Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

Современные технологии программирования

Лабораторная работа №8

Разработка и модульное тестирование класса Параметризованный абстрактный тип данных «Процессор»

Выполнил:

студент гр.ИП-213 Дмитриев Антон Александрович
ФИО студента

«__» _____ 2025 г.

Проверил:

Преподаватель

ФИО преподавателя

«__» _____ 2025 г. Оценка_____

Новосибирск 2025 г.

1. **Задание**

1. В соответствии с приведенной ниже спецификацией реализовать параметризованный абстрактный тип данных «Процессор», используя шаблон классов C++.

2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования.

3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

2. Исходные тексты программ.

Class.py:

```
from enum import Enum
from typing import TypeVar, Generic

T = TypeVar('T')

class TOprtn(Enum):
    NONE = 0
    ADD = 1
    SUB = 2
    MUL = 3
    DVD = 4

class TFunc(Enum):
    REV = 1
    SQR = 2

class TProc(Generic[T]):
    def __init__(self, default_value: T):
        self._lop_res = default_value
        self._rop = default_value
        self._operation = TOprtn.NONE

    def reset_processor(self) -> None:
        """Сброс процессора в начальное состояние"""
        self._lop_res = self._get_default_value()
        self._rop = self._get_default_value()
        self._operation = TOprtn.NONE

    def reset_operation(self) -> None:
        """Сброс текущей операции"""
        self._operation = TOprtn.NONE

    def run_operation(self) -> None:
        """Выполнение текущей операции"""
        if self._operation == TOprtn.NONE:
            return

        op_map = {
            TOprtn.ADD: lambda x, y: x + y,
            TOprtn.SUB: lambda x, y: x - y,
            TOprtn.MUL: lambda x, y: x * y,
            TOprtn.DVD: lambda x, y: x / y,
```

```
    }
    self._lop_res = op_map[self._operation](self._lop_res, self._rop)

def run_function(self, func: TFunc) -> None:
    """Выполнение функции над правым операндом"""
    func_map = {
        TFunc.REV: lambda x: 1 / x,
        TFunc.SQR: lambda x: x * x,
    }
    self._rop = func_map[func](self._rop)

@property
def left_operand(self) -> T:
    return self._lop_res

@left_operand.setter
def left_operand(self, value: T) -> None:
    self._lop_res = value

@property
def right_operand(self) -> T:
    return self._rop

@right_operand.setter
def right_operand(self, value: T) -> None:
    self._rop = value

@property
def operation(self) -> TOprtn:
    return self._operation

@operation.setter
def operation(self, value: TOprtn) -> None:
    self._operation = value

def _get_default_value(self) -> T:
    return type(self._lop_res)()
```

Tests.py:

```
import pytest
from lab8 import TProc, TOprtn, TFunc


class TestTProcInt:
    """Тесты для TProc с целыми числами"""

    @pytest.fixture
    def proc(self):
        return TProc(0)

    def test_initial_state(self, proc):
        """Тест начального состояния процессора"""
        assert proc.left_operand == 0
        assert proc.right_operand == 0
        assert proc.operation == TOprtn.NONE

    def test_basic_operations(self, proc):
        """Тест базовых арифметических операций"""
        test_cases = [
            # (left, right, operation, expected_result)
            (5, 3, TOprtn.ADD, 8),
            (10, 4, TOprtn.SUB, 6),
            (7, 6, TOprtn.MUL, 42),
            (15, 3, TOprtn.DVD, 5),
            (8, 2, TOprtn.ADD, 10),
            (20, 5, TOprtn.SUB, 15),
        ]

        for left, right, operation, expected in test_cases:
```

```
        proc.left_operand = left
        proc.right_operand = right
        proc.operation = operation
        proc.run_operation()
        assert proc.left_operand == expected

def test_functions(self, proc):
    """Тест математических функций"""
    # Тест квадрата
    proc.right_operand = 5
    proc.run_function(TFunc.SQR)
    assert proc.right_operand == 25

    # Тест обратного значения
    proc.right_operand = 4
    proc.run_function(TFunc.REV)
    assert proc.right_operand == 0.25  # 1/4

def test_operation_clear(self, proc):
    """Тест сброса операции"""
    proc.operation = TOprtn.ADD
    proc.reset_operation()
    assert proc.operation == TOprtn.NONE

def test_processor_reset(self, proc):
    """Тест полного сброса процессора"""
    proc.left_operand = 100
    proc.right_operand = 50
    proc.operation = TOprtn.MUL
    proc.reset_processor()
```

```
        assert proc.left_operand == 0
        assert proc.right_operand == 0
        assert proc.operation == TOprtn.NONE

    def test_no_operation_behavior(self, proc):
        """Тест поведения при отсутствии операции"""
        initial_left = proc.left_operand = 10
        proc.right_operand = 5
        # Operation остается NONE
        proc.run_operation()
        assert proc.left_operand == initial_left  # Не должно измениться

    def test_operation_set_get(self, proc):
        """Тест установки и получения операции"""
        operations = [TOprtn.ADD, TOprtn.SUB, TOprtn.MUL, TOprtn.DVD, TOprtn.NONE]

        for op in operations:
            proc.operation = op
            assert proc.operation == op

    def test_operand_set_get(self, proc):
        """Тест установки и получения операндов"""
        proc.left_operand = 123
        proc.right_operand = 456

        assert proc.left_operand == 123
        assert proc.right_operand == 456

class Fraction:
```

```
"""Простой класс дроби для тестирования"""
def __init__(self, numerator=0, denominator=1):
    self.numerator = numerator
    self.denominator = denominator
    self._simplify()

def _simplify(self):
    """Упрощение дроби"""
    import math
    if self.denominator < 0:
        self.numerator = -self.numerator
        self.denominator = -self.denominator

    gcd = math.gcd(abs(self.numerator), self.denominator)
    if gcd > 1:
        self.numerator //= gcd
        self.denominator //= gcd

def __add__(self, other):
    return Fraction(
        self.numerator * other.denominator + other.numerator * self.denominator,
        self.denominator * other.denominator
    )

def __sub__(self, other):
    return Fraction(
        self.numerator * other.denominator - other.numerator * self.denominator,
        self.denominator * other.denominator
    )
```

```
    def __mul__(self, other):
        return Fraction(
            self.numerator * other.numerator,
            self.denominator * other.denominator
        )

    def __truediv__(self, other):
        return Fraction(
            self.numerator * other.denominator,
            self.denominator * other.numerator
        )

    def __eq__(self, other):
        return (self.numerator == other.numerator and
                self.denominator == other.denominator)

    def __str__(self):
        return f"{self.numerator}/{self.denominator}"

    def rev(self):
        """Обратная дробь"""
        return Fraction(self.denominator, self.numerator)

    def sqr(self):
        """Квадрат дроби"""
        return Fraction(self.numerator ** 2, self.denominator ** 2)

class TestTProcFraction:
    """Тесты для TProc с дробями"""
```

```
@pytest.fixture
def proc(self):
    return TProc(Fraction(0, 1))


def test_initial_state_fraction(self, proc):
    """Тест начального состояния с дробями"""
    assert proc.left_operand == Fraction(0, 1)
    assert proc.right_operand == Fraction(0, 1)
    assert proc.operation == TOprtn.NONE


def test_fraction_operations(self, proc):
    """Тест операций с дробями"""
    test_cases = [
        # (left, right, operation, expected_result)
        (Fraction(1, 2), Fraction(1, 3), TOprtn.ADD, Fraction(5, 6)),
        (Fraction(3, 4), Fraction(1, 4), TOprtn.SUB, Fraction(1, 2)),
        (Fraction(2, 3), Fraction(3, 4), TOprtn.MUL, Fraction(1, 2)),
        (Fraction(1, 2), Fraction(2, 3), TOprtn.DVD, Fraction(3, 4)),
    ]

    for left, right, operation, expected in test_cases:
        proc.left_operand = left
        proc.right_operand = right
        proc.operation = operation
        proc.run_operation()
        assert proc.left_operand == expected


def test_fraction_functions(self, proc):
    """Тест функций с дробями"""
    # Тест квадрата дроби
```

```
    proc.right_operand = Fraction(2, 3)
    proc.run_function(TFunc.SQR)
    assert proc.right_operand == Fraction(4, 9)

    # Тест обратной дроби
    proc.right_operand = Fraction(3, 4)
    proc.run_function(TFunc.REV)
    assert proc.right_operand == Fraction(4, 3)

def test_complex_expression(self, proc):
    """Тест сложного выражения (как в примере из задания)"""
    # Эмуляция выражения: 2/1 + 3/1 * (4/1)^2

    # Шаг 1: Установка левого операнда 2/1
    proc.left_operand = Fraction(2, 1)

    # Шаг 2: Установка операции сложения
    proc.operation = TOprtn.ADD

    # Шаг 3: Установка правого операнда 3/1
    proc.right_operand = Fraction(3, 1)

    # Шаг 4: Изменение операции на умножение
    proc.operation = TOprtn.MUL

    # Шаг 5: Установка нового правого операнда 4/1
    proc.right_operand = Fraction(4, 1)

    # Шаг 6: Вычисление квадрата
    proc.run_function(TFunc.SQR)
```

```
        # Шаг 7: Выполнение операции умножения
        proc.run_operation()

        # Ожидаемый результат: 2/1 + (3/1 * (4/1)^2) = 2/1 + (3/1 * 16/1) = 2/1 + 48/1 = 50/1
        expected = Fraction(50, 1)
        assert proc.left_operand == expected

class TestTProcFloat:
    """Тесты для TProc с числами с плавающей точкой"""

    @pytest.fixture
    def proc(self):
        return TProc(0.0)

    def test_float_operations(self, proc):
        """Тест операций с числами с плавающей точкой"""
        test_cases = [
            # (left, right, operation, expected_result)
            (2.5, 3.5, TOprtn.ADD, 6.0),
            (10.5, 4.2, TOprtn.SUB, 6.3),
            (2.5, 4.0, TOprtn.MUL, 10.0),
            (15.0, 4.0, TOprtn.DVD, 3.75),
        ]

        for left, right, operation, expected in test_cases:
            proc.left_operand = left
            proc.right_operand = right
            proc.operation = operation
            proc.run_operation()
```

```
            assert proc.left_operand == pytest.approx(expected)

    def test_float_functions(self, proc):
        """Тест функций с числами с плавающей точкой"""
        proc.right_operand = 4.0
        proc.run_function(TFunc.SQR)
        assert proc.right_operand == 16.0

        proc.right_operand = 0.25
        proc.run_function(TFunc.REV)
        assert proc.right_operand == 4.0

class TestTProcExceptions:
    """Тесты исключительных ситуаций"""

    @pytest.fixture
    def proc(self):
        return TProc(0)

    def test_division_by_zero_int(self, proc):
        """Тест деления на ноль для целых чисел"""
        proc.left_operand = 10
        proc.right_operand = 0
        proc.operation = TOprtn.DVD

        with pytest.raises(ZeroDivisionError):
            proc.run_operation()

    def test_division_by_zero_fraction(self, proc_fraction):
        """Тест деления на ноль для дробей"""
```

```
        proc = TProc(Fraction(0, 1))
        proc.left_operand = Fraction(1, 2)
        proc.right_operand = Fraction(0, 1)
        proc.operation = TOprtn.DVD

        with pytest.raises(ZeroDivisionError):
            proc.run_operation()

    def test_reverse_zero(self, proc):
        """Тест обратного значения для нуля"""
        proc.right_operand = 0
        with pytest.raises(ZeroDivisionError):
            proc.run_function(TFunc.REV)

class TestTProcParametrized:
    """Параметризованные тесты для различных типов данных"""

    @pytest.mark.parametrize("left,right,operation,expected", [
        (5, 3, TOprtn.ADD, 8),
        (10, 4, TOprtn.SUB, 6),
        (7, 6, TOprtn.MUL, 42),
        (15, 3, TOprtn.DVD, 5),
    ])
    def test_parametrized_operations_int(self, left, right, operation, expected):
        """Параметризованные тесты операций для целых чисел"""
        proc = TProc(0)
        proc.left_operand = left
        proc.right_operand = right
        proc.operation = operation
        proc.run_operation()
```

```
    assert proc.left_operand == expected


@pytest.mark.parametrize("value,function,expected", [
    (5, TFunc.SQR, 25),
    (4, TFunc.SQR, 16),
    (2, TFunc.REV, 0.5),
    (10, TFunc.REV, 0.1),
])
def test_parametrized_functions_int(self, value, function, expected):
    """Параметризованные тесты функций для целых чисел"""
    proc = TProc(0)
    proc.right_operand = value
    proc.run_function(function)
    assert proc.right_operand == expected
```

3. Тестовые наборы данных для тестирования класса

TestTProcInt - Тесты для целых чисел

1. test_initial_state

Поле	Ожидаемое значение
left_operand	0
right_operand	0
operation	TOprtn.NONE

2. test_basic_operations

Левый операнд	Правый операнд	Операция	Ожидаемый результат
5	3	TOprtn.ADD	8
10	4	TOprtn.SUB	6
7	6	TOprtn.MUL	42
15	3	TOprtn.DVD	5
8	2	TOprtn.ADD	10
20	5	TOprtn.SUB	15

3. test_functions

Функция	Входное значение	Ожидаемый результат
TFunc.SQR	5	25
TFunc.REV	4	0.25

4. **test_operation_clear**

Действие	Операция до	Операция после
Установка ADD → Сброс	TOprtn.ADD	TOprtn.NONE

5. **test_processor_reset**

Поле	Значение до сброса	Значение после сброса
left_operand	100	0
right_operand	50	0
operation	TOprtn.MUL	TOprtn.NONE

6. **test_no_operation_behavior**

Параметр	Значение
left_operand до	10
right_operand	5
operation	TOprtn.NONE
left_operand после	10

7. **test_operation_set_get**

Устанавливаемая операция	Получаемая операция
TOprtn.ADD	TOprtn.ADD
TOprtn.SUB	TOprtn.SUB
TOprtn.MUL	TOprtn.MUL

Устанавливаемая операция	Получаемая операция
TOprtn.DVD	TOprtn.DVD
TOprtn.NONE	TOprtn.NONE

8. **test_operand_set_get**

Поле	Устанавливаемое значение	Получаемое значение
left_operand	123	123
right_operand	456	456

TestTProcFraction - Тесты для дробей

1. **test_initial_state_fraction**

Поле	Ожидаемое значение
left_operand	Fraction(0, 1)
right_operand	Fraction(0, 1)
operation	TOprtn.NONE

2. **test_fraction_operations**

Левый операнд	Правый операнд	Операция	Ожидаемый результат
Fraction(1, 2)	Fraction(1, 3)	TOprtn.ADD	Fraction(5, 6)
Fraction(3, 4)	Fraction(1, 4)	TOprtn.SUB	Fraction(1, 2)
Fraction(2, 3)	Fraction(3, 4)	TOprtn.MUL	Fraction(1, 2)
Fraction(1, 2)	Fraction(2, 3)	TOprtn.DVD	Fraction(3, 4)

3. **test_fraction_functions**

Функция	Входное значение	Ожидаемый результат
TFunc.SQR	Fraction(2, 3)	Fraction(4, 9)
TFunc.REV	Fraction(3, 4)	Fraction(4, 3)

4. **test_complex_expression**

Шаг	Действие	left_operand	right_operand	operation
1	Установка левого операнда	Fraction(2, 1)	Fraction(0, 1)	TOprtn.NONE
2	Установка операции	Fraction(2, 1)	Fraction(0, 1)	TOprtn.ADD
3	Установка правого операнда	Fraction(2, 1)	Fraction(3, 1)	TOprtn.ADD
4	Изменение операции	Fraction(2, 1)	Fraction(3, 1)	TOprtn.MUL
5	Установка нового правого операнда	Fraction(2, 1)	Fraction(4, 1)	TOprtn.MUL
6	Вычисление квадрата	Fraction(2, 1)	Fraction(16, 1)	TOprtn.MUL
7	Выполнение операции	Fraction(50, 1)	Fraction(16, 1)	TOprtn.MUL

Ожидаемый конечный результат: Fraction(50, 1)

TestTProcFloat - Тесты для чисел с плавающей точкой

1. **test_float_operations**

Левый операнд	Правый операнд	Операция	Ожидаемый результат
2.5	3.5	TOprtn.ADD	6.0
10.5	4.2	TOprtn.SUB	6.3
2.5	4.0	TOprtn.MUL	10.0
15.0	4.0	TOprtn.DVD	3.75

2. test_float_functions

Функция	Входное значение	Ожидаемый результат
TFunc.SQR	4.0	16.0
TFunc.REV	0.25	4.0

TestTProcExceptions - Тесты исключительных ситуаций

1. test_division_by_zero_int

Параметр	Значение	Ожидаемое исключение
left_operand	10	ZeroDivisionError
right_operand	0	
operation	TOprtn.DVD	

2. test_division_by_zero_fraction

Параметр	Значение	Ожидаемое исключение
left_operand	Fraction(1, 2)	ZeroDivisionError
right_operand	Fraction(0, 1)	

Параметр	Значение	Ожидаемое исключение
operation	TOprtn.DVD	

3. test_reverse_zero

Параметр	Значение	Ожидаемое исключение
right_operand	0	ZeroDivisionError
function	TFunc.REV	

TestTProcParametrized - Параметризованные тесты

1. test_parametrized_operations_int

Левый операнд	Правый операнд	Операция	Ожидаемый результат
5	3	TOprtn.ADD	8
10	4	TOprtn.SUB	6
7	6	TOprtn.MUL	42
15	3	TOprtn.DVD	5

2. test_parametrized_functions_int

Входное значение	Функция	Ожидаемый результат
5	TFunc.SQR	25
4	TFunc.SQR	16
2	TFunc.REV	0.5
10	TFunc.REV	0.1