Министерство Образования, Культуры и Исследований

**Молдавский Государственный Университет**

**Факультет Математики и Информатики**

**Департамент Информатики**

Algoritmi, Structuri de Date și Complexitate

Отчёт по лабораторной работе №3.14

Выполнил: Maryia Rotar IA2102

Проверил преподаватель: Mihail Croitor

Кишинев, 2023

**Задание:**

Данная лабораторная работа посвящена написанию юнит тестов.

Разрешается использование специализированных фреймворков.

Порядок выполнения

Создайте новый проект и клонируйте в него проект, полученный в результате выполнения **лабораторной работы №3**.

Создайте папку tests. Для каждого класса из проекта создайте файл с тестами.

Отредактируйте **README.md** файл, добавив пункт тестирования, поясняющий, как запускать тесты.

Начисление баллов

1. создание тестового файла для каждого класса - *2 балла*
2. прописывание тестов для каждого метода - *4 балла*
3. расширеный **README.md** файл - *2 балла*
4. проект защищен перед коллегами - *2 балла*

**Бинарное дерево поиска**

**Прямой обход**:

Для тестирования метода создаем список с продуктами, из него получаем двоичное дерево и далее в try-finally перенаправляем результаты system.out в объект класса StringBuilder, который позволяет динамически расширять и сокращать строку. Далее, с помощью метода фреймворка (Assert.*assertEquals*) сравниваем выводимый результат с ожидаемым.

@Test  
public void testPreorderTraversal() {  
 // Создание экземпляра класса BalancedBinaryTree  
 BalancedBinaryTree tree = new BalancedBinaryTree();  
  
 // Создание тестовых данных  
 List<Product> productList = new ArrayList<>();  
 productList.add(new Product(1, "Product 1", "Description 1", 10.50f, 50, Measure.*XS*));  
 productList.add(new Product(2, "Product 2", "Description 2", 30.55f, 40, Measure.*S*));  
 productList.add(new Product(3, "Product 3", "Description 3", 55.90f, 30, Measure.*XL*));  
  
 // Построение балансированного двоичного дерева  
 BinaryTree binaryTree = tree.makeBinaryTree(productList, 0, productList.size() - 1);  
  
 // Создание StringBuilder для записи вывода  
 StringBuilder output = new StringBuilder();  
  
 // Перенаправление вывода в StringBuilder  
 PrintStream printStream = new PrintStream(new ByteArrayOutputStream()) {  
 @Override  
 public void print(String s) {  
 output.append(s);  
 }  
 };  
  
 // Сохранение стандартного потока вывода  
 PrintStream oldOut = System.*out*;  
  
 try {  
 // Установка нового потока вывода  
 System.*setOut*(printStream);  
  
 // Вызов метода preorderTraversal с корневым узлом дерева  
 tree.*preorderTraversal*(binaryTree);  
  
 } finally {  
 // Восстановление стандартного потока вывода  
 System.*setOut*(oldOut);  
 }  
  
 // Создание ожидаемого результата  
 StringBuilder expectedOutput = new StringBuilder();  
 expectedOutput.append("Node---{id=2," +  
 "\n left\_for\_2 [Node---{id=1," +  
 "\n left\_for\_1 [null," +  
 "\n \t\tright\_for\_1 [null]--- },"+  
 "\n \t\tright\_for\_2 [Node---{id=3,"+  
 "\n left\_for\_3 [null,"+  
 "\n \t\tright\_for\_3 [null]--- }]--- } Node---{id=1,"+  
 "\n left\_for\_1 [null,"+  
 "\n \t\tright\_for\_1 [null]--- } Node---{id=3,"+  
 "\n left\_for\_3 [null,"+  
 "\n \t\tright\_for\_3 [null]--- } ");  
  
 // Проверка совпадения ожидаемого и фактического вывода  
 Assert.*assertEquals*(expectedOutput.toString(), output.toString());  
}

**Удаление элемента**

Для тестирования удаления элемента так же создаем тестовые данные и далее с помощью Junit сравниваем, действительно ли узел и левый потомок совпадают с ожидаемым результатом.

@Test  
public void testDelete() {  
 // Создание экземпляра класса BalancedBinaryTree  
 BalancedBinaryTree tree = new BalancedBinaryTree();  
  
 // Создание тестовых данных  
 Product product1 = new Product(1, "Product 1", "Description 1", 10.50f, 50, Measure.*XS*);  
 Product product2 = new Product(2, "Product 2", "Description 2", 30.55f, 40, Measure.*S*);  
 Product product3 = new Product(3, "Product 3", "Description 3", 55.90f, 30, Measure.*XL*);  
  
 // Вставка узлов в дерево  
 tree.insert(product2);  
 tree.insert(product1);  
 tree.insert(product3);  
  
 // Удаление узла с ключом 2  
 tree.delete(2);  
  
 // Получение корневого узла дерева  
 BinaryTree root = tree.getRoot();  
  
 // Проверка корректности удаления узла  
 Assert.*assertEquals*(product3, root.getNode());  
 Assert.*assertEquals*(product1, root.getLeftChild().getNode());  
 //Assert.assertEquals(null, root.getRightChild().getNode());  
}

**Добавление элемента**

Для тестирования вставки элемента в бинарное дерева проверяем, совпадает ли ожидаемый элемент с фактическим расположением вставленных элементов.

@Test  
public void testInsert() {  
 // Создание экземпляра класса BalancedBinaryTree  
 BalancedBinaryTree tree = new BalancedBinaryTree();  
  
 // Создание тестовых данных  
 Product product1 = new Product(1, "Product 1", "Description 1", 10.50f, 50, Measure.*XS*);  
 Product product2 = new Product(2, "Product 2", "Description 2", 30.55f, 40, Measure.*S*);  
 Product product3 = new Product(3, "Product 3", "Description 3", 55.90f, 30, Measure.*XL*);  
  
 // Вставка узлов в дерево  
 tree.insert(product2);  
 tree.insert(product1);  
 tree.insert(product3);  
  
 // Получение корневого узла дерева  
 BinaryTree root = tree.getRoot();  
  
 // Проверка корректности вставки узлов  
 Assert.*assertEquals*(product2, root.getNode());  
 Assert.*assertEquals*(product1, root.getLeftChild().getNode());  
 Assert.*assertEquals*(product3, root.getRightChild().getNode());  
}

**Очередь (Queue)**

Тестирование очереди

@BeforeEach для каждого метода инстанциирует объект класса Очереди.

**Put**

Тестируем, действительно ли после вставки элементов размер очереди получается равным 5.

public void testPut() {  
 for (int i = 1; i <= 5; i++) {  
 queue.put(i);  
 }  
 *assertEquals*(5, queue.getSize());  
}

**Get**

Проверяем, что выбрасывает ошибку, если очередь пуста. А если не пуста, совпадает ли ожидаемый get() с фактическим.

public void testGet() {  
 *assertThrows*(NoSuchElementException.class, queue::get);  
  
 for (int i = 1; i <= 5; i++) {  
 queue.put(i);  
 }  
  
 for (int i = 1; i <= 5; i++) {  
 *assertEquals*(i, queue.get());  
 }  
  
 *assertThrows*(NoSuchElementException.class, queue::get);  
}

**IsEmpty**

Проверяет, что метод работает правильно - если очередь не пуста, то возвращается true, иначе false

public void testIsEmpty() {  
 *assertTrue*(queue.isEmpty());  
 queue.put(1);  
 *assertFalse*(queue.isEmpty());  
 queue.get();  
 *assertTrue*(queue.isEmpty());  
}

**Выводы**

В данной лабораторной работе было разобрано тестирование кода с помощью фреймворка Junit, которым были протестированы методы работы с динамическими структурами данных «Очередь» и «Бинарное дерево». Тесты запускаются их среды разработки или с помощью maven test.