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 №7

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

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 «память», для хранения одного числа – объекта типа T, 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 typing import TypeVar, Generic, Callable

T = TypeVar('T')

class TMemory(Generic[T]):

    _On = "Включена"
    _Off = "Выключена"

    def __init__(self, default_value: T):

        self._default_value = default_value  # Значение по умолчанию типа T
        self._FNumber = default_value        # Текущее значение типа T
        self._FState = self._Off

    def Store(self, value: T) -> None:
        self._FNumber = value
        self._FState = self._On

    def Get(self) -> T:
        self._FState = self._On
        return self._FNumber

    def Add(self, value: T):
        try:
            self._FNumber = self._FNumber + value
            self._FState = self._On
        except TypeError as e:
            raise TypeError(f"Тип {type(self._FNumber).__name__} не
поддерживает операцию сложения") from e

    def Clear(self) -> None:
        self._FNumber = self._default_value
        self._FState = self._Off

    def ReadMemoryState(self) -> str:
```

```
    return self._FState

def ReadNumber(self) -> T:
    return self._FNumber

@property
def Number(self) -> T:
    return self._FNumber

@property
def State(self) -> str:
    return self._FState
```

Tests.py:

```
from lab7 import TMemory
import pytest


class TestsMemory:
    def test_initialization(self):
        """Тест инициализации"""
        memory = TMemory[int](0)
        assert memory.Number == 0
        assert memory.State == "Выключена"
        assert memory.ReadMemoryState() == "Выключена"
        assert memory.ReadNumber() == 0

    def test_store(self):
        """Тест записи"""
        memory = TMemory[int](0)

        memory.Store(100)
        assert memory.Number == 100
        assert memory.State == "Включена"

    def test_get(self):
        """Тест получения"""
        memory = TMemory[int](0)
        memory.Store(50)

        result = memory.Get()
```

```
    assert result == 50
    assert memory.State == "Включена"

def test_add(self):
    memory = TMemory[int](0)
    memory.Store(10)

    memory.Add(5)
    assert memory.Number == 15
    assert memory.State == "Включена"

def test_clear(self):
    """Тест операции очистки"""
    memory = TMemory[int](0)
    memory.Store(100)

    memory.Clear()
    assert memory.Number == 0
    assert memory.State == "Выключена"

def test_different_types(self):
    """Тест с разными типами данных"""

    # Целые числа
    int_memory = TMemory[int](0)
    int_memory.Store(5)
    int_memory.Add(3)
```

```
        assert int_memory.Number == 8

        # Дробные числа
        float_memory = TMemory[float](0.0)
        float_memory.Store(2.5)
        float_memory.Add(1.5)
        assert float_memory.Number == 4.0

        # Строки
        str_memory = TMemory[str]("")
        str_memory.Store("Hello")
        str_memory.Add(" World")
        assert str_memory.Number == "Hello World"

        # Списки
        list_memory = TMemory[list]([])
        list_memory.Store([1, 2])
        list_memory.Add([3, 4])
        assert list_memory.Number == [1, 2, 3, 4]

    def test_read_state(self):
        memory = TMemory[int](0)
        assert memory.ReadMemoryState() == "Выключена"

    def test_read_number(self):
        memory = TMemory[int](0)
        memory.Store(40)
```

```
        assert memory.ReadNumber() == 40

    def test_default_value(self):
        """Тест что значение по умолчанию сохраняется после очистки"""
        memory = TMemory[int](52)  # Нестандартное значение по умолчанию

        memory.Store(777)
        memory.Clear()

        assert memory.Number == 52

    def test_properties_consistency(self):
        """Тест согласованности свойств и методов"""
        memory = TMemory[int](0)

        testNum = 69

        testState = "Включена"

        memory.Store(testNum)

        assert memory.Number == testNum
        assert memory.ReadNumber() == testNum
        assert memory.Get() == testNum

        assert memory.State == testState
        assert memory.ReadMemoryState() == testState
```

```
    def test_dict_type_unsupported_add(self):
        """Тест со словарями - сложение не поддерживается"""
        dict_memory = TMemory[dict]({})
        dict_memory.Store({"a": 1})

        # Сложение словарей не поддерживается в Python
        with pytest.raises(TypeError):
            dict_memory.Add({"b": 2})

    def test_boolean_type(self):
        """Тест с булевым типом (сложение работает, так как bool наследуется от int)"""
        bool_memory = TMemory[bool](False)
        bool_memory.Store(False)
        assert bool_memory.Number == False

        # В Python bool поддерживает сложение (True=1, False=0)
        bool_memory.Add(True)
        assert bool_memory.Number == True
```

3. Тестовые наборы данных для тестирования класса

№ теста	Название теста	Тип данных	Входные данные	Ожидаемый результат
1	test_initialization	int	default_value=0	Number=0, State="Выключена"
2	test_store	int	value=100	Number=100, State="Включена"
3	test_get	int	value=50	result=50, State="Включена"
4	test_add	int	initial=10, add=5	Number=15, State="Включена"
5	test_clear	int	initial=100	Number=0, State="Выключена"
6	test_different_types (int)	int	initial=5, add=3	Number=8
7	test_different_types (float)	float	initial=2.5, add=1.5	Number=4.0
8	test_different_types (str)	str	initial="Hello", add=" World"	Number="Hello World"
9	test_different_types (list)	list	initial=[1,2], add=[3,4]	Number=[1,2,3,4]

№ теста	Название теста	Тип данных	Входные данные	Ожидаемый результат
10	test_read_state	int	-	State="Выключена"
11	test_read_number	int	value=40	Number=40
12	test_default_value	int	default=52, store=777, clear	Number=52
13	test_properties_consistency	int	value=69	Number=69, State="Включена"
14	test_dict_type_unsupported_add	dict	initial={"a": 1}, add={"b":2}	TypeError
15	test_boolean_type	bool	initial=False, add=True	Number=True