Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

# АБСТРАКТНЫЕ КЛАССЫ

Отчёт о лабораторной работе № 3

по дисциплине «Современные средства программирования»

Выполнил: студент гр. 434-М1

Колпаков Н.А.

« » 2024 г.

Проверил: доцент каф. АСУ

Алфёров С. М.

« » 2024 г.

Томск 2024

# Оглавление

1. [Цели и задачи 3](#_bookmark0)
2. [Ход работы 4](#_bookmark0)

[Вывод 5](#_bookmark1)

## 1 ЦЕЛИ И ЗАДАЧИ

1. Цель работы:

Получить навыки создания иерархии классов с использованием абстрактных классов; уяснить пользу от использования абстрактных классов.

1. Задания:

Создать указанный абстрактный класс. Создать несколько производных классов, реализующие поведение абстрактного класса. Продемонстрировать возможность работы с объектами разных классов через методы абстрактного класса.

Вариант 5

Абстрактный класс: Очередь.

Дочерние классы: Очередь на основе статического массива, Очередь на основе динамического массива.

## 2 ХОД РАБОТЫ

Листинг программы:

from abc import ABC, abstractmethod

class Queue(ABC):

    @abstractmethod

    def enqueue(self, value):

        pass

    @abstractmethod

    def dequeue(self):

        pass

    @abstractmethod

    def is\_empty(self):

        pass

    @abstractmethod

    def display(self):

        pass

class StaticArrayQueue(Queue):

    def \_\_init\_\_(self, size=5):

        self.size = size

        self.queue = [None] \* size

        self.front = 0

        self.rear = 0

        self.count = 0

    def enqueue(self, value):

        if self.count == self.size:

            raise OverflowError("Очередь полная")

        self.queue[self.rear] = value

        self.rear = (self.rear + 1) % self.size

        self.count += 1

    def dequeue(self):

        if self.is\_empty():

            raise IndexError("Очередь пустая")

        value = self.queue[self.front]

        self.front = (self.front + 1) % self.size

        self.count -= 1

        return value

    def is\_empty(self):

        return self.count == 0

    def display(self):

        if self.is\_empty():

            print("Очередь пустая")

        else:

            print("Очередь:", end=" ")

            for i in range(self.count):

                print(self.queue[(self.front + i) % self.size], end=" ")

            print()

class DynamicArrayQueue(Queue):

    def \_\_init\_\_(self, size=5):

        self.queue = [None] \* size

        self.front = 0

        self.rear = 0

        self.size = size

        self.count = 0

    def enqueue(self, value):

        if self.count == self.size:

            self.\_resize(self.size \* 2)

        self.queue[self.rear] = value

        self.rear = (self.rear + 1) % self.size

        self.count += 1

    def dequeue(self):

        if self.is\_empty():

            raise IndexError("Очередь пустая")

        value = self.queue[self.front]

        self.front = (self.front + 1) % self.size

        self.count -= 1

        return value

    def is\_empty(self):

        return self.count == 0

    def \_resize(self, new\_size):

        new\_queue = [None] \* new\_size

        for i in range(self.count):

            new\_queue[i] = self.queue[(self.front + i) % self.size]

        self.queue = new\_queue

        self.front = 0

        self.rear = self.count

        self.size = new\_size

    def display(self):

        if self.is\_empty():

            print("Очередь пустая")

        else:

            print("Очередь:", end=" ")

            for i in range(self.count):

                print(self.queue[(self.front + i) % self.size], end=" ")

            print()

def user\_select\_queue():

    print("Выберите тип очереди:")

    print("1. Статическая очередь")

    print("2. Динамическая очередь")

    choice = input("Введите номер вашего выбора: ")

    size = int(input("Введите размер очереди: "))

    match choice:

        case "1":

            return StaticArrayQueue(size)

        case "2":

            return DynamicArrayQueue(size)

        case \_:

            print("Неверный выбор, создается очередь по умолчанию (динамическая).")

            return DynamicArrayQueue(size)

if \_\_name\_\_ == "\_\_main\_\_":

    queue = user\_select\_queue()

    while True:

        print("\n1. Добавить элемент в очередь")

        print("2. Удалить элемент из очереди")

        print("3. Показать очередь")

        print("4. Выход")

        action = input("Выберите действие: ")

        match action:

            case "1":

                value = int(input("Введите элемент для добавления: "))

                try:

                    queue.enqueue(value)

                except OverflowError as e:

                    print(e)

            case "2":

                try:

                    print("Удалён элемент:", queue.dequeue())

                except IndexError as e:

                    print(e)

            case "3":

                queue.display()

            case "4":

                break

            case \_:

                print("Неверный выбор, попробуйте снова.")

Результат выполнения работы представлен на рисунке 1.1.

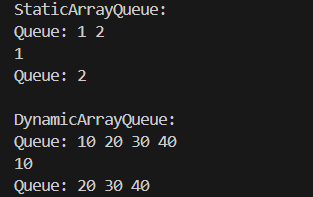


Рисунок 1.1 — Результат работы программы

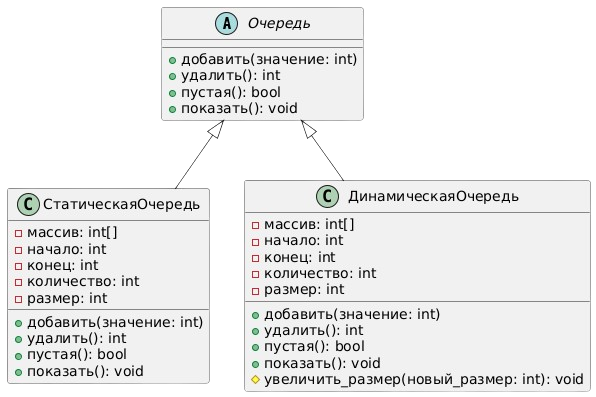


Рисунок 1.2 — UML диаграмма классов

## Вывод

В ходе работы был создан абстрактный класс «Очередь» и два дочерних класса: очередь на основе статического массива и очередь на основе динамического массива. Это позволило реализовать общие методы для работы с очередью, такие как добавление, удаление и вывод состояния очереди.

Использование абстрактного класса показало его пользу, так как позволило унифицировать взаимодействие с объектами разных типов через единый интерфейс. Дочерние классы реализовали свои особенности хранения данных, но интерфейс для работы с ними остался одинаковым, что продемонстрировало преимущество полиморфизма.