Оглавление

[Задание 1. 2](#_Toc78745088)

[Задание 2. 4](#_Toc78745089)

[Задание 3. 5](#_Toc78745090)

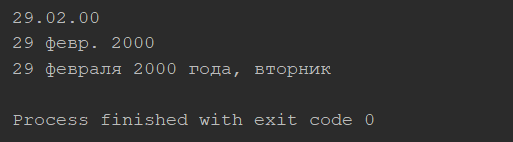
[Задание 4. 10](#_Toc78745091)

[Ссылка на GitHub 13](#_Toc78745092)

# Задание 1.

package trainingTask4.p1;  
  
import java.text.SimpleDateFormat;  
import java.util.Calendar;  
import java.util.GregorianCalendar;  
  
public class Person {  
 private String firstName;  
 private String lastName;  
 private String secondName;  
 private Calendar birthDate;  
  
 public Person(String firstName, String lastName, String secondName, int year, int month, int day) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.secondName = secondName;  
 this.setBirthDate(year, month, day);  
 }  
  
 public Person(String firstName, String lastName, String secondName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.secondName = secondName;  
 }  
  
 public Person(String firstName, String lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 }  
  
 public Person(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public Calendar getBirthdate() {  
 return birthDate;  
 }  
  
 public void setBirthDate(int year, int month, int day) {  
 Calendar date = new GregorianCalendar(year,month,day);  
 this.birthDate = date;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getSecondName() {  
 return secondName;  
 }  
  
 public void setSecondName(String secondName) {  
 this.secondName = secondName;  
 }  
  
 enum Format {  
 *shortDate*,  
 *mediumDate*,  
 *longDate* }  
  
 */\*\*  
 \* Возвращает строку с датой в заданном формате  
 \** ***@param*** *format формат даты.  
 \** ***@return*** *Строковое представление даты в заданном формате.  
 \*/* public String getDifferentFormatDate(Format format) {  
 String result;  
 switch (format){  
 case *longDate*:  
 result = new SimpleDateFormat("dd MMMM y года, EEEE").format(birthDate.getTime());  
 break;  
 case *mediumDate*:  
 result = new SimpleDateFormat("dd MMM y").format(birthDate.getTime());  
 break;  
 default:  
 result = new SimpleDateFormat("dd.MM.yy").format(birthDate.getTime());  
 break;  
 }  
 return result;  
 }  
  
 @Override  
 public String toString() {  
 return lastName+" "+firstName+" "+secondName+ ", "+getDifferentFormatDate(Format.*shortDate*);  
 }  
  
 public static void main(String[] args){  
 Person p1 = new Person("Иванов","Иван","Иванович",2000,1,29);  
 System.*out*.println(p1.getDifferentFormatDate(Format.*shortDate*));  
 System.*out*.println(p1.getDifferentFormatDate(Format.*mediumDate*));  
 System.*out*.println(p1.getDifferentFormatDate(Format.*longDate*));  
 }  
}

Результат выполнения:



# Задание 2.

package trainingTask4.p2;  
  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Calendar;  
import java.util.Date;  
  
class StringDateCalendar {  
 */\*\*  
 \* Формирует объект Date по вводимой строке.  
 \** ***@param*** *stringDate строка, содержащая данные <Год><Месяц><Число><Часы><минуты>  
 \** ***@return*** *объект Date или null, если строка не в заданном формате.  
 \*/* public static Date makeDate(String stringDate) throws ParseException {  
 return new SimpleDateFormat("dd.MM.yyyy H:m").parse(stringDate);  
 }  
  
 */\*\*  
 \* Формирует объект Calendar по вводимой строке.  
 \** ***@param*** *stringDate строка, содержащая данные <Год><Месяц><Число><Часы><минуты>  
 \** ***@return*** *объект Calendar или null, если строка не в заданном формате.  
 \*/* public static Calendar makeCalendar(String stringDate) throws ParseException {  
 Calendar calendar = Calendar.*getInstance*();  
 calendar.setTime(new SimpleDateFormat("dd.MM.yyyy H:m").parse(stringDate));  
 return calendar;  
 }  
}  
  
public class Test {  
 public static void main(String[] args){  
 String str = "2000.02.29 12:00";  
 Date date = null;  
 try {  
 date = StringDateCalendar.*makeDate*(str);  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
 Calendar calendar = null;  
 try {  
 calendar = StringDateCalendar.*makeCalendar*(str);  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println(date.toString());  
 System.*out*.println(calendar.getTime());  
 }  
}

Результат выполнения:

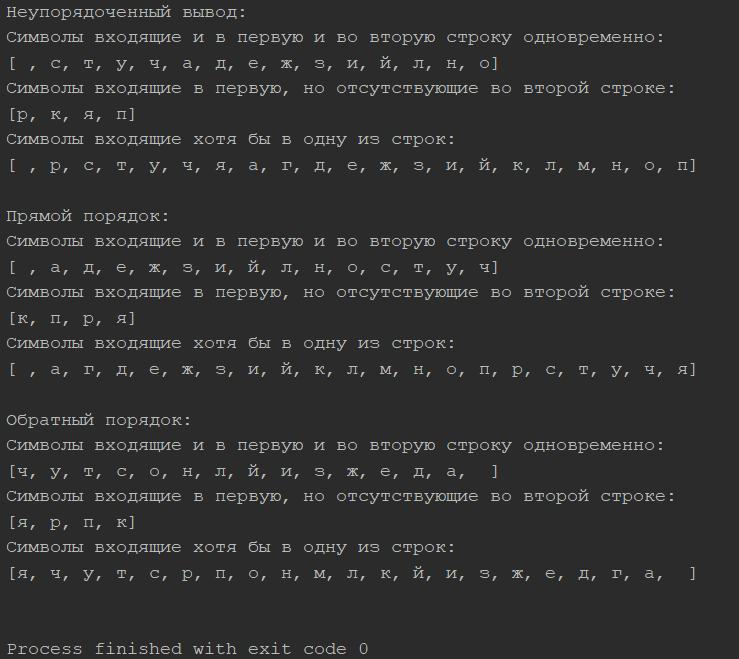


# Задание 3.

package trainingTask4.p3;  
  
import java.util.\*;  
  
public class TestAnalyze{  
 public static void main(String[] args){  
 FreqAnalyzer f3 = new FreqAnalyzer("Если я найду такое же предложение то значит копипаст " +  
 "Если же не найду то значит немного изменено");  
 System.*out*.println("Частотный анализ слов и символов строки:");  
 f3.printWordAnalysis();  
 f3.printCharAnalysis();  
  
 FreqAnalyzer f = new FreqAnalyzer("Если я найду такое же предложение то значит копипаст");  
 FreqAnalyzer f2 = new FreqAnalyzer("Если же не найду то значит немного изменено");  
 System.*out*.println("Неупорядоченный вывод:");  
 f.compareTwo(f2);  
 System.*out*.println("Прямой порядок:");  
 f.compareTwo(f2, FreqAnalyzer.Format.*regularOrder*);  
 System.*out*.println("Обратный порядок:");  
 f.compareTwo(f2, FreqAnalyzer.Format.*reverseOrder*);  
  
 }  
}  
  
class FreqAnalyzer {  
 private String str;  
 private Map<String, Integer> wordDict;  
 private Map<Character, Integer> charDict;  
  
 FreqAnalyzer(){}  
 FreqAnalyzer(String str){  
 this.str = str;  
 }  
  
 public String getStr() {  
 return str;  
 }  
  
 public void setStr(String str) {  
 this.str = str;  
 }  
  
 private Map<String, Integer> getWordDict() {  
 return wordDict;  
 }  
  
 private Map<Character, Integer> getCharDict() {  
 return charDict;  
 }  
  
 */\*\*  
 \* Возвращает Map<String, Integer>, сопоставляющую каждому  
 \* слову (длиной не менее 1 символа) анализируемой строки  
 \* количество его вхождений в данную строку.  
 \** ***@return*** *Map - результат подсчета количеств вхождений слов  
 \*/* public Map<String, Integer> wordsAnalyze(){  
 if (str == null) {  
 throw new NullPointerException();  
 }  
 wordDict = new HashMap<>();  
 String[] words = str.toLowerCase().split(" ");  
 for (String word : words) {  
 if (!wordDict.containsKey(word)) {  
 wordDict.put(word, 1);  
 } else {  
 wordDict.put(word, wordDict.get(word) + 1);  
 }  
 }  
 return wordDict;  
 }  
  
 */\*\*  
 \* Возвращает Map<Character, Integer>, сопоставляющую каждому  
 \* символу анализируемой строки количество его вхождений в данную строку.  
 \** ***@return*** *Map - результат подсчета количеств вхождений символов  
 \*/* public Map<Character, Integer> charactersAnalyze(){  
 if (str == null) {  
 throw new NullPointerException();  
 }  
 charDict = new HashMap<>();  
 char[] characters = str.toLowerCase().toCharArray();  
 for (char character : characters) {  
 if (!charDict.containsKey(character)) {  
 charDict.put(character, 1);  
 } else {  
 charDict.put(character, charDict.get(character) + 1);  
 }  
 }  
 return charDict;  
 }  
  
 */\*\*  
 \* Выводит частотный анализ по словам  
 \*/* public void printWordAnalysis() {  
 if (wordDict == null){ this.wordsAnalyze(); }  
 System.*out*.println("'Слово' - Количество вхождений:");  
 for (String word : wordDict.keySet()) {  
 System.*out*.println("'"+word + "' - " + wordDict.get(word));  
 }  
 System.*out*.println();  
 }  
  
 */\*\*  
 \* Выводит частотный анализ по символам  
 \*/* public void printCharAnalysis() {  
 if (charDict == null){ this.charactersAnalyze(); }  
 System.*out*.println("'Символ' - Количество вхождений:");  
 for (char character : charDict.keySet()) {  
 System.*out*.println("'"+character + "' - " + charDict.get(character));  
 }  
 System.*out*.println();  
 }  
  
 enum Format {  
 *regularOrder*,  
 *reverseOrder*,  
 *cyclicShift* }  
  
 */\*\*  
 \* Выводит символьное сравнение двух объектов.  
 \* Символы выводятся в заданной последовательности (Format), если указан формат.  
 \* Если формат не указан, символы не упорядочиваются.  
 \*/* public void compareTwo(FreqAnalyzer other, Format ...format){  
 if (format.length>0) {  
 Comparator<Character> cmprt;  
 switch (format[0]) {  
 case *regularOrder*:  
 cmprt = new Comparator<Character>() {  
 public int compare(Character o1, Character o2) {  
 return o1.toString().compareTo(o2.toString());  
 }  
 };  
 break;  
 case *reverseOrder*:  
 cmprt = new Comparator<Character>() {  
 public int compare(Character o1, Character o2) {  
 return o2.toString().compareTo(o1.toString());  
 }  
 };  
 break;  
 default:  
 cmprt = new Comparator<Character>() {  
 public int compare(Character o1, Character o2) {  
 return o1.toString().compareTo(o2.toString());  
 }  
 };  
 break;  
 }  
 Set<Character> sortedSet = new TreeSet<Character>(cmprt);  
 System.*out*.println("Символы входящие и в первую и во вторую строку одновременно:");  
 sortedSet.addAll(this.findSame(other));  
 System.*out*.println(sortedSet);  
 System.*out*.println("Символы входящие в первую, но отсутствующие во второй строке:");  
 sortedSet.clear();  
 sortedSet.addAll(this.findDiffer(other));  
 System.*out*.println(sortedSet);  
 System.*out*.println("Символы входящие хотя бы в одну из строк:");  
 sortedSet.clear();  
 sortedSet.addAll(this.findAny(other));  
 System.*out*.println(sortedSet);  
 System.*out*.println();  
 }  
 else {  
 System.*out*.println("Символы входящие и в первую и во вторую строку одновременно:\n"+this.findSame(other));  
 System.*out*.println("Символы входящие в первую, но отсутствующие во второй строке:\n"+this.findDiffer(other));  
 System.*out*.println("Символы входящие хотя бы в одну из строк:\n"+this.findAny(other));  
 System.*out*.println();  
 }  
 }  
  
 */\*\*  
 \* Находит символы, содержащиеся в обеих строках this и other  
 \** ***@param*** *other - с чем сравнивается FreqAnalyzer .  
 \** ***@return*** *- set символов.  
 \*/* private Set<Character> findSame(FreqAnalyzer other) {  
 if (charDict == null) { this.charactersAnalyze(); }  
 if (other.getCharDict() == null) { other.charactersAnalyze(); }  
 Set<Character> uniqChar2 = other.getCharDict().keySet();  
 Set<Character> uniqChar = this.charDict.keySet();  
 Set<Character> result = new HashSet<>(uniqChar);  
 result.retainAll(uniqChar2);  
 return result;  
 }  
  
 */\*\*  
 \* Находит символы, содержащиеся в this, но отсутствующие в other  
 \** ***@param*** *other - с чем сравнивается FreqAnalyzer .  
 \** ***@return*** *- set символов.  
 \*/* private Set<Character> findDiffer(FreqAnalyzer other) {  
 if (charDict == null) { this.charactersAnalyze(); }  
 if (other.getCharDict() == null) { other.charactersAnalyze(); }  
 Set<Character> uniqChar2 = other.getCharDict().keySet();  
 Set<Character> uniqChar = this.charDict.keySet();  
 Set<Character> result = new HashSet<>();  
 for (char character : uniqChar) {  
 if (!uniqChar2.contains(character)) {  
 result.add(character);  
 }  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Находит символы, содержащиеся хотя бы в одной из строк this и other  
 \** ***@param*** *other - с чем сравнивается FreqAnalyzer .  
 \** ***@return*** *- set символов.  
 \*/* private Set<Character> findAny(FreqAnalyzer other) {  
 if (charDict == null) { this.charactersAnalyze(); }  
 if (other.getCharDict() == null) { other.charactersAnalyze(); }  
 Set<Character> uniqChar2 = other.getCharDict().keySet();  
 Set<Character> uniqChar = this.charDict.keySet();  
 Set<Character> result = new HashSet<>(uniqChar);  
 result.addAll(uniqChar2);  
 return result;  
 }  
}

Результат выполнения:

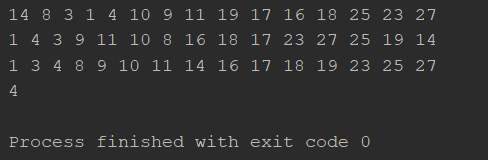




# Задание 4.

package trainingTask4.p4;  
  
public class BinTree {  
 private Node root;  
  
 private void appendNode(int val) {  
 if ( root == null )  
 root = new Node(val);  
 else {  
 Node currentNode = root;  
 while (true) {  
 if (val == currentNode.getValue()) {  
 return;  
 }  
 if (val < currentNode.getValue()) {  
 if (currentNode.getLeftChild() == null) {  
 currentNode.setLeftChild(new Node(val));  
 return;  
 } else {  
 currentNode = currentNode.getLeftChild();  
 }  
 } else {  
 if (currentNode.getRightChild() == null) {  
 currentNode.setRightChild(new Node(val));  
 return;  
 } else {  
 currentNode = currentNode.getRightChild();  
 }  
 }  
 }  
 }  
 }  
  
 public void preorder(Node currentNode) {  
 if (currentNode == null) {  
 return;  
 }  
 System.*out*.print(currentNode.getValue() + " ");  
 preorder(currentNode.getLeftChild());  
 preorder(currentNode.getRightChild());  
 }  
  
  
 public void postorder(Node currentNode) {  
 if (currentNode == null) {  
 return;  
 }  
 postorder(currentNode.getLeftChild());  
 postorder(currentNode.getRightChild());  
 System.*out*.print(currentNode.getValue() + " ");  
 }  
  
 public void inorder(Node currentNode) {  
 if (currentNode == null) {  
 return;  
 }  
 inorder(currentNode.getLeftChild());  
 System.*out*.print(currentNode.getValue() + " ");  
 inorder(currentNode.getRightChild());  
 }  
  
 public int findLength(Node currentNode) {  
 if (currentNode == null) {  
 return 0;  
 }  
 int llength = findLength(currentNode.getLeftChild())+1;  
 int rlength = findLength(currentNode.getRightChild())+1;  
 return Math.*max*(llength,rlength);  
 }  
  
 public Node getRoot(){  
 return this.root;  
 }  
  
 public static void main(String[] args){  
 BinTree tree = new BinTree();  
 tree.appendNode(14);  
 tree.appendNode(8);  
 tree.appendNode(19);  
 tree.appendNode(3);  
 tree.appendNode(10);  
 tree.appendNode(1);  
 tree.appendNode(4);  
 tree.appendNode(9);  
 tree.appendNode(11);  
 tree.appendNode(17);  
 tree.appendNode(25);  
 tree.appendNode(16);  
 tree.appendNode(18);  
 tree.appendNode(23);  
 tree.appendNode(27);  
  
 tree.preorder(tree.getRoot());  
 System.*out*.println();  
 tree.postorder(tree.getRoot());  
 System.*out*.println();  
 tree.inorder(tree.getRoot());  
 System.*out*.println();  
 System.*out*.println(tree.findLength(tree.getRoot()));  
 }  
}  
  
class Node{  
 private int value;  
 private Node leftChild;  
 private Node rightChild;  
  
 public Node(int value) {  
 this.value = value;  
 leftChild = null;  
 rightChild = null;  
 }  
  
 public Node(){}  
 public int getValue() {  
 return this.value;  
 }  
  
 public void setValue(final int value) {  
 this.value = value;  
 }  
  
 public Node getLeftChild() {  
 return this.leftChild;  
 }  
  
 public void setLeftChild(final Node leftChild) {  
 this.leftChild = leftChild;  
 }  
  
 public Node getRightChild() {  
 return this.rightChild;  
 }  
  
 public void setRightChild(final Node rightChild) {  
 this.rightChild = rightChild;  
 }  
  
 @Override  
 public String toString() {  
 return "Node{" +  
 "value=" + value +  
 ", leftChild=" + leftChild +  
 ", rightChild=" + rightChild +  
 '}';  
 }  
}

Результаты выполнения:



# Ссылка на GitHub

[yagorkas/Training-Tasks (github.com)](https://github.com/yagorkas/Training-Tasks)