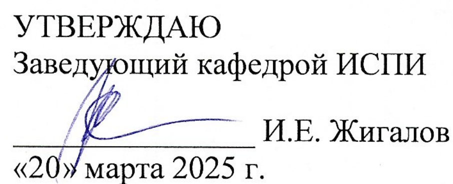
Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

УТВЕРЖДАЮ
Заведующий кафедрой ИСПИ



_________________ И.Е. Жигалов
«20» марта 2025 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ЛАБОРАТОРНЫМ РАБОТАМ
УЧЕБНОЙ ДИСЦИПЛИНЫ
ПРОФЕССИОНАЛЬНОЙ ПОДГОТОВКИ

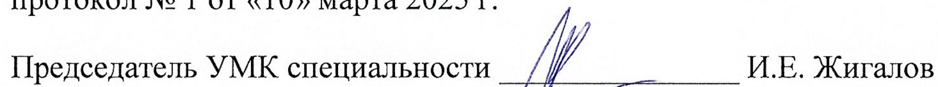
«ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»

09.02.09 Веб-разработка
Разработчик веб приложений

Владимир, 2025

Методические указания к лабораторным работам учебной дисциплины профессиональной подготовки «Основы алгоритмизации и программирования» разработал старший преподаватель кафедры ИСПИ Шамышева О.Н.

Методические указания к лабораторным работам рассмотрены и одобрены на заседании УМК специальности 09.02.09 Веб-разработка
протокол № 1 от «10» марта 2025 г.



Председатель УМК специальности _________________ И.Е. Жигалов

Методические указания к лабораторным работам рассмотрены и одобрены на заседании кафедры ИСПИ
протокол № 7а от «12» марта 2025 г.

Методические указания к лабораторной работе № 1 по теме: "Интерфейс среды разработки Java"

1. Цель выполнения работы
Целью данной лабораторной работы является ознакомление студентов с интерфейсом среды разработки для языка Java, освоение основных инструментов и функций, необходимых для создания, редактирования, компиляции и запуска программ. В процессе выполнения работы студенты научатся настраивать среду разработки, создавать проекты, писать и отлаживать простейшие программы на Java.

2. Теоретический материал
Среда разработки (Integrated Development Environment, IDE) — это программное обеспечение, которое предоставляет комплекс инструментов для разработки, отладки и тестирования программ. Для языка Java наиболее популярными средами разработки являются IntelliJ IDEA, Eclipse и NetBeans.

Основные компоненты интерфейса среды разработки:

- **Редактор кода**: область для написания и редактирования исходного кода.
- **Проект**: структура, содержащая все файлы, необходимые для разработки программы (исходные коды, библиотеки, конфигурации).
- **Панель инструментов**: содержит кнопки для выполнения часто используемых действий (запуск программы, отладка, сохранение).
- **Окно вывода**: отображает результаты компиляции и выполнения программы, а также сообщения об ошибках.
- **Навигатор проекта**: позволяет просматривать и управлять файлами проекта.
- **Отладчик**: инструмент для поиска и исправления ошибок в программе.

3. Порядок выполнения работы

Шаг 1. Установка среды разработки

1. Скачайте и установите одну из сред разработки (IntelliJ IDEA, Eclipse или NetBeans).
2. Запустите среду разработки и ознакомьтесь с интерфейсом.

Шаг 2. Создание нового проекта

1. В главном меню выберите "File" -> "New Project".
2. Укажите тип проекта (Java) и нажмите "Next".
3. Задайте имя проекта и выберите папку для его сохранения.
4. Нажмите "Finish".

Шаг 3. Создание и редактирование класса

1. В навигаторе проекта щелкните правой кнопкой мыши на папке "src" и выберите "New" -> "Java Class".
2. Введите имя класса (например, Main) и нажмите "OK".
3. В редакторе кода напишите простую программу:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
```

```
    }
}
```

Шаг 4. Компиляция и запуск программы

1. Нажмите на кнопку "Run" (или сочетание клавиш Shift + F10 в IntelliJ IDEA).
2. Убедитесь, что в окне вывода появилось сообщение "Hello, World!".

Шаг 5. Изучение отладчика

1. Установите точку останова (breakpoint) на строке с System.out.println, щелкнув на левом поле редактора кода.
2. Запустите программу в режиме отладки (кнопка "Debug" или Shift + F9).
3. Проследите за выполнением программы по шагам, используя кнопки "Step Over" и "Step Into".

4. Варианты заданий

1. Создайте программу, которая выводит ваше имя и фамилию.
2. Напишите программу, которая вычисляет сумму двух чисел, введенных пользователем.
3. Создайте программу, которая выводит таблицу умножения для числа 5.
4. Напишите программу, которая проверяет, является ли введенное число четным или нечетным.
5. Создайте программу, которая выводит все числа от 1 до 100, кратные 3.

5. Содержание отчета
Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты интерфейса среды разработки с выполненной программой.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое среда разработки (IDE)? Какие основные компоненты она включает?
2. Как создать новый проект в IntelliJ IDEA (или другой среде разработки)?
3. Какие функции выполняет отладчик в среде разработки?
4. Как установить точку останова в программе?
5. Какие основные отличия между компиляцией и запуском программы?

Методические указания к лабораторной работе № 2 по теме: "Вычисления. Составление математических выражений"

1. Цель выполнения работы

Целью данной лабораторной работы является освоение базовых принципов работы с математическими выражениями в языке Java. Студенты научатся использовать арифметические операции, математические функции, а также правильно составлять и вычислять выражения с учетом приоритета операций.

2. Теоретический материал

В языке Java поддерживаются стандартные арифметические операции, такие как сложение (+), вычитание (-), умножение (*), деление (/) и взятие остатка от деления (%). Приоритет операций соответствует математическим правилам: сначала выполняются умножение и деление, затем сложение и вычитание. Для изменения порядка выполнения операций используются круглые скобки.

Кроме того, в Java предоставляется класс Math, который содержит множество полезных математических функций, таких как:

- Math.sqrt(x) — квадратный корень из x;
- Math.pow(x, y) — возведение x в степень y;
- Math.abs(x) — модуль числа x;
- Math.sin(x), Math.cos(x), Math.tan(x) — тригонометрические функции;
- Math.log(x) — натуральный логарифм числа x;
- Math.random() — генерация случайного числа в диапазоне [0, 1).

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Написание простого математического выражения

1. В методе main напишите код для вычисления суммы двух чисел:

```
public class Main {
    public static void main(String[] args) {
        int a = 5;
        int b = 3;
        int sum = a + b;
        System.out.println("Сумма: " + sum);
    }
}
```

2. Запустите программу и убедитесь, что результат выводится корректно.

Шаг 3. Использование класса Math

1. Добавьте в программу вычисление квадратного корня из числа:

```
double x = 16;
double sqrt = Math.sqrt(x);
System.out.println("Квадратный корень из " + x + " равен " + sqrt);
```

2. Запустите программу и проверьте результат.

Шаг 4. Составление сложного выражения

1. Напишите выражение, включающее несколько операций:
```
double result = (Math.pow(2, 3) + Math.sqrt(25)) / 2;
System.out.println("Результат: " + result);
```
2. Запустите программу и проанализируйте результат.

Шаг 5. Работа с пользовательским вводом

1. Используйте класс Scanner для ввода данных с клавиатуры:
```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Введите число: ");
        double num = scanner.nextDouble();
        System.out.println("Квадрат числа: " + Math.pow(num, 2));
    }
}
```
2. Запустите программу и проверьте ее работу.

4. Варианты заданий

1. Напишите программу, которая вычисляет площадь круга по заданному радиусу.
2. Создайте программу для решения квадратного уравнения вида $ax^2+bx+c=0$.
3. Напишите программу, которая вычисляет гипотенузу прямоугольного треугольника по двум катетам.
4. Реализуйте программу, которая вычисляет факториал числа, введенного пользователем.
5. Напишите программу, которая генерирует случайное число и выводит его квадратный корень.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Какие арифметические операции поддерживаются в Java?
2. Как изменить порядок выполнения операций в выражении?
3. Какие функции предоставляет класс Math?
4. Как в Java реализовать ввод данных с клавиатуры?
5. Как вычислить квадратный корень числа в Java?

Методические указания к лабораторной работе № 3 по теме: "Операторы ветвления"

1. Цель выполнения работы
Целью данной лабораторной работы является изучение и практическое применение операторов ветвления в языке Java. Студенты научатся использовать условные конструкции (if, else, else if, switch) для управления потоком выполнения программы в зависимости от заданных условий.

2. Теоретический материал
Операторы ветвления позволяют программе принимать решения на основе определенных условий. В Java используются следующие конструкции:

1. **Оператор if**:

```
if (условие) {
    // Блок кода, выполняемый, если условие истинно
}
```

2. **Оператор if-else**:

```
if (условие) {
    // Блок кода, выполняемый, если условие истинно
} else {
    // Блок кода, выполняемый, если условие ложно
}
```

3. **Оператор else if**:

```
if (условие1) {
    // Блок кода, если условие1 истинно
} else if (условие2) {
    // Блок кода, если условие2 истинно
} else {
    // Блок кода, если все условия ложны
}
```

4. **Оператор switch**:

```
switch (выражение) {
    case значение1:
        // Блок кода для значение1
        break;
    case значение2:
        // Блок кода для значение2
        break;
    default:
        // Блок кода, если ни одно значение не совпало
}
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Использование оператора if

1. Напишите программу, которая проверяет, является ли число положительным:

```
public class Main {
```

```
    public static void main(String[] args) {
        int number = 10;
        if (number > 0) {
            System.out.println("Число положительное.");
        }
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 3. Использование оператора if-else

1. Добавьте в программу проверку на отрицательное число:

```
if (number > 0) {
    System.out.println("Число положительное.");
} else {
    System.out.println("Число отрицательное или равно нулю.");
}
```

2. Запустите программу и проверьте результат.

Шаг 4. Использование оператора else if

1. Расширьте программу для проверки нескольких условий:

```
if (number > 0) {
    System.out.println("Число положительное.");
} else if (number < 0) {
    System.out.println("Число отрицательное.");
} else {
    System.out.println("Число равно нулю.");
}
```

2. Запустите программу и проверьте результат.

Шаг 5. Использование оператора switch

1. Напишите программу, которая определяет день недели по его номеру:

```
int day = 3;
switch (day) {
    case 1:
        System.out.println("Понедельник");
        break;
    case 2:
        System.out.println("Вторник");
        break;
    case 3:
        System.out.println("Среда");
        break;
    default:
        System.out.println("Неверный номер дня.");
}
```

2. Запустите программу и проверьте результат.

4. Варианты заданий

1. Напишите программу, которая проверяет, является ли число четным или нечетным.
2. Создайте программу, которая определяет, является ли год високосным.
3. Напишите программу, которая выводит название времени года по номеру месяца.
4. Реализуйте программу, которая вычисляет корни квадратного уравнения в зависимости от дискриминанта.

5. Напишите программу, которая определяет, является ли введенный символ гласной или согласной буквой.

5. Содержание отчета
Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Какие операторы ветвления поддерживаются в Java?
2. В чем разница между операторами if и switch?
3. Как работает оператор else if?
4. Что происходит, если в операторе switch не использовать break?
5. Как можно заменить оператор switch на конструкцию if-else?

Методические указания к лабораторной работе № 4 по теме: "Операторы цикла"

1. Цель выполнения работы
Целью данной лабораторной работы является изучение и практическое применение операторов цикла в языке Java. Студенты научатся использовать циклы (for, while, do-while) для организации повторяющихся действий в программе, а также освоят работу с вложенными циклами и управляющими операторами (break, continue).

2. Теоретический материал
Операторы цикла позволяют выполнять определенный блок кода многократно. В Java используются следующие виды циклов:

1. **Цикл for**:

```
for (инициализация; условие; итерация) {
    // Блок кода, выполняемый в цикле
}
```

Пример:

```
for (int i = 0; i < 5; i++) {
    System.out.println("i = " + i);
}
```

2. **Цикл while**:

```
while (условие) {
    // Блок кода, выполняемый в цикле
}
```

Пример:

```
int i = 0;
while (i < 5) {
```

```
    System.out.println("i = " + i);
    i++;
}
```

3. **Цикл do-while**:

```
do {
    // Блок кода, выполняемый в цикле
} while (условие);
```

Пример:

```
int i = 0;
do {
    System.out.println("i = " + i);
    i++;
} while (i < 5);
```

4. **Управляющие операторы**:
   - break — завершает выполнение цикла.
   - continue — пропускает текущую итерацию цикла и переходит к следующей.

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Использование цикла for

1. Напишите программу, которая выводит числа от 1 до 10:

```
public class Main {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 3. Использование цикла while

1. Напишите программу, которая выводит числа от 1 до 10 с использованием цикла while:

```
int i = 1;
while (i <= 10) {
    System.out.println(i);
    i++;
}
```

2. Запустите программу и проверьте результат.

Шаг 4. Использование цикла do-while

1. Напишите программу, которая выводит числа от 1 до 10 с использованием цикла do-while:

```
int i = 1;
do {
    System.out.println(i);
```

```
    i++;
} while (i <= 10);
```
2. Запустите программу и проверьте результат.

Шаг 5. Использование вложенных циклов

1. Напишите программу, которая выводит таблицу умножения:
```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.print(i * j + "\t");
    }
    System.out.println();
}
```
2. Запустите программу и проверьте результат.

Шаг 6. Использование операторов break и continue

1. Напишите программу, которая выводит числа от 1 до 10, пропуская число 5:
```
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        continue;
    }
    System.out.println(i);
}
```
2. Напишите программу, которая завершает цикл при достижении числа 5:
```
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        break;
    }
    System.out.println(i);
}
```

4. Варианты заданий

1. Напишите программу, которая выводит все четные числа от 1 до 20.
2. Создайте программу, которая вычисляет сумму чисел от 1 до 100.
3. Напишите программу, которая выводит числа от 10 до 1 в обратном порядке.
4. Реализуйте программу, которая вычисляет факториал числа, введенного пользователем.
5. Напишите программу, которая выводит первые 10 чисел Фибоначчи.
6. Создайте программу, которая проверяет, является ли введенное число простым.
7. Напишите программу, которая выводит все делители числа, введенного пользователем.
8. Реализуйте программу, которая выводит треугольник из звездочек:
```
*
**
***
****
*****
```
9. Напишите программу, которая выводит ромб из звездочек.
10. Создайте программу, которая вычисляет сумму цифр числа, введенного пользователем.
11. Напишите программу, которая выводит таблицу степеней числа 2 от 0 до 10.
12. Реализуйте программу, которая находит наибольший общий делитель (НОД) двух чисел.

13. Напишите программу, которая выводит все трехзначные числа, сумма цифр которых равна 15.
14. Создайте программу, которая выводит все числа, кратные 3 и 5, в диапазоне от 1 до 100.
15. Напишите программу, которая выводит все простые числа в диапазоне от 1 до 100.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Какие операторы цикла поддерживаются в Java?
2. В чем разница между циклами while и do-while?
3. Как работает оператор break?
4. Как работает оператор continue?
5. Как можно использовать вложенные циклы?

Методические указания к лабораторной работе № 5 по теме: "Методы в Java"

1. Цель выполнения работы
Целью данной лабораторной работы является изучение и практическое применение методов в языке Java. Студенты научатся создавать методы, передавать в них параметры, возвращать значения, а также использовать методы для структурирования и упрощения кода.

2. Теоретический материал
Метод в Java — это блок кода, который выполняет определенную задачу и может быть вызван из других частей программы. Методы позволяют избежать дублирования кода, улучшают читаемость и поддерживаемость программы.

Основные элементы метода:

1. **Модификатор доступа**: определяет видимость метода (например, public, private).
2. **Тип возвращаемого значения**: указывает, какой тип данных возвращает метод (например, int, String, void).
3. **Имя метода**: должно быть уникальным и описывать его назначение.
4. **Параметры**: данные, которые передаются в метод для обработки.
5. **Тело метода**: блок кода, который выполняется при вызове метода.

Пример объявления метода:

```
public int sum(int a, int b) {
   return a + b;
}
```

Методы могут быть:

- **Статическими** (static): вызываются без создания объекта класса.
- **Нестатическими**: вызываются через объект класса.

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Создание простого метода

1. Напишите метод, который возвращает сумму двух чисел:

```
public class Main {
   public static int sum(int a, int b) {
      return a + b;
   }

   public static void main(String[] args) {
      int result = sum(5, 3);
      System.out.println("Сумма: " + result);
```

```
    }
}
```
2. Запустите программу и проверьте результат.

Шаг 3. Метод без возвращаемого значения

1. Напишите метод, который выводит приветствие:
```
public static void greet(String name) {
    System.out.println("Привет, " + name + "!");
}

public static void main(String[] args) {
    greet("Анна");
}
```
2. Запустите программу и проверьте результат.

Шаг 4. Использование статических и нестатических методов

1. Создайте нестатический метод для вычисления произведения двух чисел:
```
public class Main {
    public int multiply(int a, int b) {
        return a * b;
    }

    public static void main(String[] args) {
        Main obj = new Main();
        int result = obj.multiply(4, 5);
        System.out.println("Произведение: " + result);
    }
}
```
2. Запустите программу и проверьте результат.

Шаг 5. Перегрузка методов

1. Создайте перегруженные методы для сложения чисел:
```
public static int sum(int a, int b) {
    return a + b;
}

public static double sum(double a, double b) {
    return a + b;
}

public static void main(String[] args) {
    System.out.println("Сумма целых чисел: " + sum(5, 3));
    System.out.println("Сумма дробных чисел: " + sum(2.5, 3.7));
}
```
2. Запустите программу и проверьте результат.

4. Варианты заданий

1. Напишите метод, который возвращает наибольшее из двух чисел.
2. Создайте метод, который проверяет, является ли число четным.
3. Напишите метод, который вычисляет факториал числа.
4. Реализуйте метод, который возвращает сумму цифр числа.
5. Напишите метод, который проверяет, является ли строка палиндромом.

6. Создайте метод, который возвращает массив из первых 10 чисел Фибоначчи.
7. Напишите метод, который вычисляет площадь круга по радиусу.
8. Реализуйте метод, который находит наименьший элемент в массиве.
9. Напишите метод, который сортирует массив чисел по возрастанию.
10. Создайте метод, который возвращает количество гласных букв в строке.
11. Напишите метод, который преобразует температуру из градусов Цельсия в Фаренгейты.
12. Реализуйте метод, который вычисляет НОД двух чисел.
13. Напишите метод, который проверяет, является ли число простым.
14. Создайте метод, который возвращает разность между максимальным и минимальным элементами массива.
15. Напишите метод, который выводит таблицу умножения для заданного числа.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое метод в Java?
2. Какие элементы включает объявление метода?
3. В чем разница между статическими и нестатическими методами?
4. Что такое перегрузка методов?
5. Как передаются параметры в методы: по значению или по ссылке?

Методические указания к лабораторной работе № 6 по теме: "Рекурсивные вычисления"

1. Цель выполнения работы
Целью данной лабораторной работы является изучение и практическое применение рекурсии в языке Java. Студенты научатся разрабатывать рекурсивные методы, решать задачи с использованием рекурсии, а также анализировать преимущества и недостатки рекурсивных подходов.

2. Теоретический материал
Рекурсия — это процесс, при котором метод вызывает сам себя для решения задачи. Рекурсивный метод состоит из двух частей:

1. **Базовый случай** — условие завершения рекурсии, при котором метод возвращает результат без вызова самого себя.
2. **Рекурсивный случай** — вызов метода с измененными параметрами, приближающимися к базовому случаю.

Пример рекурсивного метода для вычисления факториала числа:

```
public static int factorial(int n) {
    if (n == 0 || n == 1) { // Базовый случай
        return 1;
    } else { // Рекурсивный случай
        return n * factorial(n - 1);
    }
}
```

Преимущества рекурсии:

- Упрощает решение задач, которые можно разбить на подзадачи.
- Улучшает читаемость кода для задач, имеющих рекурсивную природу.

Недостатки рекурсии:

- Может привести к переполнению стека вызовов при большой глубине рекурсии.
- Менее эффективна по сравнению с итеративными решениями из-за накладных расходов на вызовы методов.

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Реализация рекурсивного метода для вычисления факториала

1. Напишите рекурсивный метод для вычисления факториала числа:

```
public class Main {
    public static int factorial(int n) {
        if (n == 0 || n == 1) {
```

```
            return 1;
        } else {
            return n * factorial(n - 1);
        }
    }

    public static void main(String[] args) {
        int result = factorial(5);
        System.out.println("Факториал: " + result);
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 3. Реализация рекурсивного метода для вычисления чисел Фибоначчи

1. Напишите рекурсивный метод для вычисления n-го числа Фибоначчи:

```
public static int fibonacci(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}

public static void main(String[] args) {
    int result = fibonacci(6);
    System.out.println("Число Фибоначчи: " + result);
}
```

2. Запустите программу и проверьте результат.

Шаг 4. Реализация рекурсивного метода для вычисления суммы цифр числа

1. Напишите рекурсивный метод для вычисления суммы цифр числа:

```
public static int sumOfDigits(int n) {
    if (n == 0) {
        return 0;
    } else {
        return n % 10 + sumOfDigits(n / 10);
    }
}

public static void main(String[] args) {
    int result = sumOfDigits(1234);
    System.out.println("Сумма цифр: " + result);
}
```

2. Запустите программу и проверьте результат.

Шаг 5. Реализация рекурсивного метода для проверки палиндрома

1. Напишите рекурсивный метод для проверки, является ли строка палиндромом:

```
public static boolean isPalindrome(String s, int left, int right) {
    if (left >= right) {
        return true;
    } else if (s.charAt(left) != s.charAt(right)) {
        return false;
    } else {
```

```
        return isPalindrome(s, left + 1, right - 1);
    }
}

public static void main(String[] args) {
    String str = "мадам";
    boolean result = isPalindrome(str, 0, str.length() - 1);
    System.out.println("Является палиндромом: " + result);
}
```

2. Запустите программу и проверьте результат.

4. Варианты заданий

1. Напишите рекурсивный метод для вычисления степени числа.
2. Реализуйте рекурсивный метод для нахождения НОД двух чисел.
3. Напишите рекурсивный метод для вычисления суммы чисел от 1 до n.
4. Создайте рекурсивный метод для вычисления суммы элементов массива.
5. Напишите рекурсивный метод для поиска максимального элемента в массиве.
6. Реализуйте рекурсивный метод для вычисления длины строки.
7. Напишите рекурсивный метод для перевода числа из десятичной системы в двоичную.
8. Создайте рекурсивный метод для вычисления суммы четных чисел в диапазоне от 1 до n.
9. Напишите рекурсивный метод для проверки, является ли число простым.
10. Реализуйте рекурсивный метод для вычисления числа сочетаний (C(n, k)).
11. Напишите рекурсивный метод для вычисления числа размещений (A(n, k)).
12. Создайте рекурсивный метод для вычисления суммы квадратов чисел от 1 до n.
13. Напишите рекурсивный метод для вычисления произведения цифр числа.
14. Реализуйте рекурсивный метод для поиска минимального элемента в массиве.
15. Напишите рекурсивный метод для вычисления суммы ряда 1 + 1/2 + 1/3 + ... + 1/n.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое рекурсия?
2. Какие части включает рекурсивный метод?
3. В чем разница между рекурсивным и итеративным подходами?
4. Какие преимущества и недостатки имеет рекурсия?
5. Как избежать переполнения стека при использовании рекурсии?

Методические указания к лабораторной работе № 7 по теме: "Одномерные массивы"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение одномерных массивов в языке Java. Студенты научатся создавать, инициализировать и работать с массивами, а также решать задачи, связанные с обработкой данных, хранящихся в массивах.

2. Теоретический материал

Массив — это структура данных, которая хранит набор элементов одного типа. В Java массивы имеют фиксированную длину, которая задается при создании массива.

Основные характеристики массивов:

- **Тип массива**: определяет тип данных, которые могут храниться в массиве (например, int, double, String).
- **Длина массива**: количество элементов, которые может хранить массив. Длина задается при создании массива и не может быть изменена.
- **Индексация**: элементы массива нумеруются с 0 до length - 1.

Пример создания и инициализации массива:

```
int[] numbers = new int[5]; // Создание массива из 5 элементов
numbers[0] = 10; // Инициализация первого элемента
```

Или с использованием сокращенного синтаксиса:

```
int[] numbers = {10, 20, 30, 40, 50}; // Создание и инициализация массива
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Создание и инициализация массива

1. Создайте массив целых чисел и инициализируйте его значениями:

```
public class Main {
    public static void main(String[] args) {
        int[] numbers = {5, 10, 15, 20, 25};
        for (int i = 0; i < numbers.length; i++) {
            System.out.println("Элемент " + i + ": " + numbers[i]);
        }
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 3. Поиск максимального элемента в массиве

1. Напишите программу для поиска максимального элемента в массиве:

```
public class Main {
    public static void main(String[] args) {
        int[] numbers = {5, 10, 15, 20, 25};
        int max = numbers[0];
        for (int i = 1; i < numbers.length; i++) {
            if (numbers[i] > max) {
                max = numbers[i];
            }
        }
        System.out.println("Максимальный элемент: " + max);
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 4. Сумма элементов массива

1. Напишите программу для вычисления суммы элементов массива:

```
public class Main {
    public static void main(String[] args) {
        int[] numbers = {5, 10, 15, 20, 25};
        int sum = 0;
        for (int i = 0; i < numbers.length; i++) {
            sum += numbers[i];
        }
        System.out.println("Сумма элементов: " + sum);
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 5. Сортировка массива

1. Напишите программу для сортировки массива по возрастанию:

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] numbers = {25, 10, 15, 5, 20};
        Arrays.sort(numbers);
        System.out.println("Отсортированный массив: " + Arrays.toString(numbers));
    }
}
```

2. Запустите программу и проверьте результат.

4. Варианты заданий

1. Напишите программу для поиска минимального элемента в массиве.
2. Создайте программу для вычисления среднего арифметического элементов массива.
3. Напишите программу для поиска индекса заданного элемента в массиве.
4. Реализуйте программу для подсчета количества четных чисел в массиве.
5. Напишите программу для реверсирования массива (перестановки элементов в обратном порядке).
6. Создайте программу для поиска всех отрицательных чисел в массиве.
7. Напишите программу для умножения всех элементов массива на заданное число.
8. Реализуйте программу для поиска второго по величине элемента в массиве.

9. Напишите программу для проверки, является ли массив симметричным.
10. Создайте программу для объединения двух массивов в один.
11. Напишите программу для поиска всех уникальных элементов в массиве.
12. Реализуйте программу для сортировки массива по убыванию.
13. Напишите программу для поиска суммы элементов массива с четными индексами.
14. Создайте программу для поиска произведения элементов массива.
15. Напишите программу для поиска количества элементов массива, которые больше среднего арифметического.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое массив в Java?
2. Как создать и инициализировать массив?
3. Как получить длину массива?
4. Как найти максимальный элемент в массиве?
5. Как отсортировать массив по возрастанию?

Методические указания к лабораторной работе № 8 по теме: "Двумерные массивы"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение двумерных массивов в языке Java. Студенты научатся создавать, инициализировать и работать с двумерными массивами, а также решать задачи, связанные с обработкой данных, хранящихся в матрицах.

2. Теоретический материал

Двумерный массив (матрица) — это массив массивов, который можно представить в виде таблицы с строками и столбцами. В Java двумерные массивы создаются с использованием двух пар квадратных скобок [][].

Основные характеристики двумерных массивов:

- **Тип массива**: определяет тип данных, которые могут храниться в массиве (например, int, double, String).
- **Размерность**: задается количеством строк и столбцов (например, int[][] matrix = new int[3][4] создает матрицу 3x4).
- **Индексация**: элементы массива нумеруются с 0 до length - 1 по строкам и столбцам.

Пример создания и инициализации двумерного массива:

```
int[][] matrix = new int[3][3]; // Создание матрицы 3x3
matrix[0][0] = 1; // Инициализация элемента в первой строке и первом столбце
```

Или с использованием сокращенного синтаксиса:

```
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Создание и инициализация двумерного массива

1. Создайте двумерный массив и инициализируйте его значениями:

```
public class Main {
    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        for (int i = 0; i < matrix.length; i++) {
```

```
            for (int j = 0; j < matrix[i].length; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 3. Поиск максимального элемента в матрице

1. Напишите программу для поиска максимального элемента в двумерном массиве:

```
public class Main {
    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        int max = matrix[0][0];
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                if (matrix[i][j] > max) {
                    max = matrix[i][j];
                }
            }
        }
        System.out.println("Максимальный элемент: " + max);
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 4. Сумма элементов матрицы

1. Напишите программу для вычисления суммы всех элементов двумерного массива:

```
public class Main {
    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        int sum = 0;
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                sum += matrix[i][j];
            }
        }
        System.out.println("Сумма элементов: " + sum);
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 5. Транспонирование матрицы

1. Напишите программу для транспонирования матрицы (замены строк на столбцы):

```
public class Main {
    public static void main(String[] args) {
```

```
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        int[][] transposed = new int[matrix[0].length][matrix.length];
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                transposed[j][i] = matrix[i][j];
            }
        }
        for (int i = 0; i < transposed.length; i++) {
            for (int j = 0; j < transposed[i].length; j++) {
                System.out.print(transposed[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

2. Запустите программу и проверьте результат.

4. Варианты заданий

1. Напишите программу для поиска минимального элемента в двумерном массиве.
2. Создайте программу для вычисления суммы элементов каждой строки матрицы.
3. Напишите программу для вычисления суммы элементов каждого столбца матрицы.
4. Реализуйте программу для поиска суммы элементов главной диагонали матрицы.
5. Напишите программу для поиска суммы элементов побочной диагонали матрицы.
6. Создайте программу для проверки, является ли матрица симметричной.
7. Напишите программу для умножения двух матриц.
8. Реализуйте программу для поиска количества отрицательных элементов в матрице.
9. Напишите программу для замены всех элементов матрицы на их квадраты.
10. Создайте программу для поиска среднего арифметического всех элементов матрицы.
11. Напишите программу для поиска индексов максимального элемента в матрице.
12. Реализуйте программу для поиска индексов минимального элемента в матрице.
13. Напишите программу для поворота матрицы на 90 градусов по часовой стрелке.
14. Создайте программу для поиска количества нулевых элементов в матрице.
15. Напишите программу для проверки, является ли матрица единичной (диагональная матрица с единицами на главной диагонали).

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое двумерный массив в Java?

2. Как создать и инициализировать двумерный массив?
3. Как получить количество строк и столбцов в двумерном массиве?
4. Как найти сумму элементов главной диагонали матрицы?
5. Как транспонировать матрицу?

Методические указания к лабораторной работе № 9 по теме: "Строки"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение строк в языке Java. Студенты научатся создавать и обрабатывать строки, использовать методы класса String, а также решать задачи, связанные с обработкой текстовых данных.

2. Теоретический материал

Строка в Java — это последовательность символов, представленная классом String. Строки являются неизменяемыми (immutable), что означает, что после создания строки её содержимое нельзя изменить.

Основные методы класса String:

- length() — возвращает длину строки.
- charAt(int index) — возвращает символ по указанному индексу.
- substring(int beginIndex, int endIndex) — возвращает подстроку.
- equals(Object obj) — сравнивает строки на равенство.
- toLowerCase() и toUpperCase() — преобразуют строку в нижний и верхний регистр.
- trim() — удаляет пробелы в начале и конце строки.
- replace(char oldChar, char newChar) — заменяет символы в строке.
- split(String regex) — разбивает строку на массив подстрок по заданному разделителю.

Пример работы со строками:

```
String str = "Hello, World!";
System.out.println(str.length()); // Вывод: 13
System.out.println(str.charAt(0)); // Вывод: H
System.out.println(str.substring(7)); // Вывод: World!
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Работа с основными методами строк

1. Создайте строку и примените к ней основные методы:

```
public class Main {
    public static void main(String[] args) {
        String str = "  Hello, World!  ";
        System.out.println("Длина строки: " + str.length());
        System.out.println("Символ на позиции 1: " + str.charAt(1));
        System.out.println("Подстрока: " + str.substring(7, 12));
        System.out.println("Верхний регистр: " + str.toUpperCase());
```

```
        System.out.println("Без пробелов: " + str.trim());
    }
}
```
2. Запустите программу и проверьте результат.

Шаг 3. Сравнение строк

1. Напишите программу для сравнения строк:
```
public class Main {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "hello";
        System.out.println("Сравнение с учетом регистра: " + str1.equals(str2));
        System.out.println("Сравнение без учета регистра: " + str1.equalsIgnoreCase(str2));
    }
}
```
2. Запустите программу и проверьте результат.

Шаг 4. Разделение строки

1. Напишите программу для разделения строки на части:
```
public class Main {
    public static void main(String[] args) {
        String str = "Java,Python,C++,JavaScript";
        String[] languages = str.split(",");
        for (String lang : languages) {
            System.out.println(lang);
        }
    }
}
```
2. Запустите программу и проверьте результат.

Шаг 5. Замена символов в строке

1. Напишите программу для замены символов в строке:
```
public class Main {
    public static void main(String[] args) {
        String str = "Hello, World!";
        String newStr = str.replace('o', '0');
        System.out.println("Новая строка: " + newStr);
    }
}
```
2. Запустите программу и проверьте результат.

4. Варианты заданий

1. Напишите программу для подсчета количества гласных букв в строке.
2. Создайте программу для проверки, является ли строка палиндромом.
3. Напишите программу для удаления всех пробелов из строки.
4. Реализуйте программу для поиска количества вхождений заданного символа в строке.
5. Напишите программу для преобразования строки в обратном порядке.
6. Создайте программу для поиска самого длинного слова в строке.
7. Напишите программу для замены всех цифр в строке на символ *.
8. Реализуйте программу для подсчета количества слов в строке.
9. Напишите программу для проверки, начинается ли строка с заданного префикса.

10. Создайте программу для поиска индекса первого вхождения подстроки в строке.
11. Напишите программу для объединения двух строк через пробел.
12. Реализуйте программу для поиска всех цифр в строке и их суммы.
13. Напишите программу для преобразования строки в массив символов.
14. Создайте программу для поиска количества предложений в тексте.
15. Напишите программу для замены всех гласных букв в строке на символ #.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое строка в Java?
2. Какие основные методы класса String вы знаете?
3. Как сравнить две строки на равенство?
4. Как разбить строку на массив подстрок?
5. Как заменить символы в строке?

Методические указания к лабораторной работе № 10.1 по теме: "Классы и объекты"

1. Цель выполнения работы
Целью данной лабораторной работы является изучение и практическое применение основных принципов объектно-ориентированного программирования (ООП) в Java. Студенты научатся создавать классы, объекты, работать с полями и методами, а также понимать принципы инкапсуляции.

2. Теоретический материал
Класс в Java — это шаблон или описание для создания объектов. Класс определяет свойства (поля) и поведение (методы) объекта. Объект — это экземпляр класса, который обладает состоянием (значения полей) и поведением (методы).

Основные элементы класса:

1. **Поля (переменные класса)**: хранят состояние объекта.
2. **Методы**: определяют поведение объекта.
3. **Конструкторы**: специальные методы для инициализации объекта при его создании.

Пример класса:

```
public class Dog {
    // Поля
    String name;
    int age;

    // Конструктор
    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Метод
    public void bark() {
        System.out.println(name + " лает!");
    }
}
```

Создание объекта:

```
Dog myDog = new Dog("Бобик", 3);
myDog.bark(); // Вывод: Бобик лает!
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Создание класса и объекта

1. Создайте класс Dog с полями name и age, конструктором и методом bark:

```
public class Dog {
    String name;
    int age;

    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void bark() {
        System.out.println(name + " лает!");
    }
}
```

2. В классе Main создайте объект класса Dog и вызовите метод bark:

```
public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog("Бобик", 3);
        myDog.bark();
    }
}
```

3. Запустите программу и проверьте результат.

Шаг 3. Добавление методов в класс

1. Добавьте в класс Dog метод getAge, который возвращает возраст собаки:

```
public int getAge() {
    return age;
}
```

2. В классе Main вызовите метод getAge и выведите результат:

```
System.out.println("Возраст собаки: " + myDog.getAge());
```

3. Запустите программу и проверьте результат.

Шаг 4. Использование инкапсуляции

1. Измените поля класса Dog на приватные (private) и добавьте методы getName, setName, setAge:

```
public class Dog {
    private String name;
    private int age;

    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }
```

```
    public void setAge(int age) {
        this.age = age;
    }

    public void bark() {
        System.out.println(name + " лает!");
    }
}
```

2. В классе Main используйте методы для изменения и получения значений полей:

```
myDog.setName("Шарик");
System.out.println("Новое имя собаки: " + myDog.getName());
```

3. Запустите программу и проверьте результат.

4. Варианты заданий

1. Создайте класс Car с полями model, year, color и методами для вывода информации о машине.
2. Реализуйте класс Book с полями title, author, year и методом для вывода информации о книге.
3. Напишите класс Student с полями name, age, grade и методом для вывода информации о студенте.
4. Создайте класс BankAccount с полями accountNumber, balance и методами для пополнения и снятия средств.
5. Реализуйте класс Rectangle с полями width, height и методами для вычисления площади и периметра.
6. Напишите класс Circle с полем radius и методами для вычисления площади и длины окружности.
7. Создайте класс Employee с полями name, position, salary и методом для вывода информации о сотруднике.
8. Реализуйте класс Product с полями name, price, quantity и методом для вычисления общей стоимости товара.
9. Напишите класс Animal с полями name, age, species и методом для вывода информации о животном.
10. Создайте класс Computer с полями brand, model, price и методом для вывода информации о компьютере.
11. Реализуйте класс Bank с полями name, address и методами для добавления и удаления счетов.
12. Напишите класс Triangle с полями side1, side2, side3 и методами для вычисления периметра и площади.
13. Создайте класс Person с полями firstName, lastName, age и методом для вывода полного имени.
14. Реализуйте класс MobilePhone с полями brand, model, price и методом для вывода информации о телефоне.
15. Напишите класс BankCard с полями cardNumber, balance, ownerName и методами для пополнения и снятия средств.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.

4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое класс в Java?
2. Что такое объект и как он создается?
3. Какие элементы включает класс?
4. Что такое конструктор и для чего он используется?
5. Что такое инкапсуляция и как она реализуется в Java?

Методические указания к лабораторной работе № 10.2 по теме: " Классы и объекты: Переопределение методов Object. Статические члены класса"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение переопределения методов класса Object, а также использование статических членов класса в Java. Студенты научатся переопределять методы toString, equals и hashCode, а также понимать принципы работы статических переменных и методов.

2. Теоретический материал

Переопределение методов класса Object

Класс Object является суперклассом для всех классов в Java. Он содержит методы, которые можно переопределить в пользовательских классах:

1. **toString()**: возвращает строковое представление объекта. По умолчанию возвращает имя класса и хэш-код объекта.
2. **equals(Object obj)**: сравнивает объекты на равенство. По умолчанию сравнивает ссылки на объекты.
3. **hashCode()**: возвращает хэш-код объекта. По умолчанию возвращает целочисленное значение, основанное на адресе объекта в памяти.

Пример переопределения методов:

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Person person = (Person) obj;
        return age == person.age && name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

Статические члены класса

Статические переменные и методы принадлежат классу, а не объекту. Они используются для хранения данных или выполнения операций, общих для всех объектов класса.

- **Статические переменные**: объявляются с ключевым словом static. Общая для всех объектов класса.
- **Статические методы**: могут вызываться без создания объекта класса. Не могут обращаться к нестатическим членам класса.

Пример использования статических членов:

```
public class Counter {
    private static int count = 0;

    public Counter() {
        count++;
    }

    public static int getCount() {
        return count;
    }
}
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Переопределение метода toString

1. Создайте класс Person с полями name и age.
2. Переопределите метод toString:

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }
}
```

3. В классе Main создайте объект класса Person и выведите его:

```
public class Main {
    public static void main(String[] args) {
        Person person = new Person("Иван", 25);
        System.out.println(person);
    }
}
```

4. Запустите программу и проверьте результат.

Шаг 3. Переопределение метода equals

1. Добавьте в класс Person переопределение метода equals:
```
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Person person = (Person) obj;
    return age == person.age && name.equals(person.name);
}
```
2. В классе Main сравните два объекта:
```
Person person1 = new Person("Иван", 25);
Person person2 = new Person("Иван", 25);
System.out.println("Объекты равны: " + person1.equals(person2));
```
3. Запустите программу и проверьте результат.

Шаг 4. Переопределение метода hashCode

1. Добавьте в класс Person переопределение метода hashCode:
```
@Override
public int hashCode() {
    return Objects.hash(name, age);
}
```
2. В классе Main выведите хэш-коды объектов:
```
System.out.println("Хэш-код person1: " + person1.hashCode());
System.out.println("Хэш-код person2: " + person2.hashCode());
```
3. Запустите программу и проверьте результат.

Шаг 5. Использование статических членов класса

1. Создайте класс Counter со статической переменной count и статическим методом getCount:
```
public class Counter {
    private static int count = 0;

    public Counter() {
        count++;
    }

    public static int getCount() {
        return count;
    }
}
```
2. В классе Main создайте несколько объектов и выведите значение счетчика:
```
new Counter();
new Counter();
System.out.println("Количество объектов: " + Counter.getCount());
```
3. Запустите программу и проверьте результат.

4. Варианты заданий

1. Создайте класс Book с полями title, author, year и переопределите методы toString, equals, hashCode.
2. Реализуйте класс Student с полями name, age, grade и переопределите методы toString, equals, hashCode.

3. Напишите класс Rectangle с полями width, height и переопределите методы toString, equals, hashCode.
4. Создайте класс BankAccount с полями accountNumber, balance и статической переменной для подсчета количества счетов.
5. Реализуйте класс Circle с полем radius и статическим методом для вычисления площади круга.
6. Напишите класс Employee с полями name, position, salary и переопределите методы toString, equals, hashCode.
7. Создайте класс Product с полями name, price, quantity и статическим методом для вычисления общей стоимости всех товаров.
8. Реализуйте класс Animal с полями name, age, species и переопределите методы toString, equals, hashCode.
9. Напишите класс Computer с полями brand, model, price и статической переменной для подсчета количества компьютеров.
10. Создайте класс Bank с полями name, address и статическим методом для вывода информации о банке.
11. Реализуйте класс Triangle с полями side1, side2, side3 и переопределите методы toString, equals, hashCode.
12. Напишите класс Person с полями firstName, lastName, age и статическим методом для вывода полного имени.
13. Создайте класс MobilePhone с полями brand, model, price и переопределите методы toString, equals, hashCode.
14. Реализуйте класс BankCard с полями cardNumber, balance, ownerName и статическим методом для проверки валидности номера карты.
15. Напишите класс Car с полями model, year, color и статической переменной для подсчета количества машин.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Какие методы класса Object можно переопределить?
2. Для чего используется метод toString?
3. Как работает метод equals по умолчанию?
4. Что такое статические переменные и методы?
5. Могут ли статические методы обращаться к нестатическим членам класса?

Методические указания к лабораторной работе № 11.1 по теме: "Наследование"

1. Цель выполнения работы
Целью данной лабораторной работы является изучение и практическое применение принципа наследования в языке Java. Студенты научатся создавать иерархии классов, использовать ключевое слово extends, переопределять методы и понимать принципы полиморфизма.

2. Теоретический материал
Наследование — это механизм, позволяющий создавать новый класс на основе существующего (родительского) класса. Новый класс (подкласс) наследует поля и методы родительского класса (суперкласса) и может добавлять собственные поля и методы или изменять унаследованные.

Основные принципы наследования:

1. **Ключевое слово extends**: используется для указания родительского класса.
2. **Переопределение методов**: подкласс может переопределить методы суперкласса.
3. **Ключевое слово super**: используется для вызова конструктора или методов суперкласса.
4. **Полиморфизм**: возможность использовать объект подкласса как объект суперкласса.

Пример наследования:

```
class Animal {
    String name;

    public Animal(String name) {
        this.name = name;
    }

    public void makeSound() {
        System.out.println("Звук животного");
    }
}

class Dog extends Animal {
    public Dog(String name) {
        super(name); // Вызов конструктора суперкласса
    }

    @Override
    public void makeSound() {
        System.out.println(name + " лает");
    }
}
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.

3. Создайте класс Main в папке src.

Шаг 2. Создание базового класса

1. Создайте класс Animal с полем name и методом makeSound:

```
class Animal {
    String name;

    public Animal(String name) {
        this.name = name;
    }

    public void makeSound() {
        System.out.println("Звук животного");
    }
}
```

Шаг 3. Создание подкласса

1. Создайте класс Dog, который наследует класс Animal, и переопределите метод makeSound:

```
class Dog extends Animal {
    public Dog(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(name + " лает");
    }
}
```

Шаг 4. Использование наследования

1. В классе Main создайте объекты классов Animal и Dog и вызовите метод makeSound:

```
public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal("Животное");
        Dog dog = new Dog("Бобик");

        animal.makeSound(); // Вывод: Звук животного
        dog.makeSound();    // Вывод: Бобик лает
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 5. Использование полиморфизма

1. Создайте массив объектов типа Animal и добавьте в него объекты Animal и Dog:

```
Animal[] animals = new Animal[2];
animals[0] = new Animal("Животное");
animals[1] = new Dog("Бобик");

for (Animal a : animals) {
    a.makeSound();
}
```

2. Запустите программу и проверьте результат.

4. Варианты заданий

1. Создайте класс Vehicle с полями brand, year и методом startEngine. Наследуйте классы Car и Motorcycle, переопределив метод startEngine.
2. Реализуйте класс Shape с методом calculateArea. Наследуйте классы Circle, Rectangle, Triangle, переопределив метод calculateArea.
3. Напишите класс Person с полями name, age и методом introduce. Наследуйте классы Student и Teacher, переопределив метод introduce.
4. Создайте класс BankAccount с полями accountNumber, balance и методом withdraw. Наследуйте классы SavingsAccount и CheckingAccount, переопределив метод withdraw.
5. Реализуйте класс Animal с полями name, age и методом makeSound. Наследуйте классы Cat, Dog, Bird, переопределив метод makeSound.
6. Напишите класс Employee с полями name, salary и методом calculateBonus. Наследуйте классы Manager и Developer, переопределив метод calculateBonus.
7. Создайте класс Device с полями brand, model и методом turnOn. Наследуйте классы Phone, Laptop, Tablet, переопределив метод turnOn.
8. Реализуйте класс Book с полями title, author и методом getDescription. Наследуйте классы FictionBook и NonFictionBook, переопределив метод getDescription.
9. Напишите класс Shape с методом draw. Наследуйте классы Circle, Square, Triangle, переопределив метод draw.
10. Создайте класс Animal с полями name, age и методом eat. Наследуйте классы Herbivore и Carnivore, переопределив метод eat.
11. Реализуйте класс Person с полями firstName, lastName и методом getFullName. Наследуйте классы Student и Professor, переопределив метод getFullName.
12. Напишите класс Vehicle с полями brand, year и методом stopEngine. Наследуйте классы Car, Bike, Truck, переопределив метод stopEngine.
13. Создайте класс Animal с полями name, age и методом sleep. Наследуйте классы Dog, Cat, Bird, переопределив метод sleep.
14. Реализуйте класс Employee с полями name, salary и методом work. Наследуйте классы Manager, Developer, Designer, переопределив метод work.
15. Напишите класс Device с полями brand, model и методом turnOff. Наследуйте классы Phone, Laptop, Tablet, переопределив метод turnOff.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое наследование в Java?
2. Как используется ключевое слово extends?
3. Что такое переопределение методов?
4. Как вызвать конструктор суперкласса?
5. Что такое полиморфизм и как он реализуется в Java?

Методические указания к лабораторной работе № 11.2 по теме: "Наследование: Абстрактные и final классы. Интерфейсы. Пакеты"

1. Цель выполнения работы
Целью данной лабораторной работы является изучение и практическое применение абстрактных и final классов, интерфейсов и пакетов в Java. Студенты научатся создавать и использовать абстрактные классы, final классы, интерфейсы, а также организовывать код в пакеты.

2. Теоретический материал

Абстрактные классы

Абстрактный класс — это класс, который не может быть instantiated (создан как объект). Он используется как базовый класс для других классов. Абстрактный класс может содержать абстрактные методы (без реализации) и обычные методы.

Пример абстрактного класса:

```
abstract class Shape {
    abstract double calculateArea(); // Абстрактный метод
    void display() {
        System.out.println("Это фигура");
    }
}
```

Final классы

Final класс — это класс, который не может быть унаследован. Все его методы также являются final (не могут быть переопределены).

Пример final класса:

```
final class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```

Интерфейсы

Интерфейс — это контракт, который определяет набор методов, которые должны быть реализованы классами. Интерфейсы могут содержать только абстрактные методы (до Java 8) и методы по умолчанию (default methods).

Пример интерфейса:

```
interface Drawable {
```

```
    void draw(); // Абстрактный метод
    default void print() {
        System.out.println("Рисование объекта");
    }
}
```

Пакеты

Пакеты — это механизм для организации классов и интерфейсов в логические группы. Пакеты помогают избежать конфликтов имен и улучшают структуру проекта.

Пример использования пакетов:

```
package com.example.shapes;

public class Circle {
    // Класс Circle
}
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте пакет com.example.shapes в папке src.

Шаг 2. Создание абстрактного класса

1. В пакете com.example.shapes создайте абстрактный класс Shape:

```
package com.example.shapes;

abstract class Shape {
    abstract double calculateArea();
    void display() {
        System.out.println("Это фигура");
    }
}
```

Шаг 3. Создание final класса

1. В пакете com.example.shapes создайте final класс Circle, который наследует Shape:

```
package com.example.shapes;

final class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```

Шаг 4. Создание интерфейса

1. В пакете com.example.shapes создайте интерфейс Drawable:

```
package com.example.shapes;

interface Drawable {
    void draw();
    default void print() {
        System.out.println("Рисование объекта");
    }
}
```

Шаг 5. Реализация интерфейса

1. В пакете com.example.shapes создайте класс Rectangle, который реализует интерфейс Drawable:

```
package com.example.shapes;

class Rectangle implements Drawable {
    @Override
    public void draw() {
        System.out.println("Рисование прямоугольника");
    }
}
```

Шаг 6. Использование классов и интерфейсов

1. В классе Main создайте объекты и вызовите методы:

```
package com.example.shapes;

public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        System.out.println("Площадь круга: " + circle.calculateArea());

        Rectangle rectangle = new Rectangle();
        rectangle.draw();
        rectangle.print();
    }
}
```

2. Запустите программу и проверьте результат.

4. Варианты заданий

1. Создайте абстрактный класс Vehicle с методом startEngine. Наследуйте классы Car и Motorcycle, реализуя метод.
2. Реализуйте интерфейс Playable с методом play. Создайте классы MusicPlayer и VideoPlayer, реализующие интерфейс.
3. Напишите final класс MathUtils с методами для вычисления факториала и суммы чисел.
4. Создайте абстрактный класс Animal с методом makeSound. Наследуйте классы Dog, Cat, Bird, реализуя метод.
5. Реализуйте интерфейс Edible с методом eat. Создайте классы Apple, Banana, реализующие интерфейс.
6. Напишите final класс Constants с константами (например, PI, GRAVITY).

7. Создайте абстрактный класс Employee с методом calculateSalary. Наследуйте классы Manager, Developer, реализуя метод.
8. Реализуйте интерфейс Flyable с методом fly. Создайте классы Bird, Airplane, реализующие интерфейс.
9. Напишите final класс StringUtils с методами для работы со строками (например, reverse, isPalindrome).
10. Создайте абстрактный класс Shape с методом draw. Наследуйте классы Circle, Rectangle, реализуя метод.
11. Реализуйте интерфейс Swimmable с методом swim. Создайте классы Fish, Duck, реализующие интерфейс.
12. Напишите final класс DateUtils с методами для работы с датами (например, isLeapYear, getDayOfWeek).
13. Создайте абстрактный класс Device с методом turnOn. Наследуйте классы Phone, Laptop, реализуя метод.
14. Реализуйте интерфейс Readable с методом read. Создайте классы Book, Newspaper, реализующие интерфейс.
15. Напишите final класс ArrayUtils с методами для работы с массивами (например, sort, findMax).

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое абстрактный класс и для чего он используется?
2. Какие методы могут содержаться в абстрактном классе?
3. Что такое final класс и чем он отличается от обычного класса?
4. Что такое интерфейс и как он используется в Java?
5. Как организовать код в пакеты и зачем это нужно?

Методические указания к лабораторной работе № 12 по теме: "Исключения: Обработка исключений. Генерация исключений"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение механизма обработки исключений в Java. Студенты научатся обрабатывать исключения с помощью блоков try-catch, использовать блок finally, а также создавать и генерировать собственные исключения.

2. Теоретический материал

Исключения в Java

Исключение — это событие, которое возникает во время выполнения программы и нарушает нормальный ход выполнения инструкций. В Java исключения делятся на два типа:

1. **Проверяемые исключения (checked exceptions)**: должны быть обработаны или объявлены в методе (например, IOException).
2. **Непроверяемые исключения (unchecked exceptions)**: не требуют обязательной обработки (например, NullPointerException, ArithmeticException).

Обработка исключений

Для обработки исключений используются блоки try-catch-finally:

- **try**: блок кода, в котором может возникнуть исключение.
- **catch**: блок кода, который обрабатывает исключение.
- **finally**: блок кода, который выполняется всегда, независимо от того, возникло исключение или нет.

Пример обработки исключения:

```
try {
    int result = 10 / 0; // Деление на ноль
} catch (ArithmeticException e) {
    System.out.println("Ошибка: " + e.getMessage());
} finally {
    System.out.println("Блок finally выполнен");
}
```

Генерация исключений

Для генерации исключений используется ключевое слово throw. Также можно создавать собственные исключения, наследуя класс Exception.

Пример генерации исключения:

```
if (age < 0) {
    throw new IllegalArgumentException("Возраст не может быть отрицательным");
}
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Обработка исключений

1. Напишите код, который вызывает исключение, и обработайте его:

```
public class Main {
    public static void main(String[] args) {
        try {
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[5]); // Выход за пределы массива
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Ошибка: " + e.getMessage());
        } finally {
            System.out.println("Блок finally выполнен");
        }
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 3. Генерация исключений

1. Напишите код, который генерирует исключение:

```
public class Main {
    public static void main(String[] args) {
        int age = -5;
        try {
            if (age < 0) {
                throw new IllegalArgumentException("Возраст не может быть отрицательным");
            }
            System.out.println("Возраст: " + age);
        } catch (IllegalArgumentException e) {
            System.out.println("Ошибка: " + e.getMessage());
        }
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 4. Создание собственного исключения

1. Создайте собственное исключение NegativeNumberException:

```
class NegativeNumberException extends Exception {
    public NegativeNumberException(String message) {
        super(message);
    }
}
```

2. Используйте это исключение в программе:

```
public class Main {
    public static void main(String[] args) {
        int number = -10;
        try {
            if (number < 0) {
```

```
            throw new NegativeNumberException("Число не может быть отрицательным");
         }
         System.out.println("Число: " + number);
      } catch (NegativeNumberException e) {
         System.out.println("Ошибка: " + e.getMessage());
      }
   }
}
```

3. Запустите программу и проверьте результат.

4. Варианты заданий

1. Напишите программу, которая обрабатывает деление на ноль.
2. Создайте программу, которая обрабатывает выход за пределы массива.
3. Реализуйте программу, которая генерирует исключение при вводе отрицательного числа.
4. Напишите программу, которая обрабатывает исключение при попытке открыть несуществующий файл.
5. Создайте программу, которая генерирует исключение при вводе пустой строки.
6. Реализуйте программу, которая обрабатывает исключение при попытке преобразования строки в число.
7. Напишите программу, которая генерирует исключение при вводе числа, превышающего допустимый диапазон.
8. Создайте программу, которая обрабатывает исключение при попытке доступа к null-объекту.
9. Реализуйте программу, которая генерирует исключение при вводе некорректного email.
10. Напишите программу, которая обрабатывает исключение при попытке выполнения операции с пустой коллекцией.
11. Создайте программу, которая генерирует исключение при вводе отрицательного возраста.
12. Реализуйте программу, которая обрабатывает исключение при попытке выполнения операции с недопустимым типом данных.
13. Напишите программу, которая генерирует исключение при вводе числа, не соответствующего условию.
14. Создайте программу, которая обрабатывает исключение при попытке выполнения операции с недопустимым индексом.
15. Реализуйте программу, которая генерирует исключение при вводе некорректного пароля.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое исключение в Java?
2. Какие типы исключений существуют в Java?
3. Как обработать исключение с помощью блока try-catch?
4. Для чего используется блок finally?
5. Как создать и сгенерировать собственное исключение?

Методические указания к лабораторной работе № 13.1 по теме: "Использование классов и интерфейсов: Классы-обертки. Вложенные классы"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение классов-оберток (wrapper classes) и вложенных классов в Java. Студенты научатся использовать классы-обертки для работы с примитивными типами данных, а также создавать и использовать вложенные классы для организации кода.

2. Теоретический материал

Классы-обертки

Классы-обертки — это классы, которые представляют примитивные типы данных в виде объектов. Они используются, когда необходимо работать с примитивами как с объектами (например, в коллекциях).

Основные классы-обертки:

- Integer — для типа int.
- Double — для типа double.
- Boolean — для типа boolean.
- Character — для типа char.

Пример использования классов-оберток:

```
Integer num = 10; // Автоупаковка (autoboxing)
int n = num;      // Автораспаковка (unboxing)
```

Вложенные классы

Вложенные классы — это классы, объявленные внутри другого класса. Они делятся на:

1. **Статические вложенные классы**: объявляются с ключевым словом static.
2. **Нестатические вложенные классы (внутренние классы)**: имеют доступ к полям и методам внешнего класса.
3. **Локальные классы**: объявляются внутри метода.
4. **Анонимные классы**: классы без имени, создаваемые "на лету".

Пример вложенного класса:

```
class Outer {
    private int outerField = 10;

    class Inner {
        void display() {
            System.out.println("Значение outerField: " + outerField);
        }
    }
}
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Использование классов-оберток

1. Напишите код, демонстрирующий использование классов-оберток:

```
public class Main {
    public static void main(String[] args) {
        Integer num1 = 10; // Автоупаковка
        Integer num2 = Integer.valueOf(20); // Явное создание объекта
        int sum = num1 + num2; // Автораспаковка
        System.out.println("Сумма: " + sum);

        Double d = Double.parseDouble("3.14"); // Преобразование строки в Double
        System.out.println("Значение d: " + d);
    }
}
```

2. Запустите программу и проверьте результат.

Шаг 3. Создание вложенного класса

1. Создайте внешний класс Outer и вложенный класс Inner:

```
class Outer {
    private int outerField = 10;

    class Inner {
        void display() {
            System.out.println("Значение outerField: " + outerField);
        }
    }
}
```

2. В классе Main создайте объект вложенного класса и вызовите его метод:

```
public class Main {
    public static void main(String[] args) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();
        inner.display();
    }
}
```

3. Запустите программу и проверьте результат.

Шаг 4. Использование статического вложенного класса

1. Создайте статический вложенный класс:

```
class Outer {
    static class StaticInner {
        void display() {
            System.out.println("Это статический вложенный класс");
        }
    }
}
```

2. В классе Main создайте объект статического вложенного класса и вызовите его метод:

```
public class Main {
```

```
    public static void main(String[] args) {
        Outer.StaticInner staticInner = new Outer.StaticInner();
        staticInner.display();
    }
}
```

3. Запустите программу и проверьте результат.

4. Варианты заданий

1. Напишите программу, которая использует классы-обертки для преобразования строки в число и обратно.
2. Создайте программу, которая демонстрирует работу с классом Character (например, проверка символа на цифру или букву).
3. Реализуйте программу, которая использует классы-обертки для выполнения арифметических операций.
4. Напишите программу, которая использует вложенный класс для хранения информации о студенте и его оценках.
5. Создайте программу, которая использует статический вложенный класс для хранения констант (например, математических констант).
6. Реализуйте программу, которая использует локальный класс внутри метода для выполнения вычислений.
7. Напишите программу, которая использует анонимный класс для реализации интерфейса Runnable.
8. Создайте программу, которая использует классы-обертки для работы с коллекциями (например, ArrayList<Integer>).
9. Реализуйте программу, которая использует вложенный класс для хранения информации о книге и её авторе.
10. Напишите программу, которая использует статический вложенный класс для хранения информации о геометрических фигурах.
11. Создайте программу, которая использует классы-обертки для работы с датами (например, LocalDate).
12. Реализуйте программу, которая использует вложенный класс для хранения информации о банковском счете и его владельце.
13. Напишите программу, которая использует анонимный класс для реализации интерфейса Comparator.
14. Создайте программу, которая использует классы-обертки для работы с логическими значениями (например, Boolean).
15. Реализуйте программу, которая использует вложенный класс для хранения информации о товаре и его цене.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое классы-обертки и для чего они используются?
2. Какие классы-обертки соответствуют примитивным типам данных?
3. Что такое автоупаковка иautораспаковка?
4. Какие типы вложенных классов существуют в Java?
5. В чем разница между статическими и нестатическими вложенными классами?

Методические указания к лабораторной работе № 13.2 по теме: "Использование классов и интерфейсов: Дженерики. Стандартные интерфейсы"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение дженериков (обобщений) и стандартных интерфейсов в Java. Студенты научатся создавать обобщенные классы и методы, а также использовать стандартные интерфейсы, такие как Comparable, Comparator, Runnable и Iterable.

2. Теоретический материал

Дженерики (Generics)

Дженерики позволяют создавать классы, интерфейсы и методы, которые работают с любыми типами данных. Они обеспечивают безопасность типов и позволяют избежать приведения типов.

Пример обобщенного класса:

```
class Box<T> {
    private T value;

    public void setValue(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}
```

Пример обобщенного метода:

```
public <T> void printArray(T[] array) {
    for (T element : array) {
        System.out.println(element);
    }
}
```

Стандартные интерфейсы

1. **Comparable<T>**: используется для сравнения объектов. Содержит метод compareTo(T o).
2. **Comparator<T>**: используется для определения порядка сортировки. Содержит метод compare(T o1, T o2).
3. **Runnable**: используется для создания потоков. Содержит метод run().
4. **Iterable<T>**: используется для итерации по коллекциям. Содержит метод iterator().

Пример использования интерфейса Comparable:

```
class Person implements Comparable<Person> {
    private String name;
    private int age;

    public Person(String name, int age) {
```

```
        this.name = name;
        this.age = age;
    }

    @Override
    public int compareTo(Person other) {
        return Integer.compare(this.age, other.age);
    }
}
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Создание обобщенного класса

1. Создайте обобщенный класс Box:

```
class Box<T> {
    private T value;

    public void setValue(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}
```

2. В классе Main создайте объекты класса Box с разными типами данных:

```
public class Main {
    public static void main(String[] args) {
        Box<Integer> intBox = new Box<>();
        intBox.setValue(10);
        System.out.println("Значение intBox: " + intBox.getValue());

        Box<String> strBox = new Box<>();
        strBox.setValue("Hello");
        System.out.println("Значение strBox: " + strBox.getValue());
    }
}
```

3. Запустите программу и проверьте результат.

Шаг 3. Использование интерфейса Comparable

1. Создайте класс Person, реализующий интерфейс Comparable:

```
class Person implements Comparable<Person> {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
```

```
    @Override
    public int compareTo(Person other) {
        return Integer.compare(this.age, other.age);
    }

    @Override
    public String toString() {
        return name + " (" + age + ")";
    }
}
```

2. В классе Main создайте массив объектов Person и отсортируйте его:

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        Person[] people = {
            new Person("Иван", 25),
            new Person("Мария", 20),
            new Person("Петр", 30)
        };
        Arrays.sort(people);
        for (Person person : people) {
            System.out.println(person);
        }
    }
}
```

3. Запустите программу и проверьте результат.

Шаг 4. Использование интерфейса Comparator

1. Создайте класс PersonComparator, реализующий интерфейс Comparator:

```
import java.util.Comparator;

class PersonComparator implements Comparator<Person> {
    @Override
    public int compare(Person p1, Person p2) {
        return p1.getName().compareTo(p2.getName());
    }
}
```

2. В классе Main отсортируйте массив объектов Person по имени:

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        Person[] people = {
            new Person("Иван", 25),
            new Person("Мария", 20),
            new Person("Петр", 30)
        };
        Arrays.sort(people, new PersonComparator());
        for (Person person : people) {
            System.out.println(person);
        }
    }
}
```

3. Запустите программу и проверьте результат.

4. Варианты заданий

1. Напишите обобщенный класс Pair, который хранит два значения разных типов.
2. Создайте обобщенный метод для поиска максимального элемента в массиве.
3. Реализуйте класс Student, который реализует интерфейс Comparable для сортировки по оценке.
4. Напишите класс Book, который реализует интерфейс Comparable для сортировки по названию.
5. Создайте обобщенный класс Stack, который реализует структуру данных "стек".
6. Реализуйте класс Employee, который реализует интерфейс Comparable для сортировки по зарплате.
7. Напишите обобщенный метод для подсчета количества вхождений элемента в массиве.
8. Создайте класс Car, который реализует интерфейс Comparable для сортировки по году выпуска.
9. Реализуйте класс Product, который реализует интерфейс Comparable для сортировки по цене.
10. Напишите обобщенный класс Queue, который реализует структуру данных "очередь".
11. Создайте класс Animal, который реализует интерфейс Comparable для сортировки по возрасту.
12. Реализуйте класс City, который реализует интерфейс Comparable для сортировки по населению.
13. Напишите обобщенный метод для проверки, содержится ли элемент в массиве.
14. Создайте класс Movie, который реализует интерфейс Comparable для сортировки по рейтингу.
15. Реализуйте класс Country, который реализует интерфейс Comparable для сортировки по площади.

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое дженерики и для чего они используются?
2. Как создать обобщенный класс в Java?
3. В чем разница между интерфейсами Comparable и Comparator?
4. Как использовать интерфейс Runnable для создания потоков?
5. Какие стандартные интерфейсы вы знаете и для чего они используются?

Методические указания к лабораторной работе № 14.1 по теме: "Порождающие паттерны проектирования"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение порождающих паттернов проектирования в Java. Студенты научатся использовать паттерны для создания объектов, такие как **Singleton**, **Factory Method**, **Abstract Factory**, **Builder** и **Prototype**, а также понимать их преимущества и области применения.

2. Теоретический материал

Порождающие паттерны проектирования

Порождающие паттерны связаны с процессом создания объектов. Они помогают сделать систему независимой от способа создания, композиции и представления объектов.

1. Singleton (Одиночка)

Обеспечивает создание только одного экземпляра класса и предоставляет глобальную точку доступа к нему.
Пример:

```
public class Singleton {
    private static Singleton instance;

    private Singleton() { }

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

2. Factory Method (Фабричный метод)

Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать.
Пример:

```
interface Product {
    void use();
}

class ConcreteProduct implements Product {
    @Override
    public void use() {
        System.out.println("Используем ConcreteProduct");
    }
}

abstract class Creator {
    public abstract Product factoryMethod();
}

class ConcreteCreator extends Creator {
    @Override
```

```
    public Product factoryMethod() {
        return new ConcreteProduct();
    }
}
```

3. Abstract Factory (Абстрактная фабрика)

Предоставляет интерфейс для создания семейств связанных или зависимых объектов без указания их конкретных классов.
Пример:

```
interface Button {
    void render();
}

class WindowsButton implements Button {
    @Override
    public void render() {
        System.out.println("Отрисовка кнопки для Windows");
    }
}

interface GUIFactory {
    Button createButton();
}

class WindowsFactory implements GUIFactory {
    @Override
    public Button createButton() {
        return new WindowsButton();
    }
}
```

4. Builder (Строитель)

Позволяет создавать сложные объекты пошагово, используя один и тот же процесс строительства.
Пример:

```
class Product {
    private String part1;
    private String part2;

    public void setPart1(String part1) {
        this.part1 = part1;
    }

    public void setPart2(String part2) {
        this.part2 = part2;
    }
}

class Builder {
    private Product product = new Product();

    public Builder buildPart1(String part1) {
        product.setPart1(part1);
        return this;
    }

    public Builder buildPart2(String part2) {
        product.setPart2(part2);
```

```
        return this;
    }

    public Product getResult() {
        return product;
    }
}
```

5. Prototype (Прототип)

Позволяет копировать объекты, не вдаваясь в подробности их реализации.
Пример:

```
interface Prototype {
    Prototype clone();
}

class ConcretePrototype implements Prototype {
    private String field;

    public ConcretePrototype(String field) {
        this.field = field;
    }

    @Override
    public Prototype clone() {
        return new ConcretePrototype(this.field);
    }
}
```

3. Порядок выполнения работы

Шаг 1. Реализация Singleton

1. Создайте класс Singleton и реализуйте паттерн "Одиночка".
2. Проверьте, что создается только один экземпляр класса.

Шаг 2. Реализация Factory Method

1. Создайте интерфейс Product и класс ConcreteProduct.
2. Реализуйте класс Creator и его подкласс ConcreteCreator.
3. Проверьте создание объекта через фабричный метод.

Шаг 3. Реализация Abstract Factory

1. Создайте интерфейс Button и класс WindowsButton.
2. Реализуйте интерфейс GUIFactory и класс WindowsFactory.
3. Проверьте создание кнопки через абстрактную фабрику.

Шаг 4. Реализация Builder

1. Создайте класс Product с несколькими полями.
2. Реализуйте класс Builder для пошагового создания объекта.
3. Проверьте создание сложного объекта.

Шаг 5. Реализация Prototype

1. Создайте интерфейс Prototype и класс ConcretePrototype.
2. Реализуйте метод clone для копирования объекта.
3. Проверьте работу паттерна.

4. Варианты заданий

1. Реализуйте паттерн Singleton для класса, представляющего подключение к базе данных.
2. Создайте фабричный метод для генерации различных типов документов (PDF, Word).
3. Реализуйте абстрактную фабрику для создания элементов интерфейса (кнопки, текстовые поля) для разных ОС (Windows, macOS).
4. Используйте паттерн Builder для создания объекта "Компьютер" с различными компонентами (процессор, ОЗУ, SSD).
5. Реализуйте паттерн Prototype для копирования объектов "Фигура" (круг, квадрат).
6. Создайте Singleton для класса, управляющего настройками приложения.
7. Реализуйте фабричный метод для создания различных типов транспортных средств (автомобиль, велосипед).
8. Используйте абстрактную фабрику для создания мебели (стул, стол) в разных стилях (классический, современный).
9. Реализуйте Builder для создания объекта "Пицца" с различными ингредиентами.
10. Создайте Prototype для копирования объектов "Документ" с разным содержимым.
11. Реализуйте Singleton для класса, управляющего логгированием.
12. Используйте фабричный метод для создания различных типов уведомлений (email, SMS).
13. Реализуйте абстрактную фабрику для создания игровых персонажей (воин, маг) в разных мирах (фэнтези, научная фантастика).
14. Используйте Builder для создания объекта "Автомобиль" с различными характеристиками (двигатель, коробка передач).
15. Реализуйте Prototype для копирования объектов "Пользователь" с разными ролями (админ, гость).

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

6. Контрольные вопросы

1. Что такое порождающие паттерны проектирования?
2. В чем разница между Factory Method и Abstract Factory?
3. Как работает паттерн Singleton и в каких случаях его используют?
4. Какие преимущества предоставляет паттерн Builder?
5. Как реализуется паттерн Prototype и для чего он применяется?

Методические указания к лабораторной работе № 14.2 по теме: "Структурные паттерны проектирования"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение структурных паттернов проектирования в Java. Студенты научатся использовать паттерны, такие как **Adapter**, **Decorator**, **Proxy**, **Composite**, **Bridge**, **Facade** и **Flyweight**, для организации структуры классов и объектов, а также понимать их преимущества и области применения.

---

2. Теоретический материал

Структурные паттерны проектирования

Структурные паттерны связаны с композицией классов и объектов. Они помогают организовать структуру программы, делая её более гибкой и расширяемой.

1. Adapter (Адаптер)

Позволяет объектам с несовместимыми интерфейсами работать вместе.
Пример:

```
interface MediaPlayer {
    void play(String audioType, String fileName);
}

class Mp3Player implements MediaPlayer {
    @Override
    public void play(String audioType, String fileName) {
        if (audioType.equalsIgnoreCase("mp3")) {
            System.out.println("Воспроизведение MP3 файла: " + fileName);
        } else {
            System.out.println("Неподдерживаемый формат: " + audioType);
        }
    }
}

class MediaAdapter implements MediaPlayer {
    private AdvancedMediaPlayer advancedMediaPlayer;

    public MediaAdapter(String audioType) {
        if (audioType.equalsIgnoreCase("vlc")) {
            advancedMediaPlayer = new VlcPlayer();
        } else if (audioType.equalsIgnoreCase("mp4")) {
            advancedMediaPlayer = new Mp4Player();
        }
    }

    @Override
    public void play(String audioType, String fileName) {
        if (audioType.equalsIgnoreCase("vlc")) {
            advancedMediaPlayer.playVlc(fileName);
        } else if (audioType.equalsIgnoreCase("mp4")) {
            advancedMediaPlayer.playMp4(fileName);
        }
```

```
    }
}
```

2. Decorator (Декоратор)

Позволяет динамически добавлять объектам новую функциональность, оборачивая их в объекты-декораторы.
Пример:

```
interface Coffee {
    String getDescription();
    double getCost();
}

class SimpleCoffee implements Coffee {
    @Override
    public String getDescription() {
        return "Простой кофе";
    }

    @Override
    public double getCost() {
        return 5.0;
    }
}

class MilkDecorator implements Coffee {
    private Coffee coffee;

    public MilkDecorator(Coffee coffee) {
        this.coffee = coffee;
    }

    @Override
    public String getDescription() {
        return coffee.getDescription() + ", молоко";
    }

    @Override
    public double getCost() {
        return coffee.getCost() + 1.5;
    }
}
```

3. Proxy (Заместитель)

Предоставляет объект-заместитель, который контролирует доступ к другому объекту.
Пример:

```
interface Image {
    void display();
}

class RealImage implements Image {
    private String fileName;

    public RealImage(String fileName) {
        this.fileName = fileName;
        loadFromDisk();
    }

    private void loadFromDisk() {
```

```
        System.out.println("Загрузка изображения: " + fileName);
    }

    @Override
    public void display() {
        System.out.println("Показ изображения: " + fileName);
    }
}

class ProxyImage implements Image {
    private RealImage realImage;
    private String fileName;

    public ProxyImage(String fileName) {
        this.fileName = fileName;
    }

    @Override
    public void display() {
        if (realImage == null) {
            realImage = new RealImage(fileName);
        }
        realImage.display();
    }
}
```

4. Composite (Компоновщик)

Позволяет сгруппировать объекты в древовидную структуру и работать с ними как с единым объектом.
Пример:

```
interface Component {
    void operation();
}

class Leaf implements Component {
    @Override
    public void operation() {
        System.out.println("Операция в листе");
    }
}

class Composite implements Component {
    private List<Component> children = new ArrayList<>();

    public void add(Component component) {
        children.add(component);
    }

    @Override
    public void operation() {
        for (Component component : children) {
            component.operation();
        }
    }
}
```

5. Bridge (Мост)

Разделяет абстракцию и реализацию, позволяя им изменяться независимо.
Пример:

```
interface Renderer {
    void renderCircle(float radius);
}

class VectorRenderer implements Renderer {
    @Override
    public void renderCircle(float radius) {
        System.out.println("Рисуем круг радиусом " + radius + " в векторном формате");
    }
}

class Shape {
    protected Renderer renderer;

    public Shape(Renderer renderer) {
        this.renderer = renderer;
    }

    public void draw() {}
}

class Circle extends Shape {
    private float radius;

    public Circle(Renderer renderer, float radius) {
        super(renderer);
        this.radius = radius;
    }

    @Override
    public void draw() {
        renderer.renderCircle(radius);
    }
}
```

6. Facade (Фасад)

Предоставляет простой интерфейс для сложной системы классов.
Пример:

```
class CPU {
    void processData() {
        System.out.println("Обработка данных процессором");
    }
}

class Memory {
    void load() {
        System.out.println("Загрузка данных в память");
    }
}

class ComputerFacade {
    private CPU cpu;
    private Memory memory;

    public ComputerFacade() {
        cpu = new CPU();
        memory = new Memory();
    }

    public void start() {
```

```
        memory.load();
        cpu.processData();
    }
}
```

7. Flyweight (Приспособленец)

Позволяет эффективно использовать общие объекты для минимизации использования памяти.
Пример:

```
class TreeType {
    private String name;
    private String color;

    public TreeType(String name, String color) {
        this.name = name;
        this.color = color;
    }

    public void draw(int x, int y) {
        System.out.println("Рисуем дерево типа " + name + " цвета " + color + " в координатах (" + x + ", " + y +
")");
    }
}

class TreeFactory {
    private static Map<String, TreeType> treeTypes = new HashMap<>();

    public static TreeType getTreeType(String name, String color) {
        TreeType result = treeTypes.get(name + color);
        if (result == null) {
            result = new TreeType(name, color);
            treeTypes.put(name + color, result);
        }
        return result;
    }
}
```

---

3. Порядок выполнения работы

Шаг 1. Реализация Adapter

1. Создайте интерфейс MediaPlayer и класс Mp3Player.
2. Реализуйте класс MediaAdapter для поддержки других форматов.
3. Проверьте работу адаптера.

Шаг 2. Реализация Decorator

1. Создайте интерфейс Coffee и класс SimpleCoffee.
2. Реализуйте класс MilkDecorator для добавления функциональности.
3. Проверьте работу декоратора.

Шаг 3. Реализация Proxy

1. Создайте интерфейс Image и класс RealImage.
2. Реализуйте класс ProxyImage для контроля доступа.

3. Проверьте работу заместителя.

Шаг 4. Реализация Composite

1. Создайте интерфейс Component и классы Leaf и Composite.
2. Проверьте работу компоновщика.

Шаг 5. Реализация Bridge

1. Создайте интерфейс Renderer и класс VectorRenderer.
2. Реализуйте классы Shape и Circle.
3. Проверьте работу моста.

Шаг 6. Реализация Facade

1. Создайте классы CPU, Memory и ComputerFacade.
2. Проверьте работу фасада.

Шаг 7. Реализация Flyweight

1. Создайте класс TreeType и фабрику TreeFactory.
2. Проверьте работу приспособленца.

---

4. Варианты заданий

1. Реализуйте Adapter для преобразования данных из JSON в XML.
2. Используйте Decorator для добавления функциональности к объекту "Пицца".
3. Реализуйте Proxy для контроля доступа к базе данных.
4. Используйте Composite для создания древовидной структуры "Файловая система".
5. Реализуйте Bridge для разделения абстракции "Фигура" и её реализации "Векторный/Растровый рендеринг".
6. Используйте Facade для упрощения работы с API банка.
7. Реализуйте Flyweight для оптимизации работы с объектами "Текстуры" в игре.
8. Создайте Adapter для интеграции старой системы с новой.
9. Используйте Decorator для добавления функциональности к объекту "Чай".
10. Реализуйте Proxy для кэширования запросов к серверу.
11. Используйте Composite для создания структуры "Организация".
12. Реализуйте Bridge для разделения абстракции "Устройство" и его реализации (iOS/Android).
13. Используйте Facade для упрощения работы с библиотекой для работы с файлами.
14. Реализуйте Flyweight для оптимизации работы с объектами "Шрифты" в текстовом редакторе.
15. Используйте Adapter для преобразования данных из CSV в JSON.

---

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

---

6. Контрольные вопросы

1. Что такое структурные паттерны проектирования?
2. В чем разница между Adapter и Proxy?
3. Как работает паттерн Decorator и в каких случаях его используют?
4. Какие преимущества предоставляет паттерн Composite?
5. Как реализуется паттерн Flyweight и для чего он применяется?

Методические указания к лабораторной работе № 14.3 по теме: "Поведенческие паттерны проектирования"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение поведенческих паттернов проектирования в Java. Студенты научатся использовать паттерны, такие как **Observer**, **Strategy**, **Command**, **State**, **Chain of Responsibility**, **Iterator**, **Template Method**, **Visitor**, **Mediator** и **Memento**, для управления взаимодействием объектов и распределения ответственности, а также понимать их преимущества и области применения.

---

2. Теоретический материал

Поведенческие паттерны проектирования

Поведенческие паттерны связаны с распределением ответственности между объектами и управлением их взаимодействием. Они помогают сделать систему более гибкой и расширяемой.

1. Observer (Наблюдатель)

Позволяет объектам подписываться на изменения состояния других объектов.
Пример:

```
interface Observer {
    void update(String message);
}

class ConcreteObserver implements Observer {
    @Override
    public void update(String message) {
        System.out.println("Получено сообщение: " + message);
    }
}

class Subject {
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void notifyObservers(String message) {
        for (Observer observer : observers) {
            observer.update(message);
        }
    }
}
```

2. Strategy (Стратегия)

Позволяет выбирать алгоритм выполнения задачи во время выполнения программы.
Пример:

```
interface PaymentStrategy {
```

```
    void pay(int amount);
}

class CreditCardPayment implements PaymentStrategy {
    @Override
    public void pay(int amount) {
        System.out.println("Оплата кредитной картой: " + amount);
    }
}

class PayPalPayment implements PaymentStrategy {
    @Override
    public void pay(int amount) {
        System.out.println("Оплата через PayPal: " + amount);
    }
}

class ShoppingCart {
    private PaymentStrategy paymentStrategy;

    public void setPaymentStrategy(PaymentStrategy paymentStrategy) {
        this.paymentStrategy = paymentStrategy;
    }

    public void checkout(int amount) {
        paymentStrategy.pay(amount);
    }
}
```

3. Command (Команда)

Инкапсулирует запрос как объект, позволяя параметризовать клиентов с различными запросами.
Пример:

```
interface Command {
    void execute();
}

class LightOnCommand implements Command {
    private Light light;

    public LightOnCommand(Light light) {
        this.light = light;
    }

    @Override
    public void execute() {
        light.on();
    }
}

class RemoteControl {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void pressButton() {
        command.execute();
    }
```

```
}
```

4. State (Состояние)

Позволяет объекту изменять своё поведение при изменении внутреннего состояния.
Пример:

```
interface State {
    void handle();
}

class ConcreteStateA implements State {
    @Override
    public void handle() {
        System.out.println("Обработка в состоянии A");
    }
}

class Context {
    private State state;

    public void setState(State state) {
        this.state = state;
    }

    public void request() {
        state.handle();
    }
}
```

5. Chain of Responsibility (Цепочка обязанностей)

Позволяет передавать запросы по цепочке обработчиков.
Пример:

```
abstract class Handler {
    private Handler next;

    public void setNext(Handler next) {
        this.next = next;
    }

    public void handleRequest(String request) {
        if (next != null) {
            next.handleRequest(request);
        }
    }
}

class ConcreteHandlerA extends Handler {
    @Override
    public void handleRequest(String request) {
        if (request.equals("A")) {
            System.out.println("Обработка запроса A");
        } else {
            super.handleRequest(request);
        }
    }
}
```

6. Iterator (Итератор)

Предоставляет способ последовательного доступа к элементам коллекции без раскрытия её внутренней структуры.
Пример:

```
interface Iterator<T> {
    boolean hasNext();
    T next();
}

class ConcreteIterator implements Iterator<String> {
    private String[] items;
    private int position = 0;

    public ConcreteIterator(String[] items) {
        this.items = items;
    }

    @Override
    public boolean hasNext() {
        return position < items.length;
    }

    @Override
    public String next() {
        return items[position++];
    }
}
```

7. Template Method (Шаблонный метод)

Определяет скелет алгоритма, позволяя подклассам переопределять некоторые шаги.
Пример:

```
abstract class Game {
    abstract void initialize();
    abstract void startPlay();
    abstract void endPlay();

    public final void play() {
        initialize();
        startPlay();
        endPlay();
    }
}

class Football extends Game {
    @Override
    void initialize() {
        System.out.println("Инициализация футбола");
    }

    @Override
    void startPlay() {
        System.out.println("Начало игры в футбол");
    }

    @Override
    void endPlay() {
        System.out.println("Завершение игры в футбол");
    }
}
```

8. Visitor (Посетитель)

Позволяет добавлять новые операции к объектам без изменения их классов.
Пример:

```
interface Visitor {
    void visit(ElementA element);
    void visit(ElementB element);
}

class ConcreteVisitor implements Visitor {
    @Override
    public void visit(ElementA element) {
        System.out.println("Посещение элемента A");
    }

    @Override
    public void visit(ElementB element) {
        System.out.println("Посещение элемента B");
    }
}
```

9. Mediator (Посредник)

Упрощает взаимодействие между объектами, инкапсулируя их коммуникацию.
Пример:

```
class Mediator {
    private ColleagueA colleagueA;
    private ColleagueB colleagueB;

    public void setColleagueA(ColleagueA colleagueA) {
        this.colleagueA = colleagueA;
    }

    public void setColleagueB(ColleagueB colleagueB) {
        this.colleagueB = colleagueB;
    }

    public void send(String message, Colleague colleague) {
        if (colleague == colleagueA) {
            colleagueB.notify(message);
        } else {
            colleagueA.notify(message);
        }
    }
}
```

10. Memento (Хранитель)

Позволяет сохранять и восстанавливать состояние объекта.
Пример:

```
class Memento {
    private String state;

    public Memento(String state) {
        this.state = state;
    }

    public String getState() {
```

```
        return state;
    }
}

class Originator {
    private String state;

    public void setState(String state) {
        this.state = state;
    }

    public Memento saveState() {
        return new Memento(state);
    }

    public void restoreState(Memento memento) {
        state = memento.getState();
    }
}
```

---

3. Порядок выполнения работы

Шаг 1. Реализация Observer

1. Создайте интерфейс Observer и класс ConcreteObserver.
2. Реализуйте класс Subject для управления наблюдателями.
3. Проверьте работу паттерна.

Шаг 2. Реализация Strategy

1. Создайте интерфейс PaymentStrategy и классы CreditCardPayment, PayPalPayment.
2. Реализуйте класс ShoppingCart.
3. Проверьте работу паттерна.

Шаг 3. Реализация Command

1. Создайте интерфейс Command и класс LightOnCommand.
2. Реализуйте класс RemoteControl.
3. Проверьте работу паттерна.

Шаг 4. Реализация State

1. Создайте интерфейс State и класс ConcreteStateA.
2. Реализуйте класс Context.
3. Проверьте работу паттерна.

Шаг 5. Реализация Chain of Responsibility

1. Создайте абстрактный класс Handler и класс ConcreteHandlerA.
2. Проверьте работу паттерна.

Шаг 6. Реализация Iterator

1. Создайте интерфейс Iterator и класс ConcreteIterator.

2. Проверьте работу паттерна.

Шаг 7. Реализация Template Method

1. Создайте абстрактный класс Game и класс Football.
2. Проверьте работу паттерна.

Шаг 8. Реализация Visitor

1. Создайте интерфейс Visitor и класс ConcreteVisitor.
2. Проверьте работу паттерна.

Шаг 9. Реализация Mediator

1. Создайте класс Mediator и классы ColleagueA, ColleagueB.
2. Проверьте работу паттерна.

Шаг 10. Реализация Memento

1. Создайте классы Memento и Originator.
2. Проверьте работу паттерна.

---

4. Варианты заданий

1. Реализуйте Observer для системы уведомлений.
2. Используйте Strategy для выбора алгоритма сортировки.
3. Реализуйте Command для управления устройствами умного дома.
4. Используйте State для управления состоянием заказа в интернет-магазине.
5. Реализуйте Chain of Responsibility для обработки запросов в банке.
6. Используйте Iterator для обхода элементов коллекции.
7. Реализуйте Template Method для создания игрового процесса.
8. Используйте Visitor для добавления функциональности к элементам XML.
9. Реализуйте Mediator для управления взаимодействием между компонентами UI.
10. Используйте Memento для сохранения и восстановления состояния текстового редактора.
11. Реализуйте Observer для системы чата.
12. Используйте Strategy для выбора способа доставки.
13. Реализуйте Command для управления командами в игре.
14. Используйте State для управления состоянием игрового персонажа.
15. Реализуйте Chain of Responsibility для обработки ошибок.

---

5. Содержание отчета
Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.

4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

---

6. Контрольные вопросы

1. Что такое поведенческие паттерны проектирования?
2. В чем разница между Observer и Mediator?
3. Как работает паттерн Strategy и в каких случаях его используют?
4. Какие преимущества предоставляет паттерн Command?
5. Как реализуется паттерн Memento и для чего он применяется?

Методические указания к лабораторной работе № 15 по теме: "Работа с коллекциями: ArrayList"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение коллекции ArrayList в Java. Студенты научатся создавать, инициализировать и работать с динамическими массивами, а также использовать основные методы класса ArrayList для добавления, удаления, поиска и сортировки элементов.

---

2. Теоретический материал

Класс ArrayList

ArrayList — это реализация динамического массива в Java, который позволяет хранить элементы любого типа. Он является частью Java Collections Framework и реализует интерфейс List.

Основные характеристики ArrayList:

- Динамически изменяемый размер.
- Поддерживает хранение дубликатов.
- Позволяет доступ к элементам по индексу.
- Не синхронизирован (не потокобезопасен).

Основные методы класса ArrayList

1. **Добавление элементов**:
   - add(E element) — добавляет элемент в конец списка.
   - add(int index, E element) — добавляет элемент по указанному индексу.
2. **Удаление элементов**:
   - remove(int index) — удаляет элемент по индексу.
   - remove(Object o) — удаляет первое вхождение указанного объекта.
3. **Получение элементов**:
   - get(int index) — возвращает элемент по индексу.
4. **Изменение элементов**:
   - set(int index, E element) — заменяет элемент по указанному индексу.
5. **Поиск элементов**:
   - contains(Object o) — проверяет, содержится ли элемент в списке.
   - indexOf(Object o) — возвращает индекс первого вхождения элемента.
6. **Размер списка**:
   - size() — возвращает количество элементов в списке.
7. **Сортировка**:
   - sort(Comparator<? super E> c) — сортирует список с использованием компаратора.

Пример создания и использования ArrayList:

```
import java.util.ArrayList;
```

```
public class Main {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Java");
        list.add("Python");
        list.add("C++");

        System.out.println("Список: " + list);
        System.out.println("Размер списка: " + list.size());
        System.out.println("Элемент по индексу 1: " + list.get(1));

        list.remove("Python");
        System.out.println("Список после удаления: " + list);
    }
}
```

---

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Создание и инициализация ArrayList

1. Создайте объект ArrayList для хранения строк:
   ```
   ArrayList<String> list = new ArrayList<>();
   ```
2. Добавьте несколько элементов в список:
   ```
   list.add("Java");
   list.add("Python");
   list.add("C++");
   ```

Шаг 3. Работа с элементами списка

1. Выведите список на экран.
2. Получите элемент по индексу и выведите его.
3. Удалите элемент из списка и выведите обновленный список.

Шаг 4. Использование методов ArrayList

1. Проверьте, содержится ли элемент в списке, используя метод contains.
2. Найдите индекс элемента с помощью метода indexOf.
3. Измените элемент по индексу с помощью метода set.

Шаг 5. Сортировка списка

1. Отсортируйте список с использованием метода sort и компаратора:
   ```
   list.sort(String::compareTo);
   ```
2. Выведите отсортированный список.

---

4. Варианты заданий

1. Создайте список из 10 чисел и найдите сумму всех элементов.
2. Реализуйте программу для поиска максимального элемента в списке.
3. Напишите программу для удаления всех дубликатов из списка.
4. Создайте список строк и отсортируйте его в алфавитном порядке.
5. Реализуйте программу для поиска индекса минимального элемента в списке.
6. Напишите программу для объединения двух списков в один.
7. Создайте список чисел и найдите среднее арифметическое его элементов.
8. Реализуйте программу для проверки, является ли список пустым.
9. Напишите программу для удаления всех элементов списка, кратных 3.
10. Создайте список строк и найдите самую длинную строку.
11. Реализуйте программу для поиска количества вхождений элемента в список.
12. Напишите программу для создания списка из четных чисел от 1 до 100.
13. Создайте список объектов класса Student (с полями name и age) и отсортируйте его по возрасту.
14. Реализуйте программу для поиска всех уникальных элементов в списке.
15. Напишите программу для создания списка из случайных чисел и его сортировки по убыванию.

---

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

---

6. Контрольные вопросы

1. Что такое ArrayList и чем он отличается от обычного массива?
2. Какие основные методы класса ArrayList вы знаете?
3. Как добавить элемент в ArrayList?
4. Как удалить элемент из ArrayList?
5. Как отсортировать элементы в ArrayList?

Методические указания к лабораторной работе № 16 по теме: "Работа с коллекциями: LinkedList"

1. Цель выполнения работы
Целью данной лабораторной работы является изучение и практическое применение коллекции LinkedList в Java. Студенты научатся создавать, инициализировать и работать с двусвязными списками, а также использовать основные методы класса LinkedList для добавления, удаления, поиска и обработки элементов.

2. Теоретический материал

Класс LinkedList

LinkedList — это реализация двусвязного списка в Java, который позволяет хранить элементы любого типа. Он является частью Java Collections Framework и реализует интерфейсы List и Deque.

Основные характеристики LinkedList:

- Элементы хранятся в виде узлов, каждый из которых содержит ссылки на предыдущий и следующий элементы.
- Поддерживает быструю вставку и удаление элементов в начале и конце списка.
- Позволяет доступ к элементам по индексу, но работает медленнее, чем ArrayList.
- Не синхронизирован (не потокобезопасен).

Основные методы класса LinkedList

1. **Добавление элементов**:
   - add(E element) — добавляет элемент в конец списка.
   - addFirst(E element) — добавляет элемент в начало списка.
   - addLast(E element) — добавляет элемент в конец списка.
2. **Удаление элементов**:
   - remove(int index) — удаляет элемент по индексу.
   - removeFirst() — удаляет первый элемент.
   - removeLast() — удаляет последний элемент.
3. **Получение элементов**:
   - get(int index) — возвращает элемент по индексу.
   - getFirst() — возвращает первый элемент.
   - getLast() — возвращает последний элемент.
4. **Поиск элементов**:
   - contains(Object o) — проверяет, содержится ли элемент в списке.
   - indexOf(Object o) — возвращает индекс первого вхождения элемента.
5. **Размер списка**:
   - size() — возвращает количество элементов в списке.
6. **Дополнительные методы**:
   - offer(E element) — добавляет элемент в конец списка (аналог addLast).
   - poll() — удаляет и возвращает первый элемент.
   - peek() — возвращает первый элемент без удаления.

Пример создания и использования LinkedList:

```
import java.util.LinkedList;

public class Main {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>();
        list.add("Java");
        list.add("Python");
        list.addFirst("C++");

        System.out.println("Список: " + list);
        System.out.println("Первый элемент: " + list.getFirst());
        System.out.println("Последний элемент: " + list.getLast());

        list.removeLast();
        System.out.println("Список после удаления последнего элемента: " + list);
    }
}
```

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Создание и инициализация LinkedList

1. Создайте объект LinkedList для хранения строк:
```
LinkedList<String> list = new LinkedList<>();
```
2. Добавьте несколько элементов в список:
```
list.add("Java");
list.add("Python");
list.addFirst("C++");
```

Шаг 3. Работа с элементами списка

1. Выведите список на экран.
2. Получите первый и последний элементы списка и выведите их.
3. Удалите последний элемент из списка и выведите обновленный список.

Шаг 4. Использование методов LinkedList

1. Проверьте, содержится ли элемент в списке, используя метод contains.
2. Найдите индекс элемента с помощью метода indexOf.
3. Добавьте элемент в начало списка с помощью метода addFirst.

Шаг 5. Использование методов интерфейса Deque

1. Добавьте элемент в конец списка с помощью метода offer.
2. Удалите и верните первый элемент с помощью метода poll.
3. Верните первый элемент без удаления с помощью метода peek.

4. Варианты заданий

1. Создайте список из 10 чисел и найдите сумму всех элементов.
2. Реализуйте программу для поиска максимального элемента в списке.
3. Напишите программу для удаления всех дубликатов из списка.
4. Создайте список строк и отсортируйте его в алфавитном порядке.
5. Реализуйте программу для поиска индекса минимального элемента в списке.
6. Напишите программу для объединения двух списков в один.
7. Создайте список чисел и найдите среднее арифметическое его элементов.
8. Реализуйте программу для проверки, является ли список пустым.
9. Напишите программу для удаления всех элементов списка, кратных 3.
10. Создайте список строк и найдите самую длинную строку.
11. Реализуйте программу для поиска количества вхождений элемента в список.
12. Напишите программу для создания списка из четных чисел от 1 до 100.
13. Создайте список объектов класса Student (с полями name и age) и отсортируйте его по возрасту.
14. Реализуйте программу для поиска всех уникальных элементов в списке.
15. Напишите программу для создания списка из случайных чисел и его сортировки по убыванию.

---

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

---

6. Контрольные вопросы

1. Что такое LinkedList и чем он отличается от ArrayList?
2. Какие основные методы класса LinkedList вы знаете?
3. Как добавить элемент в начало LinkedList?
4. Как удалить последний элемент из LinkedList?
5. Какие методы интерфейса Deque поддерживает LinkedList?

Методические указания к лабораторной работе № 17 по теме: "Работа с коллекциями: HashMap"

1. Цель выполнения работы

Целью данной лабораторной работы является изучение и практическое применение коллекции HashMap в Java. Студенты научатся создавать, инициализировать и работать с ассоциативными массивами (хэш-таблицами), а также использовать основные методы класса HashMap для добавления, удаления, поиска и обработки элементов.

---

2. Теоретический материал

Класс HashMap

HashMap — это реализация ассоциативного массива (хэш-таблицы) в Java, которая хранит данные в виде пар "ключ-значение". Он является частью Java Collections Framework и реализует интерфейс Map.

Основные характеристики HashMap:

- Ключи и значения могут быть любого типа.
- Ключи должны быть уникальными, значения могут дублироваться.
- Позволяет быстрый доступ к элементам по ключу (в среднем O(1)).
- Не гарантирует порядок элементов.
- Не синхронизирован (не потокобезопасен).

Основные методы класса HashMap

1. **Добавление элементов**:
   - put(K key, V value) — добавляет пару "ключ-значение".
   - putAll(Map<? extends K, ? extends V> map) — добавляет все элементы из другой карты.
2. **Удаление элементов**:
   - remove(Object key) — удаляет элемент по ключу.
   - clear() — удаляет все элементы из карты.
3. **Получение элементов**:
   - get(Object key) — возвращает значение по ключу.
4. **Поиск элементов**:
   - containsKey(Object key) — проверяет, содержится ли ключ в карте.
   - containsValue(Object value) — проверяет, содержится ли значение в карте.
5. **Размер карты**:
   - size() — возвращает количество пар "ключ-значение".
6. **Итерация по элементам**:
   - keySet() — возвращает множество ключей.
   - values() — возвращает коллекцию значений.
   - entrySet() — возвращает множество пар "ключ-значение".

Пример создания и использования HashMap:

```
import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        HashMap<String, Integer> map = new HashMap<>();
        map.put("Java", 1);
        map.put("Python", 2);
        map.put("C++", 3);

        System.out.println("Карта: " + map);
        System.out.println("Значение для ключа 'Java': " + map.get("Java"));

        map.remove("Python");
        System.out.println("Карта после удаления: " + map);
    }
}
```

---

3. Порядок выполнения работы

Шаг 1. Создание нового проекта

1. Откройте среду разработки (например, IntelliJ IDEA).
2. Создайте новый проект Java.
3. Создайте класс Main в папке src.

Шаг 2. Создание и инициализация HashMap

1. Создайте объект HashMap для хранения пар "ключ-значение":
   ```
   HashMap<String, Integer> map = new HashMap<>();
   ```
2. Добавьте несколько пар в карту:
   ```
   map.put("Java", 1);
   map.put("Python", 2);
   map.put("C++", 3);
   ```

Шаг 3. Работа с элементами карты

1. Выведите карту на экран.
2. Получите значение по ключу и выведите его.
3. Удалите элемент по ключу и выведите обновленную карту.

Шаг 4. Использование методов HashMap

1. Проверьте, содержится ли ключ в карте, используя метод containsKey.
2. Проверьте, содержится ли значение в карте, используя метод containsValue.
3. Выведите множество ключей с помощью метода keySet.

Шаг 5. Итерация по элементам карты

1. Используйте метод entrySet для итерации по парам "ключ-значение":
   ```
   for (var entry : map.entrySet()) {
       System.out.println("Ключ: " + entry.getKey() + ", Значение: " + entry.getValue());
   }
   ```
2. Выведите коллекцию значений с помощью метода values.

4. Варианты заданий

1. Создайте карту, где ключи — это имена студентов, а значения — их оценки.
2. Реализуйте программу для поиска максимального значения в карте.
3. Напишите программу для удаления всех элементов с заданным значением.
4. Создайте карту, где ключи — это названия стран, а значения — их столицы.
5. Реализуйте программу для поиска ключа по значению.
6. Напишите программу для объединения двух карт в одну.
7. Создайте карту, где ключи — это слова, а значения — количество их вхождений в тексте.
8. Реализуйте программу для проверки, является ли карта пустой.
9. Напишите программу для удаления всех элементов с ключами, начинающимися на букву "A".
10. Создайте карту, где ключи — это даты, а значения — события.
11. Реализуйте программу для поиска количества вхождений каждого слова в тексте.
12. Напишите программу для создания карты из списка строк, где ключи — это строки, а значения — их длины.
13. Создайте карту, где ключи — это идентификаторы товаров, а значения — их цены.
14. Реализуйте программу для поиска всех уникальных значений в карте.
15. Напишите программу для создания карты из случайных чисел, где ключи — это индексы, а значения — числа.

---

5. Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист с указанием названия работы, фамилии и имени студента, группы и даты выполнения.
2. Цель работы.
3. Краткое описание выполненных шагов.
4. Исходный код программы.
5. Скриншоты результатов выполнения программы.
6. Ответы на контрольные вопросы.

---

6. Контрольные вопросы

1. Что такое HashMap и чем он отличается от ArrayList?
2. Какие основные методы класса HashMap вы знаете?
3. Как добавить пару "ключ-значение" в HashMap?
4. Как удалить элемент из HashMap по ключу?
5. Как проверить, содержится ли значение в HashMap?

Методические указания к лабораторной работе № 17 А по теме: "Работа с файлами"

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение методов работы с файлами в языке Java. В ходе выполнения работы студенты научатся:

- создавать, читать, записывать и удалять файлы;
- работать с текстовыми и бинарными файлами;
- использовать потоки ввода-вывода (I/O) для обработки данных;
- обрабатывать исключения, связанные с файловыми операциями;
- организовывать структурированное хранение данных в файлах.

---

2. Теоретический материал

Работа с файлами в Java осуществляется с использованием классов из пакетов java.io и java.nio. Эти пакеты предоставляют инструменты для чтения и записи данных, а также для управления файлами и директориями.

Основные классы для работы с файлами:

1. **Класс File:**
   - Используется для представления файлов и директорий.
   - Позволяет создавать, удалять, переименовывать файлы и директории, а также проверять их свойства (например, существование, доступность для чтения/записи).
   - Пример:

```
File file = new File("example.txt");
if (file.exists()) {
    System.out.println("Файл существует.");
}
```

2. **Потоки ввода-вывода (I/O Streams):**
   - **Байтовые потоки:**
     - InputStream и OutputStream — базовые классы для работы с бинарными данными.
     - Пример: FileInputStream, FileOutputStream.
   - **Символьные потоки:**
     - Reader и Writer — классы для работы с текстовыми данными.
     - Пример: FileReader, FileWriter.
3. **Буферизованные потоки:**
   - BufferedReader и BufferedWriter — повышают производительность за счет буферизации данных.
   - Пример:

```
try (BufferedReader reader = new BufferedReader(new FileReader("example.txt"))) {
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

4. **Класс Files (из пакета java.nio.file):**
   - Предоставляет статические методы для работы с файлами и директориями.
   - Пример:

```
Path path = Paths.get("example.txt");
List<String> lines = Files.readAllLines(path);
```

5. **Обработка исключений:**
   - При работе с файлами могут возникать исключения, такие как FileNotFoundException, IOException.
   - Для обработки исключений используется блок try-catch.

Пример работы с текстовым файлом:

```
import java.io.*;

public class FileExample {
    public static void main(String[] args) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter("example.txt"))) {
            writer.write("Привет, мир!");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Пример работы с бинарным файлом:

```
import java.io.*;

public class BinaryFileExample {
    public static void main(String[] args) {
        try (DataOutputStream dos = new DataOutputStream(new FileOutputStream("data.bin"))) {
            dos.writeInt(123);
            dos.writeDouble(45.67);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Пример работы с CSV-файлом на Java

CSV (Comma-Separated Values) — это текстовый формат для хранения табличных данных, где каждая строка представляет собой запись, а значения разделены запятыми (или другими разделителями).

Пример чтения и записи CSV-файла

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class CSVExample {
    public static void main(String[] args) {
```

```
        // Путь к CSV-файлу
        String csvFile = "data.csv";

        // Запись данных в CSV-файл
        writeCSV(csvFile);

        // Чтение данных из CSV-файла
        List<String[]> data = readCSV(csvFile);

        // Вывод данных на экран
        for (String[] row : data) {
            for (String cell : row) {
                System.out.print(cell + " ");
            }
            System.out.println();
        }
    }

    // Метод для записи данных в CSV-файл
    public static void writeCSV(String filePath) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
            // Заголовок CSV-файла
            writer.write("Name,Age,City");
            writer.newLine();

            // Данные
            writer.write("John,25,New York");
            writer.newLine();
            writer.write("Anna,30,London");
            writer.newLine();
            writer.write("Mike,22,Paris");
            writer.newLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Метод для чтения данных из CSV-файла
    public static List<String[]> readCSV(String filePath) {
        List<String[]> data = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                // Разделение строки на ячейки
                String[] row = line.split(",");
                data.add(row);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return data;
    }
}
```

Объяснение:

1. **Запись в CSV-файл:**
    - Используется BufferedWriter для записи строк в файл.
    - Каждая строка представляет собой запись, а значения разделены запятыми.
2. **Чтение из CSV-файла:**

- Используется BufferedReader для чтения строк из файла.
- Каждая строка разбивается на массив строк с помощью метода split(",").

Пример работы с JSON-файлом на Java

JSON (JavaScript Object Notation) — это текстовый формат для хранения и передачи структурированных данных. В Java для работы с JSON часто используются библиотеки, такие как **Jackson** или **Gson**.

Пример с использованием библиотеки Gson

1. Добавьте зависимость Gson в ваш проект (если используете Maven):

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.9</version>
</dependency>
```

2. Пример чтения и записи JSON-файла:

```
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.List;

class Person {
    String name;
    int age;
    String city;

    // Конструктор
    public Person(String name, int age, String city) {
        this.name = name;
        this.age = age;
        this.city = city;
    }

    // Для вывода объекта в виде строки
    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + ", city='" + city + "'}";
    }
}

public class JSONExample {
    public static void main(String[] args) {
        // Путь к JSON-файлу
        String jsonFile = "data.json";

        // Запись данных в JSON-файл
        writeJSON(jsonFile);

        // Чтение данных из JSON-файла
        List<Person> people = readJSON(jsonFile);
```

```
        // Вывод данных на экран
        for (Person person : people) {
            System.out.println(person);
        }
    }

    // Метод для записи данных в JSON-файл
    public static void writeJSON(String filePath) {
        List<Person> people = new ArrayList<>();
        people.add(new Person("John", 25, "New York"));
        people.add(new Person("Anna", 30, "London"));
        people.add(new Person("Mike", 22, "Paris"));

        Gson gson = new Gson();
        try (FileWriter writer = new FileWriter(filePath)) {
            gson.toJson(people, writer);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Метод для чтения данных из JSON-файла
    public static List<Person> readJSON(String filePath) {
        Gson gson = new Gson();
        try (FileReader reader = new FileReader(filePath)) {
            Type type = new TypeToken<ArrayList<Person>>() {}.getType();
            return gson.fromJson(reader, type);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Объяснение:

1. **Запись в JSON-файл:**
    - Используется библиотека Gson для преобразования списка объектов Person в JSON-строку.
    - JSON-строка записывается в файл с помощью FileWriter.
2. **Чтение из JSON-файла:**
    - Используется библиотека Gson для преобразования JSON-строки обратно в список объектов Person.
    - Для определения типа данных используется TypeToken.

Итог

- **CSV-файлы** удобны для хранения табличных данных и легко обрабатываются с помощью стандартных средств Java.
- **JSON-файлы** подходят для хранения сложных структур данных и требуют использования библиотек, таких как Gson или Jackson.

Эти примеры демонстрируют базовые операции работы с файлами в Java и могут быть расширены для решения более сложных задач.

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
   - Ознакомьтесь с классами и методами для работы с файлами.
   - Проанализируйте примеры кода.
2. **Реализуйте программу для работы с текстовыми файлами.**
   - Создайте текстовый файл и запишите в него данные.
   - Прочитайте данные из файла и выведите их на экран.
3. **Реализуйте программу для работы с бинарными файлами.**
   - Создайте бинарный файл и запишите в него данные.
   - Прочитайте данные из файла и выведите их на экран.
4. **Проведите тестирование.**
   - Проверьте корректность работы программ на различных файлах.
5. **Проанализируйте результаты.**
   - Сравните эффективность работы с текстовыми и бинарными файлами.

---

4. Варианты заданий

1. Создайте текстовый файл и запишите в него 10 строк текста.
2. Прочитайте текстовый файл и выведите количество строк в нем.
3. Создайте бинарный файл и запишите в него массив целых чисел.
4. Прочитайте бинарный файл и выведите сумму всех чисел.
5. Создайте программу для копирования текстового файла.
6. Создайте программу для копирования бинарного файла.
7. Реализуйте программу для поиска слова в текстовом файле.
8. Создайте программу для добавления текста в конец файла.
9. Реализуйте программу для удаления файла.
10. Создайте программу для переименования файла.
11. Реализуйте программу для чтения и записи CSV-файла.
12. Создайте программу для работы с JSON-файлом.
13. Реализуйте программу для шифрования текстового файла.
14. Создайте программу для дешифрования текстового файла.
15. Реализуйте программу для создания резервной копии файла.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием методов работы с файлами.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Какие классы используются для работы с текстовыми файлами в Java?
2. В чем разница между байтовыми и символьными потоками?
3. Как обрабатываются исключения при работе с файлами?

4. Какие преимущества предоставляет класс Files из пакета java.nio.file?
5. Как организовать чтение данных из бинарного файла?
6. Какие методы класса File используются для проверки свойств файла?
7. В чем преимущество использования буферизованных потоков?

Методические указания к лабораторной работе №18.1 по теме: "Алгоритмы сортировки. Алгоритмы внутренней сортировки"

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение алгоритмов внутренней сортировки, таких как сортировка подсчетом, вставкой, выбором, а также обменных сортировок и быстрой сортировки. В ходе выполнения работы студенты научатся:

- понимать принципы работы различных алгоритмов сортировки;
- реализовывать алгоритмы на языке Java;
- анализировать эффективность алгоритмов по времени и памяти;
- сравнивать производительность различных методов сортировки.

---

2. Теоретический материал

Алгоритмы сортировки — это методы упорядочивания элементов в списке или массиве по возрастанию или убыванию. Внутренняя сортировка предполагает, что все данные помещаются в оперативную память.

Сортировка подсчетом (Counting Sort):

- Применяется для сортировки целых чисел в ограниченном диапазоне.
- Время работы: $O(n + k)$, где n — количество элементов, k — диапазон значений.
- Не требует сравнения элементов.
- Пример: сортировка массива чисел от 0 до 100.

Сортировка вставкой (Insertion Sort):

- Элементы поочередно вставляются в отсортированную часть массива.
- Время работы: $O(n^2)$ в худшем случае, $O(n)$ — в лучшем (для почти отсортированных данных).
- Пример: сортировка массива строк.

Сортировка выбором (Selection Sort):

- На каждом шаге выбирается минимальный элемент и помещается в начало массива.
- Время работы: $O(n^2)$.
- Пример: сортировка массива вещественных чисел.

Обменные сортировки:

- **Сортировка пузырьком (Bubble Sort):**
  - Последовательно сравниваются соседние элементы и меняются местами, если они находятся в неправильном порядке.
  - Время работы: $O(n^2)$.
- **Шейкерная сортировка (Cocktail Shaker Sort):**

- Улучшенная версия пузырьковой сортировки, которая проходит массив в обоих направлениях.
- Время работы: $O(n^2)$.

Быстрая сортировка (Quick Sort):

- Использует стратегию "разделяй и властвуй".
- Выбирается опорный элемент, и массив разделяется на две части: элементы меньше опорного и больше опорного.
- Время работы: $O(n \log n)$ в среднем случае, $O(n^2)$ — в худшем.
- Пример: сортировка массива целых чисел.

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
   - Ознакомьтесь с принципами работы каждого алгоритма сортировки.
   - Проанализируйте временную и пространственную сложность алгоритмов.
2. **Реализуйте алгоритмы на языке Java.**
   - Создайте класс SortingAlgorithms, в котором реализуйте методы для каждого алгоритма сортировки.
   - Напишите тестовый класс для проверки корректности работы алгоритмов.
3. **Проведите тестирование.**
   - Создайте массивы данных различного размера (например, 10, 100, 1000 элементов).
   - Замерьте время выполнения каждого алгоритма.
4. **Проанализируйте результаты.**
   - Сравните эффективность алгоритмов по времени выполнения.
   - Сделайте выводы о применимости каждого алгоритма в зависимости от размера данных.

---

4. Варианты заданий

1. Реализуйте сортировку подсчетом для массива целых чисел в диапазоне от 0 до 100.
2. Реализуйте сортировку вставкой для массива строк.
3. Реализуйте сортировку выбором для массива вещественных чисел.
4. Реализуйте сортировку пузырьком для массива целых чисел.
5. Реализуйте шейкерную сортировку для массива целых чисел.
6. Реализуйте быструю сортировку для массива целых чисел.
7. Сравните время выполнения сортировки вставкой и сортировки выбором для массива из 1000 элементов.
8. Реализуйте сортировку подсчетом для массива целых чисел в диапазоне от -50 до 50.
9. Реализуйте сортировку вставкой для массива символов.
10. Реализуйте сортировку выбором для массива строк.
11. Реализуйте сортировку пузырьком для массива вещественных чисел.
12. Реализуйте шейкерную сортировку для массива строк.
13. Реализуйте быструю сортировку для массива вещественных чисел.

14. Сравните время выполнения сортировки пузырьком и шейкерной сортировки для массива из 500 элементов.
15. Реализуйте быструю сортировку для массива строк.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием алгоритмов.
4. Листинг реализованных программ на Java.
5. Результаты тестирования (время выполнения для каждого алгоритма).
6. Выводы по работе.

---

6. Контрольные вопросы

1. В чем преимущество сортировки подсчетом перед другими алгоритмами?
2. Какие ограничения накладываются на данные для применения сортировки подсчетом?
3. Почему сортировка вставкой эффективна для почти отсортированных данных?
4. В чем разница между сортировкой пузырьком и шейкерной сортировкой?
5. Как выбирается опорный элемент в быстрой сортировке?
6. Какие недостатки есть у быстрой сортировки?
7. В каких случаях сортировка выбором будет работать быстрее сортировки вставкой?

Методические указания к лабораторной работе № 18.2 по теме: “Алгоритмы внешней сортировки”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение алгоритмов внешней сортировки, которые используются для обработки данных, не помещающихся в оперативную память. В ходе выполнения работы студенты научатся:

- понимать принципы работы алгоритмов внешней сортировки;
- реализовывать алгоритмы на языке Java;
- анализировать эффективность алгоритмов по времени и объему используемой памяти;
- работать с большими объемами данных, хранящимися на внешних носителях.

---

2. Теоретический материал

Алгоритмы внешней сортировки применяются для сортировки данных, которые не помещаются в оперативную память и хранятся на внешних носителях (например, на жестком диске). Основная задача таких алгоритмов — минимизировать количество операций чтения и записи, так как они значительно медленнее операций в оперативной памяти.

Основные этапы внешней сортировки:

1. **Разделение данных на блоки:**
    - Данные разбиваются на блоки, которые помещаются в оперативную память.
    - Каждый блок сортируется с использованием внутренних алгоритмов сортировки (например, быстрой сортировки).
2. **Слияние отсортированных блоков:**
    - Отсортированные блоки объединяются в один отсортированный файл.

Основные алгоритмы внешней сортировки:

- **Сортировка слиянием (Merge Sort):**
    - Данные разбиваются на блоки, каждый блок сортируется, а затем блоки объединяются.
    - Время работы: O(n log n).
- **Многофазная сортировка (Polyphase Merge Sort):**
    - Улучшенная версия сортировки слиянием, которая минимизирует количество операций чтения и записи.
- **Сортировка с использованием B-деревьев:**
    - Данные хранятся в структуре B-дерева, что позволяет эффективно сортировать и обрабатывать большие объемы данных.

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
    - Ознакомьтесь с принципами работы алгоритмов внешней сортировки.
    - Проанализируйте временную и пространственную сложность алгоритмов.
2. **Реализуйте алгоритм сортировки слиянием на языке Java.**
    - Создайте класс ExternalSort, в котором реализуйте методы для чтения данных из файла, сортировки блоков и слияния блоков.
    - Напишите тестовый класс для проверки корректности работы алгоритма.
3. **Проведите тестирование.**
    - Создайте файл с большим объемом данных (например, 1 000 000 чисел).
    - Замерьте время выполнения алгоритма.
4. **Проанализируйте результаты.**
    - Сравните эффективность алгоритма по времени выполнения и объему используемой памяти.
    - Сделайте выводы о применимости алгоритма для различных объемов данных.

---

4. Варианты заданий

1. Реализуйте сортировку слиянием для файла, содержащего 1 000 000 целых чисел.
2. Реализуйте сортировку слиянием для файла, содержащего 500 000 вещественных чисел.
3. Реализуйте сортировку слиянием для файла, содержащего 100 000 строк.
4. Реализуйте многофазную сортировку для файла, содержащего 1 000 000 целых чисел.
5. Реализуйте многофазную сортировку для файла, содержащего 500 000 вещественных чисел.
6. Реализуйте сортировку с использованием B-деревьев для файла, содержащего 1 000 000 целых чисел.
7. Сравните время выполнения сортировки слиянием и многофазной сортировки для файла с 1 000 000 элементов.
8. Реализуйте сортировку слиянием для файла, содержащего 2 000 000 целых чисел.
9. Реализуйте сортировку слиянием для файла, содержащего 1 000 000 символов.
10. Реализуйте сортировку слиянием для файла, содержащего 500 000 строк.
11. Реализуйте многофазную сортировку для файла, содержащего 2 000 000 целых чисел.
12. Реализуйте многофазную сортировку для файла, содержащего 1 000 000 символов.
13. Реализуйте сортировку с использованием B-деревьев для файла, содержащего 500 000 строк.
14. Сравните время выполнения сортировки слиянием и сортировки с использованием B-деревьев для файла с 1 000 000 элементов.
15. Реализуйте сортировку слиянием для файла, содержащего 10 000 000 целых чисел.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием алгоритмов.
4. Листинг реализованных программ на Java.
5. Результаты тестирования (время выполнения для каждого алгоритма).

6. Выводы по работе.

---

6. Контрольные вопросы

1. В чем основное отличие внешней сортировки от внутренней?
2. Какие этапы включает в себя алгоритм сортировки слиянием?
3. Почему многофазная сортировка эффективнее обычной сортировки слиянием?
4. Какие преимущества имеет использование B-деревьев для внешней сортировки?
5. Как минимизировать количество операций чтения и записи при внешней сортировке?
6. В каких случаях целесообразно использовать внешнюю сортировку?
7. Какие ограничения накладываются на данные для применения внешней сортировки?

Методические указания к лабораторной работе № 19 по теме “Алгоритмы поиска”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение алгоритмов поиска данных в массивах и коллекциях. В ходе выполнения работы студенты научатся:

- понимать принципы работы различных алгоритмов поиска;
- реализовывать алгоритмы поиска на языке Java;
- анализировать эффективность алгоритмов по времени и памяти;
- применять алгоритмы поиска для решения практических задач.

---

2. Теоретический материал

Алгоритмы поиска — это методы нахождения элемента в структуре данных (например, в массиве или списке). В зависимости от структуры данных и условий задачи используются различные алгоритмы поиска.

Основные алгоритмы поиска:

Линейный поиск (Linear Search):

Простейший алгоритм поиска, который последовательно проверяет каждый элемент массива.

Время работы: O(n), где n — количество элементов.

Применяется для неотсортированных данных.

Пример реализации на Java:

```
public static int linearSearch(int[] array, int target) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == target) {
            return i; // Возвращаем индекс найденного элемента
        }
    }
    return -1; // Элемент не найден
}
```

Бинарный поиск (Binary Search):

Эффективный алгоритм поиска для отсортированных массивов.

Работает по принципу "разделяй и властвуй": массив делится на две части, и поиск продолжается в той части, где может находиться искомый элемент.

Время работы: O(log n).

Пример реализации на Java:

```
public static int binarySearch(int[] array, int target) {
```

```
    int left = 0;

    int right = array.length - 1;

    while (left <= right) {

        int mid = left + (right - left) / 2;

        if (array[mid] == target) {

            return mid;

        }

        if (array[mid] < target) {

            left = mid + 1;

        } else {

            right = mid - 1;

        }

    }

    return -1; // Элемент не найден

}
```

Поиск в хэш-таблицах (Hash Table Search):

Используется для поиска данных в структурах, основанных на хэш-функциях (например, HashMap в Java).

Время работы: O(1) в среднем случае.

Пример использования HashMap:

```
import java.util.HashMap;


public class HashSearchExample {

    public static void main(String[] args) {

        HashMap<String, Integer> map = new HashMap<>();

        map.put("Alice", 25);

        map.put("Bob", 30);

        map.put("Charlie", 35);


        // Поиск по ключу

        if (map.containsKey("Bob")) {

            System.out.println("Возраст Bob: " + map.get("Bob"));

        }

    }
```

```
}
```

Поиск в бинарных деревьях (Binary Tree Search):

Применяется для поиска в бинарных деревьях поиска (BST).

Время работы: O(log n) в среднем случае, O(n) — в худшем (если дерево несбалансированное).

Пример реализации поиска в BST:

```
class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
        left = right = null;
    }
}

class BinarySearchTree {
    Node root;

    // Поиск элемента в BST
    public Node search(Node root, int key) {
        if (root == null || root.key == key) {
            return root;
        }
        if (root.key < key) {
            return search(root.right, key);
        }
        return search(root.left, key);
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
    - Ознакомьтесь с принципами работы алгоритмов поиска.
    - Проанализируйте временную и пространственную сложность алгоритмов.

2. **Реализуйте алгоритмы поиска на языке Java.**
   - Создайте класс SearchAlgorithms, в котором реализуйте методы для линейного и бинарного поиска.
   - Напишите тестовый класс для проверки корректности работы алгоритмов.
3. **Проведите тестирование.**
   - Создайте массивы данных различного размера (например, 100, 1000, 10000 элементов).
   - Замерьте время выполнения каждого алгоритма.
4. **Проанализируйте результаты.**
   - Сравните эффективность алгоритмов по времени выполнения.
   - Сделайте выводы о применимости каждого алгоритма в зависимости от размера данных.

---

4. Варианты заданий

1. Реализуйте линейный поиск для массива целых чисел.
2. Реализуйте бинарный поиск для отсортированного массива целых чисел.
3. Сравните время выполнения линейного и бинарного поиска для массива из 1000 элементов.
4. Реализуйте поиск в хэш-таблице с использованием HashMap.
5. Реализуйте поиск в бинарном дереве поиска (BST).
6. Реализуйте линейный поиск для массива строк.
7. Реализуйте бинарный поиск для отсортированного массива строк.
8. Сравните время выполнения линейного и бинарного поиска для массива из 10000 элементов.
9. Реализуйте поиск в хэш-таблице для хранения и поиска данных о студентах.
10. Реализуйте поиск в бинарном дереве для хранения и поиска данных о книгах.
11. Реализуйте линейный поиск для массива вещественных чисел.
12. Реализуйте бинарный поиск для отсортированного массива вещественных чисел.
13. Сравните время выполнения линейного и бинарного поиска для массива из 5000 элементов.
14. Реализуйте поиск в хэш-таблице для хранения и поиска данных о сотрудниках.
15. Реализуйте поиск в бинарном дереве для хранения и поиска данных о товарах.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием алгоритмов поиска.
4. Листинг реализованных программ на Java.
5. Результаты тестирования (время выполнения для каждого алгоритма).
6. Выводы по работе.

---

6. Контрольные вопросы

1. В чем разница между линейным и бинарным поиском?
2. Какие ограничения накладываются на данные для применения бинарного поиска?
3. Почему бинарный поиск работает быстрее линейного?

4. Какие преимущества предоставляет использование хэш-таблиц для поиска?
5. В каких случаях целесообразно использовать бинарные деревья поиска?
6. Какие недостатки есть у линейного поиска?
7. Какова временная сложность поиска в хэш-таблице?

Методические указания к лабораторной работе № 20.1 по теме: “Линейные структуры данных. Стек ”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение стека как одной из базовых линейных структур данных. В ходе выполнения работы студенты научатся:

- понимать принципы работы стека;
- реализовывать стек на основе массивов и ссылочных структур;
- применять стек для решения практических задач;
- анализировать эффективность различных реализаций стека.

---

2. Теоретический материал

Стек — это линейная структура данных, которая работает по принципу **LIFO (Last In, First Out)**: последний элемент, добавленный в стек, будет извлечен первым. Основные операции со стеком:

- **push** — добавление элемента на вершину стека;
- **pop** — удаление элемента с вершины стека;
- **peek** — получение элемента с вершины стека без его удаления;
- **isEmpty** — проверка, пуст ли стек.

Реализация стека на массивах данных:

- Используется массив фиксированного или динамического размера.
- Преимущества: простота реализации, высокая производительность.
- Недостатки: ограниченный размер (если используется массив фиксированного размера).

Пример реализации стека на массиве:

```
public class ArrayStack {
    private int maxSize; // Максимальный размер стека
    private int[] stackArray;
    private int top; // Индекс вершины стека

    public ArrayStack(int size) {
        this.maxSize = size;
        this.stackArray = new int[maxSize];
        this.top = -1; // Стек пуст
    }

    // Добавление элемента на вершину стека
    public void push(int value) {
        if (top < maxSize - 1) {
            stackArray[++top] = value;
        } else {
            System.out.println("Стек переполнен!");
        }
    }
```

```
    // Удаление элемента с вершины стека
    public int pop() {
        if (top >= 0) {
            return stackArray[top--];
        } else {
            System.out.println("Стек пуст!");
            return -1;
        }
    }

    // Получение элемента с вершины стека без удаления
    public int peek() {
        if (top >= 0) {
            return stackArray[top];
        } else {
            System.out.println("Стек пуст!");
            return -1;
        }
    }

    // Проверка, пуст ли стек
    public boolean isEmpty() {
        return (top == -1);
    }
}
```

Реализация стека на ссылочных структурах:

- Используется связный список, где каждый элемент (узел) содержит данные и ссылку на следующий элемент.
- Преимущества: динамический размер, отсутствие ограничений на количество элементов.
- Недостатки: более сложная реализация, дополнительные затраты памяти на хранение ссылок.

Пример реализации стека на связном списке:

```
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class LinkedStack {
    private Node top; // Вершина стека

    public LinkedStack() {
        this.top = null;
    }

    // Добавление элемента на вершину стека
    public void push(int value) {
        Node newNode = new Node(value);
        newNode.next = top;
        top = newNode;
```

```
    }

    // Удаление элемента с вершины стека
    public int pop() {
        if (top != null) {
            int value = top.data;
            top = top.next;
            return value;
        } else {
            System.out.println("Стек пуст!");
            return -1;
        }
    }

    // Получение элемента с вершины стека без удаления
    public int peek() {
        if (top != null) {
            return top.data;
        } else {
            System.out.println("Стек пуст!");
            return -1;
        }
    }

    // Проверка, пуст ли стек
    public boolean isEmpty() {
        return (top == null);
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
   - Ознакомьтесь с принципами работы стека.
   - Проанализируйте примеры реализации стека на массиве и связном списке.
2. **Реализуйте стек на массиве.**
   - Создайте класс ArrayStack и реализуйте основные операции (push, pop, peek, isEmpty).
   - Напишите тестовый класс для проверки корректности работы стека.
3. **Реализуйте стек на связном списке.**
   - Создайте класс LinkedStack и реализуйте основные операции (push, pop, peek, isEmpty).
   - Напишите тестовый класс для проверки корректности работы стека.
4. **Проведите тестирование.**
   - Сравните производительность двух реализаций стека.
5. **Проанализируйте результаты.**
   - Сделайте выводы о преимуществах и недостатках каждой реализации.

---

4. Варианты заданий

1. Реализуйте стек на массиве и добавьте метод для вывода всех элементов стека.
2. Реализуйте стек на связном списке и добавьте метод для подсчета количества элементов в стеке.
3. Используйте стек для проверки корректности расстановки скобок в выражении.
4. Реализуйте стек на массиве с динамическим изменением размера.

5. Используйте стек для преобразования инфиксной записи выражения в постфиксную.
6. Реализуйте стек на связном списке и добавьте метод для поиска элемента в стеке.
7. Используйте стек для реверсирования строки.
8. Реализуйте стек на массиве и добавьте метод для очистки стека.
9. Используйте стек для вычисления значения постфиксного выражения.
10. Реализуйте стек на связном списке и добавьте метод для копирования стека.
11. Используйте стек для проверки, является ли строка палиндромом.
12. Реализуйте стек на массиве и добавьте метод для поиска минимального элемента в стеке.
13. Используйте стек для моделирования работы процессора (выполнение команд).
14. Реализуйте стек на связном списке и добавьте метод для сортировки элементов стека.
15. Используйте стек для решения задачи "Ханойские башни".

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием стека и его реализаций.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Что такое стек и по какому принципу он работает?
2. Какие основные операции поддерживает стек?
3. В чем разница между реализацией стека на массиве и на связном списке?
4. Какие преимущества и недостатки имеет реализация стека на массиве?
5. Какие преимущества и недостатки имеет реализация стека на связном списке?
6. В каких задачах целесообразно использовать стек?
7. Как можно реализовать стек с динамическим изменением размера?

Методические указания к лабораторной работе № 20.2 по теме “Линейные структуры данных. Очередь”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение очереди как одной из базовых линейных структур данных. В ходе выполнения работы студенты научатся:

- понимать принципы работы очереди;
- реализовывать очередь на основе массивов и ссылочных структур;
- применять очередь для решения практических задач;
- анализировать эффективность различных реализаций очереди.

---

2. Теоретический материал

Очередь — это линейная структура данных, которая работает по принципу **FIFO (First In, First Out)**: первый элемент, добавленный в очередь, будет извлечен первым. Основные операции с очередью:

- **enqueue** — добавление элемента в конец очереди;
- **dequeue** — удаление элемента из начала очереди;
- **peek** — получение элемента из начала очереди без его удаления;
- **isEmpty** — проверка, пуста ли очередь.

Реализация очереди на массивах данных:

- Используется массив фиксированного или динамического размера.
- Преимущества: простота реализации, высокая производительность.
- Недостатки: ограниченный размер (если используется массив фиксированного размера).

Пример реализации очереди на массиве:

```
public class ArrayQueue {
    private int maxSize; // Максимальный размер очереди
    private int[] queueArray;
    private int front; // Индекс начала очереди
    private int rear; // Индекс конца очереди
    private int size; // Текущий размер очереди

    public ArrayQueue(int size) {
        this.maxSize = size;
        this.queueArray = new int[maxSize];
        this.front = 0;
        this.rear = -1;
        this.size = 0;
    }

    // Добавление элемента в конец очереди
    public void enqueue(int value) {
        if (size < maxSize) {
            rear = (rear + 1) % maxSize; // Кольцевой буфер
```

```
            queueArray[rear] = value;
            size++;
        } else {
            System.out.println("Очередь переполнена!");
        }
    }

    // Удаление элемента из начала очереди
    public int dequeue() {
        if (size > 0) {
            int value = queueArray[front];
            front = (front + 1) % maxSize; // Кольцевой буфер
            size--;
            return value;
        } else {
            System.out.println("Очередь пуста!");
            return -1;
        }
    }

    // Получение элемента из начала очереди без удаления
    public int peek() {
        if (size > 0) {
            return queueArray[front];
        } else {
            System.out.println("Очередь пуста!");
            return -1;
        }
    }

    // Проверка, пуста ли очередь
    public boolean isEmpty() {
        return (size == 0);
    }
}
```

Реализация очереди на ссылочных структурах:

- Используется связный список, где каждый элемент (узел) содержит данные и ссылку на следующий элемент.
- Преимущества: динамический размер, отсутствие ограничений на количество элементов.
- Недостатки: более сложная реализация, дополнительные затраты памяти на хранение ссылок.

Пример реализации очереди на связном списке:

```
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class LinkedQueue {
    private Node front; // Начало очереди
    private Node rear; // Конец очереди
```

```
    public LinkedQueue() {
        this.front = null;
        this.rear = null;
    }

    // Добавление элемента в конец очереди
    public void enqueue(int value) {
        Node newNode = new Node(value);
        if (rear == null) { // Если очередь пуста
            front = rear = newNode;
        } else {
            rear.next = newNode;
            rear = newNode;
        }
    }

    // Удаление элемента из начала очереди
    public int dequeue() {
        if (front != null) {
            int value = front.data;
            front = front.next;
            if (front == null) { // Если очередь стала пустой
                rear = null;
            }
            return value;
        } else {
            System.out.println("Очередь пуста!");
            return -1;
        }
    }

    // Получение элемента из начала очереди без удаления
    public int peek() {
        if (front != null) {
            return front.data;
        } else {
            System.out.println("Очередь пуста!");
            return -1;
        }
    }

    // Проверка, пуста ли очередь
    public boolean isEmpty() {
        return (front == null);
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
   - Ознакомьтесь с принципами работы очереди.
   - Проанализируйте примеры реализации очереди на массиве и связном списке.
2. **Реализуйте очередь на массиве.**
   - Создайте класс ArrayQueue и реализуйте основные операции (enqueue, dequeue, peek, isEmpty).
   - Напишите тестовый класс для проверки корректности работы очереди.
3. **Реализуйте очередь на связном списке.**

- Создайте класс LinkedQueue и реализуйте основные операции (enqueue, dequeue, peek, isEmpty).
- Напишите тестовый класс для проверки корректности работы очереди.

4. **Проведите тестирование.**
    - Сравните производительность двух реализаций очереди.
5. **Проанализируйте результаты.**
    - Сделайте выводы о преимуществах и недостатках каждой реализации.

---

4. Варианты заданий

1. Реализуйте очередь на массиве и добавьте метод для вывода всех элементов очереди.
2. Реализуйте очередь на связном списке и добавьте метод для подсчета количества элементов в очереди.
3. Используйте очередь для моделирования работы принтера (задачи на печать).
4. Реализуйте очередь на массиве с динамическим изменением размера.
5. Используйте очередь для реализации алгоритма поиска в ширину (BFS).
6. Реализуйте очередь на связном списке и добавьте метод для поиска элемента в очереди.
7. Используйте очередь для моделирования работы call-центра.
8. Реализуйте очередь на массиве и добавьте метод для очистки очереди.
9. Используйте очередь для реализации системы обработки заказов.
10. Реализуйте очередь на связном списке и добавьте метод для копирования очереди.
11. Используйте очередь для моделирования работы банковского терминала.
12. Реализуйте очередь на массиве и добавьте метод для поиска минимального элемента в очереди.
13. Используйте очередь для реализации системы управления задачами.
14. Реализуйте очередь на связном списке и добавьте метод для сортировки элементов очереди.
15. Используйте очередь для моделирования работы светофора.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием очереди и ее реализаций.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Что такое очередь и по какому принципу она работает?
2. Какие основные операции поддерживает очередь?
3. В чем разница между реализацией очереди на массиве и на связном списке?
4. Какие преимущества и недостатки имеет реализация очереди на массиве?
5. Какие преимущества и недостатки имеет реализация очереди на связном списке?
6. В каких задачах целесообразно использовать очередь?

7. Как можно реализовать очередь с динамическим изменением размера?

Методические указания к лабораторной работе № 20.3 по теме “Линейные кольцевые структуры”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение кольцевых структур данных. В ходе выполнения работы студенты научатся:

- понимать принципы работы кольцевых структур;
- реализовывать кольцевые структуры на основе массивов и ссылочных структур;
- применять кольцевые структуры для решения практических задач;
- анализировать эффективность различных реализаций кольцевых структур.

---

2. Теоретический материал

Кольцевая структура данных — это линейная структура, в которой последний элемент связан с первым, образуя замкнутый цикл. Такие структуры часто используются в задачах, где требуется циклическое перемещение по данным.

Основные операции с кольцевой структурой:

- **Добавление элемента** — вставка элемента в структуру.
- **Удаление элемента** — удаление элемента из структуры.
- **Поиск элемента** — нахождение элемента в структуре.
- **Обход структуры** — циклическое перемещение по элементам.

Реализация кольцевой структуры на массиве данных:

- Используется массив фиксированного или динамического размера.
- Преимущества: простота реализации, высокая производительность.
- Недостатки: ограниченный размер (если используется массив фиксированного размера).

Пример реализации кольцевой структуры на массиве:

```
public class CircularArray {
    private int maxSize; // Максимальный размер структуры
    private int[] array;
    private int head; // Индекс начала
    private int tail; // Индекс конца
    private int size; // Текущий размер

    public CircularArray(int size) {
        this.maxSize = size;
        this.array = new int[maxSize];
        this.head = 0;
        this.tail = -1;
        this.size = 0;
    }

    // Добавление элемента в структуру
    public void add(int value) {
        if (size < maxSize) {
```

```
            tail = (tail + 1) % maxSize; // Кольцевой буфер
            array[tail] = value;
            size++;
        } else {
            System.out.println("Структура переполнена!");
        }
    }

    // Удаление элемента из структуры
    public int remove() {
        if (size > 0) {
            int value = array[head];
            head = (head + 1) % maxSize; // Кольцевой буфер
            size--;
            return value;
        } else {
            System.out.println("Структура пуста!");
            return -1;
        }
    }

    // Получение элемента без удаления
    public int peek() {
        if (size > 0) {
            return array[head];
        } else {
            System.out.println("Структура пуста!");
            return -1;
        }
    }

    // Проверка, пуста ли структура
    public boolean isEmpty() {
        return (size == 0);
    }
}
```

Реализация кольцевой структуры на ссылочных структурах:

- Используется связный список, где последний элемент ссылается на первый, образуя кольцо.
- Преимущества: динамический размер, отсутствие ограничений на количество элементов.
- Недостатки: более сложная реализация, дополнительные затраты памяти на хранение ссылок.

Пример реализации кольцевой структуры на связном списке:

```
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class CircularLinkedList {
    private Node head; // Начало структуры
```

```
    private Node tail; // Конец структуры

    public CircularLinkedList() {
        this.head = null;
        this.tail = null;
    }

    // Добавление элемента в структуру
    public void add(int value) {
        Node newNode = new Node(value);
        if (head == null) { // Если структура пуста
            head = tail = newNode;
            tail.next = head; // Замыкаем кольцо
        } else {
            tail.next = newNode;
            tail = newNode;
            tail.next = head; // Замыкаем кольцо
        }
    }

    // Удаление элемента из структуры
    public int remove() {
        if (head != null) {
            int value = head.data;
            if (head == tail) { // Если в структуре один элемент
                head = tail = null;
            } else {
                head = head.next;
                tail.next = head; // Замыкаем кольцо
            }
            return value;
        } else {
            System.out.println("Структура пуста!");
            return -1;
        }
    }

    // Получение элемента без удаления
    public int peek() {
        if (head != null) {
            return head.data;
        } else {
            System.out.println("Структура пуста!");
            return -1;
        }
    }

    // Проверка, пуста ли структура
    public boolean isEmpty() {
        return (head == null);
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
    - Ознакомьтесь с принципами работы кольцевых структур.
    - Проанализируйте примеры реализации кольцевых структур на массиве и связном списке.
2. **Реализуйте кольцевую структуру на массиве.**

    - Создайте класс CircularArray и реализуйте основные операции (add, remove, peek, isEmpty).
    - Напишите тестовый класс для проверки корректности работы структуры.
3. **Реализуйте кольцевую структуру на связном списке.**
    - Создайте класс CircularLinkedList и реализуйте основные операции (add, remove, peek, isEmpty).
    - Напишите тестовый класс для проверки корректности работы структуры.
4. **Проведите тестирование.**
    - Сравните производительность двух реализаций кольцевых структур.
5. **Проанализируйте результаты.**
    - Сделайте выводы о преимуществах и недостатках каждой реализации.

---

4. Варианты заданий

1. Реализуйте кольцевую структуру на массиве и добавьте метод для вывода всех элементов.
2. Реализуйте кольцевую структуру на связном списке и добавьте метод для подсчета количества элементов.
3. Используйте кольцевую структуру для моделирования работы карусели.
4. Реализуйте кольцевую структуру на массиве с динамическим изменением размера.
5. Используйте кольцевую структуру для реализации алгоритма Round Robin.
6. Реализуйте кольцевую структуру на связном списке и добавьте метод для поиска элемента.
7. Используйте кольцевую структуру для моделирования работы циклического буфера.
8. Реализуйте кольцевую структуру на массиве и добавьте метод для очистки структуры.
9. Используйте кольцевую структуру для реализации системы управления задачами.
10. Реализуйте кольцевую структуру на связном списке и добавьте метод для копирования структуры.
11. Используйте кольцевую структуру для моделирования работы светофора.
12. Реализуйте кольцевую структуру на массиве и добавьте метод для поиска минимального элемента.
13. Используйте кольцевую структуру для реализации системы обработки заказов.
14. Реализуйте кольцевую структуру на связном списке и добавьте метод для сортировки элементов.
15. Используйте кольцевую структуру для моделирования работы циклического планировщика.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием кольцевых структур и их реализаций.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Что такое кольцевая структура данных и в чем ее особенность?
2. Какие основные операции поддерживает кольцевая структура?
3. В чем разница между реализацией кольцевой структуры на массиве и на связном списке?
4. Какие преимущества и недостатки имеет реализация кольцевой структуры на массиве?
5. Какие преимущества и недостатки имеет реализация кольцевой структуры на связном списке?
6. В каких задачах целесообразно использовать кольцевые структуры?
7. Как можно реализовать кольцевую структуру с динамическим изменением размера?

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение структуры данных "куча" (heap) и алгоритма пирамидальной сортировки (heapsort). В ходе выполнения работы студенты научатся:

- понимать принципы работы кучи;
- реализовывать кучу на языке Java;
- применять пирамидальную сортировку для упорядочивания данных;
- анализировать эффективность алгоритма по времени и памяти.

---

2. Теоретический материал

Куча (Heap) — это специализированная структура данных, представляющая собой двоичное дерево, которое удовлетворяет свойству кучи:

- **Максимальная куча (Max Heap):** значение каждого узла больше или равно значениям его дочерних узлов.
- **Минимальная куча (Min Heap):** значение каждого узла меньше или равно значениям его дочерних узлов.

Основные операции с кучей:

- **Добавление элемента** — вставка элемента в кучу с сохранением свойства кучи.
- **Удаление корневого элемента** — извлечение корневого элемента (максимума или минимума) и восстановление свойства кучи.
- **Построение кучи** — преобразование массива в кучу.

Пирамидальная сортировка (Heapsort) — это алгоритм сортировки, который использует структуру кучи для упорядочивания данных. Алгоритм состоит из двух этапов:

1. **Построение кучи** из исходного массива.
2. **Извлечение элементов** из кучи и их размещение в отсортированном порядке.

Временная сложность:

- Построение кучи: O(n).
- Извлечение элементов: O(n log n).
- Общая сложность: O(n log n).

Пример реализации кучи и пирамидальной сортировки на Java:

```
public class HeapSort {
    // Просеивание элемента вниз для восстановления свойства кучи
    private static void heapify(int[] array, int n, int i) {
        int largest = i; // Инициализируем наибольший элемент как корень
```

```
        int left = 2 * i + 1; // Левый дочерний элемент
        int right = 2 * i + 2; // Правый дочерний элемент

        // Если левый дочерний элемент больше корня
        if (left < n && array[left] > array[largest]) {
            largest = left;
        }

        // Если правый дочерний элемент больше корня
        if (right < n && array[right] > array[largest]) {
            largest = right;
        }

        // Если наибольший элемент не корень
        if (largest != i) {
            int swap = array[i];
            array[i] = array[largest];
            array[largest] = swap;

            // Рекурсивно просеиваем затронутое поддерево
            heapify(array, n, largest);
        }
    }

    // Основная функция для выполнения пирамидальной сортировки
    public static void heapSort(int[] array) {
        int n = array.length;

        // Построение кучи (перегруппировка массива)
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(array, n, i);
        }

        // Извлечение элементов из кучи
        for (int i = n - 1; i > 0; i--) {
            // Перемещаем текущий корень в конец
            int temp = array[0];
            array[0] = array[i];
            array[i] = temp;

            // Вызываем heapify на уменьшенной куче
            heapify(array, i, 0);
        }
    }

    // Вспомогательная функция для вывода массива
    public static void printArray(int[] array) {
        for (int i : array) {
            System.out.print(i + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int[] array = {12, 11, 13, 5, 6, 7};
        System.out.println("Исходный массив:");
        printArray(array);

        heapSort(array);

        System.out.println("Отсортированный массив:");
        printArray(array);
```

```
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
   - Ознакомьтесь с принципами работы кучи и пирамидальной сортировки.
   - Проанализируйте пример реализации на Java.
2. **Реализуйте кучу и пирамидальную сортировку.**
   - Создайте класс HeapSort и реализуйте методы heapify и heapSort.
   - Напишите тестовый класс для проверки корректности работы алгоритма.
3. **Проведите тестирование.**
   - Создайте массивы данных различного размера (например, 10, 100, 1000 элементов).
   - Замерьте время выполнения пирамидальной сортировки.
4. **Проанализируйте результаты.**
   - Сравните эффективность пирамидальной сортировки с другими алгоритмами (например, быстрой сортировкой).
   - Сделайте выводы о применимости алгоритма.

---

4. Варианты заданий

1. Реализуйте пирамидальную сортировку для массива целых чисел.
2. Реализуйте минимальную кучу и отсортируйте массив по убыванию.
3. Сравните время выполнения пирамидальной сортировки и быстрой сортировки для массива из 1000 элементов.
4. Реализуйте кучу для хранения и обработки данных о студентах (по оценкам).
5. Используйте кучу для нахождения k-го наибольшего элемента в массиве.
6. Реализуйте пирамидальную сортировку для массива строк.
7. Реализуйте кучу для хранения и обработки данных о товарах (по цене).
8. Используйте кучу для реализации приоритетной очереди.
9. Реализуйте пирамидальную сортировку для массива вещественных чисел.
10. Реализуйте кучу для хранения и обработки данных о книгах (по году издания).
11. Используйте кучу для нахождения k-го наименьшего элемента в массиве.
12. Реализуйте пирамидальную сортировку для массива символов.
13. Реализуйте кучу для хранения и обработки данных о сотрудниках (по зарплате).
14. Используйте кучу для реализации системы управления задачами.
15. Реализуйте пирамидальную сортировку для массива из 10 000 элементов и замерьте время выполнения.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием кучи и пирамидальной сортировки.
4. Листинг реализованных программ на Java.
5. Результаты тестирования (время выполнения для различных массивов).
6. Выводы по работе.

6. Контрольные вопросы

1. Что такое куча и какое свойство она должна удовлетворять?
2. В чем разница между максимальной и минимальной кучей?
3. Какие основные операции поддерживает куча?
4. Какова временная сложность пирамидальной сортировки?
5. В чем преимущества пирамидальной сортировки перед другими алгоритмами?
6. Какие недостатки есть у пирамидальной сортировки?
7. В каких задачах целесообразно использовать кучу?

Методические указания к лабораторной работе № 22.1 по теме: “Бинарные деревья. Дерево поиска”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение бинарных деревьев и бинарных деревьев поиска (BST). В ходе выполнения работы студенты научатся:

- понимать принципы работы бинарных деревьев и BST;
- реализовывать BST на языке Java;
- выполнять основные операции с BST: вставка, удаление, поиск;
- анализировать эффективность операций с BST.

---

2. Теоретический материал

Бинарное дерево — это структура данных, в которой каждый узел имеет не более двух дочерних узлов (левый и правый).

Бинарное дерево поиска (BST) — это бинарное дерево, которое удовлетворяет следующим свойствам:

- Для каждого узла все элементы в левом поддереве меньше значения узла.
- Для каждого узла все элементы в правом поддереве больше значения узла.
- Левые и правые поддеревья также являются BST.

Основные операции с BST:

1. **Вставка элемента:**
    - Элемент добавляется в дерево с сохранением свойств BST.
2. **Поиск элемента:**
    - Элемент ищется в дереве, начиная с корня.
3. **Удаление элемента:**
    - Элемент удаляется из дерева с сохранением свойств BST.
4. **Обход дерева:**
    - **In-order (центрированный):** левое поддерево → корень → правое поддерево.
    - **Pre-order (прямой):** корень → левое поддерево → правое поддерево.
    - **Post-order (обратный):** левое поддерево → правое поддерево → корень.

Пример реализации BST на Java:

```
class Node {
   int key;
   Node left, right;

   public Node(int item) {
      key = item;
      left = right = null;
   }
```

```
}

public class BinarySearchTree {
    Node root;

    public BinarySearchTree() {
        root = null;
    }

    // Вставка элемента
    public void insert(int key) {
        root = insertRec(root, key);
    }

    private Node insertRec(Node root, int key) {
        if (root == null) {
            root = new Node(key);
            return root;
        }
        if (key < root.key) {
            root.left = insertRec(root.left, key);
        } else if (key > root.key) {
            root.right = insertRec(root.right, key);
        }
        return root;
    }

    // Поиск элемента
    public boolean search(int key) {
        return searchRec(root, key);
    }

    private boolean searchRec(Node root, int key) {
        if (root == null) {
            return false;
        }
        if (root.key == key) {
            return true;
        }
        return key < root.key ? searchRec(root.left, key) : searchRec(root.right, key);
    }

    // Удаление элемента
    public void delete(int key) {
        root = deleteRec(root, key);
    }

    private Node deleteRec(Node root, int key) {
        if (root == null) {
            return null;
        }
        if (key < root.key) {
            root.left = deleteRec(root.left, key);
        } else if (key > root.key) {
            root.right = deleteRec(root.right, key);
        } else {
            // Узел с одним или без дочерних узлов
            if (root.left == null) {
                return root.right;
            } else if (root.right == null) {
                return root.left;
            }
```

```
            // Узел с двумя дочерними узлами: находим минимальный элемент в правом поддереве
            root.key = minValue(root.right);
            // Удаляем минимальный элемент
            root.right = deleteRec(root.right, root.key);
        }
        return root;
    }

    private int minValue(Node root) {
        int minValue = root.key;
        while (root.left != null) {
            minValue = root.left.key;
            root = root.left;
        }
        return minValue;
    }

    // In-order обход
    public void inOrder() {
        inOrderRec(root);
    }

    private void inOrderRec(Node root) {
        if (root != null) {
            inOrderRec(root.left);
            System.out.print(root.key + " ");
            inOrderRec(root.right);
        }
    }

    public static void main(String[] args) {
        BinarySearchTree tree = new BinarySearchTree();

        tree.insert(50);
        tree.insert(30);
        tree.insert(20);
        tree.insert(40);
        tree.insert(70);
        tree.insert(60);
        tree.insert(80);

        System.out.println("In-order обход:");
        tree.inOrder();

        System.out.println("\nПоиск 40: " + tree.search(40));
        System.out.println("Поиск 90: " + tree.search(90));

        System.out.println("Удаление 20");
        tree.delete(20);
        tree.inOrder();

        System.out.println("\nУдаление 30");
        tree.delete(30);
        tree.inOrder();

        System.out.println("\nУдаление 50");
        tree.delete(50);
        tree.inOrder();
    }
}
```

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
   - Ознакомьтесь с принципами работы BST.
   - Проанализируйте пример реализации на Java.
2. **Реализуйте BST.**
   - Создайте класс BinarySearchTree и реализуйте методы для вставки, поиска, удаления и обхода.
   - Напишите тестовый класс для проверки корректности работы BST.
3. **Проведите тестирование.**
   - Создайте дерево и выполните основные операции (вставка, поиск, удаление).
   - Проверьте корректность обходов дерева.
4. **Проанализируйте результаты.**
   - Сравните эффективность операций с BST для различных наборов данных.
   - Сделайте выводы о применимости BST.

---

4. Варианты заданий

1. Реализуйте BST для хранения целых чисел.
2. Реализуйте BST для хранения строк.
3. Реализуйте BST для хранения объектов (например, студентов с полями id и name).
4. Реализуйте метод для нахождения минимального элемента в BST.
5. Реализуйте метод для нахождения максимального элемента в BST.
6. Реализуйте метод для подсчета количества узлов в BST.
7. Реализуйте метод для проверки, является ли дерево BST.
8. Реализуйте метод для нахождения k-го наименьшего элемента в BST.
9. Реализуйте метод для нахождения k-го наибольшего элемента в BST.
10. Реализуйте метод для преобразования BST в отсортированный массив.
11. Реализуйте метод для балансировки BST.
12. Реализуйте метод для поиска наименьшего общего предка двух узлов.
13. Реализуйте метод для вычисления высоты дерева.
14. Реализуйте метод для поиска всех элементов в заданном диапазоне.
15. Реализуйте метод для удаления всех элементов из BST.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием BST и его операций.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Что такое бинарное дерево поиска (BST)?
2. Какие свойства должны выполняться для BST?

3. Какие основные операции поддерживает BST?
4. Какова временная сложность операций вставки, поиска и удаления в BST?
5. В чем преимущества BST перед другими структурами данных?
6. Какие недостатки есть у BST?
7. В каких задачах целесообразно использовать BST?

Методические указания к лабораторной работе № 22.2 по теме: “Бинарные деревья: Обход дерева в глубину и ширину”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение методов обхода бинарных деревьев: в глубину (DFS) и в ширину (BFS). В ходе выполнения работы студенты научатся:

- понимать принципы работы алгоритмов обхода деревьев;
- реализовывать обходы в глубину и ширину на языке Java;
- применять обходы деревьев для решения практических задач;
- анализировать эффективность различных методов обхода.

---

2. Теоретический материал

Бинарное дерево — это структура данных, в которой каждый узел имеет не более двух дочерних узлов (левый и правый).

Обход дерева — это процесс посещения всех узлов дерева в определенном порядке. Основные методы обхода:

1. **Обход в глубину (DFS, Depth-First Search):**
   - Посещение узлов происходит по ветвям дерева, начиная с корня и углубляясь в каждую ветвь до конца, прежде чем перейти к следующей ветви.
   - Основные варианты DFS:
     - **In-order (центрированный):** левое поддерево → корень → правое поддерево.
     - **Pre-order (прямой):** корень → левое поддерево → правое поддерево.
     - **Post-order (обратный):** левое поддерево → правое поддерево → корень.
2. **Обход в ширину (BFS, Breadth-First Search):**
   - Посещение узлов происходит уровень за уровнем, начиная с корня и переходя к узлам следующего уровня.

Пример реализации бинарного дерева и обходов на Java:

```
import java.util.LinkedList;
import java.util.Queue;

class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
        left = right = null;
    }
}
```

```
public class BinaryTree {
    Node root;

    public BinaryTree() {
        root = null;
    }

    // In-order обход (левый → корень → правый)
    public void inOrder(Node node) {
        if (node != null) {
            inOrder(node.left);
            System.out.print(node.key + " ");
            inOrder(node.right);
        }
    }

    // Pre-order обход (корень → левый → правый)
    public void preOrder(Node node) {
        if (node != null) {
            System.out.print(node.key + " ");
            preOrder(node.left);
            preOrder(node.right);
        }
    }

    // Post-order обход (левый → правый → корень)
    public void postOrder(Node node) {
        if (node != null) {
            postOrder(node.left);
            postOrder(node.right);
            System.out.print(node.key + " ");
        }
    }

    // Обход в ширину (BFS)
    public void breadthFirstSearch(Node root) {
        if (root == null) {
            return;
        }
        Queue<Node> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {
            Node node = queue.poll();
            System.out.print(node.key + " ");
            if (node.left != null) {
                queue.add(node.left);
            }
            if (node.right != null) {
                queue.add(node.right);
            }
        }
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.println("In-order обход:");
```

```
        tree.inOrder(tree.root);

        System.out.println("\nPre-order обход:");
        tree.preOrder(tree.root);

        System.out.println("\nPost-order обход:");
        tree.postOrder(tree.root);

        System.out.println("\nОбход в ширину (BFS):");
        tree.breadthFirstSearch(tree.root);
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
   - Ознакомьтесь с принципами работы обходов деревьев.
   - Проанализируйте пример реализации на Java.
2. **Реализуйте бинарное дерево и методы обхода.**
   - Создайте класс BinaryTree и реализуйте методы для обходов в глубину и ширину.
   - Напишите тестовый класс для проверки корректности работы обходов.
3. **Проведите тестирование.**
   - Создайте дерево и выполните обходы в глубину и ширину.
   - Проверьте корректность результатов.
4. **Проанализируйте результаты.**
   - Сравните эффективность различных методов обхода.
   - Сделайте выводы о применимости каждого метода.

---

4. Варианты заданий

1. Реализуйте бинарное дерево и выполните in-order обход.
2. Реализуйте бинарное дерево и выполните pre-order обход.
3. Реализуйте бинарное дерево и выполните post-order обход.
4. Реализуйте бинарное дерево и выполните обход в ширину (BFS).
5. Реализуйте метод для нахождения высоты дерева.
6. Реализуйте метод для подсчета количества узлов в дереве.
7. Реализуйте метод для поиска максимального элемента в дереве.
8. Реализуйте метод для поиска минимального элемента в дереве.
9. Реализуйте метод для проверки, является ли дерево сбалансированным.
10. Реализуйте метод для поиска наименьшего общего предка двух узлов.
11. Реализуйте метод для преобразования дерева в зеркальное отражение.
12. Реализуйте метод для поиска всех путей от корня до листьев.
13. Реализуйте метод для поиска суммы всех узлов в дереве.
14. Реализуйте метод для поиска узла по значению.
15. Реализуйте метод для удаления дерева.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.

3. Теоретическая часть с описанием обходов деревьев.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Что такое бинарное дерево?
2. Какие методы обхода деревьев вы знаете?
3. В чем разница между in-order, pre-order и post-order обходами?
4. Как работает обход в ширину (BFS)?
5. В чем преимущества обхода в глубину (DFS) перед обходом в ширину (BFS)?
6. В каких задачах целесообразно использовать обход в ширину?
7. Какие недостатки есть у обхода в глубину?

Методические указания к лабораторной работе №22.3 по теме: "Бинарные деревья: Сбалансированные деревья"

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение сбалансированных бинарных деревьев, таких как AVL-деревья и красно-черные деревья. В ходе выполнения работы студенты научатся:

- понимать принципы работы сбалансированных деревьев;
- реализовывать AVL-деревья и красно-черные деревья на языке Java;
- применять сбалансированные деревья для решения практических задач;
- анализировать эффективность операций сбалансированных деревьев.

---

2. Теоретический материал

Сбалансированные деревья — это бинарные деревья поиска (BST), которые автоматически поддерживают баланс, чтобы обеспечить эффективность операций вставки, удаления и поиска. Основные типы сбалансированных деревьев:

1. **AVL-деревья:**
   - Названы в честь их создателей (Адельсон-Вельский и Ландис).
   - Балансировка достигается за счет поддержания свойства: для каждого узла высота левого и правого поддеревьев отличается не более чем на 1.
   - Основные операции:
     - **Вращения:** левое, правое, лево-правое, право-левое.
     - **Вставка:** после добавления элемента выполняется балансировка.
     - **Удаление:** после удаления элемента выполняется балансировка.
2. **Красно-черные деревья:**
   - Балансировка достигается за счет соблюдения следующих свойств:
     1. Каждый узел окрашен в красный или черный цвет.
     2. Корень всегда черный.
     3. Все листья (NIL-узлы) черные.
     4. Если узел красный, то его дочерние узлы черные.
     5. Все пути от узла до листьев содержат одинаковое количество черных узлов.
   - Основные операции:
     - **Вращения:** левое, правое.
     - **Вставка:** после добавления элемента выполняется перекрашивание и вращение.
     - **Удаление:** после удаления элемента выполняется перекрашивание и вращение.

Пример реализации AVL-дерева на Java:

```
class Node {
   int key, height;
   Node left, right;
```

```
    public Node(int key) {
        this.key = key;
        this.height = 1;
    }
}

public class AVLTree {
    Node root;

    // Получение высоты узла
    private int height(Node node) {
        return (node == null) ? 0 : node.height;
    }

    // Получение баланса узла
    private int getBalance(Node node) {
        return (node == null) ? 0 : height(node.left) - height(node.right);
    }

    // Правое вращение
    private Node rightRotate(Node y) {
        Node x = y.left;
        Node T2 = x.right;

        x.right = y;
        y.left = T2;

        y.height = Math.max(height(y.left), height(y.right)) + 1;
        x.height = Math.max(height(x.left), height(x.right)) + 1;

        return x;
    }

    // Левое вращение
    private Node leftRotate(Node x) {
        Node y = x.right;
        Node T2 = y.left;

        y.left = x;
        x.right = T2;

        x.height = Math.max(height(x.left), height(x.right)) + 1;
        y.height = Math.max(height(y.left), height(y.right)) + 1;

        return y;
    }

    // Вставка элемента
    public Node insert(Node node, int key) {
        if (node == null) {
            return new Node(key);
        }
        if (key < node.key) {
            node.left = insert(node.left, key);
        } else if (key > node.key) {
            node.right = insert(node.right, key);
        } else {
            return node; // Дубликаты не допускаются
        }

        node.height = 1 + Math.max(height(node.left), height(node.right));
```

```
        int balance = getBalance(node);

        // Лево-левое вращение
        if (balance > 1 && key < node.left.key) {
            return rightRotate(node);
        }

        // Право-правое вращение
        if (balance < -1 && key > node.right.key) {
            return leftRotate(node);
        }

        // Лево-правое вращение
        if (balance > 1 && key > node.left.key) {
            node.left = leftRotate(node.left);
            return rightRotate(node);
        }

        // Право-левое вращение
        if (balance < -1 && key < node.right.key) {
            node.right = rightRotate(node.right);
            return leftRotate(node);
        }

        return node;
    }

    // In-order обход
    public void inOrder(Node node) {
        if (node != null) {
            inOrder(node.left);
            System.out.print(node.key + " ");
            inOrder(node.right);
        }
    }

    public static void main(String[] args) {
        AVLTree tree = new AVLTree();
        tree.root = tree.insert(tree.root, 10);
        tree.root = tree.insert(tree.root, 20);
        tree.root = tree.insert(tree.root, 30);
        tree.root = tree.insert(tree.root, 40);
        tree.root = tree.insert(tree.root, 50);
        tree.root = tree.insert(tree.root, 25);

        System.out.println("In-order обход AVL-дерева:");
        tree.inOrder(tree.root);
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
    - Ознакомьтесь с принципами работы AVL-деревьев и красно-черных деревьев.
    - Проанализируйте пример реализации AVL-дерева на Java.
2. **Реализуйте AVL-дерево.**
    - Создайте класс AVLTree и реализуйте методы для вставки, удаления и балансировки.

- Напишите тестовый класс для проверки корректности работы AVL-дерева.

3. **Реализуйте красно-черное дерево.**
   - Создайте класс RedBlackTree и реализуйте методы для вставки, удаления и балансировки.
   - Напишите тестовый класс для проверки корректности работы красно-черного дерева.
4. **Проведите тестирование.**
   - Сравните производительность AVL-дерева и красно-черного дерева для различных наборов данных.
5. **Проанализируйте результаты.**
   - Сделайте выводы о применимости каждого типа сбалансированных деревьев.

---

4. Варианты заданий

1. Реализуйте AVL-дерево и выполните вставку элементов.
2. Реализуйте AVL-дерево и выполните удаление элементов.
3. Реализуйте красно-черное дерево и выполните вставку элементов.
4. Реализуйте красно-черное дерево и выполните удаление элементов.
5. Сравните время выполнения операций вставки в AVL-дерево и красно-черное дерево.
6. Реализуйте метод для нахождения высоты AVL-дерева.
7. Реализуйте метод для подсчета количества узлов в красно-черном дереве.
8. Реализуйте метод для поиска элемента в AVL-дереве.
9. Реализуйте метод для поиска элемента в красно-черном дереве.
10. Реализуйте метод для проверки, является ли дерево AVL-деревом.
11. Реализуйте метод для проверки, является ли дерево красно-черным деревом.
12. Реализуйте метод для преобразования AVL-дерева в массив.
13. Реализуйте метод для преобразования красно-черного дерева в массив.
14. Реализуйте метод для поиска минимального элемента в AVL-дереве.
15. Реализуйте метод для поиска максимального элемента в красно-черном дереве.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием сбалансированных деревьев.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Что такое сбалансированное дерево?
2. Какие типы сбалансированных деревьев вы знаете?
3. В чем разница между AVL-деревом и красно-черным деревом?
4. Как работает балансировка в AVL-дереве?
5. Какие свойства должны выполняться для красно-черного дерева?

6. В чем преимущества сбалансированных деревьев перед обычными BST?
7. В каких задачах целесообразно использовать сбалансированные деревья?

Методические указания к лабораторной работе № 23 по теме: “Графы. Основные понятия. Обходы графов в глубину и ширину”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение графов как структуры данных, а также освоение алгоритмов обхода графов в глубину (DFS) и в ширину (BFS). В ходе выполнения работы студенты научатся:

- понимать основные понятия теории графов;
- реализовывать графы на языке Java;
- применять алгоритмы DFS и BFS для обхода графов;
- анализировать эффективность алгоритмов обхода.

---

2. Теоретический материал

Граф — это совокупность вершин (узлов) и ребер (связей), соединяющих пары вершин. Графы используются для моделирования различных систем и процессов, таких как социальные сети, транспортные сети, компьютерные сети и т.д.

Основные понятия:

- **Вершина (узел):** элемент графа.
- **Ребро (дуга):** связь между двумя вершинами.
- **Ориентированный граф:** граф, в котором ребра имеют направление.
- **Неориентированный граф:** граф, в котором ребра не имеют направления.
- **Степень вершины:** количество ребер, инцидентных вершине.
- **Путь:** последовательность вершин, соединенных ребрами.
- **Цикл:** путь, начальная и конечная вершины которого совпадают.

Представление графов в памяти:

1. **Матрица смежности:**
    - Двумерный массив, где элемент `matrix[i][j]` указывает на наличие ребра между вершинами `i` и `j`.
    - Подходит для плотных графов.
2. **Список смежности:**
    - Массив списков, где каждый элемент массива содержит список вершин, смежных с данной.
    - Подходит для разреженных графов.

Алгоритмы обхода графов:

1. **Обход в глубину (DFS, Depth-First Search):**
    - Посещение вершин происходит по принципу "идти вглубь", пока это возможно.
    - Используется стек (явно или через рекурсию).
    - Применяется для поиска компонент связности, проверки наличия циклов, топологической сортировки.

2. **Обход в ширину (BFS, Breadth-First Search):**
   - Посещение вершин происходит по уровням, начиная с начальной вершины.
   - Используется очередь.
   - Применяется для поиска кратчайшего пути в невзвешенном графе, поиска компонент связности.

Пример реализации графа и обходов на Java:

```
import java.util.*;

class Graph {
    private int V; // Количество вершин
    private LinkedList<Integer> adj[]; // Список смежности

    // Конструктор
    public Graph(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList();
        }
    }

    // Добавление ребра в граф
    void addEdge(int v, int w) {
        adj[v].add(w);
    }

    // Обход в глубину (DFS)
    void DFS(int v) {
        boolean visited[] = new boolean[V];
        DFSUtil(v, visited);
    }

    private void DFSUtil(int v, boolean visited[]) {
        visited[v] = true;
        System.out.print(v + " ");

        for (int n : adj[v]) {
            if (!visited[n]) {
                DFSUtil(n, visited);
            }
        }
    }

    // Обход в ширину (BFS)
    void BFS(int v) {
        boolean visited[] = new boolean[V];
        Queue<Integer> queue = new LinkedList<>();

        visited[v] = true;
        queue.add(v);

        while (!queue.isEmpty()) {
            v = queue.poll();
            System.out.print(v + " ");

            for (int n : adj[v]) {
                if (!visited[n]) {
                    visited[n] = true;
```

```
                queue.add(n);
            }
        }
    }
}

public static void main(String args[]) {
    Graph g = new Graph(4);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    System.out.println("Обход в глубину (DFS), начиная с вершины 2:");
    g.DFS(2);

    System.out.println("\nОбход в ширину (BFS), начиная с вершины 2:");
    g.BFS(2);
}
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
   - Ознакомьтесь с основными понятиями теории графов.
   - Проанализируйте пример реализации графа и обходов на Java.
2. **Реализуйте граф и алгоритмы обхода.**
   - Создайте класс Graph и реализуйте методы для добавления ребер и обходов DFS и BFS.
   - Напишите тестовый класс для проверки корректности работы графа и обходов.
3. **Проведите тестирование.**
   - Создайте граф и выполните обходы DFS и BFS.
   - Проверьте корректность результатов.
4. **Проанализируйте результаты.**
   - Сравните эффективность DFS и BFS для различных графов.
   - Сделайте выводы о применимости каждого алгоритма.

---

4. Варианты заданий

1. Реализуйте граф и выполните обход DFS.
2. Реализуйте граф и выполните обход BFS.
3. Реализуйте ориентированный граф и выполните обход DFS.
4. Реализуйте ориентированный граф и выполните обход BFS.
5. Реализуйте метод для поиска компонент связности в графе.
6. Реализуйте метод для проверки наличия циклов в графе.
7. Реализуйте метод для поиска кратчайшего пути в невзвешенном графе.
8. Реализуйте метод для топологической сортировки графа.
9. Реализуйте метод для поиска всех путей между двумя вершинами.
10. Реализуйте метод для подсчета количества вершин в графе.
11. Реализуйте метод для подсчета количества ребер в графе.

12. Реализуйте метод для поиска вершины с максимальной степенью.
13. Реализуйте метод для поиска вершины с минимальной степенью.
14. Реализуйте метод для проверки, является ли граф двудольным.
15. Реализуйте метод для поиска эйлерова пути в графе.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием графов и алгоритмов обхода.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Что такое граф?
2. Какие способы представления графов вы знаете?
3. В чем разница между ориентированным и неориентированным графом?
4. Как работает алгоритм обхода в глубину (DFS)?
5. Как работает алгоритм обхода в ширину (BFS)?
6. В чем преимущества DFS перед BFS?
7. В каких задачах целесообразно использовать BFS?

Методические указания к лабораторной работе № 24.1 по теме:"Графы. Поиск мостов, точек сочленения."

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение алгоритмов поиска мостов и точек сочленения в графах. В ходе выполнения работы студенты научатся:

- понимать основные понятия теории графов, связанные с мостами и точками сочленения;
- реализовывать алгоритмы поиска мостов и точек сочленения на языке Java;
- применять эти алгоритмы для анализа структуры графов;
- анализировать эффективность алгоритмов.

---

2. Теоретический материал

Граф — это совокупность вершин (узлов) и ребер (связей), соединяющих пары вершин. В данной работе рассматриваются неориентированные графы.

Основные понятия:

- **Мост (bridge):** ребро, удаление которого увеличивает количество компонент связности графа.
- **Точка сочленения (articulation point):** вершина, удаление которой увеличивает количество компонент связности графа.

Алгоритмы поиска мостов и точек сочленения:

1. **Поиск мостов:**
   - Используется модификация алгоритма обхода в глубину (DFS).
   - Для каждой вершины хранятся два значения:
     - disc[u] — время обнаружения вершины.
     - low[u] — минимальное время обнаружения, достижимое из поддерева вершины.
   - Ребро (u, v) является мостом, если low[v] > disc[u].
2. **Поиск точек сочленения:**
   - Также используется модификация DFS.
   - Вершина u является точкой сочленения, если:
     - u — корень дерева DFS и имеет более одного ребенка.
     - u — не корень, и существует ребенок v, такой что low[v] >= disc[u].

Пример реализации на Java:

```
import java.util.*;

class Graph {
    private int V; // Количество вершин
    private LinkedList<Integer> adj[]; // Список смежности
```

```
private int time = 0; // Время обнаружения вершин

public Graph(int v) {
    V = v;
    adj = new LinkedList[v];
    for (int i = 0; i < v; ++i) {
        adj[i] = new LinkedList();
    }
}

// Добавление ребра в граф
void addEdge(int v, int w) {
    adj[v].add(w);
    adj[w].add(v);
}

// Рекурсивная функция для поиска мостов
private void bridgeUtil(int u, boolean visited[], int disc[], int low[], int parent[]) {
    visited[u] = true;
    disc[u] = low[u] = ++time;

    for (int v : adj[u]) {
        if (!visited[v]) {
            parent[v] = u;
            bridgeUtil(v, visited, disc, low, parent);

            low[u] = Math.min(low[u], low[v]);

            // Проверка на мост
            if (low[v] > disc[u]) {
                System.out.println("Мост: " + u + " - " + v);
            }
        } else if (v != parent[u]) {
            low[u] = Math.min(low[u], disc[v]);
        }
    }
}

// Поиск мостов
void findBridges() {
    boolean visited[] = new boolean[V];
    int disc[] = new int[V];
    int low[] = new int[V];
    int parent[] = new int[V];

    for (int i = 0; i < V; i++) {
        parent[i] = -1;
        visited[i] = false;
    }

    for (int i = 0; i < V; i++) {
        if (!visited[i]) {
            bridgeUtil(i, visited, disc, low, parent);
        }
    }
}

// Рекурсивная функция для поиска точек сочленения
private void articulationPointUtil(int u, boolean visited[], int disc[], int low[], int parent[], boolean ap[]) {
    int children = 0;
    visited[u] = true;
    disc[u] = low[u] = ++time;
```

```
    for (int v : adj[u]) {
        if (!visited[v]) {
            children++;
            parent[v] = u;
            articulationPointUtil(v, visited, disc, low, parent, ap);

            low[u] = Math.min(low[u], low[v]);

            // Проверка на точку сочленения
            if (parent[u] == -1 && children > 1) {
                ap[u] = true;
            }
            if (parent[u] != -1 && low[v] >= disc[u]) {
                ap[u] = true;
            }
        } else if (v != parent[u]) {
            low[u] = Math.min(low[u], disc[v]);
        }
    }
}

// Поиск точек сочленения
void findArticulationPoints() {
    boolean visited[] = new boolean[V];
    int disc[] = new int[V];
    int low[] = new int[V];
    int parent[] = new int[V];
    boolean ap[] = new boolean[V];

    for (int i = 0; i < V; i++) {
        parent[i] = -1;
        visited[i] = false;
        ap[i] = false;
    }

    for (int i = 0; i < V; i++) {
        if (!visited[i]) {
            articulationPointUtil(i, visited, disc, low, parent, ap);
        }
    }

    System.out.println("Точки сочленения:");
    for (int i = 0; i < V; i++) {
        if (ap[i]) {
            System.out.print(i + " ");
        }
    }
    System.out.println();
}

public static void main(String args[]) {
    Graph g = new Graph(5);
    g.addEdge(1, 0);
    g.addEdge(0, 2);
    g.addEdge(2, 1);
    g.addEdge(0, 3);
    g.addEdge(3, 4);

    System.out.println("Мосты в графе:");
    g.findBridges();
```

```
        System.out.println("\nТочки сочленения в графе:");
        g.findArticulationPoints();
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
   - Ознакомьтесь с понятиями мостов и точек сочленения.
   - Проанализируйте пример реализации на Java.
2. **Реализуйте алгоритмы поиска мостов и точек сочленения.**
   - Создайте класс Graph и реализуйте методы findBridges и findArticulationPoints.
   - Напишите тестовый класс для проверки корректности работы алгоритмов.
3. **Проведите тестирование.**
   - Создайте граф и выполните поиск мостов и точек сочленения.
   - Проверьте корректность результатов.
4. **Проанализируйте результаты.**
   - Сравните эффективность алгоритмов для различных графов.
   - Сделайте выводы о применимости алгоритмов.

---

4. Варианты заданий

1. Реализуйте поиск мостов в графе.
2. Реализуйте поиск точек сочленения в графе.
3. Реализуйте поиск мостов и точек сочленения в ориентированном графе.
4. Реализуйте метод для подсчета количества мостов в графе.
5. Реализуйте метод для подсчета количества точек сочленения в графе.
6. Реализуйте метод для поиска всех мостов в графе.
7. Реализуйте метод для поиска всех точек сочленения в графе.
8. Реализуйте метод для проверки, является ли граф двусвязным.
9. Реализуйте метод для проверки, является ли граф трёхсвязным.
10. Реализуйте метод для поиска всех компонент связности в графе.
11. Реализуйте метод для поиска всех циклов в графе.
12. Реализуйте метод для поиска кратчайшего пути в графе.
13. Реализуйте метод для поиска эйлерова пути в графе.
14. Реализуйте метод для поиска гамильтонова пути в графе.
15. Реализуйте метод для поиска всех путей между двумя вершинами.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием мостов и точек сочленения.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Что такое мост в графе?

2. Что такое точка сочленения в графе?
3. Как работает алгоритм поиска мостов?
4. Как работает алгоритм поиска точек сочленения?
5. В чем разница между мостами и точками сочленения?
6. В каких задачах целесообразно использовать поиск мостов?
7. В каких задачах целесообразно использовать поиск точек сочленения?

Методические указания к лабораторной работе № 24.2 по теме: “Графы. Поиск кратчайшего пути методами Дейкстры, Форда, Флойда”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение алгоритмов поиска кратчайшего пути в графах: алгоритма Дейкстры, алгоритма Беллмана-Форда и алгоритма Флойда-Уоршелла. В ходе выполнения работы студенты научатся:

- понимать принципы работы алгоритмов поиска кратчайшего пути;
- реализовывать алгоритмы на языке Java;
- применять алгоритмы для решения практических задач;
- анализировать эффективность алгоритмов по времени и памяти.

---

2. Теоретический материал

Граф — это совокупность вершин (узлов) и ребер (связей), соединяющих пары вершин. В данной работе рассматриваются взвешенные графы, где каждому ребру присвоен вес (стоимость).

Основные понятия:

- **Кратчайший путь:** путь между двумя вершинами с минимальной суммой весов ребер.
- **Отрицательные веса:** веса ребер могут быть отрицательными, что усложняет задачу поиска кратчайшего пути.

Алгоритмы поиска кратчайшего пути:

1. **Алгоритм Дейкстры:**
    - Находит кратчайшие пути от одной вершины до всех остальных.
    - Работает только для графов с неотрицательными весами.
    - Временная сложность: $O(V^2)$ для наивной реализации, $O(E + V \log V)$ с использованием приоритетной очереди.
2. **Алгоритм Беллмана-Форда:**
    - Находит кратчайшие пути от одной вершины до всех остальных.
    - Работает для графов с отрицательными весами (но без отрицательных циклов).
    - Временная сложность: $O(V * E)$.
3. **Алгоритм Флойда-Уоршелла:**
    - Находит кратчайшие пути между всеми парами вершин.
    - Работает для графов с отрицательными весами (но без отрицательных циклов).
    - Временная сложность: $O(V^3)$.

Пример реализации алгоритмов на Java:

```
import java.util.*;
```

```
class Graph {
    private int V; // Количество вершин
    private LinkedList<Edge> adj[]; // Список смежности

    class Edge {
        int dest, weight;
        Edge(int dest, int weight) {
            this.dest = dest;
            this.weight = weight;
        }
    }

    public Graph(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList<>();
        }
    }

    // Добавление ребра в граф
    void addEdge(int src, int dest, int weight) {
        adj[src].add(new Edge(dest, weight));
    }

    // Алгоритм Дейкстры
    void dijkstra(int src) {
        int dist[] = new int[V];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[src] = 0;

        PriorityQueue<Node> pq = new PriorityQueue<>(V, Comparator.comparingInt(n -> n.dist));
        pq.add(new Node(src, 0));

        while (!pq.isEmpty()) {
            Node node = pq.poll();
            int u = node.vertex;

            for (Edge e : adj[u]) {
                int v = e.dest;
                int weight = e.weight;

                if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                    dist[v] = dist[u] + weight;
                    pq.add(new Node(v, dist[v]));
                }
            }
        }

        System.out.println("Кратчайшие пути (Дейкстра):");
        for (int i = 0; i < V; i++) {
            System.out.println(src + " -> " + i + " = " + dist[i]);
        }
    }

    // Алгоритм Беллмана-Форда
    void bellmanFord(int src) {
        int dist[] = new int[V];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[src] = 0;
```

```
    for (int i = 1; i < V; i++) {
        for (int u = 0; u < V; u++) {
            for (Edge e : adj[u]) {
                int v = e.dest;
                int weight = e.weight;

                if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                    dist[v] = dist[u] + weight;
                }
            }
        }
    }

    // Проверка на отрицательные циклы
    for (int u = 0; u < V; u++) {
        for (Edge e : adj[u]) {
            int v = e.dest;
            int weight = e.weight;

            if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                System.out.println("Граф содержит отрицательный цикл!");
                return;
            }
        }
    }

    System.out.println("Кратчайшие пути (Беллман-Форд):");
    for (int i = 0; i < V; i++) {
        System.out.println(src + " -> " + i + " = " + dist[i]);
    }
}

// Алгоритм Флойда-Уоршелла
void floydWarshall() {
    int dist[][] = new int[V][V];
    for (int i = 0; i < V; i++) {
        Arrays.fill(dist[i], Integer.MAX_VALUE);
        dist[i][i] = 0;
    }

    for (int u = 0; u < V; u++) {
        for (Edge e : adj[u]) {
            dist[u][e.dest] = e.weight;
        }
    }

    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] != Integer.MAX_VALUE && dist[k][j] != Integer.MAX_VALUE &&
                    dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    System.out.println("Кратчайшие пути (Флойд-Уоршелл):");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            System.out.print(dist[i][j] == Integer.MAX_VALUE ? "INF " : dist[i][j] + " ");
        }
```

```
            System.out.println();
        }
    }

    class Node {
        int vertex, dist;
        Node(int vertex, int dist) {
            this.vertex = vertex;
            this.dist = dist;
        }
    }

    public static void main(String args[]) {
        Graph g = new Graph(5);
        g.addEdge(0, 1, 4);
        g.addEdge(0, 2, 1);
        g.addEdge(1, 3, 1);
        g.addEdge(2, 1, 2);
        g.addEdge(2, 3, 5);
        g.addEdge(3, 4, 3);

        g.dijkstra(0);
        System.out.println();
        g.bellmanFord(0);
        System.out.println();
        g.floydWarshall();
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
    - Ознакомьтесь с принципами работы алгоритмов Дейкстры, Беллмана-Форда и Флойда-Уоршелла.
    - Проанализируйте пример реализации на Java.
2. **Реализуйте алгоритмы поиска кратчайшего пути.**
    - Создайте класс Graph и реализуйте методы dijkstra, bellmanFord и floydWarshall.
    - Напишите тестовый класс для проверки корректности работы алгоритмов.
3. **Проведите тестирование.**
    - Создайте граф и выполните поиск кратчайших путей с использованием каждого алгоритма.
    - Проверьте корректность результатов.
4. **Проанализируйте результаты.**
    - Сравните эффективность алгоритмов для различных графов.
    - Сделайте выводы о применимости каждого алгоритма.

---

4. Варианты заданий

1. Реализуйте алгоритм Дейкстры для поиска кратчайших путей.
2. Реализуйте алгоритм Беллмана-Форда для поиска кратчайших путей.
3. Реализуйте алгоритм Флойда-Уоршелла для поиска кратчайших путей.
4. Сравните время выполнения алгоритмов Дейкстры и Беллмана-Форда.
5. Сравните время выполнения алгоритмов Беллмана-Форда и Флойда-Уоршелла.
6. Реализуйте метод для проверки наличия отрицательных циклов в графе.

7. Реализуйте метод для поиска кратчайшего пути в графе с отрицательными весами.
8. Реализуйте метод для поиска кратчайшего пути в графе с неотрицательными весами.
9. Реализуйте метод для поиска всех кратчайших путей между всеми парами вершин.
10. Реализуйте метод для поиска кратчайшего пути в ориентированном графе.
11. Реализуйте метод для поиска кратчайшего пути в неориентированном графе.
12. Реализуйте метод для поиска кратчайшего пути в графе с большим количеством вершин.
13. Реализуйте метод для поиска кратчайшего пути в графе с большим количеством ребер.
14. Реализуйте метод для поиска кратчайшего пути в графе с отрицательными циклами.
15. Реализуйте метод для визуализации графа и кратчайших путей.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием алгоритмов поиска кратчайшего пути.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. В чем разница между алгоритмами Дейкстры и Беллмана-Форда?
2. В каких случаях используется алгоритм Флойда-Уоршелла?
3. Почему алгоритм Дейкстры не работает с отрицательными весами?
4. Как алгоритм Беллмана-Форда обнаруживает отрицательные циклы?
5. В чем преимущество алгоритма Флойда-Уоршелла перед другими алгоритмами?
6. Какие ограничения на графы накладывает алгоритм Дейкстры?
7. В каких задачах целесообразно использовать алгоритм Беллмана-Форда?

Методические указания к лабораторной работе № 24.3 по теме: “Графы. Алгоритмы нахождения остовного дерева: Прима, Крускала”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение алгоритмов нахождения минимального остовного дерева (MST) в графах: алгоритма Прима и алгоритма Крускала. В ходе выполнения работы студенты научатся:

- понимать принципы работы алгоритмов Прима и Крускала;
- реализовывать алгоритмы на языке Java;
- применять алгоритмы для решения практических задач;
- анализировать эффективность алгоритмов по времени и памяти.

---

2. Теоретический материал

Остовное дерево — это подграф, который является деревом и содержит все вершины исходного графа. **Минимальное остовное дерево (MST)** — это остовное дерево с минимальной суммой весов ребер.

Основные алгоритмы нахождения MST:

1. **Алгоритм Прима:**
    - Постепенно строит MST, начиная с произвольной вершины и добавляя на каждом шаге ребро с минимальным весом, соединяющее уже включенные в MST вершины с оставшимися.
    - Использует приоритетную очередь для выбора ребра с минимальным весом.
    - Временная сложность: O(E log V) с использованием приоритетной очереди.
2. **Алгоритм Крускала:**
    - Строит MST, добавляя ребра в порядке возрастания их весов, избегая образования циклов.
    - Использует структуру данных "система непересекающихся множеств" (Disjoint Set Union, DSU) для проверки циклов.
    - Временная сложность: O(E log E) или O(E log V).

Пример реализации алгоритмов на Java:

```
import java.util.*;

class Graph {
    private int V; // Количество вершин
    private LinkedList<Edge> edges; // Список ребер

    class Edge implements Comparable<Edge> {
        int src, dest, weight;
        Edge(int src, int dest, int weight) {
            this.src = src;
            this.dest = dest;
            this.weight = weight;
        }
        public int compareTo(Edge other) {
```

```
        return this.weight - other.weight;
    }
}

public Graph(int v) {
    V = v;
    edges = new LinkedList<>();
}

// Добавление ребра в граф
void addEdge(int src, int dest, int weight) {
    edges.add(new Edge(src, dest, weight));
}

// Алгоритм Прима
void primMST() {
    boolean[] inMST = new boolean[V];
    int[] parent = new int[V];
    int[] key = new int[V];
    Arrays.fill(key, Integer.MAX_VALUE);

    PriorityQueue<Node> pq = new PriorityQueue<>(V, Comparator.comparingInt(n -> n.key));
    key[0] = 0;
    pq.add(new Node(0, 0));

    while (!pq.isEmpty()) {
        Node node = pq.poll();
        int u = node.vertex;
        inMST[u] = true;

        for (Edge e : edges) {
            if (e.src == u && !inMST[e.dest] && e.weight < key[e.dest]) {
                key[e.dest] = e.weight;
                parent[e.dest] = u;
                pq.add(new Node(e.dest, key[e.dest]));
            }
            if (e.dest == u && !inMST[e.src] && e.weight < key[e.src]) {
                key[e.src] = e.weight;
                parent[e.src] = u;
                pq.add(new Node(e.src, key[e.src]));
            }
        }
    }

    System.out.println("Минимальное остовное дерево (Прим):");
    for (int i = 1; i < V; i++) {
        System.out.println(parent[i] + " - " + i + " = " + key[i]);
    }
}

// Алгоритм Крускала
void kruskalMST() {
    Collections.sort(edges);
    int[] parent = new int[V];
    Arrays.fill(parent, -1);

    System.out.println("Минимальное остовное дерево (Крускал):");
    for (Edge e : edges) {
        int x = find(parent, e.src);
        int y = find(parent, e.dest);

        if (x != y) {
```

```
                System.out.println(e.src + " - " + e.dest + " = " + e.weight);
                union(parent, x, y);
            }
        }
    }

    // Поиск корня множества
    private int find(int[] parent, int i) {
        if (parent[i] == -1) {
            return i;
        }
        return find(parent, parent[i]);
    }

    // Объединение двух множеств
    private void union(int[] parent, int x, int y) {
        int xset = find(parent, x);
        int yset = find(parent, y);
        parent[xset] = yset;
    }

    class Node {
        int vertex, key;
        Node(int vertex, int key) {
            this.vertex = vertex;
            this.key = key;
        }
    }

    public static void main(String args[]) {
        Graph g = new Graph(4);
        g.addEdge(0, 1, 10);
        g.addEdge(0, 2, 6);
        g.addEdge(0, 3, 5);
        g.addEdge(1, 3, 15);
        g.addEdge(2, 3, 4);

        g.primMST();
        System.out.println();
        g.kruskalMST();
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
    - Ознакомьтесь с принципами работы алгоритмов Прима и Крускала.
    - Проанализируйте пример реализации на Java.
2. **Реализуйте алгоритмы нахождения MST.**
    - Создайте класс Graph и реализуйте методы primMST и kruskalMST.
    - Напишите тестовый класс для проверки корректности работы алгоритмов.
3. **Проведите тестирование.**
    - Создайте граф и выполните поиск минимального остовного дерева с использованием каждого алгоритма.
    - Проверьте корректность результатов.
4. **Проанализируйте результаты.**
    - Сравните эффективность алгоритмов для различных графов.
    - Сделайте выводы о применимости каждого алгоритма.

4. Варианты заданий

1. Реализуйте алгоритм Прима для нахождения MST.
2. Реализуйте алгоритм Крускала для нахождения MST.
3. Сравните время выполнения алгоритмов Прима и Крускала.
4. Реализуйте метод для проверки корректности MST.
5. Реализуйте метод для поиска MST в ориентированном графе.
6. Реализуйте метод для поиска MST в неориентированном графе.
7. Реализуйте метод для поиска MST в графе с большим количеством вершин.
8. Реализуйте метод для поиска MST в графе с большим количеством ребер.
9. Реализуйте метод для поиска MST в графе с отрицательными весами.
10. Реализуйте метод для поиска MST в графе с циклами.
11. Реализуйте метод для поиска всех возможных MST в графе.
12. Реализуйте метод для визуализации графа и MST.
13. Реализуйте метод для поиска MST в графе с весами, заданными случайным образом.
14. Реализуйте метод для поиска MST в графе с весами, заданными пользователем.
15. Реализуйте метод для поиска MST в графе с весами, заданными в файле.

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием алгоритмов Прима и Крускала.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

6. Контрольные вопросы

1. Что такое минимальное остовное дерево (MST)?
2. В чем разница между алгоритмами Прима и Крускала?
3. Почему алгоритм Прима использует приоритетную очередь?
4. Как алгоритм Крускала избегает образования циклов?
5. В каких случаях целесообразно использовать алгоритм Прима?
6. В каких случаях целесообразно использовать алгоритм Крускала?
7. Какие ограничения на графы накладывают алгоритмы Прима и Крускала?

Методические указания к лабораторной работе № 24.4 по теме: “Графы. Поиск циклов в графах”

---

1. Цель выполнения работы

Целью лабораторной работы является изучение и практическое применение алгоритмов поиска циклов в графах. В ходе выполнения работы студенты научатся:

- понимать основные понятия теории графов, связанные с циклами;
- реализовывать алгоритмы поиска циклов на языке Java;
- применять алгоритмы для анализа структуры графов;
- анализировать эффективность алгоритмов по времени и памяти.

---

2. Теоретический материал

Граф — это совокупность вершин (узлов) и ребер (связей), соединяющих пары вершин. В данной работе рассматриваются как ориентированные, так и неориентированные графы.

Основные понятия:

- **Цикл:** путь, начальная и конечная вершины которого совпадают.
- **Ориентированный граф:** граф, в котором ребра имеют направление.
- **Неориентированный граф:** граф, в котором ребра не имеют направления.

Алгоритмы поиска циклов:

1. **Поиск циклов в неориентированном графе:**
   - Используется модификация алгоритма обхода в глубину (DFS).
   - Если при обходе обнаруживается ребро, ведущее к уже посещенной вершине, которая не является родительской, то граф содержит цикл.
2. **Поиск циклов в ориентированном графе:**
   - Также используется модификация DFS.
   - Если при обходе обнаруживается ребро, ведущее к вершине, которая находится в текущем пути обхода, то граф содержит цикл.

Пример реализации поиска циклов на Java:

```
import java.util.*;

class Graph {
    private int V; // Количество вершин
    private LinkedList<Integer> adj[]; // Список смежности

    public Graph(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList<>();
        }
    }
```

```
// Добавление ребра в граф
void addEdge(int src, int dest) {
    adj[src].add(dest);
}

// Поиск циклов в неориентированном графе
boolean isCyclicUndirected() {
    boolean visited[] = new boolean[V];
    for (int i = 0; i < V; i++) {
        if (!visited[i]) {
            if (isCyclicUndirectedUtil(i, visited, -1)) {
                return true;
            }
        }
    }
    return false;
}

private boolean isCyclicUndirectedUtil(int v, boolean visited[], int parent) {
    visited[v] = true;

    for (int i : adj[v]) {
        if (!visited[i]) {
            if (isCyclicUndirectedUtil(i, visited, v)) {
                return true;
            }
        } else if (i != parent) {
            return true;
        }
    }
    return false;
}

// Поиск циклов в ориентированном графе
boolean isCyclicDirected() {
    boolean visited[] = new boolean[V];
    boolean recStack[] = new boolean[V];

    for (int i = 0; i < V; i++) {
        if (!visited[i]) {
            if (isCyclicDirectedUtil(i, visited, recStack)) {
                return true;
            }
        }
    }
    return false;
}

private boolean isCyclicDirectedUtil(int v, boolean visited[], boolean recStack[]) {
    visited[v] = true;
    recStack[v] = true;

    for (int i : adj[v]) {
        if (!visited[i]) {
            if (isCyclicDirectedUtil(i, visited, recStack)) {
                return true;
            }
        } else if (recStack[i]) {
            return true;
        }
    }
```

```
        recStack[v] = false;
        return false;
    }

    public static void main(String args[]) {
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Неориентированный граф содержит цикл: " + g.isCyclicUndirected());

        Graph g2 = new Graph(4);
        g2.addEdge(0, 1);
        g2.addEdge(0, 2);
        g2.addEdge(1, 2);
        g2.addEdge(2, 3);

        System.out.println("Ориентированный граф содержит цикл: " + g2.isCyclicDirected());
    }
}
```

---

3. Порядок выполнения работы

1. **Изучите теоретический материал.**
    - Ознакомьтесь с принципами работы алгоритмов поиска циклов.
    - Проанализируйте пример реализации на Java.
2. **Реализуйте алгоритмы поиска циклов.**
    - Создайте класс Graph и реализуйте методы isCyclicUndirected и isCyclicDirected.
    - Напишите тестовый класс для проверки корректности работы алгоритмов.
3. **Проведите тестирование.**
    - Создайте граф и выполните поиск циклов.
    - Проверьте корректность результатов.
4. **Проанализируйте результаты.**
    - Сравните эффективность алгоритмов для различных графов.
    - Сделайте выводы о применимости каждого алгоритма.

---

4. Варианты заданий

1. Реализуйте поиск циклов в неориентированном графе.
2. Реализуйте поиск циклов в ориентированном графе.
3. Сравните время выполнения алгоритмов для неориентированных и ориентированных графов.
4. Реализуйте метод для поиска всех циклов в неориентированном графе.
5. Реализуйте метод для поиска всех циклов в ориентированном графе.
6. Реализуйте метод для поиска циклов в графе с большим количеством вершин.
7. Реализуйте метод для поиска циклов в графе с большим количеством ребер.
8. Реализуйте метод для поиска циклов в графе с отрицательными весами.
9. Реализуйте метод для поиска циклов в графе с весами, заданными случайным образом.
10. Реализуйте метод для поиска циклов в графе с весами, заданными пользователем.
11. Реализуйте метод для поиска циклов в графе с весами, заданными в файле.

12. Реализуйте метод для визуализации графа и найденных циклов.
13. Реализуйте метод для поиска циклов в графе с использованием BFS.
14. Реализуйте метод для поиска циклов в графе с использованием DFS.
15. Реализуйте метод для поиска циклов в графе с использованием алгоритма Флойда.

---

5. Содержание отчета

1. Титульный лист с указанием названия работы, группы и ФИО студента.
2. Цель работы.
3. Теоретическая часть с описанием алгоритмов поиска циклов.
4. Листинг реализованных программ на Java.
5. Результаты тестирования.
6. Выводы по работе.

---

6. Контрольные вопросы

1. Что такое цикл в графе?
2. В чем разница между циклами в ориентированных и неориентированных графах?
3. Как работает алгоритм поиска циклов в неориентированном графе?
4. Как работает алгоритм поиска циклов в ориентированном графе?
5. В чем преимущества DFS перед BFS для поиска циклов?
6. В каких задачах целесообразно использовать поиск циклов?
7. Какие ограничения на графы накладывают алгоритмы поиска циклов?

СПИСОК ЛИТЕРАТУРЫ

1. Мухаметзянов, Р.Р. Основы программирования на Java: учебное пособие / Р.Р. Мухаметзянов, И. Д. Минегалиева. - Набережные Челны: НГПУ. URL: https://www.iprbookshop.ru/66812.html (дата обращения 10.03.2025)

2. Вирт, Н. Алгоритмы и структуры данных: учебник / Н. Вирт; пер. с англ. Ф. В. Ткачева. - 3-е изд. – М.: ДМК Пресс. URL: https://www.studentlibrary.ru/book/ISBN9785898183134.html (дата обращения:10.03.2025)

3. Гуськова, О.И. Объектно ориентированное программирование в Java: учебное пособие / О.И. Гуськова. - М: МПГУ. URL: https://znanium.com/catalog/product/1020593 (дата обращения 10.03.2025)

4. Грудина, О. Н. Основы бережливого производства: учебное пособие / О. Н. Грудина, Д. В. Запорожец [и др.] – Ставрополь: АГРУС Ставропольского гос. аграрного ун-та. URL: https://www.studentlibrary.ru/book/stavgau_230531.html (дата обращения 10.03.2025)