Санкт-Петербургский политехнический университет Петра Великого Институт компьютерных наук и технологи  
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВОЙ ПРОЕКТ**

**Дисциплина**: Алгоритмы и структуры данных

**Тема**: разработка GUI приложения Splay-Tree

Выполнил студент гр. 3530901/80003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Курняков П.М.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Глухих М.И.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019г.

Санкт-Петербург

2019

**Оглавление**

[Техническое задание 3](#_Toc28095897)

[Метод решения 3](#_Toc28095898)

[Листинг программы 5](#_Toc28095899)

[SplayNode.kt 6](#_Toc28095900)

[SplayTree.kt 6](#_Toc28095901)

[Views: 14](#_Toc28095902)

[ActionView.kt 14](#_Toc28095903)

[MainView.kt 15](#_Toc28095904)

[MenuBar.kt 15](#_Toc28095905)

[SplayTreeView.kt 16](#_Toc28095906)

[Контроллеры: 18](#_Toc28095907)

[MenuController.kt 18](#_Toc28095908)

[ParseController.kt 19](#_Toc28095909)

[SplayTreeController.kt 20](#_Toc28095910)

[Main-класс: 22](#_Toc28095911)

[Тесты: 22](#_Toc28095912)

[SplayTreeTest.kt 22](#_Toc28095913)

[build.gradle: 23](#_Toc28095914)

[Скриншоты графического приложения 24](#_Toc28095915)

# 

## Техническое задание

Реализовать графическое приложение “Splay-Tree”, задачей которого является реализация класса расширяющегося дерева и его отображения.

В приложение используются текстовое поле: input, несколько функциональных кнопок, канвас и меню бар.

Пример работы:

В поле input мы заносим целое число, которое хотим добавить, удалить или найти в дереве, после чего дерево выполняет заданную задачу и отображает новую версию дерева (пере-балансированную) в поле канвас.

GitHub Repository

<https://github.com/pupptmstr/splaytree>

## Метод решения

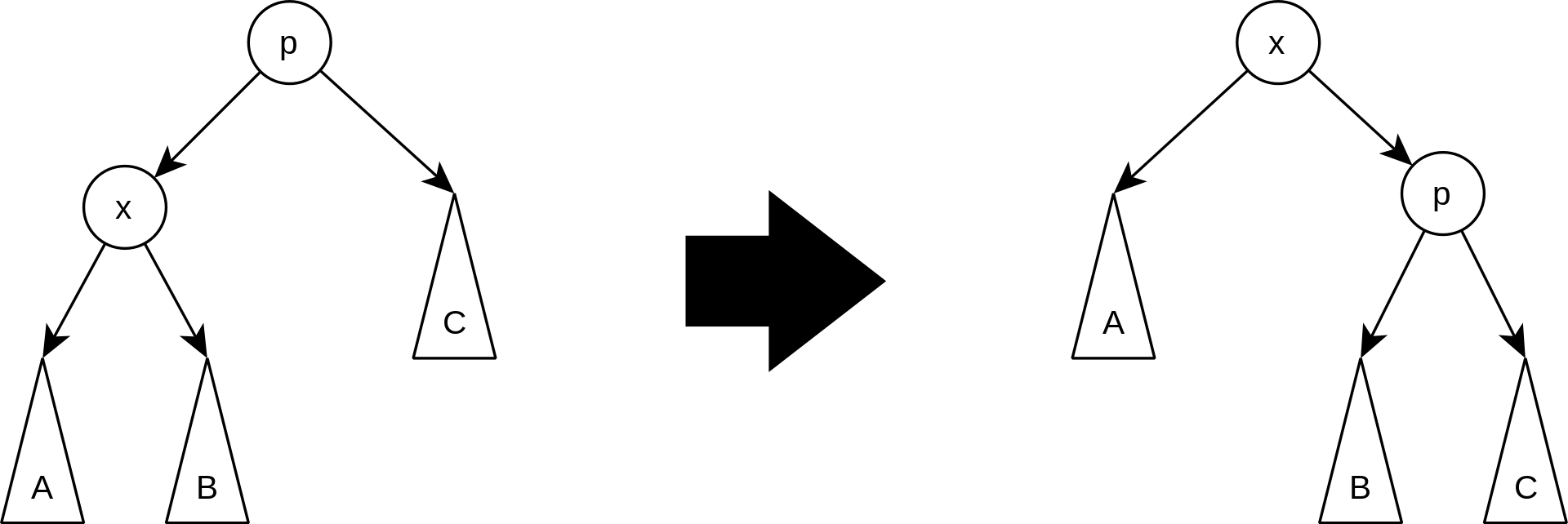
В данном API основным алгоритмом является алгоритм работы splay-дерева (<https://ru.m.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%88%D0%B8%D1%80%D1%8F%D1%8E%D1%89%D0%B5%D0%B5%D1%81%D1%8F_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE>). При реализации я наследовался и реализовывал Java-интерфейс SortedSet.

*Описание алгоритма*

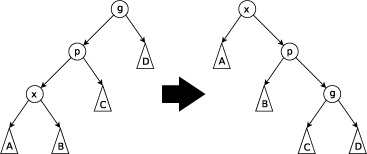
Основной алгоритм можно разделить на 3 условные части. Особенностью и главным отличием расширяющегося дерева от других популярных бинарных деревьев является то, что оно после каждого запроса к нему, будь то добавление нового элемента, удаление какого-либо элемента или поиск элемента, вне зависимости от успешности этих операций, перебаллансируется, вынося последний элемент, к которому оно обращалось, в корень дерева. Соответственно каждую операцию с деревом можно разделить на классическую операцию с бинарных деревом (добавление, удаление, поиск) и операцию splay, выносящую последний элемент в корень дерева.

Операция splay делается за счёт других важных для нее операций: zig, zig-zig и zig-zag.

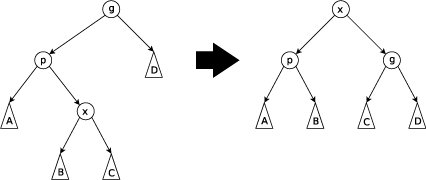
**Zig:** выполняется, когда *p* является корнем. Дерево поворачивается по ребру между *x* и *p*, где x – нынешний элемент, который необходимо вынести в корень.

Существует лишь для разбора крайнего случая и выполняется только один раз в конце, когда изначальная глубина *x* была нечётна

**Zig-Zig:** выполняется, когда и *x*, и *p* являются левыми (или правыми) сыновьями. Дерево поворачивается по ребру между *g* и *p*, а потом — по ребру между *p* и *x*.



**Zig-Zag:** выполняется, когда *x* является правым сыном, а *p* — левым (или наоборот). Дерево поворачивается по ребру между *p* и *x*, а затем — по ребру между *x* и *g*.



За что какие классы отвечают:

Все классы в пакете views отвечают за разные отображаемые элементы (вьюхи). Основной среди них является – MainView (содержит основное окно и подтягивает в себя все остальные вьюхи)

ActionView – класс, отвечающий за отображение поля ввода и активных кнопок.

MenuBar – класс отвечающий за меню

SplayTreeView – класс, собержащий в себе канвас и отвечающий за правильную отрисовку дерева

Пакет controllers – это контроллеры, служащие прослойками и контроллирующиеми классами

MenuController – контроллер, отвечающий за все, что происходит с меню

ParseController – вспомогательный контроллер, отвечающий за правильный разбор текстовых файлов и их представление в программе

SplayTreeController – контроллер, отвечающий за связь SplayTreeView и самой модели дерева

Соответственно пакет model содержит в себе модели самого дерева

SplayNode – узел дерева

SplayTree – само дерево

## 

## Листинг программы

Программа была написана в соответствии с парадигмой MVC.

Модели:

#### SplayNode.kt



#### SplayTree.kt

package com.pupptmstr.splaytree.model  
  
import java.util.\*  
  
class SplayTree : SortedSet<SplayNode> {  
 override var size = 0  
 private var root: SplayNode? = null  
 private val listOfHeadSets = ArrayList<SudoSet>()  
 private val listOfTailSets = ArrayList<SudoSet>()  
 private val listOfSubSets = ArrayList<SudoSet>()  
  
  
 fun add(element: Int): Boolean{  
 return add(SplayNode(element, null, null, null))  
 }  
  
 @Throws(java.lang.IllegalArgumentException::class)  
 override fun add(element: SplayNode?): Boolean {  
 try {  
 var current: SplayNode? = root  
 var parent: SplayNode? = null  
 while (current != null) {  
 parent = current  
 current = when {  
 (element!!.element > parent.element) -> current.right  
 (element.element < parent.element) -> current.left  
 else -> throw java.lang.IllegalArgumentException()  
 }  
 }  
 current = SplayNode(element!!.element, null, null, parent)  
 when {  
 (parent == null) -> root = current  
 (element.element > parent.element) -> parent.right = current  
 (element.element < parent.element) -> parent.left = current  
 }  
 splay(current)  
 size++  
 return true  
 } catch (e: Exception) {  
 e.printStackTrace()  
 return false  
 }  
 }  
  
 override fun addAll(elements: Collection<SplayNode>): Boolean {  
 elements.*forEach* **{** val answer = add(**it**)  
 if (!answer) {  
 return false  
 }  
 **}** return true  
 }  
  
 fun remove(element: Int): Boolean {  
 val node = findNode(element)  
 return remove(node)  
 }  
  
 override fun remove(element: SplayNode?): Boolean {  
 try {  
 if (element == null) {  
 throw IllegalArgumentException()  
 }  
 if (!contains(element)) {  
 throw IllegalArgumentException()  
 }  
  
 splay(element)  
  
 if ((element.left != null) && (element.right != null)) {  
 var min = element.left  
 while (min!!.right != null) {  
 min = min.right  
 }  
  
 min.right = element.right  
 element.right!!.parent = min  
 element.left!!.parent = null  
 root = element.left  
  
 } else if (element.right != null) {  
  
 element.right!!.parent = null  
 root = element.right  
  
 } else if (element.left != null) {  
  
 element.left!!.parent = null  
 root = element.left  
  
 } else {  
 root = null  
 }  
  
 element.parent = null  
 element.left = null  
 element.right = null  
 size--  
 return true  
 } catch (e: Exception) {  
 return false  
 }  
 }  
  
 override fun removeAll(elements: Collection<SplayNode>): Boolean {  
 elements.*forEach* **{** val answer = remove(**it**)  
 if (!answer) {  
 return false  
 }  
 **}** return true  
 }  
  
 override fun clear() {  
 root = null  
 size = 0  
 }  
  
 fun contains(element: Int): Boolean {  
 return contains(SplayNode(element, null, null, null))  
 }  
  
 override fun contains(element: SplayNode?): Boolean {  
 return findNode(element!!.element) != null  
 }  
  
 override fun containsAll(elements: Collection<SplayNode>): Boolean {  
 elements.*forEach* **{** val answer = contains(**it**)  
 if (!answer) {  
 return false  
 }  
 **}** return true  
 }  
  
 private fun findNode(element: Int): SplayNode? {  
 var prevNode: SplayNode? = null  
 var z = root  
  
 while (z != null) {  
 prevNode = z  
 z = when {  
 element > z.element -> z.right  
 element < z.element -> z.left  
 else -> {  
 splay(z)  
 return z  
 }  
 }  
  
 }  
  
 if (prevNode != null) {  
 splay(prevNode)  
 return null  
 }  
 return null  
 }  
  
 override fun iterator(): MutableIterator<SplayNode> = Iterator()  
  
 inner class Iterator internal constructor(): MutableIterator<SplayNode> {  
 private var nodes: Stack<SplayNode> = Stack()  
 private lateinit var current: SplayNode  
  
 init {  
 leftMostInorder(root)  
 }  
  
 private fun leftMostInorder(mainRoot: SplayNode?) {  
 var root = mainRoot  
 while (root != null) {  
 nodes.push(root)  
 root = root.left  
 }  
 }  
  
 override fun hasNext(): Boolean = nodes.size > 0  
  
 override fun next(): SplayNode {  
 current = nodes.pop();  
 if (current.right != null) {  
 leftMostInorder(current.right);  
 }  
 return current;  
 }  
  
 override fun remove() {  
 if (current != null) {  
 this@SplayTree.remove(current)  
 }  
 }  
 }  
  
 private fun inorderIterator(): List<String> {  
 return inorderIterator(root, 0)  
 }  
  
 private fun inorderIterator(r: SplayNode?, counter: Int): List<String> {  
 val res = *mutableListOf*<String>()  
 if (r != null) {  
 res.addAll(inorderIterator(r.left, counter+1))  
 res.add("${r.element}-$counter")  
 res.addAll(inorderIterator(r.right, counter + 1))  
 }  
 return res  
 }  
  
 override fun first(): SplayNode {  
 var current: SplayNode = root ?: throw NoSuchElementException()  
 while (current.left != null) {  
 current = current.left!!  
 }  
 return current  
 }  
  
 override fun last(): SplayNode {  
 var current: SplayNode = root ?: throw NoSuchElementException()  
 while (current.right != null) {  
 current = current.right!!  
 }  
 return current  
 }  
  
 inner class SudoSet(val fromElement: Int?, val toElement: Int?) : TreeSet<Int>() {  
 override fun add(element: Int): Boolean {  
 val value = element as Int? ?: return false  
 if (toElement == null) {  
 *require*(value >= fromElement!!)  
 if (!this@SplayTree.contains(value)) {  
 this@SplayTree.add(value)  
 }  
 super.add(value)  
 return true  
 }  
  
 if (fromElement == null) {  
 *require*(value < toElement)  
 if (!this@SplayTree.contains(value)) {  
 this@SplayTree.add(value)  
 }  
 super.add(value)  
 return true  
 }  
  
 if (value < toElement && value >= fromElement) {  
 if (!this@SplayTree.contains(value)) {  
 this@SplayTree.add(value)  
 }  
 super.add(value)  
 } else {  
 throw IllegalArgumentException()  
 }  
 return true  
 }  
  
 override fun remove(element: Int): Boolean {  
 val value = element as Int? ?: return false  
 this@SplayTree.remove(value)  
 return super.remove(value)  
 }  
 }  
  
 private fun updateSets() {  
 for (k in listOfSubSets) {  
 for (value in this) {  
 if (value < k.toElement!! && value >= k.fromElement!!) {  
 k.add(value.element)  
 }  
 }  
 }  
 for (k in listOfHeadSets) {  
 for (value in this) {  
 if (value < k.toElement!!) {  
 k.add(value.element)  
 } else {  
 break  
 }  
 }  
 }  
 for (k in listOfTailSets) {  
 for (value in this) {  
 if (value >= k.fromElement!!) {  
 k.add(value.element)  
 }  
 }  
 }  
 }  
 override fun tailSet(fromElement: SplayNode?): SortedSet<SplayNode> {  
 val tailSet = SudoSet(fromElement!!.element, null)  
 listOfTailSets.add(tailSet)  
 updateSets()  
 return tailSet as SortedSet<SplayNode>  
 }  
  
 override fun headSet(toElement: SplayNode?): SortedSet<SplayNode> {  
 val headSet = SudoSet(null, toElement!!.element)  
 listOfHeadSets.add(headSet)  
 updateSets()  
 return headSet as SortedSet<SplayNode>  
 }  
  
 override fun subSet(fromElement: SplayNode?, toElement: SplayNode?): SortedSet<SplayNode> {  
 *require*(fromElement!! < toElement!!.element)  
 val subSet = SudoSet(fromElement.element, toElement.element)  
 listOfSubSets.add(subSet)  
 updateSets()  
 return subSet as SortedSet<SplayNode>  
 }  
  
 override fun retainAll(elements: Collection<SplayNode>): Boolean {  
 try {  
 val existingElements = *mutableListOf*<Int>()  
 elements.*forEach* **{** if (contains(**it**.element)) {  
 existingElements.add(**it**.element)  
 }  
 **}** clear()  
 existingElements.*forEach* **{** add(**it**)  
 **}** return true  
 } catch (e: java.lang.Exception) {  
 e.printStackTrace()  
 return false  
 }  
 }  
  
 override fun isEmpty(): Boolean {  
 return root == null  
 }  
  
 override fun comparator(): Comparator<in SplayNode>? = null  
  
 private fun splay(aim: SplayNode) {  
 while (aim.parent != null) {  
 val parent = aim.parent  
 val grandParent = parent!!.parent  
  
 if (grandParent == null) {  
 if (aim == parent.left) {  
 makeLeftChildParent(aim, parent)  
 } else {  
 makeRightChildParent(aim, parent)  
 }  
 } else {  
 if (aim == parent.left) {  
 if (parent == grandParent.left) {  
 makeLeftChildParent(parent, grandParent)  
 makeLeftChildParent(aim, parent)  
 } else {  
 makeLeftChildParent(aim, aim.parent)  
 makeRightChildParent(aim, aim.parent)  
 }  
 } else {  
 if (parent == grandParent.left) {  
 makeRightChildParent(aim, aim.parent)  
 makeLeftChildParent(aim, aim.parent)  
 } else {  
 makeRightChildParent(parent, grandParent)  
 makeRightChildParent(aim, parent)  
 }  
 }  
 }  
 }  
 root = aim  
 }  
  
 private fun makeLeftChildParent(child: SplayNode?, parent: SplayNode?) {  
 if ((child == null) || (parent == null) || (parent.left != child) || (child.parent != parent)) {  
 throw RuntimeException("WRONG")  
 }  
  
 if (parent.parent != null) {  
 if (parent == parent.parent!!.left) {  
 parent.parent!!.left = child  
 } else {  
 parent.parent!!.right = child  
 }  
 }  
  
 if (child.right != null) {  
 child.right!!.parent = parent  
 }  
  
 child.parent = parent.parent  
 parent.parent = child  
 parent.left = child.right  
 child.right = parent  
 }  
  
 private fun makeRightChildParent(child: SplayNode?, parent: SplayNode?) {  
 if ((child == null) || (parent == null) || (parent.right != child) || (child.parent != parent)) {  
 throw RuntimeException("WRONG")  
 }  
  
 if (parent.parent != null) {  
 if (parent.parent!!.left == parent) {  
 parent.parent!!.left = child  
 } else {  
 parent.parent!!.right = child  
 }  
 }  
  
 if (child.left != null) {  
 child.left!!.parent = parent  
 }  
  
 child.parent = parent.parent  
 parent.parent = child  
 parent.right = child.left  
 child.left = parent  
 }  
  
 override fun toString() : String = inorderIterator().*joinToString*(separator = "//")  
  
 fun toList(): List<String> = inorderIterator()  
  
  
 override fun equals(other: Any?): Boolean {  
 if (this === other) return true  
 if (*javaClass* != other?.*javaClass*) return false  
  
 other as SplayTree  
  
 if (size != other.size) return false  
 if (root != other.root) return false  
  
 if (!*containsAll*(*listOf*(other.root))) return false  
  
 return true  
 }  
  
 override fun hashCode(): Int {  
 var result = size  
 result = 31 \* result + (root?.hashCode() ?: 0)  
 return result  
 }  
  
  
}

### Views:

#### ActionView.kt

package com.pupptmstr.splaytree.views  
  
import com.pupptmstr.splaytree.controllers.SplayTreeController  
import javafx.beans.property.SimpleStringProperty  
import javafx.scene.control.Alert  
import tornadofx.\*  
import java.lang.Exception  
import java.lang.IllegalArgumentException  
import java.lang.NumberFormatException  
import java.util.\*  
  
class ActionView : View() {  
 private val controller: SplayTreeController by inject()  
 private val input = SimpleStringProperty()  
  
 override val root = *form* **{** *fieldset* **{** *field*("Enter your integer") **{** *maxWidth* = 400.0  
 *textfield*(input)  
 **}** *button*("Add") **{** *action* **{** try {  
 *alert*(Alert.AlertType.INFORMATION, controller.addDataToTree(input.*value*.*toInt*()))  
 } catch (e: Exception) {  
 e.printStackTrace()  
 *alert*(Alert.AlertType.ERROR, "Неверный формат ввода")  
 }  
 **}  
 }** *button*("Remove") **{** *action* **{** try {  
 *alert*(Alert.AlertType.INFORMATION, controller.removeDataFromTree(input.*value*.*toInt*()))  
 } catch (e: NumberFormatException) {  
 e.printStackTrace()  
 *alert*(Alert.AlertType.ERROR, "Неверный формат ввода")  
 }  
  
 **}  
 }** *button*("Find") **{** *action* **{** try {  
 *alert*(Alert.AlertType.INFORMATION, controller.searchDataInTree(input.*value*.*toInt*()))  
 } catch (e: NumberFormatException) {  
 e.printStackTrace()  
 *alert*(Alert.AlertType.ERROR, "Неверный формат ввода")  
 }  
 **}  
 }  
 }  
 }**}

#### MainView.kt

package com.pupptmstr.splaytree.views  
  
import tornadofx.\*  
  
class MainView: View() {  
 private val actionView = find<ActionView>()  
 private val splayTreeView = find<SplayTreeView>()  
 private val menuBar = find<MenuBar>()  
   
 override val root = *borderpane* **{** *top* = menuBar.root  
 *center* = actionView.root  
 *bottom* = splayTreeView.root  
 **}**}

#### MenuBar.kt

package com.pupptmstr.splaytree.views  
  
import com.pupptmstr.splaytree.controllers.MenuController  
import com.pupptmstr.splaytree.controllers.SplayTreeController  
import javafx.scene.control.Alert  
import javafx.scene.control.ButtonType  
import tornadofx.\*  
import java.util.\*  
  
class MenuBar : View() {  
 private val controller: MenuController by inject()  
 private val treeController: SplayTreeController by inject()  
  
 override val root = menubar **{** menu("File") **{** *item*("Open").*action* **{** val content = controller.openFile()  
 val parsedData = controller.parser.parseData(content)  
 if (treeController.isNotEmpty()) {  
 val alert = Alert(Alert.AlertType.CONFIRMATION)  
 alert.*title* = "Сохранить?"  
 alert.*headerText* = "У вас есть несохраненное дерево"  
 alert.*contentText* = "Сохранить?"  
 alert.*buttonTypes*.clear()  
 alert.*buttonTypes*.addAll(ButtonType.*OK*, ButtonType.*NO*, ButtonType.*CANCEL*)  
 val option: Optional<ButtonType> = alert.showAndWait()  
 when (option.get()) {  
 ButtonType.*OK* -> {  
 controller.saveTreeToFile(treeController.getTree())  
 treeController.clearTree()  
 parsedData.*forEach* **{** treeController.addDataToTree(**it**) **}** }  
  
 ButtonType.*NO* -> {  
 treeController.clearTree()  
 parsedData.*forEach* **{** treeController.addDataToTree(**it**) **}** }  
 }  
 } else {  
 parsedData.*forEach* **{** treeController.addDataToTree(**it**) **}** }  
 **}** *item*("Save").*action* **{** controller.saveTreeToFile(treeController.getTree())  
 **}** item("//Export as picture")  
 **}** menu("Settings") **{** item("//For Project")  
  
 item("//For Program")  
  
 item("About").action **{** controller.showAboutWindow()  
 **}  
 }  
  
  
 }**}

#### SplayTreeView.kt

package com.pupptmstr.splaytree.views  
  
import javafx.scene.Parent  
import javafx.scene.canvas.GraphicsContext  
import javafx.scene.control.ScrollPane  
import javafx.scene.paint.Color  
import tornadofx.\*  
  
class SplayTreeView : View() {  
  
 private lateinit var gc: GraphicsContext  
  
 override val root: Parent = *scrollpane* **{** *hbarPolicy* = ScrollPane.ScrollBarPolicy.ALWAYS  
 *prefViewportHeight* = 1000.0  
 *prefViewportWidth* = 1000.0  
 *canvas*(5000.0, 5000.0) **{** gc = *graphicsContext2D* **}  
 }** fun drawShapes(list: List<String>) {  
 gc.clearRect(0.0, 0.0, 1000.0, 1000.0)  
 gc.*fill* = Color.*ALICEBLUE* gc.*stroke* = Color.*BLACK* gc.*lineWidth* = 1.0  
  
 var indexOfRoot: Int? = null  
 var levelNums = 0  
 for (i in list.*indices*) {  
 val level = list[i].*split*("-")[1].*toInt*()  
 if (level > levelNums) {  
 levelNums = level  
 }  
 }  
 //centerX = ((2.0.pow(levelNums) + levelNums) \* 50).toInt() + 10  
 val relationList = *mutableListOf*<Pair<Int, Int>>()//лист пар значение-уровень  
 //элементы расставлены для удобного определения отношений между узлами дерева  
 for (i in list.*indices*) {  
 val splitted = list[i].*split*("-")  
 val level = splitted[1].*toInt*()  
 val value = splitted[0].*toInt*()  
 if (level == 0) {  
 indexOfRoot = i  
 }  
 gc.strokeOval(findPlaceX(i), findPlaceY(level), 50.0, 50.0)  
 gc.strokeText(value.toString(), findPlaceX(i), findPlaceY(level) - 1.0)  
 relationList.add(Pair(value, level))  
 }  
 if (indexOfRoot != null) {  
 drawRelations(indexOfRoot, relationList)  
 }  
 }  
  
 private fun drawRelations(indexNow: Int, listOfElements: List<Pair<Int, Int>>) {  
 if (indexNow > 0) {  
 var leftChildIndex: Int? = null  
  
 for (i in indexNow *downTo* 0) {  
 if (listOfElements[i].second < listOfElements[indexNow].second)  
 break  
 if (listOfElements[i].second == listOfElements[indexNow].second + 1)  
 leftChildIndex = i  
 }  
 if (leftChildIndex != null) {  
   
 gc.strokeLine(  
 findPlaceX(indexNow) - 5.0,  
 findPlaceY(listOfElements[indexNow].second) + 5.0,  
 findPlaceX(leftChildIndex) + 10.0,  
 findPlaceY(listOfElements[leftChildIndex].second) - 15.0  
 )  
 drawRelations(leftChildIndex, listOfElements)  
 }  
  
  
 }  
 if (indexNow < listOfElements.size) {  
 var rightChildIndex: Int? = null  
  
 for (i in indexNow..listOfElements.*lastIndex*) {  
 if (listOfElements[i].second < listOfElements[indexNow].second)  
 break  
 if (listOfElements[i].second == listOfElements[indexNow].second + 1)  
 rightChildIndex = i  
 }  
  
 if (rightChildIndex != null) {  
 gc.strokeLine(  
 findPlaceX(indexNow) + 50.0,  
 findPlaceY(listOfElements[indexNow].second) + 50.0,  
 findPlaceX(rightChildIndex) - 5.0,  
 findPlaceY(listOfElements[rightChildIndex].second) - 5.0  
 )  
 drawRelations(rightChildIndex, listOfElements)  
 }  
  
 }  
 }  
  
  
 private fun findPlaceX(positionInList: Int): Double {  
 return ((75 \* positionInList) + 15).toDouble()  
  
 }  
  
 private fun findPlaceY(level: Int): Double {  
 return (level \* 75 + 15).toDouble()  
  
 }  
  
  
}

### Контроллеры:

#### MenuController.kt

package com.pupptmstr.splaytree.controllers  
  
import com.pupptmstr.splaytree.model.SplayTree  
import javafx.scene.control.Alert  
import javafx.scene.control.ButtonType  
import javafx.scene.layout.Region  
import javafx.stage.FileChooser  
import tornadofx.\*  
import java.io.BufferedWriter  
import java.io.File  
import java.io.FileWriter  
import java.lang.IllegalArgumentException  
import java.lang.NumberFormatException  
  
class MenuController: Controller() {  
 //вынести парсинг в отдельный контроллер  
 private val aboutText = "Создатель: Курняков Пётр Михайлович(pupptmstr)\n" +  
 "студент группы 3530901/80003, а так же молодой и подающий надежды программист по совместительству\n" +  
 "Это приложение создано в качестве курсовой работы по предмету `Алгоритмы и структуры данных`\n\n" +  
 "Визуализация алгоритмов работы `splay` дерева\n\n" +  
 "Курняков Пётр 2019 СПбПУ ©"  
 val parser: ParseController by inject()  
  
 fun openFile(): List<String> {  
 val files: List<File> = *chooseFile*("Choose `.tr` or `.txt` file", *emptyArray*(), FileChooserMode.Single)  
 val res = *mutableListOf*<String>()  
  
 if (files.*isNotEmpty*()) {  
 files.*forEach* **{** res.add(**it**.*absolutePath*)  
 **}** }  
  
 return res  
 }  
  
  
 //Метод для общего определения(какого типа файл -> как его нужно парсить  
  
  
 @Throws(FileSystemException::class)  
 fun saveTreeToFile(tree: SplayTree): Boolean {  
 val fileChooser = FileChooser()  
 fileChooser.*title* = "Save Document"  
 val file = fileChooser.showSaveDialog(primaryStage)  
 return if (file != null) {  
 val writer = BufferedWriter(FileWriter(file, true))  
 writer.write(tree.toString())  
 writer.close()  
 true  
 } else {  
 false  
 }  
 }  
  
 fun showAboutWindow() {  
 val alert = Alert(Alert.AlertType.INFORMATION, aboutText, ButtonType.*OK*)  
 alert.*dialogPane*.*minHeight* = Region.*USE\_PREF\_SIZE* alert.showAndWait()  
 }  
}

#### ParseController.kt

package com.pupptmstr.splaytree.controllers  
  
import javafx.scene.control.Alert  
import tornadofx.Controller  
import tornadofx.alert  
import java.io.File  
import java.lang.IllegalArgumentException  
import java.lang.NumberFormatException  
  
class ParseController: Controller() {  
 @Throws(IllegalArgumentException::class)  
 fun parseData(input: List<String>): List<Int> {  
 val res = *mutableListOf*<Int>()  
 input.*forEach* **{** val file = File(**it**)  
 if (file.*isFile*) {  
 *println*(file.*extension*)  
 if (file.*extension* == "tr") {  
 res += this.parseDataTr(file.*readText*())  
 } else if(file.*extension* == "txt") {  
 res += this.parseDataTxt(file.*readText*())  
 }  
  
 } else {  
 throw IllegalArgumentException()  
 }  
 **}** return res  
 }  
  
 //Метод, достающий данные и правильно парсящий файлы, с расширением ".tr"  
 private fun parseDataTr(input: String): List<Int> {  
 val res = *mutableListOf*<Int>()  
 val splitted = input.*split*("//")  
 for (i in splitted.*indices*) {  
 try {  
 val num = splitted[i].*split*("-")[0].*toInt*()  
 res.add(num)  
 } catch (e: NumberFormatException) {  
 *alert*(  
 Alert.AlertType.ERROR,  
 "${splitted[i]} is not a Number",  
 "It will not be added to the Tree")  
 }  
 }  
 return res  
 }  
  
 //Метод, достающий данные и правильно парсящий файлы, с расширением ".txt"  
 private fun parseDataTxt(input: String): List<Int> {  
 val res = *mutableListOf*<Int>()  
 val splitted = input.split(Regex("""[\t\n\r\s-]+"""))  
 for (i in splitted.indices) {  
 try {  
 val num = splitted[i].toInt()  
 res.add(num)  
 } catch (e: NumberFormatException) {  
 alert(  
 Alert.AlertType.ERROR,  
 "${splitted[i]} is not a Number",  
 "It will not be added to the Tree")  
 }  
 }  
 return res  
 }  
}

#### SplayTreeController.kt

package com.pupptmstr.splaytree.controllers  
  
import com.pupptmstr.splaytree.model.SplayTree  
import com.pupptmstr.splaytree.views.SplayTreeView  
import tornadofx.Controller  
import java.lang.Exception  
import java.util.\*  
  
class SplayTreeController : Controller() {  
  
 private val splayTree = SplayTree()  
  
 fun addDataToTree(inputValue: Int): String {  
 val ans = splayTree.add(inputValue)  
 return if (ans) {  
 *println*(splayTree.toString())  
 find<SplayTreeView>().drawShapes(splayTree.toList())  
 "Добавлено"  
 } else {  
 "Такой элемент уже существует"  
 }  
 }  
  
 fun removeDataFromTree(inputValue: Int): String {  
 val ans = splayTree.remove(inputValue)  
 return if (ans) {  
 *println*(splayTree.toString())  
 find<SplayTreeView>().drawShapes(splayTree.toList())  
 "Удалено"  
 } else {  
 "Удалять нечего. Такого элемента не существует."  
 }  
 }  
  
 fun searchDataInTree(inputValue: Int): String {  
 val ans = splayTree.contains(inputValue)  
 *println*(splayTree.toString())  
 find<SplayTreeView>().drawShapes(splayTree.toList())  
 return when (ans) {  
 true -> "Элемент найден"  
 false -> "Элемента нет в дереве"  
 }  
 }  
  
 fun getTree(): SplayTree {  
 return splayTree  
 }  
  
 fun clearTree() {  
 splayTree.clear()  
 }  
  
 fun isNotEmpty(): Boolean {  
 return splayTree.*isNotEmpty*()  
 }  
}

### Main-класс:

package com.pupptmstr.splaytree.views  
  
import javafx.scene.control.Alert  
import tornadofx.\*  
import java.text.ParseException  
  
fun main(args: Array<String>) {  
 try {  
 *launch*<MyApp>(args)  
 } catch (e: ParseException) {  
 *alert*(Alert.AlertType.ERROR, "Неправильное число в строке, ну ка введи норально")  
 }  
  
}  
  
class MyApp : App(MainView::class)

### Тесты:

#### SplayTreeTest.kt

package com.pupptmstr.splaytree  
  
import com.pupptmstr.splaytree.model.SplayTree  
import junit.framework.Assert.assertEquals  
import org.junit.Test  
  
class SplayTreeTest {  
  
 @Test  
 fun testAdd() {  
 val splayTree = SplayTree()  
 splayTree.add(5)  
 splayTree.add(18)  
 splayTree.add(45)  
 splayTree.add(20)  
 assertEquals("5-2//18-1//20-0//45-1", splayTree.toString())  
 }  
  
 @Test  
 fun testDelete() {  
 val splayTree = SplayTree()  
 splayTree.add(5)  
 splayTree.add(18)  
 splayTree.add(45)  
 splayTree.add(20)  
 splayTree.remove(18)  
 assertEquals("5-0//20-1//45-2", splayTree.toString())  
 }  
  
 @Test  
 fun testFind() {  
 val splayTree = SplayTree()  
 splayTree.add(5)  
 splayTree.add(18)  
 splayTree.add(45)  
 splayTree.add(20)  
 val beforeFind = splayTree.toString()  
 splayTree.contains(18)  
 assertEquals("5-2//18-1//20-0//45-1 -- 5-1//18-0//20-1//45-2",  
 "$beforeFind -- $splayTree"  
 )  
 }  
  
 }

### build.gradle:

plugins **{** id 'java'  
 id 'org.jetbrains.kotlin.jvm' version '1.3.41'  
 id 'com.github.johnrengelman.shadow' version '4.0.4'  
**}**group 'com.pupptmstr'  
version '0.3'  
  
sourceCompatibility = 1.8  
  
repositories **{** mavenCentral()  
**}**dependencies **{** implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8"  
 testCompile group: 'junit', name: 'junit', version: '4.12'  
 compile 'no.tornado:tornadofx:1.7.19'  
**}**compileKotlin **{** kotlinOptions.jvmTarget = "1.8"  
**}**compileTestKotlin **{** kotlinOptions.jvmTarget = "1.8"  
**}**jar **{** manifest **{** attributes 'Main-Class': 'com.pupptmstr.splaytree.MainKt'  
 **}** from **{** configurations.compile.collect **{** it.isDirectory() ? it : zipTree(it) **} }  
}**

## Скриншоты графического приложения

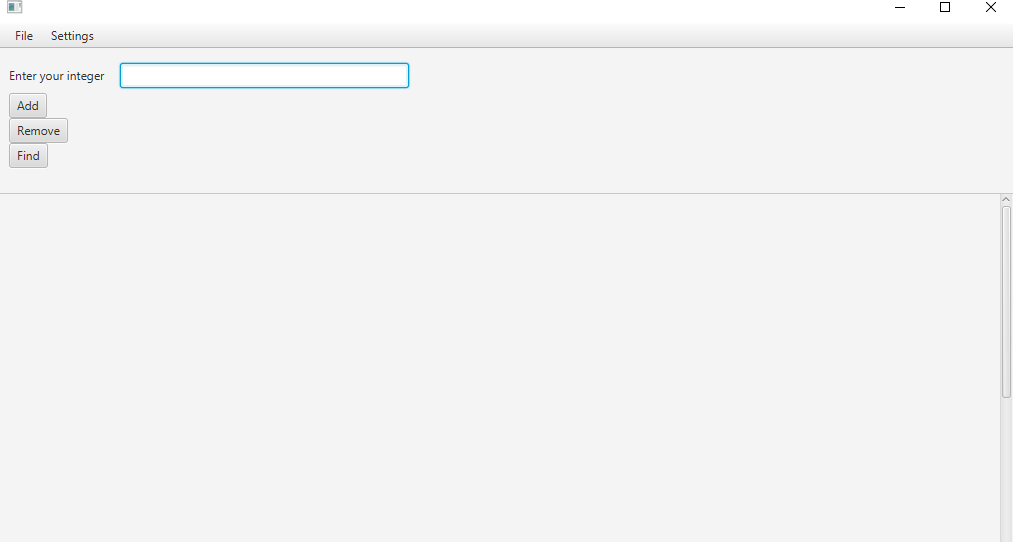


Рис.1 Главное окно приложения (при запуске)

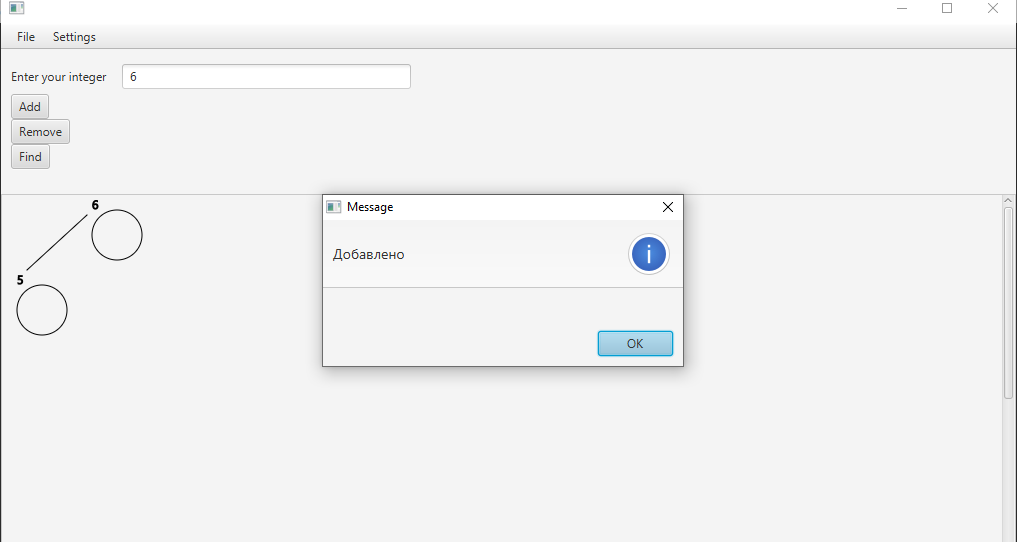


Рис.2 Последовательно добавлены элементы 5 и 6

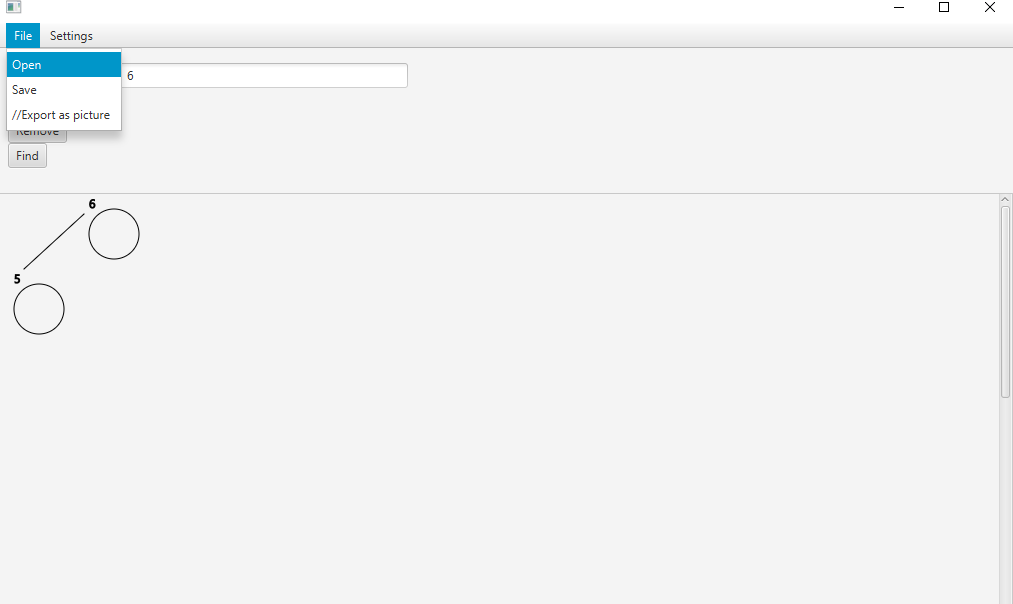


Рис.3 Наведена мышь на пункт меню Open

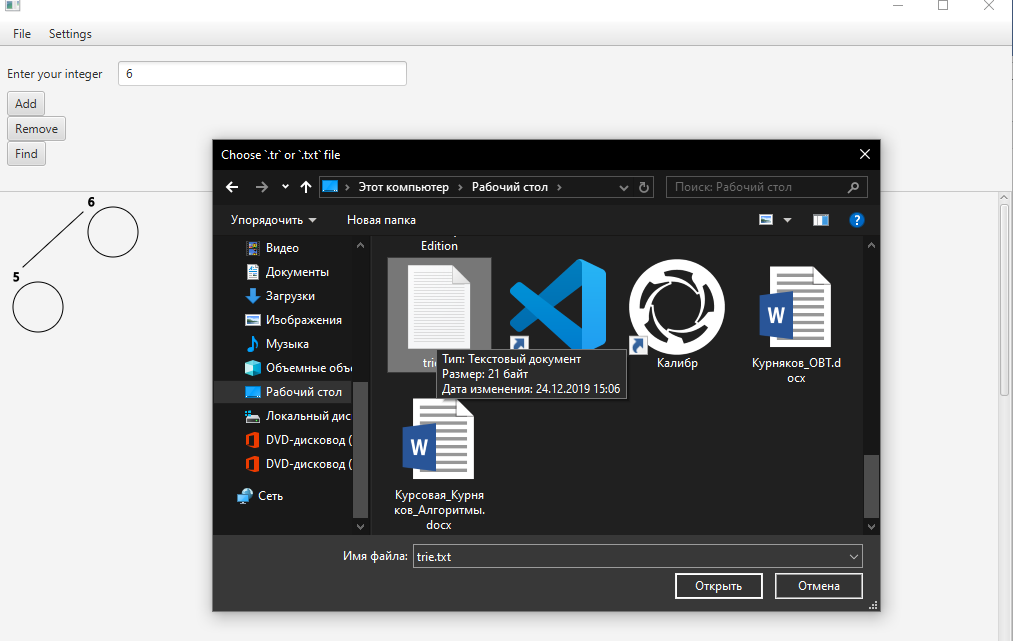


Рис.4 Процесс открытия файла

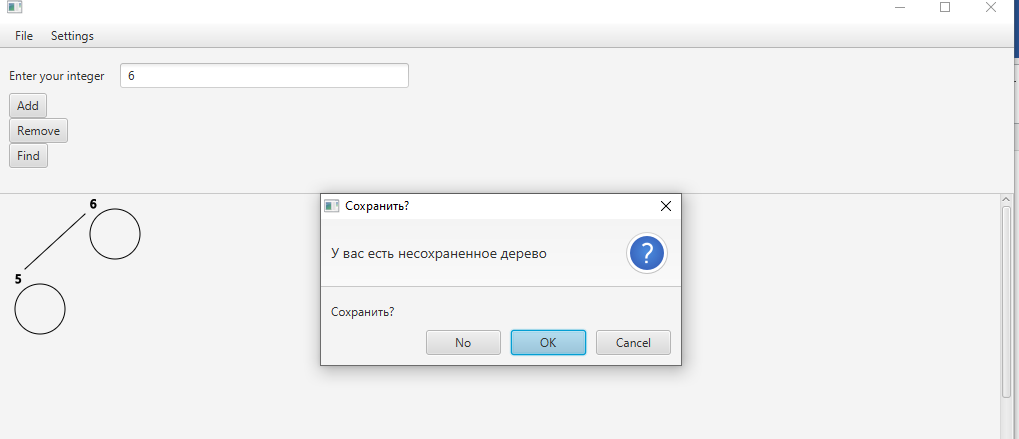


Рис.5 Выбрано прочтение дерево из файла, программа предлагает сохранить текущее дерево

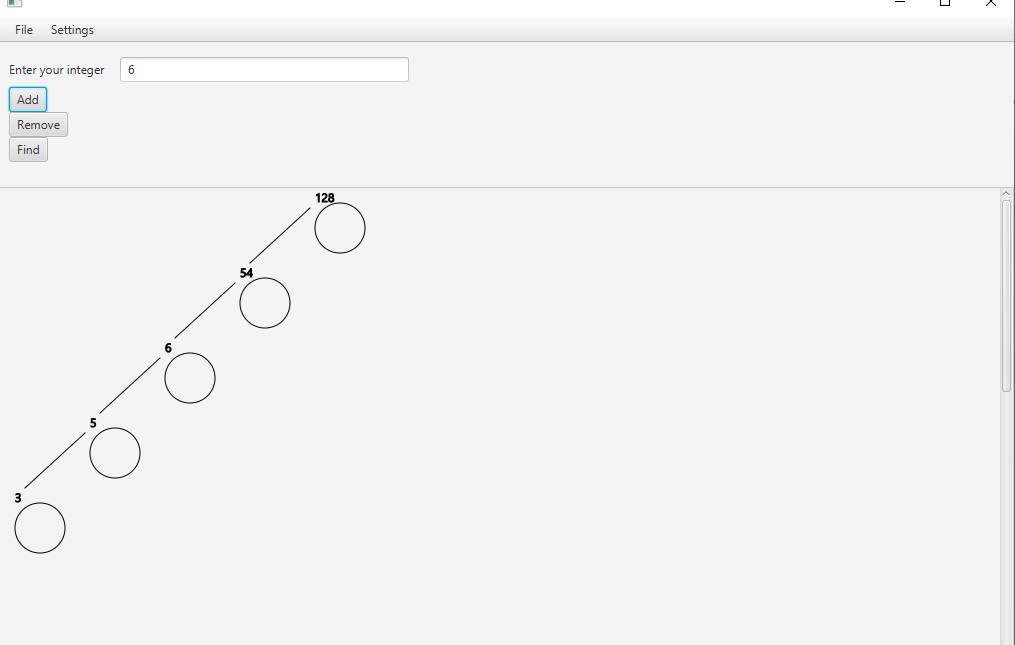


Рис.6 Прочитано дерево из файла

Содержание файла:

3 5 6 54 128

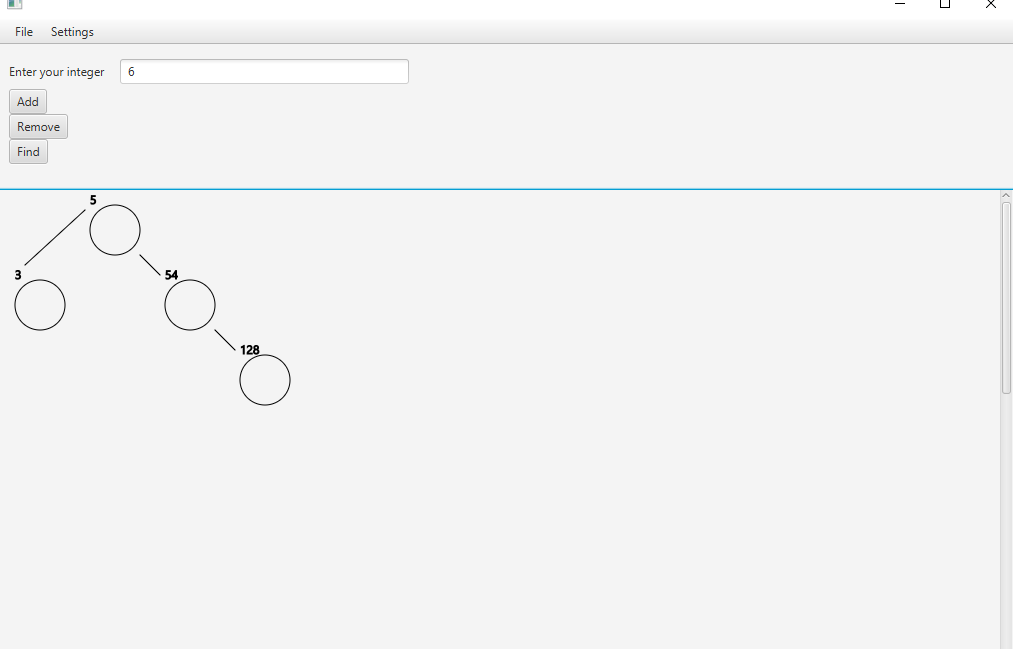


Рис.7 Выполнено удаление элемента 6 в дереве



Рис.8 Выбран пункт About

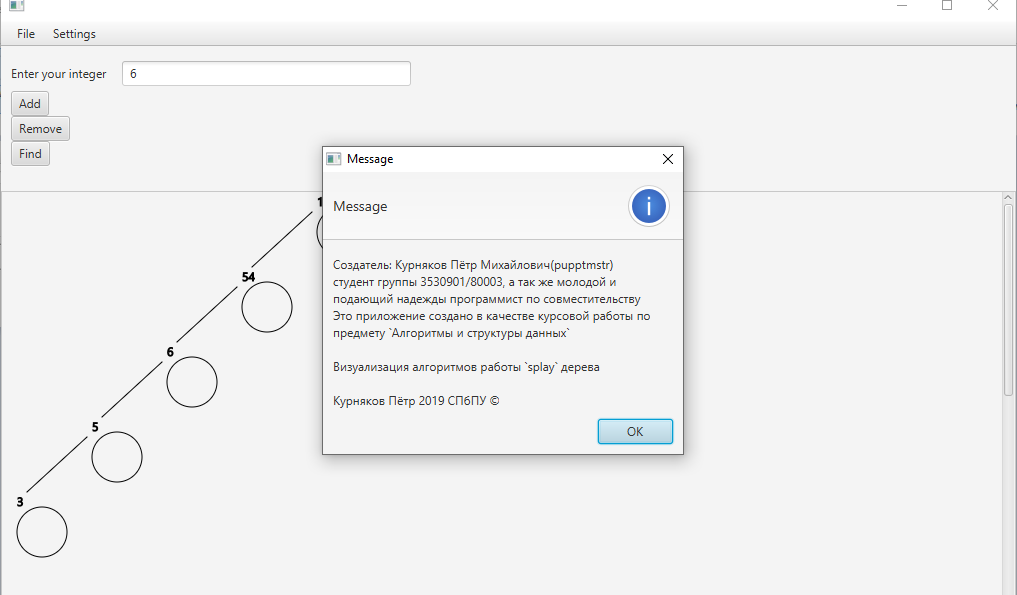


Рис.9 Показ окна About