**Контролно No. 2**

**Инструкции:**

1. Решението на **всички задачи** да се качи със **студенсткия акаунт на Мооdle**.-

3. **Използвайте дадените означения** за *класове*, *променливи* и *методи*.

### *Скала за оценяване:*

### *от 0 до 54 точки*

### *от 55 до 64 точки*

### *от 65 до 74 точки*

### *от 75 до 84 точки*

### *от 85 до 100 точки*

**Забележка**: При установено **преписване** се пише **0 точки за контролното**

**Задание за програмиране**

1. **Напишете** **class TreeNode** (**данни**, **конструктор и toString() метод извеждане на данните, съхранявани** *във възела на дървото*) за дефиниране на **възел** на **бинарно дърво за търсене (данните в лявото поддърво са по- големи от данните в дясното поддърво)**, в което да може се **съхраняват обекти от класове, които имплементират интерфейса Comparable**.. (**да се използва подходящо дефиниран параметър за тип**)

public class TreeNode<T extends Comparable<T>> implements Serializable// missing &Serializable {

 private TreeNode leftTreeNode; // TreeNode<T> !!!

 private TreeNode rightTreeNode; // TreeNode<T> !!!

 private T data;

 public void setData(T data) {

 if (data == null) {

 // при невалидна нова стойност – стойност по подразбиране.

 System.out.println("Invalid value for data");

 } else {

 this.data = data;

 }

 }

 public TreeNode getLeftTreeNode() { // TreeNode<T> !!!

 return leftTreeNode;

 }

 public TreeNode getRightTreeNode() { // TreeNode<T> !!!

 return rightTreeNode;

 }

 public T getData() {

 return data;

 }

 public TreeNode(T data) {

 leftTreeNode = rightTreeNode = null;

 setData(data);

 }

 public String toString(){

 String result;

 return result = String.format("Data = %s", data);

 }

}

**Точки**: 11 / 15

1. **В class TreeNode** **напишете void метод insertData(), който взима един аргумент-референция към**  **обекта** за **вмъкване** в **лявото** или **дясното поддърво** на **текущия обект** от TreeNode като се **спазва определението за бинарно дърво за търсене**

public void insertData(T newData) {

 // необходима е проверка за newData != null.

 if (newData.compareTo(data) > 0) {

 if (leftTreeNode == null) {

 leftTreeNode = new TreeNode(newData);

 } else {

 leftTreeNode.insertData(newData);

 }

 } else {

 if (rightTreeNode == null) {

 rightTreeNode = new TreeNode(newData);

 }else{

 rightTreeNode.insertData(newData);

 }

 }

 }

**Точки**: 11 / 12

1. **Напишете** **class ChristmasTree** за дефиниране на **бинарно дърво за търсене с TreeNode** възли (**данни**, **конструктор, isEmpty() метод**)-**да се използва подходящо дефиниран параметър за тип**

public class ChristmasTree<T extends Comparable<T>> // missing &Serializable {

 private TreeNode root; // TreeNode<T> !!!

 public ChristmasTree() {

 root = null;

 }

 public boolean isEmpty() {

 if (root == null) {

 return true;

 } else {

 return false;

 }

 }

**Точки**: 10 / 12

1. **В class ChristmasTree** **напишете void метод insertNode(),** който взима като аргумент референция към данна за съхранение във възел на дървото. Методът **вмъква** **възел в** бинарното дърво с реферираната в аргумента на метода данна като се **използва методът insertData ()**  **на class TreeNode**.

public void insertNode(T obeject) {

 if (isEmpty()) {

 root = new TreeNode(obeject);

 } else {

 root.insertData(obeject);

 }

 }

**Точки**: 6 / 6

1. **В class ChristmasTree** **напишете рекурсивен метод**

**public void size()**

**за пресмятане на броят на всички върхове на бинарното дърво.**

public int size() {

 return sizeHelper(root);

 }

 public int sizeHelper(TreeNode node) { // private и TreeNode<T>!!!

 if (node == null) {

 return 0;

 } else {

 return 1 + sizeHelper(node.getLeftTreeNode()) + sizeHelper(node.getRightTreeNode());

 }

 }

**Точки**: 9 / 10

1. **В class ChristmasTree** **напишете** метод (рекурсивен или итеративен- по избор)

**public** void writeTree(File filename)

за **сериализирано** записване на **данните**, съхранявани във **всички** върхове на дървото в **бинарен файл**, зададен **като аргумент** на метода. **Върховете** на дървото да се **обхождат** в ***postorder*** порядък

// липсва

**Точки**: 0 / 10

1. **В class ChristmasTree** **напишете** **рекурсивен** метод

public String toString() за извеждане в текстов низ на **всички** върхове на дървото**. Върховете** на дървото да се **обхождат** в ***inorder*** порядък

public String toString() {

 StringBuilder builder = new StringBuilder();

 return toString(root, builder);

 }

 public String toString(TreeNode node, StringBuilder builder) { // TreeNode<T>!!!

 if (node == null) {

 return builder.toString();

 } else {

 **// ????**

 builder.append(root); // **node.data** вместо **root!!!!**

 builder.append(node.getLeftTreeNode().toString());

 builder.append(node.getRightTreeNode().toString());

 // липсва рекурсивно извикване за дясното поддърво

 return toString(node.getLeftTreeNode(), builder);

 }

 }

**Точки**: 3 / 10

1. **Напишете** **class Toy, който има име name** и **цена** **price** на играчката. (**данни**, **конструктор, get и set методи и toString() метод** ). **Имплементирайте** интерфейс **Comparable** в **class Toy**, при което обектите се подреждат в **низходящ ред на цената им.**

**//** тази дефиниция е ок: public class Toy implements Comparable<Toy> {….}

public class Toy<T extends Comparable<T>> // missing ,Serializable {

 private T name; // String!!!

 private T price; // double!!!

 public void setName(T name) { // String!!!

 if (name == null) {

 System.out.println("Invalid value for Name");

 } else {

 this.name = name;

 }

 }

 public T getName() { // String!!!

 return this.name;

 }

 public void setPrice(T price) { // double!!!

 if (price.compareTo(this.price) < 0) {

 System.out.println("Invalid Price");

 } else {

 this.price = price;

 }

 }

 public T getPrice() { // double!!!

 return this.price;

 }

 public Toy(T newPrice, T newName) {

 setPrice(price);

 setName(name);

 }

 public String toString(String result) {

 return String.format("Name of product%s\nPrice of product%d ", name, price);

 }

}

**Точки:** 3 / 15

1. Напишете **конзолно приложение class** **TreeTest**, което:
2. Създава **бинарно дърво ChristmasTree** **за търсене** с **3 върха**, които са **Toy**  **обекти** с данни (**име** и **цена**) **по избор** на студента
3. Извежда в  **диалогов прозорец**
* **броят на върховете-** използва метод **size**() на **ChristmasTree**,
* **списък с данните във върховете на дървото-** използва метод **toString**() на **ChristmasTree**

public class TreeTest {

 public static void main(String[] args) {

 Toy newToy = new Toy(13.0, "Igrachka1");

 Toy newToy1 = new Toy(13.3, "Igrachka2");

 Toy newToy2 = new Toy(18.01, "Igrachka4");

 ChristmasTree<Toy> newTree = new ChristmasTree<>();

 newTree.insertNode(newToy);

 newTree.insertNode(newToy1);

 newTree.insertNode(newToy2);

// Беше необходимо да се използва JOptionPane за извеждане на резултатите в диалогов прозорец

 System.out.println("Size of the tree");

 newTree.size();

 System.out.println(" In order");

 newTree.inOrder();

 newTree.toString();

 }

}

**Точки**: 7 / 10

Общбройточкизаконтролнатаработа**: 61 / 100** точки