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

Лабораторная работа №10

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. В соответствии с приведенной ниже спецификацией реализуйте шаблон классов «множество». Для тестирования в качестве параметра шаблона T выберите типы:
   - int;
   - TFrac (простая дробь), разработанный вами ранее.
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций

Спецификация типа данных «множество»
ADT tset

Данные
Множества - это изменяемые неограниченные наборы элементов типа T.
Содержимое множества изменяется следующими операциями:
Опустошить (опустошение множества);
Добавить (добавление элемента во множество);
Удалить (извлечение элемента из множества).
Множество поддерживает следующую дисциплину записи и извлечения элементов: элемент может присутствовать во множестве только в одном экземпляре, при извлечении выбирается заданный элемент множества и удаляется из множества.

Операции
Операции могут вызываться только объектом «множество» (тип tset), указатель на который передаётся в них по умолчанию. При описании операций этот объект в разделе «Вход» не указывается.

Исходный код программы

```
from typing import TypeVar, Generic, Iterable, Iterator

T = TypeVar("T")


class TFrac:
    """Класс для представления простой дроби"""

    def __init__(self, numerator=0, denominator=1):
        """
        Инициализация дроби
        Args:
            numerator: числитель
            denominator: знаменатель (не может быть 0)
        Raises:
            ValueError: если знаменатель равен 0
        """
        if denominator == 0:
            raise ValueError("Denominator cannot be zero")
        self.numerator = numerator
        self.denominator = denominator
        self._normalize()

    def _normalize(self):
        """Приведение дроби к нормальной форме (сокращение)"""
        if self.numerator == 0:
            self.denominator = 1
            return

        a, b = abs(self.numerator), abs(self.denominator)
        while b:
            a, b = b, a % b
        gcd = a

        self.numerator //= gcd
        self.denominator //= gcd

        if self.denominator < 0:
            self.numerator = -self.numerator
            self.denominator = -self.denominator

    def __eq__(self, other):
        """Проверка равенства дробей"""
        if isinstance(other, TFrac):
            return (
                self.numerator * other.denominator == other.numerator *
self.denominator
            )
        return False
```

```
    def __hash__(self):
        """Хеш для использования в множествах"""
        return hash((self.numerator, self.denominator))

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

    def __repr__(self):
        """Представление для отладки"""
        return f"TFrac({self.numerator}, {self.denominator})"


class TSet(Generic[T], set):
    """
    Универсальное множество элементов типа T
    Наследуется от встроенного типа set с добавлением типизации и дополнительных методов
    """

    def __init__(self, elements: Iterable[T] = None):
        """
        Конструктор множества
        Args:
            elements: итерируемый объект с элементами (опционально)
        """
        if elements is None:
            super().__init__()
        else:
            super().__init__(elements)

    def clear(self) -> None:
        """
        Опустошить множество - удалить все элементы
        """
        super().clear()

    def add(self, element: T) -> None:
        """
        Добавить элемент в множество, если его там нет
        Args:
            element: элемент для добавления
        """
        super().add(element)

    def remove(self, element: T) -> None:
        """
        Удалить элемент из множества, если он присутствует
        Args:
            element: элемент для удаления
        Raises:
            KeyError: если элемент не найден в множестве
```

```
        """
        super().remove(element)

    def is_empty(self) -> bool:
        """
        Проверить, пусто ли множество
        Returns:
            True если множество пустое, иначе False
        """
        return len(self) == 0

    def contains(self, element: T) -> bool:
        """
        Проверить принадлежность элемента множеству
        Args:
            element: элемент для проверки
        Returns:
            True если элемент принадлежит множеству, иначе False
        """
        return element in self

    def union(self, other: "TSet[T]") -> "TSet[T]":
        """
        Объединить с другим множеством
        Args:
            other: другое множество для объединения
        Returns:
            Новое множество - объединение текущего с другим
        """
        if not isinstance(other, TSet):
            raise TypeError("Argument must be TSet")
        return TSet(super().__or__(other))

    def difference(self, other: "TSet[T]") -> "TSet[T]":
        """
        Вычесть другое множество
        Args:
            other: множество для вычитания
        Returns:
            Новое множество - разность текущего с другим
        """
        if not isinstance(other, TSet):
            raise TypeError("Argument must be TSet")
        return TSet(super().__sub__(other))

    def intersection(self, other: "TSet[T]") -> "TSet[T]":
        """
        Пересечь с другим множеством
        Args:
            other: множество для пересечения
        Returns:
            Новое множество - пересечение текущего с другим
```

```
        """
        if not isinstance(other, TSet):
            raise TypeError("Argument must be TSet")
        return TSet(super().__and__(other))

    def size(self) -> int:
        """
        Получить количество элементов в множестве
        Returns:
            Количество элементов
        """
        return len(self)

    def get_element(self, index: int) -> T:
        """
        Получить элемент по индексу (для итерации)
        Args:
            index: индекс элемента
        Returns:
            Элемент по указанному индексу
        Raises:
            IndexError: если индекс вне диапазона
        """
        if index < 0 or index >= len(self):
            raise IndexError("Index out of range")
        return list(self)[index]

    def __iter__(self) -> Iterator[T]:
        """Итератор по элементам множества"""
        return super().__iter__()

    def __str__(self) -> str:
        """Строковое представление множества"""
        elements = ", ".join(str(x) for x in self)
        return f"TSet{{{elements}}}"

    def __repr__(self) -> str:
        """Представление для отладки"""
        return f"TSet({super().__repr__()})"


def demonstrate_int_set():
    """Демонстрация работы с целыми числами"""
    print("=== Демонстрация TSet с целыми числами ===")

    set_a = TSet([1, 2, 3, 4, 5])
    set_b = TSet([3, 4, 5, 6, 7])
    print(f"Множество A: {set_a}")
    print(f"Множество B: {set_b}")
    print(f"Размер A: {set_a.size()}")
    print(f"Пусто ли A: {set_a.is_empty()}")
    print(f"Содержит ли A число 3: {set_a.contains(3)}")
```

```
    union_set = set_a.union(set_b)
    diff_set = set_a.difference(set_b)
    inter_set = set_a.intersection(set_b)
    print(f"Объединение A и B: {union_set}")
    print(f"Разность A и B: {diff_set}")
    print(f"Пересечение A и B: {inter_set}")

    print("Элементы множества A:")
    for i in range(set_a.size()):
        element = set_a.get_element(i)
        print(f" Элемент {i}: {element}")

    print("\nДемонстрация добавления и удаления:")
    set_a.add(10)
    print(f"После добавления 10: {set_a}")
    set_a.remove(3)
    print(f"После удаления 3: {set_a}")

    set_a.clear()
    print(f"После очистки: {set_a}, пустое: {set_a.is_empty()}")
    print()


def demonstrate_frac_set():
    """Демонстрация работы с дробями"""
    print("=== Демонстрация TSet с дробями TFrac ===")

    frac1 = TFrac(1, 2)
    frac2 = TFrac(3, 4)
    frac3 = TFrac(2, 4)
    frac4 = TFrac(5, 6)
    print(f"Дробь 1: {frac1}")
    print(f"Дробь 2: {frac2}")
    print(f"Дробь 3: {frac3}")
    print(f"Дробь 4: {frac4}")
    print(f"Дробь 1 == Дробь 3: {frac1 == frac3}")

    set_x = TSet([frac1, frac2])
    set_y = TSet([frac3, frac4])
    print(f"Множество X: {set_x}")
    print(f"Множество Y: {set_y}")

    union_frac = set_x.union(set_y)
    diff_frac = set_x.difference(set_y)
    inter_frac = set_x.intersection(set_y)
    print(f"Объединение X и Y: {union_frac}")
    print(f"Разность X и Y: {diff_frac}")
    print(f"Пересечение X и Y: {inter_frac}")

    test_frac = TFrac(2, 4)
    print(f"Содержит ли X дробь 2/4: {set_x.contains(test_frac)}")
```

```
    print(f"Содержит ли Y дробь 1/2: {set_y.contains(TFrac(1, 2))}")

    print("Элементы множества X:")
    for i in range(set_x.size()):
        element = set_x.get_element(i)
        print(f" Элемент {i}: {element}")
    print()


def demonstrate_error_handling():
    """Демонстрация обработки ошибок"""
    print("=== Демонстрация обработки ошибок ===")

    set_empty = TSet()
    set_numbers = TSet([1, 2, 3])
    print("Пустое множество:", set_empty)
    print("Множество с числами:", set_numbers)

    print("\n1. Попытка удаления несуществующего элемента:")
    try:
        set_empty.remove(1)
        print("Удаление выполнено успешно")
    except KeyError as e:
        print(f"Ошибка при удалении: {e}")

    print("\n2. Попытка получения элемента по неверному индексу:")
    try:
        element = set_empty.get_element(0)
        print(f"Получен элемент: {element}")
    except IndexError as e:
        print(f"Ошибка при получении элемента: {e}")

    print("\n3. Попытка создания дроби с нулевым знаменателем:")
    try:
        invalid_frac = TFrac(1, 0)
        print(f"Дробь создана: {invalid_frac}")
    except ValueError as e:
        print(f"Ошибка создания дроби: {e}")

    print("\n4. Попытка объединения с неправильным типом:")
    try:
        result = set_numbers.union([4, 5, 6])
        print(f"Объединение выполнено: {result}")
    except TypeError as e:
        print(f"Ошибка при объединении: {e}")
    print()


def demonstrate_all_methods():
    """Демонстрация всех методов класса TSet"""
    print("=== Демонстрация всех методов TSet ===")
```

```
    print("1. Конструктор:")
    set1 = TSet([10, 20, 30])
    set2 = TSet()
    print(f" Множество 1: {set1}")
    print(f" Множество 2: {set2}")

    print("\n2. Метод add():")
    set2.add(100)
    set2.add(200)
    set2.add(100)
    print(f" После добавления: {set2}")

    print("\n3. Метод remove():")
    set1.remove(20)
    print(f" После удаления 20: {set1}")

    print("\n4. Метод is_empty():")
    print(f" set1 пустое: {set1.is_empty()}")
    print(f" set2 пустое: {set2.is_empty()}")

    print("\n5. Метод contains():")
    print(f" set1 содержит 10: {set1.contains(10)}")
    print(f" set1 содержит 50: {set1.contains(50)}")

    print("\n6. Метод union():")
    set3 = TSet([30, 40, 50])
    union_set = set1.union(set3)
    print(f" {set1} ∪ {set3} = {union_set}")

    print("\n7. Метод difference():")
    diff_set = set1.difference(set3)
    print(f" {set1} - {set3} = {diff_set}")

    print("\n8. Метод intersection():")
    inter_set = set1.intersection(set3)
    print(f" {set1} ∩ {set3} = {inter_set}")

    print("\n9. Метод size():")
    print(f" Размер set1: {set1.size()}")
    print(f" Размер union_set: {union_set.size()}")

    print("\n10. Метод get_element():")
    print(" Элементы set1 по индексам:")
    for i in range(set1.size()):
        print(f" Индекс {i}: {set1.get_element(i)}")

    print("\n11. Метод clear():")
    set1.clear()
    print(f" После очистки: {set1}, пустое: {set1.is_empty()}")


def main():
```

```
    """Основная функция демонстрации"""
    demonstrate_int_set()
    demonstrate_frac_set()
    demonstrate_error_handling()
    demonstrate_all_methods()
    print("\n=== Демонстрация завершена ===")


if __name__ == "__main__":
    main()
```

Модульные тесты для тестирования класса

```
import unittest
from tset import TSet, TFrac


class TestTFrac(unittest.TestCase):
    """Тесты для класса TFrac"""

    def test_initialization(self):
        """Тест инициализации дроби"""
        frac = TFrac(3, 4)
        self.assertEqual(frac.numerator, 3)
        self.assertEqual(frac.denominator, 4)

    def test_normalization(self):
        """Тест нормализации дроби"""
        frac = TFrac(2, 4)
        self.assertEqual(frac.numerator, 1)
        self.assertEqual(frac.denominator, 2)

        frac2 = TFrac(6, 9)
        self.assertEqual(frac2.numerator, 2)
        self.assertEqual(frac2.denominator, 3)

    def test_zero_denominator(self):
        """Тест создания дроби с нулевым знаменателем"""
        with self.assertRaises(ValueError):
            TFrac(1, 0)

    def test_equality(self):
        """Тест сравнения дробей"""
        frac1 = TFrac(1, 2)
        frac2 = TFrac(2, 4)
        frac3 = TFrac(3, 4)
        self.assertEqual(frac1, frac2)
        self.assertNotEqual(frac1, frac3)
```

```
    def test_hash(self):
        """Тест хеширования дробей"""
        frac1 = TFrac(1, 2)
        frac2 = TFrac(2, 4)
        self.assertEqual(hash(frac1), hash(frac2))

    def test_string_representation(self):
        """Тест строкового представления"""
        frac = TFrac(3, 4)
        self.assertEqual(str(frac), "3/4")


class TestTSet(unittest.TestCase):
    """Тесты для класса TSet"""

    def test_initialization(self):
        """Тест инициализации множества"""
        set1 = TSet([1, 2, 3])
        self.assertEqual(set1.size(), 3)

        set2 = TSet()
        self.assertEqual(set2.size(), 0)

    def test_add(self):
        """Тест добавления элемента"""
        set1 = TSet()
        set1.add(10)
        set1.add(20)
        set1.add(10)
        self.assertEqual(set1.size(), 2)

    def test_remove(self):
        """Тест удаления элемента"""
        set1 = TSet([1, 2, 3])
        set1.remove(2)
        self.assertEqual(set1.size(), 2)
        self.assertFalse(set1.contains(2))

        with self.assertRaises(KeyError):
            set1.remove(10)

    def test_is_empty(self):
        """Тест проверки пустоты"""
        set1 = TSet()
        self.assertTrue(set1.is_empty())

        set1.add(1)
        self.assertFalse(set1.is_empty())

    def test_contains(self):
        """Тест проверки принадлежности"""
        set1 = TSet([1, 2, 3])
```

```
        self.assertTrue(set1.contains(2))
        self.assertFalse(set1.contains(10))

    def test_union(self):
        """Тест объединения множеств"""
        set1 = TSet([1, 2, 3])
        set2 = TSet([3, 4, 5])
        union_set = set1.union(set2)
        self.assertEqual(union_set.size(), 5)
        self.assertTrue(union_set.contains(1))
        self.assertTrue(union_set.contains(5))

    def test_difference(self):
        """Тест разности множеств"""
        set1 = TSet([1, 2, 3, 4])
        set2 = TSet([3, 4, 5])
        diff_set = set1.difference(set2)
        self.assertEqual(diff_set.size(), 2)
        self.assertTrue(diff_set.contains(1))
        self.assertFalse(diff_set.contains(3))

    def test_intersection(self):
        """Тест пересечения множеств"""
        set1 = TSet([1, 2, 3, 4])
        set2 = TSet([3, 4, 5])
        inter_set = set1.intersection(set2)
        self.assertEqual(inter_set.size(), 2)
        self.assertTrue(inter_set.contains(3))
        self.assertFalse(inter_set.contains(1))

    def test_get_element(self):
        """Тест получения элемента по индексу"""
        set1 = TSet([10, 20, 30])
        element = set1.get_element(0)
        self.assertIn(element, [10, 20, 30])

        with self.assertRaises(IndexError):
            set1.get_element(10)

    def test_clear(self):
        """Тест очистки множества"""
        set1 = TSet([1, 2, 3])
        set1.clear()
        self.assertTrue(set1.is_empty())
        self.assertEqual(set1.size(), 0)

    def test_with_fractions(self):
        """Тест множества с дробями"""
        frac1 = TFrac(1, 2)
        frac2 = TFrac(3, 4)
        frac3 = TFrac(2, 4)
```

```
        set1 = TSet([frac1, frac2])
        self.assertEqual(set1.size(), 2)
        self.assertTrue(set1.contains(frac3))


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

    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}")

    loader = unittest.defaultTestLoader
    suite = TestSuite()
    suite.addTests(loader.loadTestsFromTestCase(TestTFrac))
    suite.addTests(loader.loadTestsFromTestCase(TestTSet))

    runner = TextTestRunner(resultclass=RichTestResult, verbosity=0)
    result = runner.run(suite)

    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

```
❯ python test_tset.py
Running: test_equality
✓ PASS: test_equality
Running: test_hash
✓ PASS: test_hash
Running: test_initialization
✓ PASS: test_initialization
Running: test_normalization
✓ PASS: test_normalization
Running: test_string_representation
✓ PASS: test_string_representation
Running: test_zero_denominator
✓ PASS: test_zero_denominator
Running: test_add
✓ PASS: test_add
Running: test_clear
✓ PASS: test_clear
Running: test_contains
✓ PASS: test_contains
Running: test_difference
✓ PASS: test_difference
Running: test_get_element
✓ PASS: test_get_element
Running: test_initialization
✓ PASS: test_initialization
Running: test_intersection
✓ PASS: test_intersection
Running: test_is_empty
✓ PASS: test_is_empty
Running: test_remove
✓ PASS: test_remove
Running: test_union
✓ PASS: test_union
Running: test_with_fractions
✓ PASS: test_with_fractions
Ran 17 tests in 0.005s

OK
┌─ Test Results ─┐
  Passed: 17
  Failed: 0
  Errors: 0
  Total: 17
└────────────────┘
```