Министерство науки и высшего образования РФ

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа «Киберфизических систем и управления»

**ОТЧЕТ**

по дисциплине «Системный подход в разработке ПО»

Задания по материалу занятия 10

**Выполнил:**

студент гр. 3530902/90201 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В. Позолотин

подпись, дата

**Проверил:**

доцент, к.т.н \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Нестеров

подпись, дата

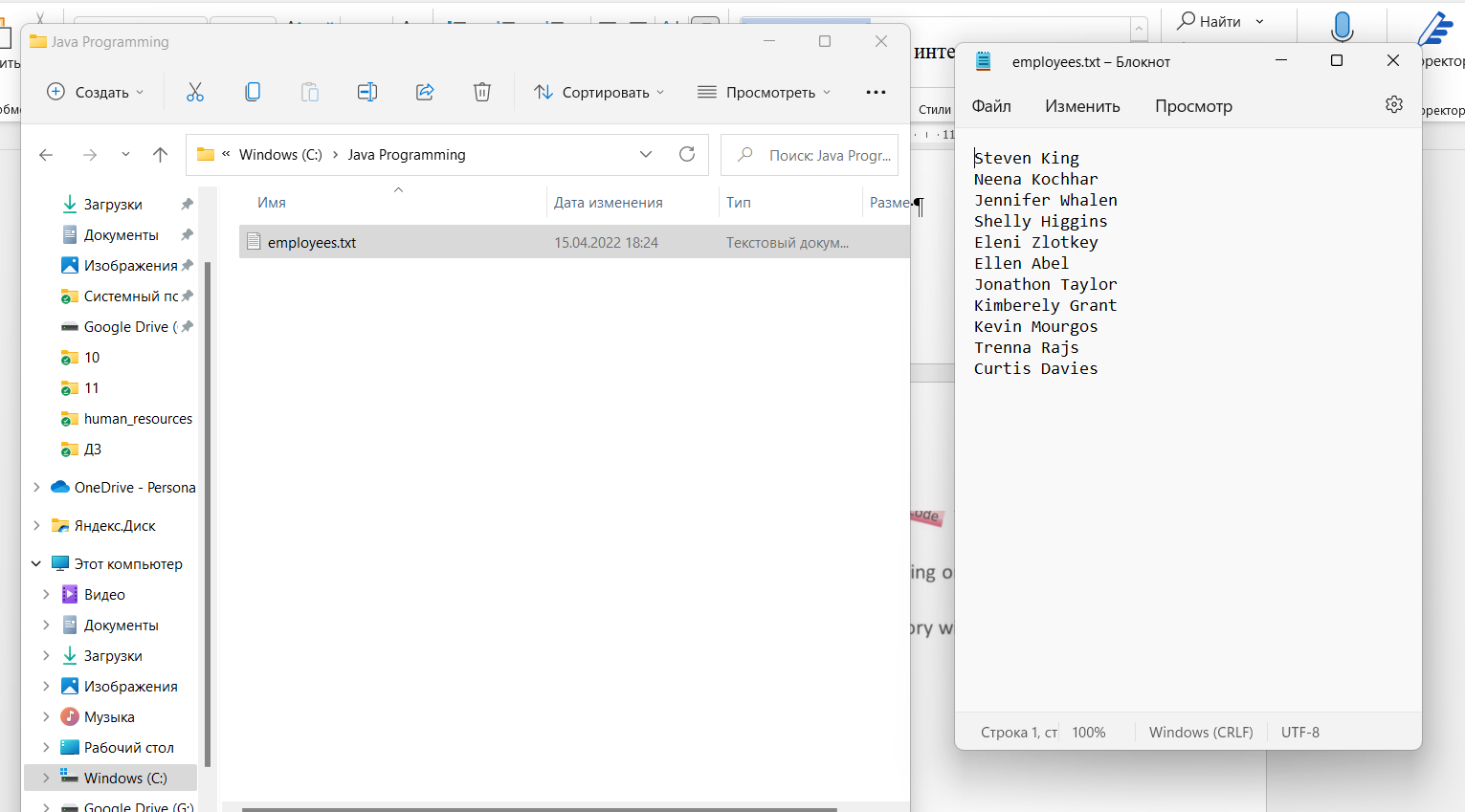
Санкт-Петербург

2022 г.

**5-1 Basics of Input and Output**

**Изображение выглядит как текст

Автоматически созданное описание**

****

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

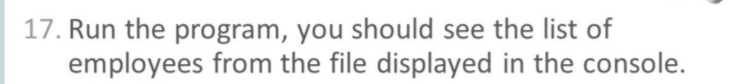
Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

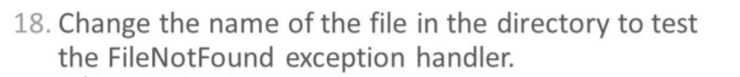
**Изображение выглядит как текст, внутренний, снимок экрана

Автоматически созданное описание**

****

**Изображение выглядит как текст

Автоматически созданное описание**

****

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст, внутренний, снимок экрана

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

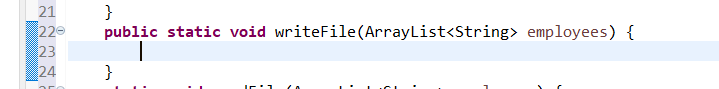
**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

****

****

**Изображение выглядит как текст

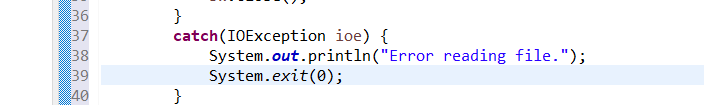
Автоматически созданное описание**

**Изображение выглядит как текст

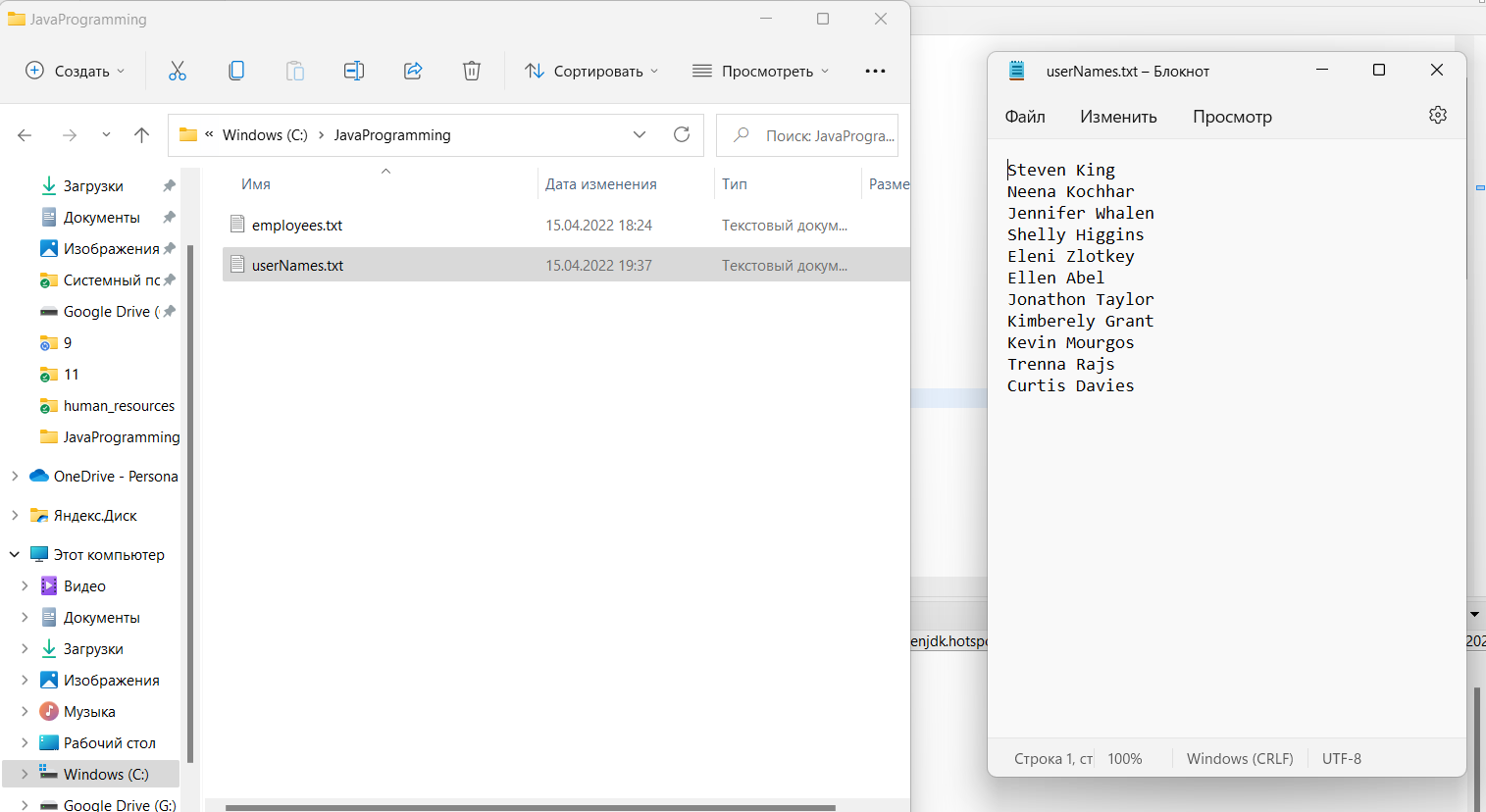
Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

****

**Изображение выглядит как текст

Автоматически созданное описание  
Изображение выглядит как текст

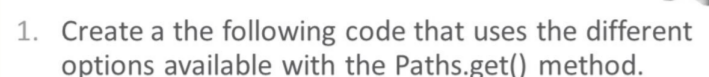
Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

****

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

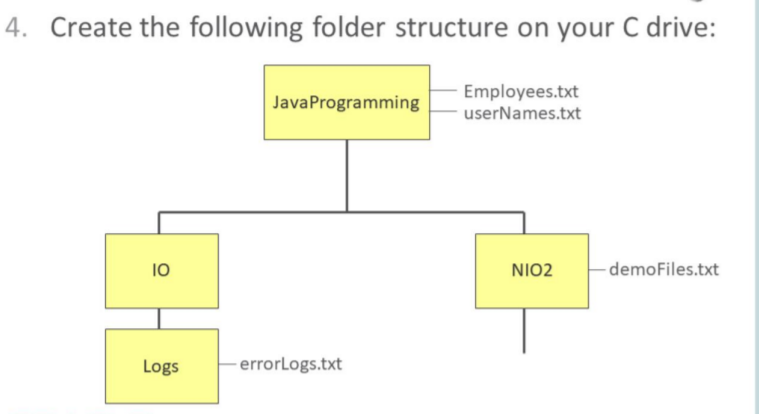
Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

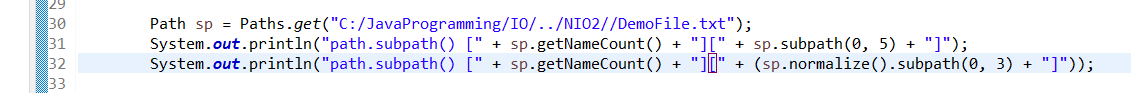
****

****

****

**Изображение выглядит как текст

Автоматически созданное описание**

****

****

****

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

****

**Изображение выглядит как текст

Автоматически созданное описание**

**Практика 5.1**

1. Create a class with a static main that tests the ability to resolve and print a Path:

• Create an instance of a FileSystem class.

• Create an instance of the following Path interface.

**Изображение выглядит как текст

Автоматически созданное описание**

• Print the constructed Path with System.out.println() method.

**Изображение выглядит как текст

Автоматически созданное описание**

2. Identify the main limitations of the Java.io Package.

Ограничения пакета java.io:

• Многие методы не выбрасывают исключений.

• Отсутствуют некоторые операции (например копирование и перемещение).

• Не поддерживаются символические ссылки.

• Многие методы не масштабируются на большие файлы.

3. Create a class that does the following:

• Using a pre-Java 7 solution, create a class that tests streams in the static main.

• The class should instantiate a new File class, a new FileReader class, and new BufferedReader class.

• Read lines by using the readLine() method call.

• The file path used should be: C:/JavaProgramming/employees.txt

• The file should handle errors when the file is not found as well as reading the contents of the file when it is found.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

**Конспект**

ФАЙЛЫ И АРГУМЕНТЫ КОМАНДНОЙ СТРОКИ

**Аргументы командной строки**

Аргументом методу main() передается текстовый массив, через значения которого реализуются параметры, которые передаются программе при запуске на выполнение. Дело все в том, что, запуская программу на выполнение, ей можно передать параметры. Если бы мы запускали откомпилированную программу из командной строки, то после имени файла с откомпилированным главным классом программы через пробел указать передать некоторые параметры. Эти параметры считываются и их можно использовать в программе. Другими словами, в программном коде метода main() можно обращаться к элементам массива, указанного аргументом метода. Сам массив формируется при запуске программы на выполнение на основе параметров, которые передаются программе.

GetMoneyDemo:

**Изображение выглядит как текст

Автоматически созданное описание**

**Работа с файлами**

Нередко приходится иметь дело с файлами, из которых программа получает данные, записывает в них данные или просто обрабатывает информацию, связанную с файлами. Далее мы рассмотрим некоторые базовые операции, которые важны с прикладной точки зрения. Среди этих операций, разумеется операции, связанные с файловым вводом и выводом.

GetFileInfoDemo:

**Изображение выглядит как текст

Автоматически созданное описание**

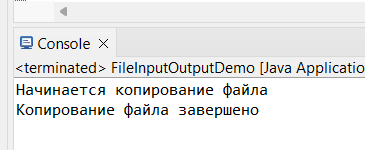
**Получение информации о файле**

Если необходимо просто получить некоторую информацию о файле или директории, можем использовать класс File. Класс доступен после импорта пакета java.io.

Объект класса File создается несколькими способами. Например, при создании объекта класса File аргументом конструктору можно передать полный путь к файлу. Также отдельно первым аргументом может быть указан путь к родительской директории (той, в которой содержится файл или поддиректория), а вторым аргументом — имя файла. Вместо первого текстового аргумента, определяющего путь К родительской директории, может быть указан объект класса File, соответствующий этой директории. Здесь важно отметить, что в принципе файл или директория, указанные при создании объекта класса File, могут и не существовать.

В принципе, объект класса File представляет собой некую «сущность», связанною с определенным местом в файловой иерархии, даже если такого места в иерархии нет. То есть на самом деле это не объект для файла, а объект для пути к файлу.

**FileInputOutputDemo:**

****

**Чтение из файла и запись в файл**

В Java существуют разные способы считывания данных из файла и записи данных в файл, но все они в итоге подразумевают, что используется поток ввода или вывода данных, и этот поток связывается с файлом.

Для начала рассмотрим наиболее простой случай, когда данные из файла считываются по отдельным байтам и записываются в файл байт за байтом. В таком случае используют байтовый поток. Для реализации байтового файлового потока ввода используют класс FileInputStream из пакета java.io. Для создания потока ввода создается объект данного класса. Аргументом конструктору передается имя файла (полный путь к файлу), из которого считываются данные. Байтовый файловый поток вывода реализуется через объект класса FileOutputStream. Аргументом конструктору при создании объекта потока вывода указывается полный путь к файлу, в который выполняется запись. Если указанного аргументом конструктора файла нет, то он будет создан.

Конструкторы классов FileInputStream и FileOutputStream могут выбрасывать контролируемое исключение класса FileNotFoundException. После того как потоки ввода и/или вывода созданы, через соответствующие объекты можно получать данные из файла и записывать данные в файл. Так, для считывания очередного байта из файла из соответствующего потока ввода вызывается метод read(). Результатом метод возвращает считанный из файла байт.

Обычно байты считываются последовательно один за другим, вплоть до окончания файла. Признаком окончания файла является считанное значение -1.

Для записи байта в файл используют метод write(), который вызывается из объекта файлового потока вывода. Аргументом методу write() передается записываемое в файл значение. После завершения работы с файлом необходимо вызвать из объекта связанного с файлом потока метод close(). Если этого не сделать, то некоторые операции по чтению и записи данных могут быть выполнены некорректно. Методы read(), write() и close() могут вызвать контролируемое исключение класса IOException.

Байтовые потоки не всегда удобны в плане прикладного использования, особенно если речь идет о работе с символьными файлами (файлами, содержащими символы). Проблема в том, что символ — это два байта. Поэтому при побайтовом считывании данных могут возникнуть проблемы. Однако есть возможность считывать данные посимвольно (то есть по два байта за один раз). В таком случае говорят о символьных потоках ввода и вывода.

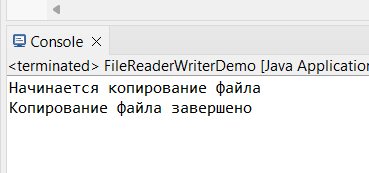
Символьный файловый поток ввода реализуется с помощью класса FileReader, а символьный файловый поток вывода реализуется с помощью класса FileWriter. Принципы работы с классами FileReader и FileWriter во многом напоминает методы работы с классами FileInputStream и FileOutputStream, включая наличие методов read(), write() и close().

Еще один способ работы с файлами при чтении и записи данных, который мы рассмотрим, базируется на использовании буферизированных потоков ввода и вывода.

Особенность буферизированного потока в том, что запись и считывание данных выполняется через буфер. Благодаря этому количество обращений к файлам на диске уменьшается и скорость выполнения операций увеличивается. Есть и дополнительный «бонус», состоящий в том, что считывание и запись данных может выполняться блоками большими, чем байт или символ, — во всяком случае, на программном уровне.

Буферизированные потоки ввода и вывода реализуются с помощью классов BufferedReader и BufferedWriter. При создании объекта буферизированного потока аргументом конструктору будет передаваться анонимный объект символьного потока. При считывании содержимого текстового файла через буферизированный поток мы используем метод readLine(), позволяющий считывать целые строки.

**FileReaderWriterDemo:**

****

**Средства выбора файлов**

Ранее ссылки на файлы мы указывали явно путем записи в текстовое значение пути к файлу. На практике намного удобнее искать и выбирать файл в файловой системе с помощью специального окна выбора файлов.

Мы познакомимся с классом JFileChooser из пакета javax.swing. С помощью объекта класса JFileChooser легко организовать процесс выбора файла через отображение специального диалогового окна, позволяющего выбрать нужный файл непосредственно в файловой системе. Наибольший интерес для нас представляют методы showOpenDialog() и showSaveDialog() класса JFileChooser. С помощью этих методов, собственно, и открывается диалоговое окно для выбора файла.

**FileChooserDemo:**

**Изображение выглядит как текст

Автоматически созданное описание**