = data;
        this.next = next;
    }


}
public class task12 {
    public static void main(String[] args) {
        ListNode12 list = new ListNode12(1, new ListNode12(2));
        list.next.next = new ListNode12(3);
```

```
    }

}
```

Билет 14

Напишите метод equals, который принимает в качестве параметров два стека целых чисел, метод возвращает true, если два стека равны, в противном случае возвращает false. Чтобы считаться равными, два стека должны хранить одинаковую последовательность целочисленных значений в одном и том же порядке. Ваш метод заключается в проверке двух стеков, но перед завершением работы метода необходимо вернуть их в исходное состояние. Вы можете использовать один стек в качестве вспомогательного хранения.

```
import java.util.Stack;

public class Num_14 {
    static boolean equals(Stack<Integer> stack1,Stack<Integer> stack2){
        Stack<Integer> stack3 = new Stack<>();

        System.out.println(stack1);
        System.out.println(stack2);

        boolean f = true;

        if(stack1.size() != stack2.size()){
            return f;
        }

        while(!stack1.isEmpty()){
            if(stack1.peek() == stack2.peek()){
                stack3.push(stack1.pop());
                stack2.pop();
            }
            else {
                f = false;
                break;
            }
        }

        System.out.println(stack3);

        while(!stack3.isEmpty()){
            stack1.push(stack3.peek());
            stack2.push(stack3.pop());
        }

        System.out.println(stack1);
        System.out.println(stack2);
        return f;
    }


    public static void main(String[] args) {
        Stack<Integer> stack1 = new Stack<>();
```

```
        Stack<Integer> stack2 = new Stack<>();
        Stack<Integer> stack3 = new Stack<>();

        stack1.push(1);
        stack1.push(6);
        stack1.push(4);
        stack1.push(3);
        stack1.push(3);
        stack1.push(-2);

        stack2.push(1);
        stack2.push(-9);
        stack2.push(18);
        stack2.push(0);
        stack2.push(4);
        stack2.push(1);

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

Билет 15

Напишите метод splitStack, который принимает стек целых чисел в качестве параметра и разбивает его на отрицательные и неотрицательные значения. Числа в стеке должны быть переставлены так, чтобы все отрицательные значения появлялись в нижней части стека, а все неотрицательные - в верхней части. Другим словами, если после вызова этого метода вам нужно будет вытолкнуть числа из стека, вы сначала получите все неотрицательные числа, а затем получите все отрицательные числа. Неважно, в каком порядке появляются числа, если все отрицательные находятся в стеке всегда ниже, чем все неотрицательные числа. Вы можете использовать одну очередь в качестве вспомогательного хранения.

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

public class Main {

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(-1);
        stack.push(2);
        stack.push(2);
        stack.push(0);
        stack.push(-2);
        stack.push(-3);
        stack.push(-3);
        stack.push(4);
        stack.push(5);
        stack = splitStack(stack);
```

```
        System.out.println(stack);
    }

    private static Stack<Integer> splitStack(Stack<Integer> stack) {

        Queue<Integer> queue = new ArrayDeque<>();

        while (!stack.empty()) {
            queue.add(stack.pop());
        }

        int element;
        int amount = queue.size();
        int counter = 0;
        while (!queue.isEmpty()) {
            if (counter == amount) {
                while (!queue.isEmpty()) {
                    stack.push(queue.poll());
                }

                break;
            }

            element = queue.poll();
            counter++;

            if (element < 0) {
                stack.push(element);
            } else {
                queue.add(element);
            }
        }
        return stack;
    }
}
```

Билет 16

Напишите метод copyStack, который принимает стек целых чисел в качестве параметра и возвращает копию оригинального стека (т.е. новый стек с теми же значениями, что и у оригинала, сохраненный в том же порядке, что и оригинал). Ваш метод должен создать новый стек и заполнять его теми же значениями, которые хранятся в исходном стеке. Недопустимо возвращать тот же стек, переданный методу. Вы должны создать, заполнить и вернуть новый стек. Вы будете удалять значения из исходного стека, чтобы сделать копию, но вы должны быть уверены, что поместите их обратно в исходный стек в том же порядке, прежде чем завершите с ним работу. Другими словами, когда ваш метод будет выполнен, исходный стек должен быть восстановлен в исходное состояние, и вы вернете новый независимый стек, который находится в том же состоянии. Вы можете использовать одну очередь в качестве вспомогательного хранения.

```
public class Num_16 {
```

```
    public static Stack<Integer> CopyStack(Stack<Integer> stack) {
        Queue<Integer> queue = new LinkedList<>();
        Stack<Integer> Newstack = new Stack<>();
        int size = stack.size();
        System.out.println("\n" + stack );
        System.out.println(queue);
        while (!stack.isEmpty()){
            queue.add(stack.pop());
        }
        while (!queue.isEmpty()){
            stack.push(queue.remove());
        }
        while (!stack.isEmpty()){
            queue.add(stack.pop());
        }
        while (!queue.isEmpty()){
            stack.push(queue.element());
            Newstack.push(queue.remove());
        }
        System.out.println("\n" + stack );
        System.out.println(Newstack);
        return Newstack;
    }

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        CopyStack(stack);
    }
}
```

Билет 18

Напишите метод longestSortedSequence, который возвращает длину самой длинной отсортированной последовательности в списке целых чисел. Например, если переменная с именем list хранит следующую последовательность значений:
[1, 3, 5, 2, 9, 7, -3, 0, 42, 308, 17]
Тогда вызов: list.longestSortedSequence() вернет значение 4, поскольку это длина самой длинной отсортированной последовательности в этом списке (последовательность -3, 0, 42, 308). Если список пуст, ваш метод должен вернуть 0. Обратите внимание, что для непустого списка метод всегда будет возвращать значение по крайней мере 1, потому что любой отдельный элемент составляет отсортированнпоследовательность.

```
public class ArrayIntList {
private int[] elementData;
private int size;
// your code goes hereio ppl
}
```

```
public class Num_18 {
```

```
    public static int num18_longestSortedSequence(int[] list) {
        int maxLength = 0;
        int currentLength = 0;
        for (int i = 0; i < list.length; i++) {
            if (i == 0 || list[i] >= list[i - 1]) {
                currentLength++;
                if (currentLength > maxLength) {
                    maxLength = currentLength;
                }
            } else {
                currentLength = 1;
            }
        }
        return maxLength;
    }

    public static void main(String[] args) {
        int[] a = new int[]{1,3,5,2,9,7,-3,0,42,308,17};
        int[] b = new int[]{1231231};

        System.out.println(num18_longestSortedSequence(b));
    }
}
```

Билет 19

Класс Деньги для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа double – для копеек. Дробная часть (копейки) при выводе на экран должна быть отделена от целой части запятой. Реализовать сложение, вычитание, деление сумм, деление суммы на дробное число, умножение на дробное число и операции сравнения. В классе Тестер проверить эти методы.

```
package Билет19_Деньги;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.Arrays;

/*Класс Деньги для работы с денежными суммами.
Число должно быть представлено двумя полями: типа long для рублей и типа double – для копеек.
 Дробная часть (копейки) при выводе на экран должна быть отделена от целой части запятой.
  Реализовать сложение, вычитание,деление сумм,
 деление суммы на дробное число, умножение на дробное число и операции сравнения.
 В классе Тестер проверить эти методы.*/
public class Money {
    private long rubles;
    private double kopeks;

    public Money(long rubles, double kopeks) {
        this.rubles = rubles;
        this.kopeks = kopeks;
    }

    public long getRubles() {
```

```
    return rubles;
}

public double getKopeks() {
    return kopeks;
}

public void setRubles(int rubles) {
    this.rubles = rubles;
}

public void setKopeks(double kopeks) {
    this.kopeks = kopeks;
}

public Money add(Money money) {
    long newRubles = this.rubles + money.getRubles();
    double newKopeks = this.kopeks + money.getKopeks();
    return new Money(newRubles, newKopeks);
}

public Money subtract(Money money) {
    long newRubles = this.rubles - money.getRubles();
    double newKopeks = this.kopeks - money.getKopeks();
    return new Money(newRubles, newKopeks);
}

public Money divide(Money money) {
    long newRubles = this.rubles / money.getRubles();
    double newKopeks = this.kopeks / money.getKopeks();
    return new Money(newRubles, newKopeks);
}

public Money divide(double number) {
    long newRubles = (long) (this.rubles / number);
    double newKopeks = this.kopeks / number;
    return new Money(newRubles, newKopeks);
}

public Money multiply(double number) {
    int newRubles = (int) (this.rubles * number);
    double newKopeks = this.kopeks * number;
    return new Money(newRubles, newKopeks);
}

public int compareTo(Money money) {
    if (this.rubles > money.getRubles()) {
        return 1;
    } else if (this.rubles < money.getRubles()) {
        return -1;
    } else {
        if (this.kopeks > money.getKopeks()) {
            return 1;
        } else if (this.kopeks < money.getKopeks()) {
            return -1;
        } else {
            return 0;
```

```
        }
      }
   }

   @Override
   public String toString() {
      String str=Double.toString(Math.abs(kopeks));
      return rubles + "," +str.substring(2);
   }

   public static void main(String args[]){
      Money money1 = new Money(100L, (double) 0.10);
      Money money2 = new Money(200L, (double) 0.20);
      System.out.println(money1.add(money2));
      System.out.println(money1.subtract(money2));
      System.out.println(money1.divide(money2));
      System.out.println(money1.divide(2));
      System.out.println(money1.multiply(1.1));
      System.out.println(money1.equals(money2));//1==2? 1 объект, для которого вызывается
метод
      if (money1.compareTo(money2)==-1){
         System.out.println("2 число больше первого");
      }else if(money1.compareTo(money2)==1){
         System.out.println("1 число больше второго");
      }
   }
}
```

Билет 21

Напишите универсальный класс для реализации алгоритмов поиска. В качестве параметров используйте массив интерфейсных ссылок

```
class Sorter{
   public static Comparable find(Comparable[] list, Comparable obj){
      for (int i = 0; i < list.length; i++) {
         for (int j = 0; j < list.length - 1; j++) {
            if (list[j].equals(obj)){
               return list[j];
            }
         }
      }
      return obj;
   }
}

public class task21 {
   public static void main(String[] args) {
      Integer[] list = {10, 2, 3, 4, 5, 6, 7, 8, 9, 1};
      System.out.println(Sorter.find(list, 5));
      System.out.println("----------------");
      String[] listS = {"10", "2", "3", "4", "5", "6", "7", "8", "9", "1"};
      Sorter.find(listS, "5");
      System.out.println(Sorter.find(listS, "5"));
   }
}
```

Билет 22

Напишите универсальный класс для реализации алгоритмов сортировок. В качестве параметров используйте массив интерфейсных ссылок

```
public class test{
    public static void main(String[] args) {
        Integer[] arr = new Integer[5];
        String[] arr2 = new String[5];

        Sorts.sort1(arr);
        Sorts.sort1(arr2);
    }

    public static class Sorts {
        public static <T> void sort1(Comparable<T>[] array) {}
        public static <T> void sort2(Comparable<T>[] array) {}
        public static <T> void sort3(Comparable<T>[] array) {}
        public static <T> void sort4(Comparable<T>[] array) {}
    }
}
```

Билет 23

Разработайте класс иерархию классов Геометрическая фигура, Прямоугольник, Круг. Используйте паттерн Фабрика

```
enum figureType {
    RECTANGLE,
    CIRCLE,
}
class figure{
    void echo(){
        System.out.println("I'm a figure");
    }
}

class rectangle extends figure{
    @Override
    void echo() {
        System.out.println("I'm a rectangle");
    }
}

class circle extends figure{
    @Override
    void echo() {
        System.out.println("I'm a circle");
    }
}

class figureFactory{
    figure getFigure(figureType type){
        return switch (type) {
            case RECTANGLE -> new rectangle();
            case CIRCLE -> new circle();
            default -> null;
```

```
    };
  }
}

public class task23 {
}
```

Билет 24

Разработайте класс иерархию классов Комплексное число, Рациональное число. Используйте паттерн Фабрика

Билет 25

Напишите реализацию структуры данных Stack на списке

```
public static class stack<T> {
    private class Node {
        T data;
        Node next;

        public Node(T data, Node next) {
            this.data = data;
            this.next = next;
        }
    }

    private Node root;
    int size;

    public stack() {
        root = null;
        size = 0;
    }

    public boolean isEmpty() {return size == 0;}

    public int size() {return size;}

    public void push(T element) {
        size++;
        if (isEmpty()) {root = new Node(element, null); return;}

        root = new Node(element, root);
    }

    public T pop() {
        if (isEmpty()) {return null;}

        size--;
        Node node = root;
        root = root.next;

        return node.data;
    }
```

```
    public T peek() {
        return root.data;
    }
}
```

Билет 26

Напишите реализацию структуры ArrayList на списке

```
public static class arrayList<T> {
    private class Node {
        T data;
        Node next;

        public Node(T data, Node next) {
            this.data = data;
            this.next = next;
        }
    }

    private Node root;
    private int size;

    public arrayList() {
        root = null;
        size = 0;
    }

    public boolean isEmpty() {return size == 0;}

    public void add(T element) {
        size++;

        if (isEmpty()) {root = new Node(element, null); return;}

        Node currentNode = root;
        while (currentNode.next != null) {
            currentNode = currentNode.next;
        }

        currentNode.next = new Node(element, null);
    }

    public int size() {return size;}

    public void remove(int index) {
        if (isEmpty()) {return;}

        Node node = root;
        for (int i = 0; i < index - 1; i++) {
            node = node.next;
        }

        node.next = node.next.next;
        size--;
```

```
    }

    public T get(int index) {
        int i = 0;
        Node node = root;

        while(i < index) {
            node = node.next; i++;
        }

        return node.data;
    }

}
```

Билет 27

Напишите реализацию структуры ArrayList на массиве.

```
package lab3_4;

public class arrayList<T> {
    private T[] arr;
    private int size;

    public arrayList() {
        T[] arr = (T[])new Object[100];
        size = 0;
    }

    public boolean isEmpty() {return size == 0;}

    public void add(T element) {
        arr[size] = element;
        size++;
    }

    public int size() {return size;}

    public void remove(int index) {
        if (isEmpty()) {return;}

        for (int i = index; i  < size - 1; i++)
            arr[i] = arr[i + 1];
        size--;
    }

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

}
```

Билет 28

Дано натуральное число n. вычислите сумму его цифр. при решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется).

```
public class Main {
   static int sum(int num){

   }
   public static void main(String[] args) {
      System.out.println(sum(123));
   }
}
```

Билет 29

Напишите реализацию структуры данных Stack на массиве. При реализации необходимо использовать дженерики

```
static class stack<T> {
      T[] array;
      int topIndex;
      int maxSize;

      public stack(int size) {
         this.maxSize = size;
         topIndex = -1;

         array = (T[]) new Object[size];
      }
      public boolean isEmpty() {return topIndex == -1;}
      public boolean isFull() {return topIndex + 1 == maxSize;}
      public int size() {return topIndex + 1;}

      public void push(T newElement) throws Exception {
         if (isFull()) {throw new Exception("Stack is Full");}

         topIndex++;
         array[topIndex] = newElement;
      }

      public T peek() throws Exception {
         if (isEmpty()) {throw new Exception("Stack is Empty");}

         return array[topIndex];
      }

      public T pop() throws Exception {
         if (isEmpty()) {throw new Exception("Stack is Empty");}

         topIndex--;
         return array[topIndex + 1];
      }
   }
```

Билет 30

Задача на рекурсию: дано натуральное число n. выведите слово yes, если число n является точной степенью двойки, или слово no в противном случае. операцией возведения в степень пользоваться нельзя!

```
public class Num_30{
   public static void chechPow2(int N,int powTwo){
      if(powTwo==N){
         System.out.println("YES");
      } else if (powTwo>N) {
         System.out.println("NO");
      }else {
         chechPow2(N,powTwo*2);
      }

   }

   public static void main(String[] args) {
      chechPow2(32,1);
   }
}
```

Билет N

Напишите метод reverseHalf, который меняет порядок на половину элементов очереди целых чисел. Ваш метод должен изменить порядок всех элементов в нечетных позициях, помним, что первое значение в очереди имеет позицию 0. Например, если очередь изначально хранит эту последовательность чисел, когда метод вызывается:
Index: 0 1 2 3 4 5 6 7
Front [1, 8, 7, 2, 9, 18, 12, 0] back
Очередь должна хранить следующие значения после завершения выполнения метода:
Index: 0 1 2 3 4 5 6 7
Front [1, 0, 7, 18, 9, 2, 12, 8] back
Обратите внимание, что числа в четных позициях не сместились. Эта последовательность чисел по-прежнему: (1,7,9,12). Но обратите внимание, что числа в нечетных позициях теперь в обратном порядке относительно оригинала. Другими словами, исходная последовательность (8,2,18,0) – стала такой: (0, 18, 2, 8). Вы можете использовать стек в качестве вспомогательного хранения элементов.

```
import java.lang.reflect.Array;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class n19 {
   public static void reversedHalf(Queue<Integer> numbers){
      Stack<Integer> half1 = new Stack<>();
      Queue<Integer> half2 = new LinkedList<>();
      int i = 0;
      while (true) {
         if (numbers.isEmpty()){
```

```
            break;
        }
        if(i % 2 == 0){
            half2.add(numbers.poll());
        } else {
            half1.add(numbers.poll());
        }
        i++;
    }
    while (true){
        if (half1.empty() && half2.isEmpty()){
            break;
        }
        if (!half2.isEmpty()){
            numbers.add(half2.poll());
        }
        if (!half1.empty()){
            numbers.add(half1.peek());
            half1.pop();
        }
     }
    System.out.println(numbers);
}

public static void main(String[] args){
    Queue<Integer> numbers = new LinkedList<>(Arrays.asList(1,2,3,4,5,6,7,8,9,10));
    reversedHalf(numbers);  } }
```