**19. Потоковый ввод/вывод, работа с файловой системой**

Java предоставляет мощный набор инструментов для работы с файлами, потоками и вводом-выводом.

Основные классы:

1. File: Для управления файлами и директориями.
2. FileReader и FileWriter: Для работы с текстовыми данными.
3. FileInputStream и FileOutputStream: Для работы с байтовыми потоками.
4. BufferedReader и BufferedWriter: Для повышения производительности.

Принципы работы с файлами

1. **Чтение и запись данных**: Используются потоки ввода (InputStream) и вывода (OutputStream).
2. **Буферизация**: Улучшает производительность за счёт уменьшения количества операций с диском.
3. **Сериализация объектов**: Позволяет сохранять объекты в файлы и восстанавливать их из файлов.

Работа с NIO

API NIO предоставляет современный способ работы с файлами и потоками:

* Классы Paths и Files для работы с путями.
* Каналы (FileChannel) для асинхронного ввода-вывода.

**Пример:**

File file = new File("example.txt");

// Проверка существования

if (!file.exists()) {

try {

file.createNewFile();

System.out.println("Файл создан: " + file.getAbsolutePath());

} catch (IOException e) {

e.printStackTrace();

}

} else {

System.out.println("Файл уже существует");

}

**2. Классы потоков ввода/вывода**

Позволяют читать и записывать данные в файлы.

Чтение и запись побайтово:

* **FileInputStream и FileOutputStream** – для работы с байтами.
* Используются для работы с бинарными файлами, изображениями, аудиофайлами и т.д.

**Пример:**

// Запись в файл

try (FileOutputStream fos = new FileOutputStream("example.bin")) {

fos.write(new byte[]{1, 2, 3, 4});

System.out.println("Данные записаны в файл");

} catch (IOException e) {

e.printStackTrace();

}

// Чтение из файла

try (FileInputStream fis = new FileInputStream("example.bin")) {

int data;

while ((data = fis.read()) != -1) {

System.out.print(data + " ");

}

} catch (IOException e) {

e.printStackTrace();

}

Чтение и запись текстовых данных:

* **FileReader и FileWriter** – для работы с текстом.
* Читают и пишут символы.

**Пример:**

// Запись в текстовый файл

try (FileWriter writer = new FileWriter("example.txt")) {

writer.write("Привет, Java!");

System.out.println("Данные записаны в файл");

} catch (IOException e) {

e.printStackTrace();

}

// Чтение из текстового файла

try (FileReader reader = new FileReader("example.txt")) {

int character;

while ((character = reader.read()) != -1) {

System.out.print((char) character);

}

} catch (IOException e) {

e.printStackTrace();

}

**3. Буферизованные потоки**

Позволяют улучшить производительность за счет использования буферов.

* **BufferedReader и BufferedWriter** – для работы с текстом построчно.
* **BufferedInputStream и BufferedOutputStream** – для работы с байтами с буферизацией.

**Пример работы с BufferedReader и BufferedWriter:**

// Запись в файл

try (BufferedWriter writer = new BufferedWriter(new FileWriter("example.txt"))) {

writer.write("Первая строка");

writer.newLine();

writer.write("Вторая строка");

} catch (IOException e) {

e.printStackTrace();

}

// Чтение из файла

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. Работа с объектами**

* **ObjectInputStream и ObjectOutputStream** – для сериализации и десериализации объектов (сохранение и восстановление объектов).

**Пример:**

// Сериализация объекта

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("object.dat"))) {

oos.writeObject("Пример строки");

System.out.println("Объект сохранен");

} catch (IOException e) {

e.printStackTrace();

}

// Десериализация объекта

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("object.dat"))) {

String data = (String) ois.readObject();

System.out.println("Прочитанный объект: " + data);

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

**5. Работа с NIO (New Input/Output)**

NIO – это расширенный API для работы с файлами и потоками, включающий:

* **Класс Paths и Files** для работы с файлами и директориями.
* **Буферы (ByteBuffer, CharBuffer)** и каналы (FileChannel).

**Пример с NIO API:**

import java.nio.file.\*;

Path path = Paths.get("example.txt");

// Создание файла

try {

Files.write(path, "Привет, NIO!".getBytes());

System.out.println("Файл создан и данные записаны");

} catch (IOException e) {

e.printStackTrace();

}

// Чтение файла

try {

String content = Files.readString(path);

System.out.println("Содержимое файла: " + content);

} catch (IOException e) {

e.printStackTrace();

}

**6. Работа с файлами в потоках (java.util.stream)**

NIO поддерживает обработку файлов через потоки.

**Пример чтения файла через потоки:**

import java.nio.file.\*;

import java.util.stream.Stream;

try (Stream<String> lines = Files.lines(Paths.get("example.txt"))) {

lines.forEach(System.out::println);

} catch (IOException e) {

e.printStackTrace();

}

Пример который нам давал Шараф Асрорович на уроке

Файловый ввод и вывод

public static void main(String[] args) throws FileNotFoundException {

/\* Scanner in = new Scanner(System.in); //для ввода

int a,b,c;

a = in.nextInt();

b = in.nextInt();

c = a + b;

System.out.print(c) ;\*\

/\* String fn;

fn = “c:\\1.txt”;

FileInputStream fin = new FileInputStream(fn);

DataInputStream din = new DataInputStream(fin);

int x,y;

try { x.din.readInt(); } catch (Exception ex) {};

try { y.din.readInt(); } catch (Exception ex) {};

System.out.println(x+y);

fn = “c:\\2.txt”;

FileOutputStream fout = new FileOutputStream(fn);

DataOutputStream dout = new DataOutputStream(fout);

try { dout.writeByte(5); } catch (Exception ex) {};

try { fout.flush(); } catch (Exception ex) {};\*\

/\* string fn;

fn = “c:\\1.txt”;

FileRead frd = new FileReader (fn);

Scanner in = new Scanner(frd);

Int n;

n = in.nextInt();

fn = “c:\\2.txt”;

FileWriter fwr = new FileWriter(fn);

PrintWriter pwr = new PrintWriter(fwr);

pwr.println(15);

pwr.flush();

pwr.close();\*/

string fn = “”;

//fn = “ c:\\1.txt”;

JFileChooser fch = new JFileChooser( “c:\\”);

int fc\_res = fch.showDialog( null, “Выбирайте файл”);

If (fc\_res == JFileChooser.APPROVE\_OPTION)

fn = fch.getSelectedFile().toString

FileRead frd = new FileReader (fn);

Scanner in = new Scanner(frd);

Int n;

n = in.nextInt();

fn = “c:\\2.txt”;

FileWriter fwr = new FileWriter(fn);

PrintWriter pwr = new PrintWriter(fwr);

pwr.println(15);

pwr.flush();

pwr.close();

}

}