

Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

Пояснительная записка

к курсовой работе

по дисциплине: "Программирование на языке Java"

на тему: "Разработка многомодульного приложения на языке Java"

Выполнил студент группы 23BVB3:

Саранцев Е. А.

Принял:

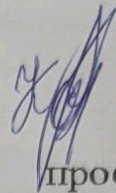
к.т.н., доцент Юрова О.В.

25.05.26
отлично
ф.и.о.

Пенза, 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"


Зав. кафедрой ВТ
профессор М. А. Митрохин

«___» _____ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Проектирование по излож. Чале
Студенту Сергееву Егору Александровичу Группа 23ВВТ1
Тема проекта Разработка многопользовательского приложения по излож. Чале

Исходные данные (технические требования) на проектирование

Разработка приложения архиватор
Изыскать программное Чале, среда разработки
Net Beans
Приложение должно откорректировать
интерфейс и использовать следующие технологии:
1 Чале Collection Framework
2 Механизм обработки исключительных ситуаций
3 Чале JFreeChart API
4 Чале Multi-threading
5 Сервис взаимодействия

Объем работы по курсу

1. Расчетная часть

2. Графическая часть

Расчетные языки ИМ: 1) Расчетные
массы 2) Расчетные рыбности 3) Расчетные
варианты использования 4) Расчетные
развертки 5) Расчетные последовательности

3. Экспериментальная часть

Тестирование нового модульного приложения

Срок выполнения проекта по разделам

- 1 В соответствии с графиком
- 2
- 3
- 4
- 5
- 6
- 7
- 8

Дата выдачи задания “ ”

Дата защиты проекта “ ”

Руководитель Норов О.В.

Задание получил “ 24 ” февраля 202 г.

Студент Сорокин Евгений Александрович

Содержание

Введение.....	5
1. Постановка задачи.....	6
2. Теоретические сведения.....	7
3. Выбор решения.....	8
4. Моделирование системы.....	9
5. Описание разработки программы и протокола взаимодействия.....	10
6. Руководство пользователя.....	12
Заключение.....	13
Список литературы.....	14
Приложение А.....	14
Приложение Б.....	28

Введение

В современных операционных системах задачи эффективного хранения данных и оптимизации пропускной способности каналов связи занимают центральное место. Одним из классических способов уменьшения избыточности информации является статистическое кодирование.

Целью данной курсовой работы является проектирование и разработка распределенного программного комплекса для архивации и декомпрессии файлов и каталогов. Система построена по архитектуре «клиент-сервер», где трудоемкие вычислительные операции (анализ частот байт, построение кодового дерева Хаффмана, битовая упаковка и распаковка) вынесены на сторону выделенного высокопроизводительного сервера. Взаимодействие узлов обеспечивается сетевыми сокетами, а параллельная обработка запросов от нескольких пользователей реализуется механизмами многопоточности операционной системы.

1. Постановка задачи.

В рамках курсового проектирования необходимо реализовать кроссплатформенную систему сетевого сжатия данных. Программный комплекс разделяется на два независимых компонента:

1. Серверный модуль (HuffmanServer): консольное многопоточное приложение, работающее в фоновом режиме. Сервер должен параллельно принимать бинарные потоки данных от клиентов, определять тип операции (сжатие/распаковка), выполнять алгоритм Хаффмана и возвращать результат.

2. Клиентский модуль (HuffmanClient): графическое приложение (GUI) на базе Java Swing. Клиент отвечает за локальный обход дерева каталогов, сериализацию структуры файлов в плоский поток байт, сетевой обмен с сервером и сохранение готовых архивов (.myarc) или распакованных файлов на диск. реализуется механизмами многопоточности операционной системы.

2. Теоретические сведения.

Алгоритм Хаффмана — это метод префиксного кодирования с переменной длиной кодового слова. В его основе лежит жадный алгоритм построения бинарного дерева, где минимальная длина кода присваивается наиболее часто встречающимся символам (байтам), а максимальная — редким.

С точки зрения операционной системы сетевой обмен реализуется через транспортный протокол TCP. Так как TCP обеспечивает потоковую передачу (Stream-oriented), границы сообщений в сети отсутствуют. Для корректной сборки пакетов на стороне сервера и клиента применен протокол явного указания длины: перед передачей массива байт в сокет записывается 4-байтовое целочисленное значение (int), определяющее точный размер последующего буфера.

Многопоточность на сервере организуется посредством системного пула потоков (ExecutorService). Это позволяет избежать блокировки главного сетевого цикла (accept()) и выделяет под каждую клиентскую сессию отдельный рабочий поток ядра ОС.

3. Выбор решения.

Для реализации выбран язык программирования Java (JDK 17) благодаря встроенной кроссплатформенной поддержке сетевых сокетов (java.net.Socket), механизмов сериализации объектов и многопоточного пула CachedThreadPool.

Архитектура системы исключает необходимость хранения баз данных на сервере. Сервер выступает в роли «чистого» вычислительного узла, обрабатывая поступающие байты «на лету», что снижает нагрузку на дисковую подсистему ОС.

4. Моделирование системы.

4.1. Диаграмма вариантов использования (Use Case Diagram)

Описывает функциональные границы системы и взаимодействие пользователя с клиентским GUI, а также скрытые системные прецеденты со стороны сервера. (см. Приложение Б- Рисунок Б.1)

4.2. Диаграмма деятельности

Иллюстрирует логику работы потока обработки клиента на сервере при диспетчеризации входящих команд.(см. Приложение Б- Рисунок Б.2)

4.3. Диаграмма классов

Отражает архитектуру объектно-ориентированного кода системы. Программа разделена на два независимых исполняемых класса. (см. Приложение Б- Рисунок Б.3)

4.4. Диаграмма развертывания

Показывает физическое распределение компонентов распределенной системы по аппаратным узлам вычислительной сети.(см. Приложение Б- Рисунок Б.4)

4.5. Диаграмма последовательности

Показывает динамику взаимодействий объектов во время работы: от нажатия кнопки пользователем, до конца работы приложения. (см. Приложение Б- Рисунок Б.5)

5. Описание разработки программы и протокола взаимодействия.

Обмен данными строго детерминирован и защищен от эффекта «разрывов» пакетов внутри сетевых буферов ОС. На прикладном уровне определен бинарный формат кадра запроса/ответа.

Таблица 1. Формат сетевого кадра прикладного уровня

Компонент кадра	Тип данных	Размер	Назначение
OpCode	Boolean	1 байт	Направление операции: <code>true</code> — сжатие, <code>false</code> — распаковка.
Length	Integer	4 байта	Длина передаваемого бинарного массива (в байтах).
Payload	Byte Array	Переменный	Полезная нагрузка (структурированные сырые файлы или архив).

Структура бинарного архива `.muarc` формируется сервером и имеет следующий вид:

MAGIC_NUMBER (4 байта) — сигнатура 0x4D594143 ("MYARC") для идентификации формата.

Сериализованный Java-объект `HashMap<Byte, Integer>` — таблица частот символов исходного файла, необходимая для однозначного восстановления дерева кодов при распаковке.

Bit Count (4 байта) — точное количество значащих битов в упакованной строке (защита от лишних нулевых битов в последнем байте).

Массив байт сжатого содержимого.

6. Руководство пользователя.

Запуск сервера: Сервер компилируется и запускается в фоновом режиме или внутри системного терминала командой `java HuffmanServer`. После успешной инициализации сетевого интерфейса выводится лог: [СЕРВЕР] Слушает порт: 12345.

Запуск клиента: Клиент запускается на рабочей станции пользователя. Открывается графическое окно «Huffman Archiver Pro (Network Arch Linux Edition)». В центре окна располагается лог-зона.

Процесс сжатия: Пользователь нажимает кнопку «Сжать (Файл/Папка)». В открывшемся диалоговом окне выбирается целевой объект. Клиент упаковывает его, передает серверу и выводит статистику эффективности (Эффективность сжатия: XX.XX%). Рядом с исходным файлом появляется сжатый архив с расширением `.muarc`.

Процесс распаковки: Пользователь нажимает кнопку «Распаковать `.muarc`», указывает архивный файл. Сервер декодирует данные, а клиент воссоздает оригинальную иерархию папок в каталоге `EXTRACTED_[timestamp]`.

Заключение

В ходе выполнения курсовой работы была успешно спроектирована и реализована многопоточная клиент-серверная система сетевой архивации данных на основе алгоритма Хаффмана. Были освоены принципы работы с сетевыми потоками ввода-вывода низкого уровня, протоколы разграничения бинарных кадров в потоке ТСР, координация пула потоков центрального процессора в ОС для обеспечения параллельной работы множества клиентов, а также сериализация сложных древовидных структур данных. Программа показала высокую скорость обработки и стабильность при одновременной нагрузке.

Список литературы

1. Седжвик, Р. Алгоритмы на Java. — М.: Вильямс, 2013. — 848 с.
2. Таненбаум, Э., Бос, Х. Современные операционные системы. — СПб.: Питер, 2015. — 1120 с.
3. Шилдт, Г. Java 8. Полное руководство. — М.: Вильямс, 2015. — 1376 с.
4. Олифер, В. Г., Олифер, Н. А. Компьютерные сети. Принципы, технологии, протоколы. — СПб.: Питер, 2016. — 992 с.

Приложение А

HuffmanServer.java

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class HuffmanServer {
    private static final int PORT = 12345;
    private static final int MAGIC_NUMBER = 0x4D594143;

    public static void main(String[] args) {
        System.out.println("[СЕРВЕР] Запуск сервера Хаффмана...");
        ExecutorService threadPool =
            Executors.newCachedThreadPool();

        try (ServerSocket serverSocket = new ServerSocket(PORT))
        {
            System.out.println("[СЕРВЕР] Слушает порт: " +
                PORT);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("[СЕРВЕР] Подключился новый клиент: " + clientSocket.getRemoteSocketAddress());
                threadPool.execute(new ClientHandler(clientSocket));
            }
        } catch (IOException e) {
            System.err.println("[СЕРВЕР] Критическая ошибка: " + e.getMessage());
        } finally {
            threadPool.shutdown();
        }
    }
}
```

```

private static class ClientHandler implements Runnable {
    private final Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try (DataInputStream dis = new
DataInputStream(socket.getInputStream());
            DataOutputStream dos = new
DataOutputStream(socket.getOutputStream())) {

            boolean compress = dis.readBoolean();
            int dataLength = dis.readInt();
            byte[] inputData = new byte[dataLength];
            dis.readFully(inputData);

            System.out.println("[ПОТОК " +
Thread.currentThread().getId() + "] Получено данных: " +
dataLength + " байт. Операция: " + (compress ? "Сжатие" :
"Распаковка"));

            byte[] resultData;
            if (compress) {
                resultData = compressData(inputData);
            } else {
                resultData = decompressData(inputData);
            }

            dos.writeInt(resultData.length);
            dos.write(resultData);
            dos.flush();
            System.out.println("[ПОТОК " +
Thread.currentThread().getId() + "] Ответ успешно отправлен.");

        } catch (Exception e) {
            System.err.println("[ПОТОК " +
Thread.currentThread().getId() + "] Ошибка обработки: " +
e.getMessage());
        }
    }
}

```

```

        } finally {
            try {
                socket.close();
                System.out.println("[СЕРВЕР] Соединение
закрыто: " + socket.getRemoteSocketAddress());
            } catch (IOException ignored) {}
        }
    }

    private byte[] compressData(byte[] rawData) throws
IOException {
        Map<Byte, Integer> freqs = new HashMap<>();
        for (byte b : rawData) freqs.put(b,
freqs.getOrDefault(b, 0) + 1);

        Node root = buildTree(freqs);
        Map<Byte, String> codes = new HashMap<>();
        generateCodes(root, "", codes);

        StringBuilder sb = new StringBuilder();
        for (byte b : rawData) sb.append(codes.get(b));
        byte[] compressed = bitStringToBytes(sb.toString());

        ByteArrayOutputStream outBaos = new
ByteArrayOutputStream();
        try (DataOutputStream packDos = new
DataOutputStream(outBaos)) {
            packDos.writeInt(MAGIC_NUMBER);

            ObjectOutputStream oos = new
ObjectOutputStream(packDos);
            oos.writeObject(freqs);
            oos.flush();

            packDos.writeInt(sb.length());
            packDos.write(compressed);
        }
        return outBaos.toByteArray();
    }
}

```

```

        private byte[] decompressData(byte[] compressedPack)
throws Exception {
            DataInputStream packDis = new DataInputStream(new
ByteArrayInputStream(compressedPack));

            if (packDis.readInt() != MAGIC_NUMBER) {
                throw new Exception("Неверный формат архива
(Magic Number не совпадает)!");
            }

            ObjectInputStream ois = new
ObjectInputStream(packDis);
            @SuppressWarnings("unchecked")
            Map<Byte, Integer> freqs = (Map<Byte, Integer>)
ois.readObject();

            int bitCount = packDis.readInt();
            byte[] compressed = packDis.readAllBytes();

            Node root = buildTree(freqs);
            String bitString =
bytesToBitString(compressed).substring(0, bitCount);

            ByteArrayOutputStream baos = new
ByteArrayOutputStream();
            Node curr = root;
            for (char bit : bitString.toCharArray()) {
                curr = (bit == '0') ? curr.left : curr.right;
                if (curr.isLeaf()) {
                    baos.write(curr.b);
                    curr = root;
                }
            }
            return baos.toByteArray();
        }

/ static class Node implements Comparable<Node> {
    byte b; int freq; Node left, right;
    Node(byte b, int f) { this.b = b; this.freq = f; }
}

```

```

        Node(int f, Node l, Node r) { this.freq = f;
this.left = l; this.right = r; }
        boolean isLeaf() { return left == null && right ==
null; }

        @Override public int compareTo(Node o) { return
Integer.compare(this.freq, o.freq); }
    }

    private Node buildTree(Map<Byte, Integer> freqs) {
        PriorityQueue<Node> pq = new PriorityQueue<>();
        freqs.forEach((b, f) -> pq.add(new Node(b, f)));
        if (pq.isEmpty()) return new Node((byte)0, 0);
        while (pq.size() > 1) {
            Node l = pq.poll(), r = pq.poll();
            pq.add(new Node(l.freq + r.freq, l, r));
        }
        return pq.poll();
    }

    private void generateCodes(Node n, String s, Map<Byte,
String> codes) {
        if (n.isLeaf()) codes.put(n.b, s.isEmpty() ? "0" :
s);

        else {
            generateCodes(n.left, s + "0", codes);
            generateCodes(n.right, s + "1", codes);
        }
    }

    private byte[] bitStringToBytes(String s) {
        byte[] res = new byte[(s.length() + 7) / 8];
        for (int i = 0; i < s.length(); i++)
            if (s.charAt(i) == '1') res[i / 8] |= (1 << (7 -
(i % 8)));
        return res;
    }

    private String bytesToBitString(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
        for (byte b : bytes)

```

```
        for (int i = 7; i >= 0; i--) sb.append((b >> i &
1) == 1 ? "1" : "0");
        return sb.toString();
    }
}
```

HuffmanClient.java

```
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.net.Socket;

public class HuffmanClient extends JFrame {
    private JTextArea logArea;
    private static final String EXTENSION = ".myarc";
    private static final String SERVER_HOST = "localhost";
    private static final int SERVER_PORT = 12345;

    public HuffmanClient() {
        setTitle("Huffman Archiver Pro (Network Arch Linux
Edition)");
        setSize(800, 550);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout(10, 10));

        logArea = new JTextArea();
        logArea.setEditable(false);
        logArea.setBackground(new Color(20, 20, 20));
        logArea.setForeground(new Color(50, 255, 50));
        logArea.setFont(new Font("Monospaced", Font.PLAIN, 13));
        logArea.setMargin(new Insets(10, 10, 10, 10));

        JButton btnCompress = new JButton("Сжать через Сервер
(Файл/Папка)");
        JButton btnDecompress = new JButton("Распаковать через
Сервер .myarc");
```

```

        JPanel btnPanel = new JPanel();
        btnPanel.add(btnCompress);
        btnPanel.add(btnDecompress);

        add(btnPanel, BorderLayout.NORTH);
        add(new JScrollPane(logArea), BorderLayout.CENTER);

        btnCompress.addActionListener(e -> process(true));
        btnDecompress.addActionListener(e -> process(false));
    }

    private void log(String msg) {
        SwingUtilities.invokeLater(() -> {
            logArea.append(msg + "\n");

            logArea.setCaretPosition(logArea.getDocument().getLength());
        });
    }

    private void process(boolean compress) {
        JFileChooser chooser = new JFileChooser();
        chooser.setFileSelectionMode(compress ?
JFileChooser.FILES_AND_DIRECTORIES : JFileChooser.FILES_ONLY);
        if (chooser.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION) {
            File selected = chooser.getSelectedFile();
            new Thread(() -> {
                try {
                    if (compress) compressTask(selected);
                    else decompressTask(selected);
                } catch (Exception ex) {

```

```

        log("\n!!! ОШИБКА: " + ex.getMessage());
        ex.printStackTrace();
    }
    }).start();
}
}

private void compressTask(File input) throws IOException {
    long startTime = System.currentTimeMillis();
    log(">>> ЗАПУСК СЕТЕВОГО СЖАТИЯ: " + input.getName());

    ByteArrayOutputStream baos = new
ByteArrayOutputStream();
    DataOutputStream packDos = new DataOutputStream(baos);
    serialize(input, "", packDos);
    packDos.flush();
    byte[] rawData = baos.toByteArray();
    log("1. Данные структуры собраны. Размер для отправки: "
+ rawData.length + " байт");

    log("2. Подключение к серверу " + SERVER_HOST + ":" +
SERVER_PORT + "...");
    byte[] compressedFromServer = sendRequestToServer(true,
rawData);
    log("3. Ответ от сервера получен.");

    File outFile = new File(input.getAbsolutePath() +
EXTENSION);

```

```

        try (FileOutputStream fos = new
FileOutputStream(outFile)) {
            fos.write(compressedFromServer);
        }

        log("-----");
        log("ИТОГО (Сеть + Сжатие): " +
(System.currentTimeMillis() - startTime) + " мс");
        log("Создан сетевой архив: " + outFile.length() + "
байт");
        double ratio = 100.0 - (outFile.length() * 100.0 /
rawData.length());
        log(String.format("Эффективность сжатия: %.2f%%",
ratio));
    }

    private void decompressTask(File file) throws Exception {
        long startTime = System.currentTimeMillis();
        log(">>> ЗАПУСК СЕТЕВОЙ РАСПАКОВКИ: " + file.getName());

        byte[] archiveData =
java.nio.file.Files.readAllBytes(file.toPath());

        log("1. Отправка архива на сервер...");
        byte[] unpackedData = sendRequestToServer(false,
archiveData);
        log("2. Сервер успешно распаковал данные.");
    }

```

```

        DataInputStream unpackDis = new DataInputStream(new
ByteArrayInputStream(unpackedData));

        String outRoot = file.getParent() + "/EXTRACTED_" +
System.currentTimeMillis();

        new File(outRoot).mkdirs();

        while (unpackDis.available() > 0) {
            String fileName = unpackDis.readUTF();
            int fileLen = unpackDis.readInt();
            byte[] content = new byte[fileLen];
            unpackDis.readFully(content);

            File f = new File(outRoot, fileName);
            f.getParentFile().mkdirs();
            try (FileOutputStream fos = new FileOutputStream(f))
            {
                fos.write(content);
            }
            log("Извлечен: " + fileName);
        }
        log("-----");
        log("--- ВСЕ ФАЙЛЫ ИЗВЛЕЧЕНЫ В: " + outRoot);
        log("ВРЕМЯ РАБОТЫ: " + (System.currentTimeMillis() -
startTime) + " мс");
    }

    private byte[] sendRequestToServer(boolean isCompress,
byte[] dataToAsks) throws IOException {
        try (Socket socket = new Socket(SERVER_HOST,
SERVER_PORT);

```

```

                                DataOutputStream dos = new
DataOutputStream(socket.getOutputStream());
                                DataInputStream dis = new
DataInputStream(socket.getInputStream())) {

    dos.writeBoolean(isCompress);
    dos.writeInt(dataToAsks.length);
    dos.write(dataToAsks);
    dos.flush();

    int responseLength = dis.readInt();
    byte[] responseData = new byte[responseLength];
    dis.readFully(responseData);

    return responseData;
}

private void serialize(File file, String path,
DataOutputStream dos) throws IOException {
    if (file.isDirectory()) {
        File[] files = file.listFiles();
        if (files != null) {
            for (File f : files) serialize(f, path +
file.getName() + "/", dos);
        }
    } else {
                                byte[] data =
java.nio.file.Files.readAllBytes(file.toPath());
        dos.writeUTF(path + file.getName());
        dos.writeInt(data.length);
    }
}

```

```

        dos.write(data);
    }
}

public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName
        ()); }
    catch (Exception ignored) {}

    SwingUtilities.invokeLater(() -> new
    HuffmanClient().setVisible(true));
}
}

```

Приложение Б

UML-диаграммы

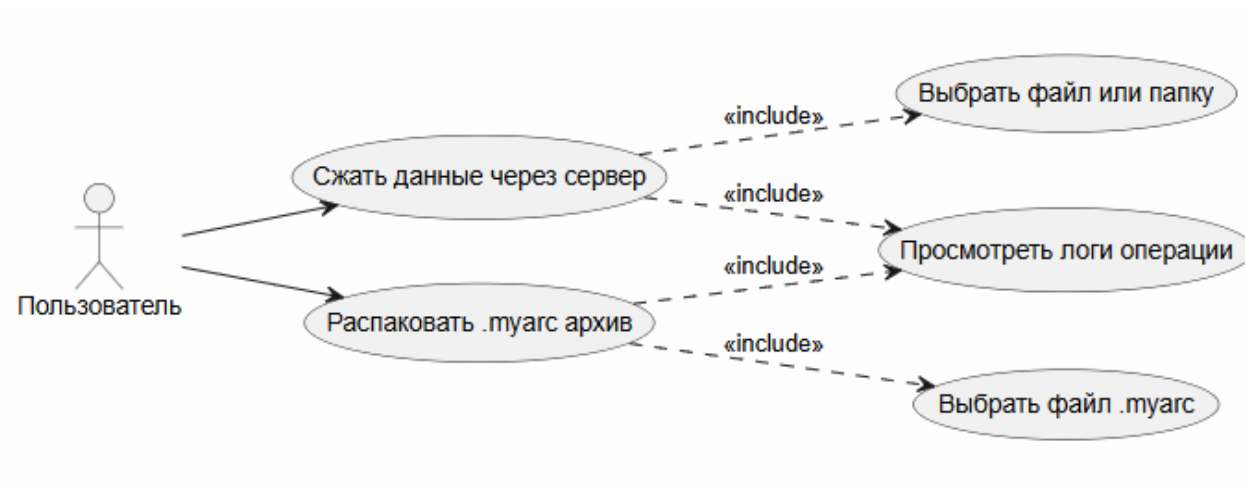


Рисунок Б.1- Диаграмма вариантов использования



Рисунок Б.2- Диаграмма деятельности сервера

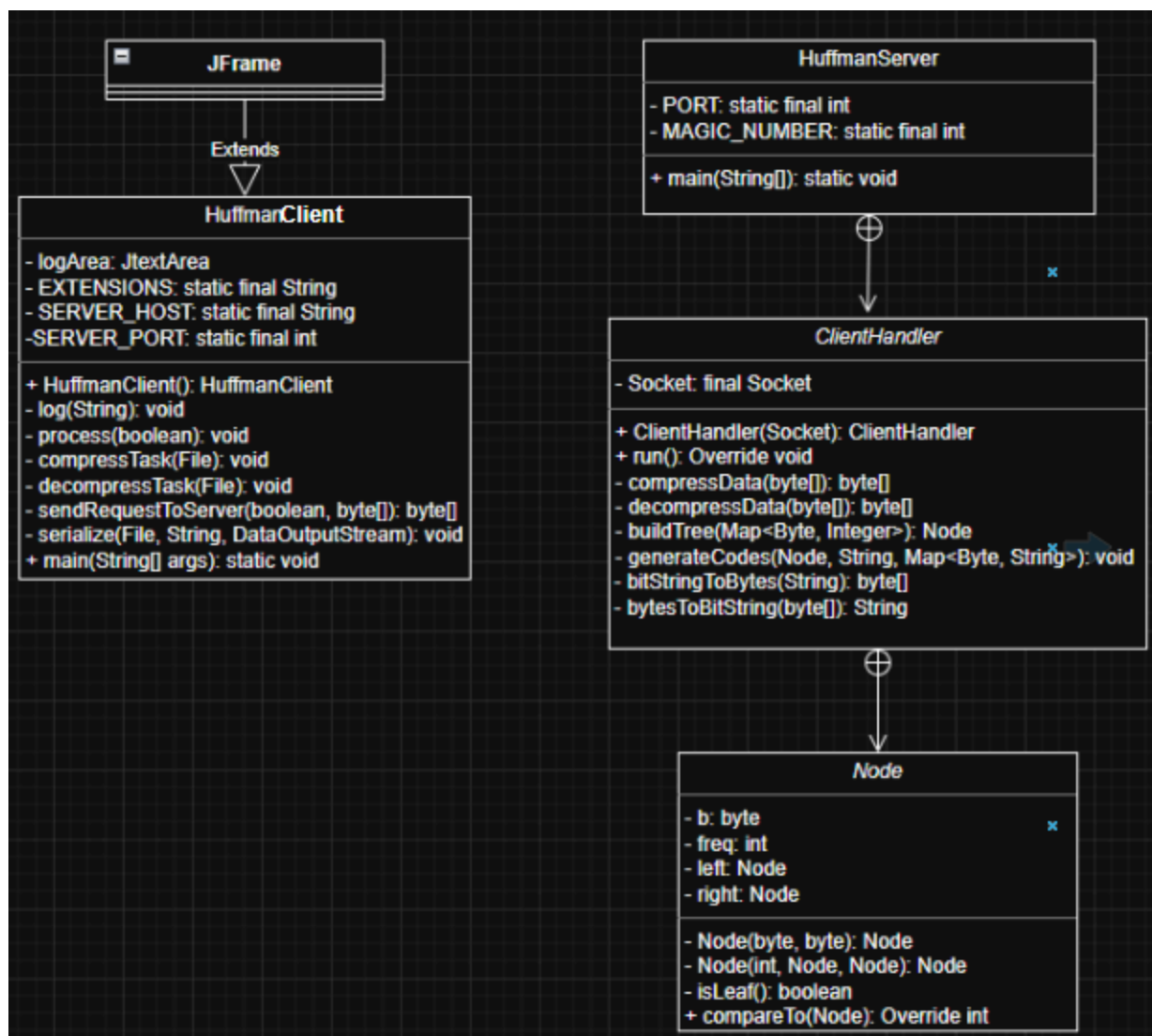


Рисунок Б.3- Диаграмма классов

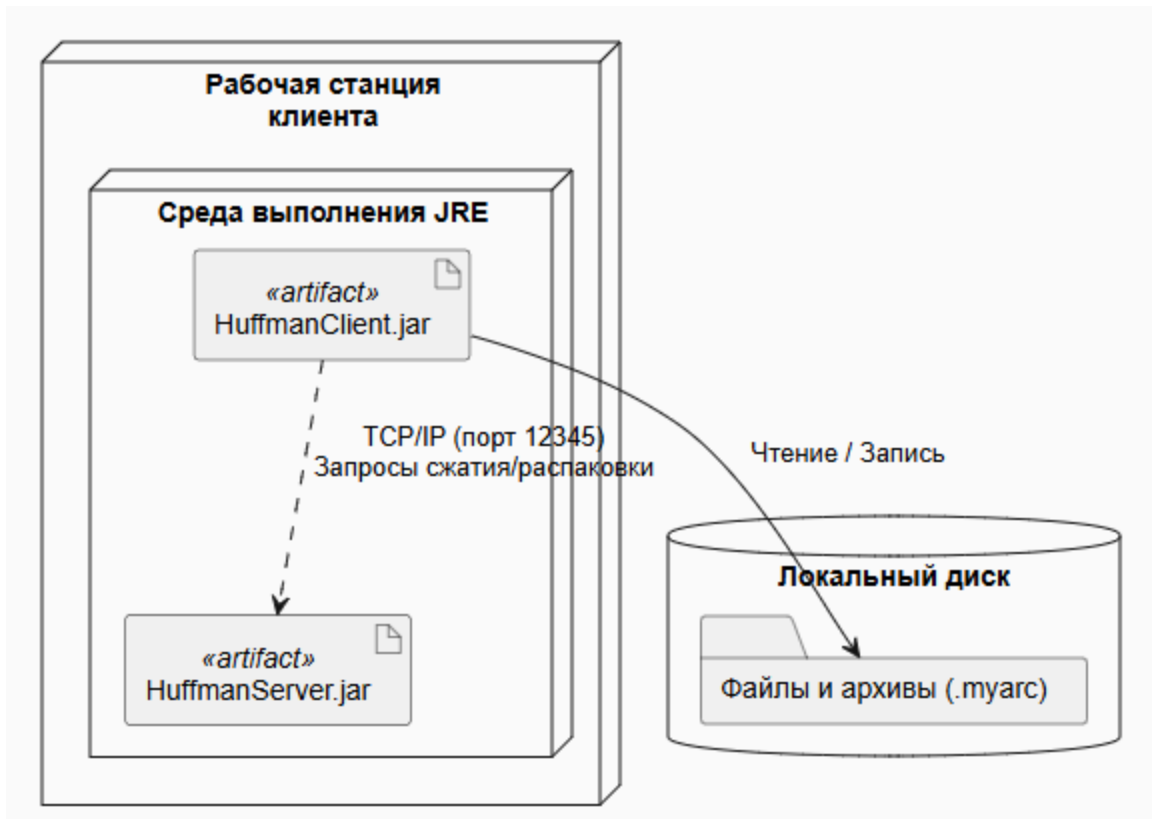


Рисунок Б.4- Диаграмма развертывания

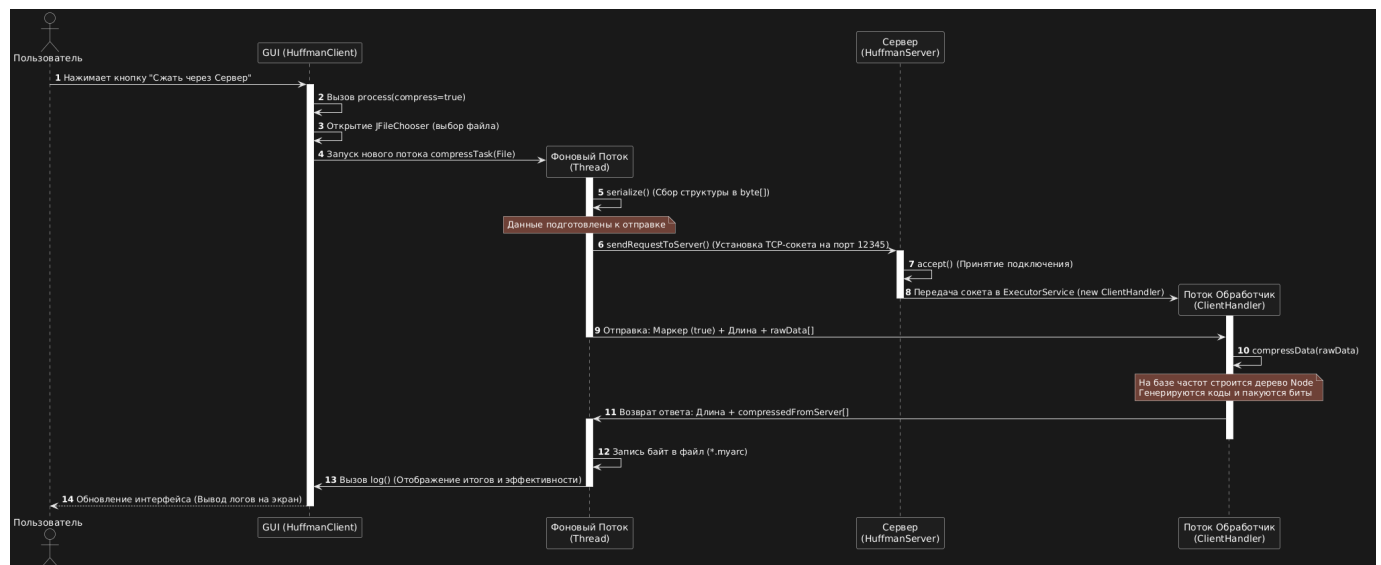


Рисунок Б.5-Диаграмма последовательности