ФГАОУ ВО «Севастопольский государственный университет»

Институт информационных технологий   
и управления в технических системах

ОТЧЁТ  
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Исследование способов реализации классов и объектов в языке Scala»

по дисциплине «Парадигмы современных языков программирования»

Выполнила:  
студентка группы ИС/м-21-1-з  
Ускова Екатерина Дмитриевна

Севастополь

2022

# ЦЕЛЬ РАБОТЫ

Исследовать особенности реализации классов, объектов и трейтов в языке Scala. Реализовать программные модули для обработки сложных структур данных, используя объектно-ориентированный и функциональный подходы.

# Ход работы

Реализована структура бинарного дерева поиска. Для реализации:

Написан трейт Tree, определяющий абстрактные методы getLeftSubtree: Tree, getRightSubtree: Tree и getNodeData: Int.

trait Tree {  
 def getLeftSubtree: Tree  
 def getRightSubtree: Tree  
 def getNodeData: Int  
}

Написан класс Node, который реализует трейт Tree и представляет собой узел дерева.

class Node (data: Int, var left: Tree, var right: Tree) extends Tree{  
 override def getLeftSubtree: Tree = left  
 override def getRightSubtree: Tree = right  
 override def getNodeData: Int = data  
  
 def add(el: Int): Unit = {  
 if(getNodeData > el) {  
 if(getLeftSubtree == null)  
 left = new Leaf(el)  
 else {  
 if(left.isInstanceOf[Leaf])  
 left = new Node(left.getNodeData, null, null)  
 left.asInstanceOf[Node].add(el)  
 }  
 } else {  
 if (getRightSubtree == null)  
 right = new Leaf(el)  
 else {  
 if (right.isInstanceOf[Leaf])  
 right = new Node(right.getNodeData, null, null)  
 right.asInstanceOf[Node].add(el)  
 }  
 }  
 }  
}

Написан класс Leaf, который реализует трейт Tree и представляет собой лист дерева.

class Leaf (data: Int) extends Tree {  
 override def getLeftSubtree: Tree = null  
 override def getRightSubtree: Tree = null  
 override def getNodeData: Int = data  
}

Для реализованной структуры дерева написаны следующие функции:

Функция printTree(Tree): Unit, которая выводит на экран дерево.

def printTree(tree: Tree): Unit = {  
 def printTreeLevel(tree: Tree, lv: Int): Unit = {  
 println("-" \* lv + tree.getNodeData)  
 if (tree.getLeftSubtree != null)  
 printTreeLevel(tree.getLeftSubtree, lv + 1)  
 if (tree.getRightSubtree != null)  
 printTreeLevel(tree.getRightSubtree, lv + 1)  
 }  
 printTreeLevel(tree, 0)  
}

Функция insert(Int, Tree): Tree, которая принимает элемент для вставки и корень дерева, возвращает корень нового дерева со вставленным элементом (при этом изначальное дерево не изменяется).

def insert(el: Int, tree: Tree): Tree = {  
 var newTree: Node = null //new Node(tree.getNodeData, null, null)  
 def copyTree(tree: Tree): Node = {  
 if(newTree == null)  
 newTree = new Node(tree.getNodeData, null, null)  
 else  
 newTree.add(tree.getNodeData)  
 if (tree.getLeftSubtree != null) {  
 copyTree(tree.getLeftSubtree)  
 }  
 if (tree.getRightSubtree != null)  
 copyTree(tree.getRightSubtree)  
 newTree  
 }  
 newTree = copyTree(tree)  
 newTree.add(el)  
 newTree  
}

Функция contains(Int, Tree): Boolean, которая возвращает true или false в зависимости от того, содержится ли заданное число в дереве, или нет.

def contains(el: Int, tree: Tree): Boolean = {  
 var res = false  
 def seekInNodes(node: Tree): Boolean = {  
 res = res || (node.getNodeData == el)  
 if (node.getLeftSubtree != null) {  
 seekInNodes(node.getLeftSubtree)  
 }  
 if (node.getRightSubtree != null) {  
 seekInNodes(node.getRightSubtree)  
 }  
 res  
 }  
 res = seekInNodes(tree)  
 res  
}

Функция sum(Tree): Int, которая возвращает сумму всех элементов в дереве.

def sum(tree: Tree): Int = {  
 var sum = 0  
 def sumInNodes(node: Tree): Int = {  
 sum += node.getNodeData  
 if (node.getLeftSubtree != null) {  
 sumInNodes(node.getLeftSubtree)  
 }  
 if (node.getRightSubtree != null) {  
 sumInNodes(node.getRightSubtree)  
 }  
 sum  
 }  
 sum = sumInNodes(tree)  
 sum  
}

# Вывод

Выполняя эту работу, я исследовала особенности реализации классов, объектов и трейтов в языке Scala. Реализовала программные модули для обработки сложных структур данных, используя объектно-ориентированный и функциональный подходы.

# Приложение А

Листинг программного кода

package main.scala  
  
trait Tree {  
 def getLeftSubtree: Tree  
 def getRightSubtree: Tree  
 def getNodeData: Int  
}  
  
class Node (data: Int, var left: Tree, var right: Tree) extends Tree{  
 override def getLeftSubtree: Tree = left  
 override def getRightSubtree: Tree = right  
 override def getNodeData: Int = data  
  
 def add(el: Int): Unit = {  
 if(getNodeData > el) {  
 if(getLeftSubtree == null)  
 left = new Leaf(el)  
 else {  
 if(left.isInstanceOf[Leaf])  
 left = new Node(left.getNodeData, null, null)  
 left.asInstanceOf[Node].add(el)  
 }  
 } else {  
 if (getRightSubtree == null)  
 right = new Leaf(el)  
 else {  
 if (right.isInstanceOf[Leaf])  
 right = new Node(right.getNodeData, null, null)  
 right.asInstanceOf[Node].add(el)  
 }  
 }  
 }  
}  
  
class Leaf (data: Int) extends Tree {  
 override def getLeftSubtree: Tree = null  
 override def getRightSubtree: Tree = null  
 override def getNodeData: Int = data  
}  
  
object Application {  
 var tree = new Node(8, new Node(3, new Leaf(2), new Leaf(5)), new Node(11, new Leaf(9), null))  
  
 def printTree(tree: Tree): Unit = {  
 def printTreeLevel(tree: Tree, lv: Int): Unit = {  
 println("-" \* lv + tree.getNodeData)  
 if (tree.getLeftSubtree != null)  
 printTreeLevel(tree.getLeftSubtree, lv + 1)  
 if (tree.getRightSubtree != null)  
 printTreeLevel(tree.getRightSubtree, lv + 1)  
 }  
 printTreeLevel(tree, 0)  
 }  
  
 def insert(el: Int, tree: Tree): Tree = {  
 var newTree: Node = null //new Node(tree.getNodeData, null, null)  
 def copyTree(tree: Tree): Node = {  
 if(newTree == null)  
 newTree = new Node(tree.getNodeData, null, null)  
 else  
 newTree.add(tree.getNodeData)  
 if (tree.getLeftSubtree != null) {  
 copyTree(tree.getLeftSubtree)  
 }  
 if (tree.getRightSubtree != null)  
 copyTree(tree.getRightSubtree)  
 newTree  
 }  
 newTree = copyTree(tree)  
 newTree.add(el)  
 newTree  
 }  
  
 def contains(el: Int, tree: Tree): Boolean = {  
 var res = false  
  
 def seekInNodes(node: Tree): Boolean = {  
 res = res || (node.getNodeData == el)  
 if (node.getLeftSubtree != null) {  
 seekInNodes(node.getLeftSubtree)  
 }  
 if (node.getRightSubtree != null) {  
 seekInNodes(node.getRightSubtree)  
 }  
  
 res  
 }  
  
 res = seekInNodes(tree)  
 res  
 }  
  
 def sum(tree: Tree): Int = {  
 var sum = 0  
  
 def sumInNodes(node: Tree): Int = {  
 sum += node.getNodeData  
 if (node.getLeftSubtree != null) {  
 sumInNodes(node.getLeftSubtree)  
 }  
 if (node.getRightSubtree != null) {  
 sumInNodes(node.getRightSubtree)  
 }  
  
 sum  
 }  
  
 sum = sumInNodes(tree)  
 sum  
 }  
  
 def main (args: Array[String]): Unit = {  
 printTree(tree)  
// printTree(insert(4, tree))  
// printTree(tree)  
// println(contains(9, tree))  
// println(sum(tree))  
 }  
  
}