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) extends Tree{  
 var left: Node = null  
 var right: Node = null  
  
 override def getLeftSubtree: Node = left  
 override def getRightSubtree: Node = right  
 override def getNodeData: Int = data  
}

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  
}

Класс для дерева:

class BSTree extends Tree{  
 var head: Node = null  
  
 def Add(el: Int): Unit = {  
 if(head == null)  
 head = new Node(el)  
 else  
 AddTo(head, el)  
 }  
  
 def AddTo(node: Node, el: Int): Unit = {  
 if(node.getNodeData > el) {  
 if(node.getLeftSubtree == null)  
 node.left = new Node(el)  
 else AddTo(node.left, el)  
 } else {  
 if (node.getRightSubtree== null)  
 node.right = new Node(el)  
 else AddTo(node.right, el)  
 }  
 }  
  
 override def getLeftSubtree: Node = {  
 if (head == null)  
 null  
 else  
 head.getLeftSubtree  
 }  
  
 override def getRightSubtree: Node = {  
 if (head == null)  
 null  
 else  
 head.getRightSubtree  
 }  
  
 override def getNodeData: Int = {  
 if (head == null)  
 -1  
 else  
 head.getNodeData  
 }  
  
}

Для реализованной структуры дерева написаны следующие функции:

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

def printTree(tree: BSTree): Unit = {  
 def printNode(node: Node): Unit = {  
 println(node.getNodeData)  
  
 if (node.getLeftSubtree != null) {  
 printNode(node.getLeftSubtree)  
 }  
 else println(s"${node.getNodeData} no left")  
  
 if (node.getRightSubtree != null) {  
 printNode(node.getRightSubtree)  
 }  
 else println(s"${node.getNodeData} no right")  
 }  
  
 if(tree.getNodeData == -1) println("empty")  
 else printNode(tree.head)  
  
 }

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: BSTree): BSTree = {  
 var newTree = new BSTree  
  
 def copyNodes(node: Node): BSTree = {  
 newTree.Add(node.getNodeData)  
  
 if (node.getLeftSubtree != null) {  
 copyNodes(node.getLeftSubtree)  
 }  
  
 if (node.getRightSubtree != null) {  
 copyNodes(node.getRightSubtree)  
 }  
  
 newTree  
 }  
  
 if(tree.getNodeData == -1) println("empty")  
 else newTree = copyNodes(tree.head)  
 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: BSTree): Boolean = {  
 var res = false  
  
 def seekInNodes(node: Node): Boolean = {  
 res = res || (node.getNodeData == el)  
  
 if (node.getLeftSubtree != null) {  
 seekInNodes(node.getLeftSubtree)  
 }  
  
 if (node.getRightSubtree != null) {  
 seekInNodes(node.getRightSubtree)  
 }  
  
 res  
 }  
  
 if (tree.getNodeData == -1) println("empty")  
 else res = seekInNodes(tree.head)  
 res  
}

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

def sum(tree: BSTree): Int = {  
 var sum = 0  
  
 def sumInNodes(node: Node): Int = {  
 sum += node.getNodeData  
  
 if (node.getLeftSubtree != null) {  
 sumInNodes(node.getLeftSubtree)  
 }  
  
 if (node.getRightSubtree != null) {  
 sumInNodes(node.getRightSubtree)  
 }  
  
 sum  
 }  
  
 if (tree.getNodeData == -1) println("empty")  
 else sum = sumInNodes(tree.head)  
 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) extends Tree{  
 var left: Node = null  
 var right: Node = null  
  
 override def getLeftSubtree: Node = left  
 override def getRightSubtree: Node = right  
 override def getNodeData: Int = data  
}  
  
class Leaf (data: Int) extends Tree {  
 override def getLeftSubtree: Tree = null  
 override def getRightSubtree: Tree = null  
 override def getNodeData: Int = data  
}  
  
class BSTree extends Tree{  
 var head: Node = null  
  
 def Add(el: Int): Unit = {  
 if(head == null)  
 head = new Node(el)  
 else  
 AddTo(head, el)  
 }  
  
 def AddTo(node: Node, el: Int): Unit = {  
 if(node.getNodeData > el) {  
 if(node.getLeftSubtree == null)  
 node.left = new Node(el)  
 else AddTo(node.left, el)  
 } else {  
 if (node.getRightSubtree== null)  
 node.right = new Node(el)  
 else AddTo(node.right, el)  
 }  
 }  
  
 override def getLeftSubtree: Node = {  
 if (head == null)  
 null  
 else  
 head.getLeftSubtree  
 }  
  
 override def getRightSubtree: Node = {  
 if (head == null)  
 null  
 else  
 head.getRightSubtree  
 }  
  
 override def getNodeData: Int = {  
 if (head == null)  
 -1  
 else  
 head.getNodeData  
 }  
  
}  
  
object Application {  
 def main (args: Array[String]): Unit = {  
  
 def printTree(tree: BSTree): Unit = {  
 def printNode(node: Node): Unit = {  
 println(node.getNodeData)  
  
// println("/")  
 if (node.getLeftSubtree != null) {  
// print("-")  
 printNode(node.getLeftSubtree)  
 }  
 else println(s"${node.getNodeData} no left")  
  
// println("\\")  
 if (node.getRightSubtree != null) {  
// print("-")  
 printNode(node.getRightSubtree)  
 }  
 else println(s"${node.getNodeData} no right")  
 }  
  
 if(tree.getNodeData == -1) println("empty")  
 else printNode(tree.head)  
  
 }  
  
 def insert(el: Int, tree: BSTree): BSTree = {  
 var newTree = new BSTree  
  
 def copyNodes(node: Node): BSTree = {  
 newTree.Add(node.getNodeData)  
  
 if (node.getLeftSubtree != null) {  
 copyNodes(node.getLeftSubtree)  
 }  
  
 if (node.getRightSubtree != null) {  
 copyNodes(node.getRightSubtree)  
 }  
  
 newTree  
 }  
  
 if(tree.getNodeData == -1) println("empty")  
 else newTree = copyNodes(tree.head)  
 newTree.Add(el)  
 newTree  
 }  
  
 def contains(el: Int, tree: BSTree): Boolean = {  
 var res = false  
  
 def seekInNodes(node: Node): Boolean = {  
 res = res || (node.getNodeData == el)  
  
 if (node.getLeftSubtree != null) {  
 seekInNodes(node.getLeftSubtree)  
 }  
  
 if (node.getRightSubtree != null) {  
 seekInNodes(node.getRightSubtree)  
 }  
  
 res  
 }  
  
 if (tree.getNodeData == -1) println("empty")  
 else res = seekInNodes(tree.head)  
 res  
 }  
  
 def sum(tree: BSTree): Int = {  
 var sum = 0  
  
 def sumInNodes(node: Node): Int = {  
 sum += node.getNodeData  
  
 if (node.getLeftSubtree != null) {  
 sumInNodes(node.getLeftSubtree)  
 }  
  
 if (node.getRightSubtree != null) {  
 sumInNodes(node.getRightSubtree)  
 }  
  
 sum  
 }  
  
 if (tree.getNodeData == -1) println("empty")  
 else sum = sumInNodes(tree.head)  
 sum  
 }  
  
 val testTree = new BSTree  
  
 testTree.Add(8)  
 testTree.Add(3)  
 testTree.Add(5)  
 testTree.Add(2)  
 testTree.Add(11)  
 testTree.Add(9)  
  
 printTree(testTree)  
// printTree(insert(4, testTree))  
// println(contains(4, testTree))  
 println(sum(testTree))  
 }  
  
}