Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Кубанский государственный технологический университет»

(ФГБОУ ВО «КубГТУ»)

Факультет информационных технологий и кибербезопасности

Кафедра информационных систем и программирования

**ОТЧЕТ**

по лабораторной работе №7.

Тема работы: «Модульное тестирование объектно-ориентированных программ».

Наименование дисциплины: «Тестирование и отладка программного обеспечения».

Подготовил:

Студент группы 23-КБ-ПР1

Юданов М. Д.

Проверил:

Волик А. Г.

Краснодар

**2025**

**Цель работы**: изучить подход к автоматизации процесса модульного тестирования объектно-ориентированных программ.

**2 Задание**

1) Создать класс (в соответствии с вариантом задания из п.5), реализующий работу с АСД (абстрактная структура данных).

2) Построить диаграмму состояний класса.

3) Составить тестовые требования к методам класса не основе полученной диаграммы состояний.

4) Определить наборы тестов на основе полученных тестовых требований.

5) Создать проект для автоматизированного модульного тестирования на основе тестовых наборов.

6) Запустить тестирование и проверить результаты работы.

7) Составить отчет о результатах проведенного тестирования.

**3 Вариант задания**

Создать класс (в соответствии с вариантом задания из п.5), реализующий работу с АСД (абстрактная структура данных).

**Вариант 14**

13) Дек, односвязный список

****

**4 Выполнение работы**

1. Создаю класс, реализующий работу с абстрактной структурой данных, соответствующей варианту задания. В моем случае абстрактная структура данных – очередь.

**Код класса:**

// Узел односвязного списка

class Node[T](var value: T, var next: Option[Node[T]] = None)

trait Dequeue[T] {

def pushFront(value: T): Unit

def pushBack(value: T): Unit

def popFront(): Option[T]

def popBack(): Option[T]

def getFront(): Option[T]

def getBack(): Option[T]

def isEmpty(): Boolean

def count(): Int

def clear(): Unit

}

// Реализация на односвязном списке

class SinglyLinkedListDequeue[T] extends Dequeue[T] {

private var head: Option[Node[T]] = None

private var tail: Option[Node[T]] = None

private var currentCount: Int = 0

override def pushFront(value: T): Unit = {

val newNode = new Node[T](value, head)

head = Some(newNode)

if (tail.isEmpty) {

tail = Some(newNode)

}

currentCount += 1

}

override def pushBack(value: T): Unit = {

val newNode = new Node[T](value, None)

tail match {

case Some(currentTail) => currentTail.next = Some(newNode)

case None => head = Some(newNode)

}

tail = Some(newNode)

currentCount += 1

}

override def popFront(): Option[T] = {

head match {

case Some(oldHead) =>

head = oldHead.next

if (head.isEmpty) tail = None

currentCount -= 1

Some(oldHead.value)

case None => None

}

}

override def popBack(): Option[T] = {

head match {

case None => None

case Some(single) if single.next.isEmpty =>

head = None

tail = None

currentCount -= 1

Some(single.value)

case Some(\_) =>

var currentNode = head.get

var previousNode: Option[Node[T]] = None

while (currentNode.next.nonEmpty) {

previousNode = Some(currentNode)

currentNode = currentNode.next.get

}

previousNode.foreach(\_.next = None)

tail = previousNode

currentCount -= 1

Some(currentNode.value)

}

}

override def getFront(): Option[T] = head.map(\_.value)

override def getBack(): Option[T] = tail.map(\_.value)

override def isEmpty(): Boolean = head.isEmpty

override def count(): Int = currentCount

override def clear(): Unit = {

head = None

tail = None

currentCount = 0

}

}

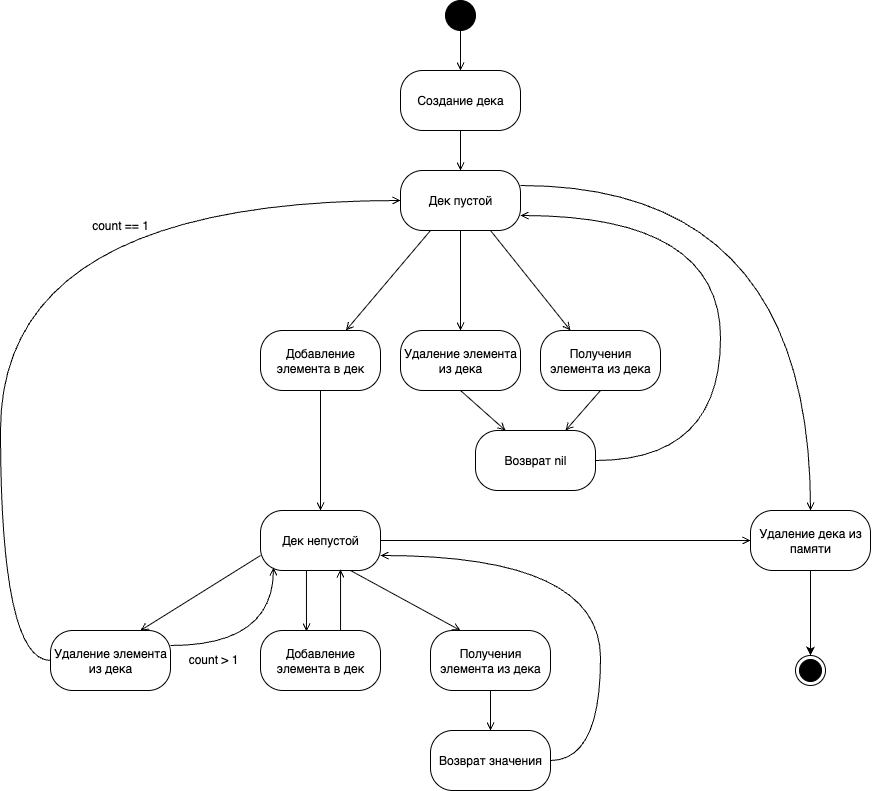
2. Составляю диаграмму состояний класса. 

Рисунок 1 – Диаграмма состояний класса

Тестовые требования к методам класса не основе полученной диаграммы состояний:

1. Конструктор: при инициализации дек пустой (isEmpty() == true, count() == 0, getFront() == nil, getBack() == nil).
2. pushFront: после pushFront(value) дек непустой, count увеличивается на 1, getFront() возвращает добавленное значение; при единственном элементе getBack() == getFront().
3. pushBack: после pushBack(value) дек непустой, count увеличивается на 1, getBack() возвращает добавленное значение; при единственном элементе getFront() == getBack().
4. Комбинация pushFront + pushBack: порядок фронта и бэка соответствует последовательности добавлений, count корректен.
5. popFront на непустом: возвращает значение с фронта, уменьшает count; при удалении последнего элемента дек становится пустым и getFront()/getBack() == nil.
6. popFront на пустом: возвращает nil, count остаётся 0, состояние остаётся пустым.
7. popBack на непустом: возвращает значение с бэка, уменьшает count; при удалении последнего элемента дек становится пустым и getFront()/getBack() == nil.
8. popBack на пустом: возвращает nil, count остаётся 0, состояние остаётся пустым.
9. Peek/инспекция: getFront() и getBack() возвращают корректные значения без изменения count и состояния (работают и для одиночного элемента).
10. Смешанные операции (pushFront/pushBack + попы с противоположных концов): структура корректно поддерживает порядок элементов и count (проверки popBack после pushFront и popFront после pushBack).
11. Множественные push/pop: последовательные pushFront/pushBack и popFront/popBack приводят к ожидаемому пустому состоянию и count == 0 при завершении.
12. Нагрузочный сценарий: при большом числе операций (массовые pushBack/pushFront и последующие popFront/popBack) count, getFront/getBack и порядок остаются корректными; clear() очищает дек (isEmpty() == true, count() == 0).

Напишем тесты. Проверим сценарии на основании вышеописанных требований.

**Код тестирований:**

import org.scalatest.funsuite.AnyFunSuite

import org.scalatest.funsuite.AnyFunSuite

import org.scalatest.matchers.should.Matchers

class IntSinglyLinkedListDequeueTests extends AnyFunSuite with Matchers {

private var intDequeue: SinglyLinkedListDequeue[Int] = \_

// Аналог setUp()

def beforeEach(): Unit = {

intDequeue = new SinglyLinkedListDequeue[Int]()

}

// Аналог tearDown()

def afterEach(): Unit = {

intDequeue = null

}

test("Дек должен быть пустым при инициализации") {

beforeEach()

intDequeue.isEmpty() shouldBe true

intDequeue.count() shouldBe 0

afterEach()

}

test("pushFront и getFront") {

beforeEach()

intDequeue.pushFront(10)

intDequeue.isEmpty() shouldBe false

intDequeue.count() shouldBe 1

intDequeue.getFront() shouldBe Some(10)

intDequeue.getBack() shouldBe Some(10)

afterEach()

}

test("pushBack и getBack") {

beforeEach()

intDequeue.pushBack(20)

intDequeue.isEmpty() shouldBe false

intDequeue.count() shouldBe 1

intDequeue.getBack() shouldBe Some(20)

intDequeue.getFront() shouldBe Some(20)

afterEach()

}

test("pushFront и pushBack комбинация") {

beforeEach()

intDequeue.pushFront(5)

intDequeue.pushBack(15)

intDequeue.getFront() shouldBe Some(5)

intDequeue.getBack() shouldBe Some(15)

intDequeue.count() shouldBe 2

afterEach()

}

test("popFront работает корректно") {

beforeEach()

intDequeue.pushBack(10)

intDequeue.pushBack(20)

intDequeue.popFront() shouldBe Some(10)

intDequeue.count() shouldBe 1

intDequeue.getFront() shouldBe Some(20)

intDequeue.getBack() shouldBe Some(20)

intDequeue.isEmpty() shouldBe false

intDequeue.popFront() shouldBe Some(20)

intDequeue.isEmpty() shouldBe true

intDequeue.count() shouldBe 0

intDequeue.getFront() shouldBe None

intDequeue.getBack() shouldBe None

intDequeue.popFront() shouldBe None

afterEach()

}

test("popBack работает корректно") {

beforeEach()

intDequeue.pushBack(30)

intDequeue.pushBack(40)

intDequeue.popBack() shouldBe Some(40)

intDequeue.count() shouldBe 1

intDequeue.getBack() shouldBe Some(30)

intDequeue.getFront() shouldBe Some(30)

intDequeue.popBack() shouldBe Some(30)

intDequeue.isEmpty() shouldBe true

intDequeue.count() shouldBe 0

intDequeue.getFront() shouldBe None

intDequeue.getBack() shouldBe None

intDequeue.popBack() shouldBe None

afterEach()

}

test("clear очищает дек") {

beforeEach()

intDequeue.pushBack(1)

intDequeue.pushFront(2)

intDequeue.pushBack(3)

intDequeue.clear()

intDequeue.isEmpty() shouldBe true

intDequeue.count() shouldBe 0

intDequeue.getFront() shouldBe None

intDequeue.getBack() shouldBe None

afterEach()

}

test("pushFront/popBack комбинации") {

beforeEach()

intDequeue.pushFront(1)

intDequeue.pushBack(2)

intDequeue.pushFront(0)

intDequeue.count() shouldBe 3

intDequeue.getFront() shouldBe Some(0)

intDequeue.getBack() shouldBe Some(2)

intDequeue.popBack() shouldBe Some(2)

intDequeue.popBack() shouldBe Some(1)

intDequeue.popBack() shouldBe Some(0)

intDequeue.isEmpty() shouldBe true

afterEach()

}

test("pushBack/popFront комбинации") {

beforeEach()

intDequeue.pushBack(1)

intDequeue.pushFront(2)

intDequeue.pushBack(3)

intDequeue.popFront() shouldBe Some(2)

intDequeue.popFront() shouldBe Some(1)

intDequeue.popFront() shouldBe Some(3)

intDequeue.isEmpty() shouldBe true

afterEach()

}

test("Много операций push/pop") {

beforeEach()

for (i <- 1 to 100) intDequeue.pushBack(i)

intDequeue.count() shouldBe 100

intDequeue.getFront() shouldBe Some(1)

intDequeue.getBack() shouldBe Some(100)

for (i <- 1 to 50) intDequeue.popFront() shouldBe Some(i)

intDequeue.count() shouldBe 50

intDequeue.getFront() shouldBe Some(51)

intDequeue.getBack() shouldBe Some(100)

for (i <- 101 to 150) intDequeue.pushFront(i)

intDequeue.count() shouldBe 100

intDequeue.getFront() shouldBe Some(150)

intDequeue.getBack() shouldBe Some(100)

intDequeue.clear()

intDequeue.isEmpty() shouldBe true

afterEach()

}

}

// ===================== String-тесты =====================

class StringSinglyLinkedListDequeueTests extends AnyFunSuite with Matchers {

private var stringDequeue: SinglyLinkedListDequeue[String] = \_

def beforeEach(): Unit = {

stringDequeue = new SinglyLinkedListDequeue[String]()

}

def afterEach(): Unit = {

stringDequeue = null

}

test("Инициализация пустого дека строк") {

beforeEach()

stringDequeue.isEmpty() shouldBe true

stringDequeue.count() shouldBe 0

afterEach()

}

test("pushFront и getFront для строк") {

beforeEach()

stringDequeue.pushFront("hello")

stringDequeue.isEmpty() shouldBe false

stringDequeue.count() shouldBe 1

stringDequeue.getFront() shouldBe Some("hello")

stringDequeue.getBack() shouldBe Some("hello")

afterEach()

}

test("pushBack и getBack для строк") {

beforeEach()

stringDequeue.pushBack("world")

stringDequeue.getBack() shouldBe Some("world")

stringDequeue.getFront() shouldBe Some("world")

afterEach()

}

test("pushFront и pushBack строковые комбинации") {

beforeEach()

stringDequeue.pushFront("a")

stringDequeue.pushBack("b")

stringDequeue.getFront() shouldBe Some("a")

stringDequeue.getBack() shouldBe Some("b")

stringDequeue.count() shouldBe 2

afterEach()

}

test("popFront для строк") {

beforeEach()

stringDequeue.pushBack("first")

stringDequeue.pushBack("second")

stringDequeue.popFront() shouldBe Some("first")

stringDequeue.popFront() shouldBe Some("second")

stringDequeue.isEmpty() shouldBe true

stringDequeue.popFront() shouldBe None

afterEach()

}

test("popBack для строк") {

beforeEach()

stringDequeue.pushBack("third")

stringDequeue.pushBack("fourth")

stringDequeue.popBack() shouldBe Some("fourth")

stringDequeue.popBack() shouldBe Some("third")

stringDequeue.isEmpty() shouldBe true

stringDequeue.popBack() shouldBe None

afterEach()

}

test("clear очищает строковый дек") {

beforeEach()

stringDequeue.pushBack("x")

stringDequeue.pushFront("y")

stringDequeue.pushBack("z")

stringDequeue.clear()

stringDequeue.isEmpty() shouldBe true

stringDequeue.count() shouldBe 0

stringDequeue.getFront() shouldBe None

stringDequeue.getBack() shouldBe None

afterEach()

}

test("pushFront/popBack строковые комбинации") {

beforeEach()

stringDequeue.pushFront("one")

stringDequeue.pushBack("two")

stringDequeue.pushFront("zero")

stringDequeue.getFront() shouldBe Some("zero")

stringDequeue.getBack() shouldBe Some("two")

stringDequeue.popBack() shouldBe Some("two")

stringDequeue.getBack() shouldBe Some("one")

stringDequeue.getFront() shouldBe Some("zero")

afterEach()

}

test("Большое количество операций со строками") {

beforeEach()

for (i <- 1 to 50) stringDequeue.pushBack(s"Item $i")

stringDequeue.count() shouldBe 50

stringDequeue.getFront() shouldBe Some("Item 1")

stringDequeue.getBack() shouldBe Some("Item 50")

for (i <- 1 to 25) stringDequeue.popFront() shouldBe Some(s"Item $i")

stringDequeue.count() shouldBe 25

stringDequeue.getFront() shouldBe Some("Item 26")

for (i <- 51 to 75) stringDequeue.pushFront(s"New Item $i")

stringDequeue.count() shouldBe 50

stringDequeue.getFront() shouldBe Some("New Item 75")

stringDequeue.getBack() shouldBe Some("Item 50")

stringDequeue.clear()

stringDequeue.isEmpty() shouldBe true

afterEach()

}

}

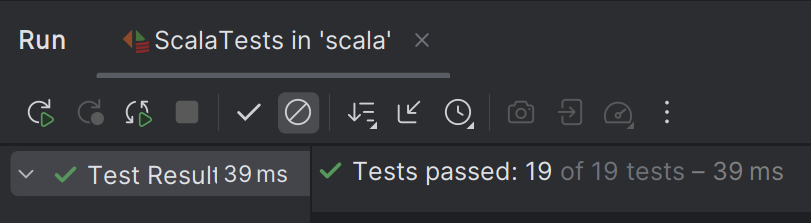


Рисунок 2 – Результаты тестирований.

5. Делаю вывод касательно проделанной работы.

**Вывод:** в ходе выполнения лабораторной работы я изучил подход к автоматизации процесса модульного тестирования объектно-ориентированных программ.