

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 №6

Разработка и модульное тестирование класса

Редактор комплексных чисел

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. Разработать и реализовать класс «Ввод и редактирование комплексных чисел» (TEditor), используя класс C++.
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования Visual Studio по критерию C2.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

2. Исходные тексты программ.

Class.py:

class TEditor:

"""Класс для ввода и редактирования комплексных чисел"""

Константы

DECIMAL_SEPARATOR = "," # разделитель целой и дробной частей

IMAGINARY_SEPARATOR = "i*" # разделитель действительной и мнимой частей

ZERO REPRESENTATION = "0, i* 0," # строковое представление нуля

def __init__(self):

"""Конструктор - инициализирует строку нулевым значением"""

self._str = self.ZERO REPRESENTATION

@property

def _Str_(self):

"""Чтение строки в формате строки (метод свойства)"""

return self._str

@_Str_.setter

def _Str_(self, value):

"""Запись строки в формате строки (метод свойства)"""

if not isinstance(value, str):

raise ValueError("_Str_ должна быть строкового типа")

self._str = value

def complexNumberIsZero(self):

"""Проверяет, равно ли комплексное число нулю"""

try:

Более точная проверка - парсим и проверяем числовые значения

parts = self._parse_complex_string()

Проверяем, что обе части равны нулю

real_zero = (parts["real_integer"] == "0" and parts["real_fraction"] == "")

imag_zero = (parts["imaginary_integer"] == "0" and parts["imaginary_fraction"] == "")

```

        return real_zero and imag_zero
    except:
        return False

def _normalize_string(self, s):
    """Нормализует строку для сравнения"""
    return s.replace(" ", "").lower()

def addSign(self, part="real"):
    """Добавляет или удаляет знак '-' у указанной части"""
    parts = self._parse_complex_string()

    if part == "real":
        if parts["real_sign"] == "-":
            parts["real_sign"] = ""
        else:
            parts["real_sign"] = "-"
    elif part == "imaginary":
        if parts["imaginary_sign"] == "-":
            parts["imaginary_sign"] = ""
        else:
            parts["imaginary_sign"] = "-"
    else:
        raise ValueError("Часть должна быть 'real' или 'imaginary'")

    self._str = self._build_complex_string(parts)
    return self._str

def addValue(self, digit):
    """Добавляет цифру к строке"""
    if not isinstance(digit, int) or digit < 0 or digit > 9:
        raise ValueError("Цифра должна быть целым числом от 0 до 9)")

    char_digit = str(digit)
    parts = self._parse_complex_string()

    # Определяем, к какой части добавлять цифру
    if self._can_add_to_real(parts):

```

```

        if parts["real_integer"] == "0" and parts["real_fraction"] == "":
            parts["real_integer"] = char_digit
        elif parts["real_fraction"] == "":
            parts["real_integer"] += char_digit
        else:
            parts["real_fraction"] += char_digit
    elif self._can_add_to_imaginary(parts):
        if parts["imaginary_integer"] == "0" and parts["imaginary_fraction"] == "":
            parts["imaginary_integer"] = char_digit
        elif parts["imaginary_fraction"] == "":
            parts["imaginary_integer"] += char_digit
        else:
            parts["imaginary_fraction"] += char_digit

    self._str = self._build_complex_string(parts)
    return self._str

```

```

def addZero(self):
    """Добавляет ноль к строке"""
    return self.addValue(0)

def rmVal(self):
    """Удаляет крайний правый символ"""
    if len(self._str) <= len(self.ZERO REPRESENTATION):
        # Если _Str_ уже минимальной длины, сбрасываем к нулю
        self.clear()
    else:
        parts = self._parse_complex_string()

        # Удаляем символ из соответствующей части
        if parts["imaginary_fraction"] != "":
            parts["imaginary_fraction"] = parts["imaginary_fraction"][:-1]
        elif parts["imaginary_integer"] != "0":
            if len(parts["imaginary_integer"]) > 1:
                parts["imaginary_integer"] = parts["imaginary_integer"][:-1]
            else:
                parts["imaginary_integer"] = "0"
        elif parts["real_fraction"] != "":

```

```

        parts["real_fraction"] = parts["real_fraction"][:-1]
    elif parts["real_integer"] != "0":
        if len(parts["real_integer"]) > 1:
            parts["real_integer"] = parts["real_integer"][:-1]
        else:
            parts["real_integer"] = "0"
    else:
        # Если больше ничего удалять, сбрасываем к нулю
        self.clear()
        return self._str

    self._str = self._build_complex_string(parts)

    return self._str

def clear(self):
    """Устанавливает нулевое значение комплексного числа"""
    self._str = self.ZERO REPRESENTATION
    return self._str

def redact(self, command):
    """Выполняет команду редактирования"""
    commands = {
        10: self.clear,
        11: lambda: self.addSign("real"),
        12: self.addZero,
        13: self.rmVal
    }

    if command in commands:
        return commands[command]()
    elif command < 10:
        digit = command
        return self.addValue(digit)
    else:
        raise ValueError(f"Неизвестная команда: {command}")

def _parse_complex_string(self):
    """Парсит строку комплексного числа на составляющие части"""

```

```

# Базовая структура для нулевого значения
parts = {
    "real_sign": "", 
    "real_integer": "0", 
    "real_fraction": "", 
    "imaginary_sign": "", 
    "imaginary_integer": "0", 
    "imaginary_fraction": ""}

try:
    # Нормализуем строку
    normalized = self._str.replace(" ", "")

    # Разделяем действительную и мнимую части
    if self.IMAGINARY_SEPARATOR in normalized:
        real_part, imag_part =
            normalized.split(self.IMAGINARY_SEPARATOR, 1)
    else:
        real_part = normalized
        imag_part = "0,"

    # Парсим действительную часть
    if real_part.startswith("-"):
        parts["real_sign"] = "-"
        real_part = real_part[1:]
    elif real_part.startswith("+"):
        parts["real_sign"] = ""
        real_part = real_part[1:]
    else:
        parts["real_sign"] = ""

    if self.DECIMAL_SEPARATOR in real_part:
        real_int, real_frac = real_part.split(self.DECIMAL_SEPARATOR, 1)
        parts["real_integer"] = real_int if real_int else "0"
        parts["real_fraction"] = real_frac
    else:
        # Если нет разделителя, вся часть - целая
        parts["real_integer"] = real_part if real_part else "0"

```

```

parts["real_fraction"] = ""

# Парсим мнимую часть
if imag_part.startswith("-"):
    parts["imaginary_sign"] = "-"
    imag_part = imag_part[1:]
elif imag_part.startswith("+"):
    parts["imaginary_sign"] = ""
    imag_part = imag_part[1:]
else:
    parts["imaginary_sign"] = ""

if self.DECIMAL_SEPARATOR in imag_part:
    imag_int, imag_frac =
        imag_part.split(self.DECIMAL_SEPARATOR, 1)
    parts["imaginary_integer"] = imag_int if imag_int else "0"
    parts["imaginary_fraction"] = imag_frac
else:
    # Если нет разделителя, вся часть - целая
    parts["imaginary_integer"] = imag_part if imag_part else "0"
    parts["imaginary_fraction"] = ""

except Exception as e:
    # В случае ошибки парсинга возвращаем нулевое значение
    print(f"Ошибка парсинга: {e}")
    pass

return parts

def _build_complex_string(self, parts):
    """Собирает строку комплексного числа из составляющих частей"""
    # Собираем действительную часть
    real_sign = parts['real_sign']
    real_int = parts['real_integer']
    real_frac = parts['real_fraction']

    # Собираем мнимую часть
    imag_sign = parts['imaginary_sign']
    imag_int = parts['imaginary_integer']

```

```

imag_frac = parts['imaginary_fraction']

# Форматируем строку (без лишних плюсов)
real_part =
f'{real_sign}{real_int}{self.DECIMAL_SEPARATOR}{real_frac}'
    imag_part =
f'{imag_sign}{imag_int}{self.DECIMAL_SEPARATOR}{imag_frac}'

# Убираем лишние запятые если дробной части нет
if not real_frac:
    real_part = real_part.rstrip(self.DECIMAL_SEPARATOR)
if not imag_frac:
    imag_part = imag_part.rstrip(self.DECIMAL_SEPARATOR)

return f'{real_part} {self.IMAGINARY_SEPARATOR} {imag_part}'


def _can_add_to_real(self, parts):
    """Проверяет, можно ли добавить символ к действительной части"""
    return parts["imaginary_integer"] == "0" and parts["imaginary_fraction"]
== ""

def _can_add_to_imaginary(self, parts):
    """Проверяет, можно ли добавить символ к мнимой части"""
    # Можно добавлять к мнимой части, если действительная уже
    заполнена
    return parts["real_integer"] != "0" or parts["real_fraction"] != ""

```

Tests.py:

```

import pytest

from lab6 import TEditor

```

```

class TestTEditorC2:

```

```
def test_complex_number_is_zero(self):
    """Тест проверки на нулевое значение"""
    editor = TEditor()

    # Тест 1: Начальное значение - ноль
    assert editor.complexNumberIsZero() == True

    # Тест 2: После добавления цифры - не ноль
    editor.addValue(1)
    assert editor.complexNumberIsZero() == False

    # Тест 3: После очистки - снова ноль
    editor.clear()
    assert editor.complexNumberIsZero() == True

    # Тест 4: Различные нулевые представления
    zero_representations = ["0, i* 0,", "0 i* 0", "0 i* 0,"]
    for zero_repr in zero_representations:
        editor._Str_ = zero_repr
        assert editor.complexNumberIsZero() == True

def test_add_sign_real_part(self):
    """Тест изменения знака действительной части"""
    editor = TEditor()

    # Тест 1: Смена знака с "+" на "-"
    editor.addValue(1)
    editor.addValue(-1)
    assert editor.complexNumberIsZero() == True
```

```
editor._Str_ = "5, i* 3,"  
result = editor.addSign("real")  
assert result == "-5 i* 3"  
  
# Тест 2: Смена знака с '-' на ''  
result = editor.addSign("real")  
assert result == "5 i* 3"  
  
def test_add_sign_imaginary_part(self):  
    """Тест изменения знака мнимой части""""  
    editor = TEditor()  
  
    # Тест 1: Смена знака с '' на '-'  
    editor._Str_ = "5, i* 3,"  
    result = editor.addSign("imaginary")  
    assert result == "5 i* -3"  
  
    # Тест 2: Смена знака с '-' на ''  
    result = editor.addSign("imaginary")  
    assert result == "5 i* 3"  
  
    # Тест 3: Неверный параметр  
    with pytest.raises(ValueError):  
        editor.addSign("invalid_part")  
  
def test_property_setter_validation(self):  
    """Тест валидации в сеттере свойства"""
```

```
editor = TEditor()

# Тест 1: Корректные значения
valid_values = ["1, i* 2,", "3, i* 4,", "0, i* 0,"]
for value in valid_values:
    editor._Str_ = value
    assert editor._Str_ == value

# Тест 2: Некорректные типы данных
invalid_values = [123, 45.67, None, [], {}]
for value in invalid_values:
    with pytest.raises(ValueError):
        editor._Str_ = value

def test_add_value_validation(self):
    """Тест валидации входных данных addValue"""
    editor = TEditor()

    # Тест 1: Корректные цифры
    for digit in range(10):
        result = editor.addValue(digit)
        assert str(digit) in result

    # Тест 2: Некорректные цифры
    invalid_digits = [-1, 10, 15, -5, 100]
    for digit in invalid_digits:
        with pytest.raises(ValueError):
            editor.addValue(digit)
```

```
    editor.addValue(digit)

def test_clear_operation(self):
    """Тест операции очистки"""
    editor = TEditor()

    # Тест 1: Очистка ненулевого значения
    editor.addValue(5)
    editor.addValue(3)
    assert editor.complexNumberIsZero() == False

    result = editor.clear()
    assert result == "0, i* 0,"
    assert editor.complexNumberIsZero() == True

def test_redact_commands(self):
    """Тест команд редактирования"""
    editor = TEditor()

    # Тест 1: Команды цифр (0-9)
    for digit in range(10):
        result = editor.redact(digit)
        assert result is not None

    # Тест 2: Специальные команды
    commands = [10, 11, 12, 13]
```

```
for cmd in commands:  
    result = editor.redact(cmd)  
    assert result is not None  
  
# Тест 3: Неверные команды  
invalid_commands = [-1, 14, 15, 100, -5]  
for cmd in invalid_commands:  
    with pytest.raises(ValueError):  
        editor.redact(cmd)
```

3. Тестовые наборы данных для тестирования класса

Категория тестирования	Метод	Входные данные	Ожидаемый результат	Описание теста
Свойства строки	<code>_Str_ (setter)</code>	"1, i* 2,"	"1, i* 2,"	Установка корректного строкового значения
Свойства строки	<code>_Str_ (setter)</code>	123	ValueError	Исключение при неверном типе данных
Проверка нуля	<code>complexNumberI sZero</code>	"0, i* 0,"	True	Нулевое значение определяется корректно
Проверка нуля	<code>complexNumberI sZero</code>	"1, i* 0,"	False	Ненулевое значение определяется корректно
Проверка нуля	<code>complexNumberI sZero</code>	"0, i* 1,"	False	Ненулевая мнимая часть
Добавление цифр	<code>addValue</code>	5	Содержит "5"	Добавление корректной цифры
Добавление цифр	<code>addValue</code>	15	ValueError	Исключение при цифре > 9
Добавление цифр	<code>addValue</code>	-1	ValueError	Исключение при цифре < 0
Добавление цифр	<code>addValue</code>	"5"	ValueError	Исключение при не int типе

Категория тестирования	Метод	Входные данные	Ожидаемый результат	Описание теста
Последовательность цифр	addValue	[1, 2, 3]	"123, i* 0,"	Последовательное добавление к действительной части
Последовательность цифр	addValue	[1, 2, 3, 4, 5]	"123, i* 45,"	Автоматический переход к мнимой части
Добавление нуля	addZero	-	Содержит "0"	Добавление нулевой цифры
Изменение знака	addSign	"real"	Изменение знака действительной части	Смена знака действительной части
Изменение знака	addSign	"imaginary"	Изменение знака мнимой части	Смена знака мнимой части
Изменение знака	addSign	"invalid"	ValueError	Исключение при неверном параметре
Удаление символов	rmVal	-	Уменьшение длины строки	Удаление одного символа
Удаление символов	rmVal	Многократный вызов	"0, i* 0,"	Сброс к нулю после многократного удаления

Категория тестирования	Метод	Входные данные	Ожидаемый результат	Описание теста
Очистка	clear	-	"0, i* 0,"	Сброс к нулевому значению
Команды редактирования	redact	0-9	Добавление цифры	Цифровые команды 0-9
Команды редактирования	redact	10	"0, i* 0,"	Команда очистки
Команды редактирования	redact	11	Изменение знака	Команда смены знака
Команды редактирования	redact	12	Добавление нуля	Команда добавления нуля
Команды редактирования	redact	13	Удаление символа	Команда backspace
Команды редактирования	redact	14	ValueError	Неверная команда
Границные значения	addValue	0	Содержит "0"	Минимальная цифра
Границные значения	addValue	9	Содержит "9"	Максимальная цифра

Категория тестирования	Метод	Входные данные	Ожидаемый результат	Описание теста
Границочные значения	rmVal	На пустой строке	"0, i* 0,"	Удаление из минимальной строки
Интеграционные тесты	Последовательность	[1, addSign, 2, 3]	"-123, i* 0,"	Комбинация операций
Интеграционные тесты	Последовательность	[1, 2, 3, 4, 5]	"123, i* 45,"	Автоматический переход частей