Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Лабораторная работа №5

Выполнил:
студенты 4 курса
группы ИП-216
Андрущенко Ф.А.
Литвинов А. Е.
Русецкий А. С.

Проверил:
преподаватель кафедры ПМиК
Агалаков Антон Александрович

Новосибирск, 2025 г.

Задание

1. Реализовать абстрактный тип данных «р-ичное число», используя класс, в соответствии с приведенной ниже спецификацией.
2. Протестировать каждую операцию, определенную на типе данных по критерию С2, используя средства модульного тестирования Visual Studio.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

Спецификация типа данных «р-ичное число».

ADT TPNumber

Данные

Р-ичное число (тип TPNumber) - это действительное число (n) со знаком в системе счисления с основанием (b) в диапазоне 2..16, содержащее целую и дробную части. Точность представления числа – (c >= 0). Р-ичные числа неизменяемые.

Операции

Операции могут вызываться только объектом р-ичное число (тип TPNumber), указатель на который в них передаётся по умолчанию. При описании операций этот объект называется this «само число».

КонструкторЧисло	
Вход:	Вещественное число (a), основание системы счисления (b), точность представления числа (c)
Предусловия:	Основание системы счисления (b) должно принадлежать интервалу [2..16], точность представления числа c >= 0.

Процесс:	Инициализирует поля объекта this p-ичное число: система счисления (b), точность представления (c). В поле (n) числа заносится (a). Например: TPNumber(a,3,3) = число a в системе счисления 3 с тремя разрядами после троичной точки. TPNumber (a,3,2) = число a в системе счисления 3 с двумя разрядами после троичной точки.
Постусловия:	Объект инициализирован начальными значениями.
Выход:	Нет.
КонструкторСтрока	
Вход:	Строковые представления: p–ичного числа (a), основания системы счисления (b), точности представления числа (c)
Предусловия:	Основание системы счисления (b) должно принадлежать интервалу [2..16], точность представления числа c >= 0.
Процесс:	Инициализирует поля объекта this p-ичное число: основание системы счисления (b), точностью представления (c). В поле (n) числа this заносится результат преобразования строки (a) в числовое представление. b-ичное число (a) и основание системы счисления (b) представлены в формате строки. Например: TPNumber (“20”,”3”,”6”) = 20 в системе счисления 3, точность 6 знаков после запятой.

	TPNumber (“0”,”3”,”8”) = 0 в системе счисления 3, точность 8 знаков после запятой.
Постусловия:	Объект инициализирован начальными значениями.
Выход:	Нет.
Копировать:	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт копию самого числа this (тип TPNumber).
Выход:	p-ичное число.
Постусловия:	Нет.
Сложить	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа this.
Предусловия:	Нет.
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное сложением полей (n) самого числа this и числа d.
Выход:	p-ичное число.
Постусловия:	Нет
Умножить	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа this.
Предусловия:	Нет.
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное умножением полей (n)

	самого числа this и числа d.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
Вычесть	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа this.
Предусловия:	Нет.
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное вычитанием полей (n) самого числа this и числа d.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
Делить	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа.
Предусловия:	Поле (n) числа (d) не равно 0.
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное делением полей (n) самого числа this на поле (n) числа d.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
Обратить	
Вход:	Нет.
Предусловия:	Поле (n) самого числа не равно 0.
Процесс:	Создаёт p-ичное число, в поле (n) которого заносится значение, полученное как 1/(n) самого

	числа this.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
Квадрат	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт p-ичное число, в поле (n) которого заносится значение, полученное как квадрат поля (n) самого числа this.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
ВзятьРЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (n) самого числа this.
Выход:	Вещественное значение.
Постусловия:	Нет.
ВзятьРСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает p-ичное число (q) в формате строки, изображающей значение поля (n) самого числа this в системе счисления (b) с точностью (c).
Выход:	Строка.
Постусловия:	Нет.

ВзятьОснованиеЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (b) самого числа this
Выход:	Целочисленное значение
Постусловия:	Нет.
ВзятьОснованиеСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (b) самого числа this в формате строки, изображающей (b) в десятичной системе счисления.
Выход:	Строка.
Постусловия:	Нет.
ВзятьТочностьЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (c) самого числа this.
Выход:	Целое значение.
Постусловия:	Нет.
ВзятьТочностьСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (c) самого числа this в формате строки, изображающей (c) в десятичной системе счисления.

Выход:	Строка.
Постусловия:	Нет.
УстановитьОснованиеЧисло	
Вход:	Целое число (newb).
Предусловия:	2 <= newb <= 16.
Процесс:	Устанавливает в поле (b) самого числа this значение (newb).
Выход:	Нет.
Постусловия:	Нет.
УстановитьОснованиеСтрока	
Вход:	Строка (bs), изображающая основание (b) р-ичного числа в десятичной системе счисления.
Предусловия:	Допустимый диапазон числа, изображаемого строкой (bs) - 2,,16.
Процесс:	Устанавливает значение поля (b) самого числа this значением, полученным в результате преобразования строки (bs).
Выход:	Строка.
Постусловия:	Нет.
УстановитьТочностьЧисло	
Вход:	Целое число (newc).
Предусловия:	newc >= 0.
Процесс:	Устанавливает в поле (c) самого числа значение (newc).
Выход:	Нет.
Постусловия:	Нет.

УстановитьТочностьСтрока	
Вход:	Строка (newc).
Предусловия:	Строка (newc) изображает десятичное целое >= 0.
Процесс:	Устанавливает в поле (c) самого числа this значение, полученное преобразованием строки (newc).
Выход:	Нет.
Постусловия:	Нет.
end TPNumber	

Исходный код программы

```
import math


class TPNumberException(Exception):
    """Класс для исключительных ситуаций TPNumber"""
    pass


class TPNumber:
    def __init__(self, *args):
        """
        Конструкторы:
        1. TPNumber(a: float, b: int, c: int)
        2. TPNumber(a_str: str, b_str: str, c_str: str)
        """
        if len(args) == 3:
            if isinstance(args[0], (int, float)) and isinstance(args[1], int) and
isinstance(args[2], int):
                self._init_from_numbers(args[0], args[1], args[2])
            elif isinstance(args[0], str) and isinstance(args[1], str) and
isinstance(args[2], str):
                self._init_from_strings(args[0], args[1], args[2])
            else:
                raise TPNumberException("Неверные типы аргументов")
        else:
            raise TPNumberException("Неверное количество аргументов")

    def _init_from_numbers(self, a: float, b: int, c: int):
        """Конструктор из чисел"""
        if not (2 <= b <= 16):
            raise TPNumberException("Основание системы счисления должно быть в
диапазоне [2..16]")
        if c < 0:
            raise TPNumberException("Точность представления должна быть >= 0")
        self._n = float(a)
        self._b = b
        self._c = c

    def _init_from_strings(self, a_str: str, b_str: str, c_str: str):
        """Конструктор из строк"""
        try:
            b = int(b_str)
            c = int(c_str)
        except ValueError:
            raise TPNumberException("Основание и точность должны быть целыми
числами")

        if not (2 <= b <= 16):
            raise TPNumberException("Основание системы счисления должно быть в
диапазоне [2..16]")
```

```
        if c < 0:
            raise TPNumberException("Точность представления должна быть >= 0")

        # Преобразование строки в число
        try:
            n = self._convert_string_to_number(a_str, b)
        except ValueError:
            raise TPNumberException(f"Невозможно преобразовать строку '{a_str}' в число
с основанием {b}")

        self._n = n
        self._b = b
        self._c = c

    def _convert_string_to_number(self, s: str, base: int) -> float:
        """Преобразование строки в число с заданным основанием"""
        s = s.strip().lower()

        # Проверка на отрицательное число
        negative = False
        if s.startswith('-'):
            negative = True
            s = s[1:]

        # Разделение на целую и дробную части
        parts = s.split('.')
        if len(parts) > 2:
            raise ValueError("Неверный формат числа")

        integer_part = parts[0]
        fractional_part = parts[1] if len(parts) > 1 else ""

        # Преобразование целой части
        integer_value = 0
        for char in integer_part:
            digit = self._char_to_digit(char)
            if digit >= base:
                raise ValueError(f"Цифра {char} недопустима для основания {base}")
            integer_value = integer_value * base + digit

        # Преобразование дробной части
        fractional_value = 0.0
        if fractional_part:
            for i, char in enumerate(fractional_part, 1):
                digit = self._char_to_digit(char)
                if digit >= base:
                    raise ValueError(f"Цифра {char} недопустима для основания {base}")
                fractional_value += digit / (base ** i)

        result = integer_value + fractional_value
        return -result if negative else result
```

```
def _char_to_digit(self, char: str) -> int:
    """Преобразование символа в цифру"""
    if '0' <= char <= '9':
        return ord(char) - ord('0')
    elif 'a' <= char <= 'f':
        return 10 + ord(char) - ord('a')
    else:
        raise ValueError(f"Недопустимый символ: {char}")

def _digit_to_char(self, digit: int) -> str:
    """Преобразование цифры в символ"""
    if 0 <= digit <= 9:
        return str(digit)
    elif 10 <= digit <= 15:
        return chr(ord('a') + digit - 10)
    else:
        raise ValueError(f"Недопустимая цифра: {digit}")

def copy(self):
    """Создать копию числа"""
    return TPNumber(self._n, self._b, self._c)

def add(self, other):
    """Сложение"""
    if not isinstance(other, TPNumber):
        raise TPNumberException("Можно складывать только с TPNumber")
    if self._b != other._b or self._c != other._c:
        raise TPNumberException("Основания и точности должны совпадать")
    return TPNumber(self._n + other._n, self._b, self._c)

def multiply(self, other):
    """Умножение"""
    if not isinstance(other, TPNumber):
        raise TPNumberException("Можно умножать только на TPNumber")
    if self._b != other._b or self._c != other._c:
        raise TPNumberException("Основания и точности должны совпадать")
    return TPNumber(self._n * other._n, self._b, self._c)

def subtract(self, other):
    """Вычитание"""
    if not isinstance(other, TPNumber):
        raise TPNumberException("Можно вычитать только TPNumber")
    if self._b != other._b or self._c != other._c:
        raise TPNumberException("Основания и точности должны совпадать")
    return TPNumber(self._n - other._n, self._b, self._c)

def divide(self, other):
    """Деление"""
    if not isinstance(other, TPNumber):
        raise TPNumberException("Можно делить только на TPNumber")
    if self._b != other._b or self._c != other._c:
        raise TPNumberException("Основания и точности должны совпадать")
```

```
        if other._n == 0:
            raise TPNumberException("Деление на ноль невозможно")
        return TPNumber(self._n / other._n, self._b, self._c)

    def invert(self):
        """Обратное число (1/n)"""
        if self._n == 0:
            raise TPNumberException("Невозможно вычислить обратное число для нуля")
        return TPNumber(1.0 / self._n, self._b, self._c)

    def square(self):
        """Квадрат числа"""
        return TPNumber(self._n * self._n, self._b, self._c)

    def get_number(self) -> float:
        """Получить число как вещественное значение"""
        return self._n

    def get_number_string(self) -> str:
        """Получить число как строку в системе счисления с основанием b"""
        return self._convert_number_to_string(self._n, self._b, self._c)

    def _convert_number_to_string(self, number: float, base: int, precision: int) ->
str:
        """Преобразование числа в строку с заданным основанием и точностью"""
        if number == 0:
            return "0" + ("." + "0" * precision) if precision > 0 else "")

        negative = number < 0
        number = abs(number)

        # Целая часть
        integer_part = int(number)
        fractional_part = number - integer_part

        # Преобразование целой части
        integer_str = ""
        if integer_part == 0:
            integer_str = "0"
        else:
            temp = integer_part
            while temp > 0:
                digit = temp % base
                integer_str = self._digit_to_char(digit) + integer_str
                temp //= base

        # Преобразование дробной части с дополнением нулями
        fractional_str = ""
        if precision > 0:
            fractional_str = "."
            temp = fractional_part
            for _ in range(precision):
```

```
                temp *= base
                digit = int(temp)
                fractional_str += self._digit_to_char(digit)
                temp -= digit

        result = integer_str + fractional_str
        return ("-" + result) if negative else result

    def get_base_number(self) -> int:
        """Получить основание как число"""
        return self._b

    def get_base_string(self) -> str:
        """Получить основание как строку"""
        return str(self._b)

    def get_precision_number(self) -> int:
        """Получить точность как число"""
        return self._c

    def get_precision_string(self) -> str:
        """Получить точность как строку"""
        return str(self._c)

    def set_base_number(self, new_b: int):
        """Установить основание (число)"""
        if not (2 <= new_b <= 16):
            raise TPNumberException("Основание системы счисления должно быть в
диапазоне [2..16]")
        self._b = new_b

    def set_base_string(self, bs: str):
        """Установить основание (строка)"""
        try:
            new_b = int(bs)
        except ValueError:
            raise TPNumberException("Основание должно быть целым числом")
        if not (2 <= new_b <= 16):
            raise TPNumberException("Основание системы счисления должно быть в
диапазоне [2..16]")
        self._b = new_b

    def set_precision_number(self, new_c: int):
        """Установить точность (число)"""
        if new_c < 0:
            raise TPNumberException("Точность представления должна быть >= 0")
        self._c = new_c

    def set_precision_string(self, new_c: str):
        """Установить точность (строка)"""
        try:
            c = int(new_c)
```

```
        except ValueError:
            raise TPNumberException("Точность должна быть целым числом")
        if c < 0:
            raise TPNumberException("Точность представления должна быть >= 0")
        self._c = c

    def __str__(self):
        return f"TPNumber({self.get_number_string()}, base={self._b},
precision={self._c})"

    def __repr__(self):
        return self.__str__()
```

Модульные тесты для тестирования класса

```
import unittest
from tpnumber import TPNumber, TPNumberException


class TestTPNumber(unittest.TestCase):
    def test_constructor_numbers(self):
        # Тест конструктора с числами
        num = TPNumber(10.5, 10, 2)
        self.assertEqual(num.get_number(), 10.5)
        self.assertEqual(num.get_base_number(), 10)
        self.assertEqual(num.get_precision_number(), 2)

    def test_constructor_strings(self):
        # Тест конструктора со строками
        num = TPNumber("10.5", "10", "2")
        self.assertEqual(num.get_number(), 10.5)
        self.assertEqual(num.get_base_number(), 10)
        self.assertEqual(num.get_precision_number(), 2)

    def test_constructor_invalid_base(self):
        # Тест с недопустимым основанием
        with self.assertRaises(TPNumberException):
            TPNumber(10, 1, 2)
        with self.assertRaises(TPNumberException):
            TPNumber(10, 17, 2)

    def test_constructor_invalid_precision(self):
        # Тест с отрицательной точностью
        with self.assertRaises(TPNumberException):
            TPNumber(10, 10, -1)

    def test_copy(self):
        # Тест копирования
        num1 = TPNumber(10.5, 10, 2)
        num2 = num1.copy()
        self.assertEqual(num1.get_number(), num2.get_number())
        self.assertEqual(num1.get_base_number(), num2.get_base_number())
        self.assertEqual(num1.get_precision_number(), num2.get_precision_number())

    def test_add(self):
        # Тест сложения
        num1 = TPNumber(10.5, 10, 2)
        num2 = TPNumber(5.5, 10, 2)
        result = num1.add(num2)
        self.assertEqual(result.get_number(), 16.0)

    def test_multiply(self):
        # Тест умножения
        num1 = TPNumber(10.0, 10, 2)
        num2 = TPNumber(2.5, 10, 2)
```

```
        result = num1.multiply(num2)
        self.assertEqual(result.get_number(), 25.0)

    def test_subtract(self):
        # Тест вычитания
        num1 = TPNumber(10.0, 10, 2)
        num2 = TPNumber(3.5, 10, 2)
        result = num1.subtract(num2)
        self.assertEqual(result.get_number(), 6.5)

    def test_divide(self):
        # Тест деления
        num1 = TPNumber(10.0, 10, 2)
        num2 = TPNumber(4.0, 10, 2)
        result = num1.divide(num2)
        self.assertEqual(result.get_number(), 2.5)

    def test_divide_by_zero(self):
        # Тест деления на ноль
        num1 = TPNumber(10.0, 10, 2)
        num2 = TPNumber(0.0, 10, 2)
        with self.assertRaises(TPNumberException):
            num1.divide(num2)

    def test_invert(self):
        # Тест обратного числа
        num = TPNumber(4.0, 10, 2)
        result = num.invert()
        self.assertEqual(result.get_number(), 0.25)

    def test_invert_zero(self):
        # Тест обратного числа для нуля
        num = TPNumber(0.0, 10, 2)
        with self.assertRaises(TPNumberException):
            num.invert()

    def test_square(self):
        # Тест квадрата
        num = TPNumber(5.0, 10, 2)
        result = num.square()
        self.assertEqual(result.get_number(), 25.0)

    def test_get_number_string(self):
        # Тест получения числа как строки
        num = TPNumber(10.5, 10, 2)
        self.assertEqual(num.get_number_string(), "10.50")

    def test_get_number_string_binary(self):
        # Тест получения числа в двоичной системе
        num = TPNumber(5.5, 2, 4)
        self.assertEqual(num.get_number_string(), "101.1000")
```

```
    def test_get_number_string_hex(self):
        # Тест получения числа в шестнадцатеричной системе
        num = TPNumber(255.5, 16, 2)
        self.assertEqual(num.get_number_string(), "ff.80")

    def test_set_base_number(self):
        # Тест установки основания (число)
        num = TPNumber(10.0, 10, 2)
        num.set_base_number(16)
        self.assertEqual(num.get_base_number(), 16)

    def test_set_base_string(self):
        # Тест установки основания (строка)
        num = TPNumber(10.0, 10, 2)
        num.set_base_string("16")
        self.assertEqual(num.get_base_number(), 16)

    def test_set_precision_number(self):
        # Тест установки точности (число)
        num = TPNumber(10.0, 10, 2)
        num.set_precision_number(4)
        self.assertEqual(num.get_precision_number(), 4)

    def test_set_precision_string(self):
        # Тест установки точности (строка)
        num = TPNumber(10.0, 10, 2)
        num.set_precision_string("4")
        self.assertEqual(num.get_precision_number(), 4)

    def test_different_bases_error(self):
        # Тест ошибки при разных основаниях
        num1 = TPNumber(10.0, 10, 2)
        num2 = TPNumber(5.0, 16, 2)
        with self.assertRaises(TPNumberException):
            num1.add(num2)

    def test_different_precisions_error(self):
        # Тест ошибки при разных точностях
        num1 = TPNumber(10.0, 10, 2)
        num2 = TPNumber(5.0, 10, 4)
        with self.assertRaises(TPNumberException):
            num1.add(num2)


if __name__ == "__main__":
    from rich import print
    from rich.panel import Panel
    from rich.console import Console
    from unittest import TextTestRunner, TestResult

    console = Console()
```

```
    class RichTestResult(TestResult):
        def __init__(self, stream, descriptions, verbosity):
            super().__init__(stream, descriptions, verbosity)

        def startTest(self, test):
            super().startTest(test)
            console.print(f"[cyan]Running:[/cyan] {test._testMethodName}")

        def addSuccess(self, test):
            super().addSuccess(test)
            console.print(f"[green]✓ PASS:[/green] {test._testMethodName}")

        def addFailure(self, test, err):
            super().addFailure(test, err)
            console.print(f"[red]✗ FAIL:[/red] {test._testMethodName}")

        def addError(self, test, err):
            super().addError(test, err)
            console.print(f"[magenta]💥 ERROR:[/magenta] {test._testMethodName}")

    # Запуск с Rich
    runner = TextTestRunner(resultclass=RichTestResult, verbosity=0)
    result = runner.run(unittest.defaultTestLoader.loadTestsFromTestCase(TestTPNumber))

    # Красивая статистика
    console.print(
        Panel.fit(
            f"[green]Passed: {result.testsRun - len(result.failures) -
len(result.errors)}[/green]\n"
            f"[red]Failed: {len(result.failures)}[/red]\n"
            f"[magenta]Errors: {len(result.errors)}[/magenta]\n"
            f"[yellow]Total: {result.testsRun}[/yellow]",
            title="Test Results",
        )
    )
```

Результат тестирования методов класса