Потоки ввода вывода

System

**System** – это класс предоставляет стандартный ввод, стандартный вывод и потоки вывода ошибок; доступ к внешним свойствам и переменным среды; средство загрузки файлов и библиотек; и служебный метод для быстрого копирования части массива.

**Поля:**

**PrintStream err** - "стандартный" поток вывода ошибок.

**InputStream in** - "стандартный" поток ввода.

**PrintStream out** - "стандартный" поток вывода.

**Методы:**

**arraycopy(Object src, int srcPos, Object dest, int destPos, int length)** - копирует массив из указанного исходного массива, начиная с указанной позиции, в указанную позицию целевого массива.

**clearProperty(String key)** - удаляет системное свойство, указанное указанным ключом.

**console()** - возвращает уникальный Console объект, связанный с текущей виртуальной машиной Java, если таковой имеется.

**currentTimeMillis()** - возвращает текущее время в миллисекундах.

**exit(int status)** - завершает текущую виртуальную машину Java.

**gc()** - запускает сборщик мусора на виртуальной машине Java.

**getenv()** - возвращает неизменяемый вид строковой карты текущей системной среды.

**getenv(String name)** - возвращает значение указанной переменной среды.

**getLogger(String name)** - возвращает экземпляр Logger для использования вызывающей стороной.

**getLogger(String name, ResourceBundle bundle)** - возвращает локализуемый экземпляр Logger для использования вызывающей стороной.

**getProperties()** - определяет текущие системные свойства.

**getProperty(String key)** - возвращает системное свойство, указанное указанным ключом.

**getProperty(String key, String def)** - возвращает системное свойство, указанное указанным ключом.

**getSecurityManager()** - устаревший, для удаления: Этот элемент API подлежит удалению в будущей версии. Этот метод полезен только в сочетании с Security Manager, который устарел и подлежит удалению в будущей версии.

**identityHashCode(Object x)** - возвращает тот же хэш-код для данного объекта, который был бы возвращен методом по умолчанию hashCode(), независимо от того, переопределяет класс данного объекта hashCode() или нет.

**inheritedChannel()** - возвращает канал, унаследованный от объекта, который создал эту виртуальную машину Java.

**lineSeparator()** - возвращает строку разделителя строк, зависящую от системы.

**load(String filename)** - загружает встроенную библиотеку, указанную в аргументе filename.

**loadLibrary(String libname)** - загружает встроенную библиотеку, указанную в libname аргументе.

**mapLibraryName(String libname)** - преобразует имя библиотеки в строку, зависящую от платформы, представляющую собственную библиотеку.

**nanoTime()** - возвращает текущее значение источника времени с высоким разрешением для запущенной виртуальной машины Java в наносекундах.

**runFinalization()** - устаревший, для удаления: Этот элемент API подлежит удалению в будущей версии. Доработка устарела и подлежит удалению.

**setErr(PrintStream err)** - переназначает "стандартный" поток вывода ошибок.

**setIn(InputStream in)** - переназначает "стандартный" поток ввода.

**setOut(PrintStream out)** - переназначает "стандартный" поток вывода.

**setProperties(Properties props)** - устанавливает системные свойства в Properties аргумент.

**setProperty(String key, String value)** - устанавливает системное свойство, указанное указанным ключом.

**setSecurityManager(SecurityManager sm)** - устаревший, для удаления: Этот элемент API подлежит удалению в будущей версии. Этот метод полезен только в сочетании с Security Manager, который устарел и подлежит удалению в будущей версии.

**Вложенные классы:**

**System.Logger –** это вложенный классэкземпляры регистрируют сообщения, которые будут перенаправлены в базовую платформу ведения журнала, которую LoggerFinder использует.

**Методы:**

**getName()** - возвращает имя этого регистратора.

**isLoggable(System.Logger.Level level)** - проверяет, будет ли сообщение данного уровня зарегистрировано этим регистратором.

**log(System.Logger.Level level, Object obj)** - регистрирует сообщение, созданное из данного объекта.

**log(System.Logger.Level level, String msg)** - регистрирует сообщение.

**log(System.Logger.Level level, String format, Object... params)** - регистрирует сообщение с необязательным списком параметров.

**log(System.Logger.Level level, String msg, Throwable thrown)** - регистрирует сообщение, связанное с данным throwable.

**log(System.Logger.Level level, Supplier<String> msgSupplier)** - регистрирует лениво отправленное сообщение.

**log(System.Logger.Level level, Supplier<String> msgSupplier, Throwable thrown)** - регистрирует лениво предоставленное сообщение, связанное с данным throwable.

**log(System.Logger.Level level, ResourceBundle bundle, String format, Object... params)** - регистрирует сообщение с пакетом ресурсов и необязательным списком параметров.

**log(System.Logger.Level level, ResourceBundle bundle, String msg, Throwable thrown)** - регистрирует локализованное сообщение, связанное с данным throwable.

**Имеет вложенный класс:**

**System.Logger.Level** – это вложенный класс который хранит уровни системных регистраторов. Уровень имеет название и серьезность.

**Поля:**

**ALL** - маркер, указывающий, что все уровни включены.

**DEBUG** - уровень: обычно используется для регистрации трассировок отладочной информации.

**ERROR** - уровень: обычно используется для регистрации сообщений об ошибках.

**INFO** - уровень: обычно используется для регистрации информационных сообщений.

**OFF** - маркер, указывающий на то, что все уровни отключены.

**TRACE** - уровень: обычно используется для регистрации диагностической информации.

**WARNING** - уровень: обычно используется для регистрации предупреждающих сообщений.

**Методы**:

**getName()** - возвращает имя этого уровня.

**getSeverity()** - возвращает степень серьезности этого уровня.

**valueOf(String name)** - возвращает константу enum этого класса с указанным именем.

**values()** - возвращает массив, содержащий константы этого класса enum, в порядке их объявления.

**System.LoggerFinder** – это вложенный классслужба отвечает за создание, управление и настройку регистраторов в соответствии с базовой платформой, которую она использует.

**Методы:**

**getLocalizedLogger(String name, ResourceBundle bundle, Module module) -** возвращает локализуемый экземпляр Logger для данного module.

**getLogger(String name, Module module) -** возвращает экземпляр Logger для данного module.

**getLoggerFinder()** - возвращает LoggerFinder экземпляр.

Потоки для работы с байтами

OutputStream

**OutputStream** — это базовый абстрактный класс является суперклассом всех классов, представляющих выходной поток байтов.

**Методы:**

**close()** - закрывает этот выходной поток и освобождает все системные ресурсы, связанные с этим потоком.

**flush()** - очищает этот выходной поток и принудительно записывает все буферизованные выходные байты.

**nullOutputStream()** - возвращает новый, OutputStream который отбрасывает все байты.

**write(byte[] b)** - записывает b.length байты из указанного массива байтов в этот выходной поток.

**write(byte[] b, int off, int len)** - записывает len байты из указанного массива байтов, начиная со смещения off, в этот выходной поток.

**write(int b)** - записывает указанный байт в этот выходной поток.

**Расширяющие его классы:**

**BufferedOutputStream -** этокласс который реализует буферизованный выходной поток.

**ByteArrayOutputStream -** это класс который реализует выходной поток, в котором данные записываются в массив байтов.

**DataOutputStream** – это класс который представляет поток вывода данных позволяющий приложению записывать примитивные типы данных Java в outputstream переносимым способом.

**FileOutputStream** – этокласс который предоставляет поток вывода для записи данных в файл или в FileDescriptor.

**FilterOutputStream -** это класс который является суперклассом всех классов, которые фильтруют выходные потоки.

**ObjectOutputStream –** это класс который записывает примитивные типы данных и графики объектов Java в OutputStream.

**ObjectOutputStream.PutField –** это класс который обеспечьте программный доступ к постоянным полям, которые будут записаны в ObjectOutput.

**PipedOutputStream** – этокласскоторый позволяет создать канал связи подключив к нему конвейерный выходной поток.

FilterOutputStream

**FilterOutputStream** — это класс который расширяет OutputStream и фильтруют выходные потоки.

**Методы:**

**close()** - закрывает этот выходной поток и освобождает все системные ресурсы, связанные с этим потоком.

**flush()** - очищает этот выходной поток и принудительно записывает все буферизованные выходные байты.

**write(byte[] b)** - записывает b.length байты из указанного массива байтов в этот выходной поток.

**write(byte[] b, int off, int len)** - записывает len байты из указанного массива байтов, начиная со смещения off, в этот выходной поток.

**write(int b)** - записывает указанный байт в этот выходной поток.

PrintStream

**PrintStream** – это класс расширяющий FilterOutputStream которыйдобавляет функциональность к другому выходному потоку, а именно возможность удобной печати представлений различных значений данных. В отличие от других выходных потоков никогда не выдает IOException, вместо этого устанавливается внутренний флаг.

**Поля:**

**OutputStream out -** базовый выходной поток, подлежащий фильтрации.

**Методы:**

**append(char c)** - добавляет указанный символ в этот выходной поток.

**append(CharSequence csq)** - добавляет указанную последовательность символов к этому выходному потоку.

**append(CharSequence csq, int start, int end)** - добавляет подпоследовательность указанной последовательности символов к этому выходному потоку.

**charset()** - возвращает кодировку, используемую в этом PrintStream экземпляре.

**checkError()** - сбрасывает поток, если он не закрыт, и проверяет его состояние ошибки.

**clearError()** - очищает состояние ошибки этого потока.

**close()** - закрывает поток.

**flush()** - сбрасывает поток.

**format(String format, Object... args)** - записывает форматированную строку в этот выходной поток, используя указанную строку формата и аргументы.

**format(Locale l, String format, Object... args)** - записывает форматированную строку в этот выходной поток, используя указанную строку формата и аргументы.

**print(b)** - выводит значение.

**print(Object obj)** - печатает объект.

**print(String s)** - печатает строку.

**printf(String format, Object... args)** - удобный метод для записи форматированной строки в этот выходной поток с использованием указанной строки формата и аргументов.

**printf(Locale l, String format, Object... args)** - удобный метод для записи форматированной строки в этот выходной поток с использованием указанной строки формата и аргументов.

**println()** - завершает текущую строку, записывая строку-разделитель строк.

**println(x)** - выводит значение, а затем завершает строку.

**println(Object x)** - печатает объект, а затем завершает строку.

**println(String x)** - печатает строку, а затем завершает строку.

**setError()** - устанавливает состояние ошибки потока в true.

**write(byte[] buf)** - записывает все байты из указанного массива байтов в этот поток.

**write(byte[] buf, int off, int len)** - записывает len байты из указанного массива байтов, начиная со смещения off, в этот поток.

**write(int b)** - записывает указанный байт в этот поток.

**writeBytes(byte[] buf)** - записывает все байты из указанного массива байтов в этот поток.

InputStream

**InputStream** — это абстрактный класс, который является суперклассом всех классов, представляющих входной поток байтов.

**Методы:**

**available()** - возвращает оценку количества байтов, которые могут быть прочитаны (или пропущены) из этого входного потока без блокировки, которое может быть равно 0 или 0 при обнаружении конца потока.

**close()** - закрывает этот входной поток и освобождает все системные ресурсы, связанные с потоком.

**mark(int readlimit)** - отмечает текущую позицию в этом входном потоке.

**markSupported()** - проверяет, поддерживает ли этот входной поток методы mark и reset.

**nullInputStream()** - возвращает новое значение, InputStream которое не считывает байты.

**read()** - считывает следующий байт данных из входного потока.

**read(byte[] b)** - считывает некоторое количество байтов из входного потока и сохраняет их в буферном массиве b.

**read(byte[] b, int off, int len)** - считывает до len байт данных из входного потока в массив байтов.

**readAllBytes()** - считывает все оставшиеся байты из входного потока.

**readNBytes(byte[] b, int off, int len)** - считывает запрошенное количество байтов из входного потока в заданный массив байтов.

**readNBytes(int len)** - считывает до указанного количества байтов из входного потока.

**reset()** - перемещает этот поток в положение на момент последнего вызова mark метода в этом входном потоке.

**skip(long n)** - пропускает и отбрасывает n байты данных из этого входного потока.

**skipNBytes(long n)** - пропускает и отбрасывает ровно n байты данных из этого входного потока.

**transferTo(OutputStream out)** - считывает все байты из этого входного потока и записывает байты в данный выходной поток в том порядке, в котором они были прочитаны.

**Расширяющие его классы:**

**BufferedInputStream** – это класс который добавляет функциональность к другому входному потоку, а именно, возможность буферизации входных данных и поддержки методов mark и reset.

**ByteArrayInputStream** - это класс который содержит внутренний буфер, содержащий байты, которые могут быть считаны из потока.

**DataInputStream** - это класс который позволяет приложению считывать примитивные типы данных Java из базового входного потока машинно-независимым способом.

**FileInputStream** - это класс который получает входные байты из файла в файловой системе.

**FilterInputStream** - это класс который оборачивает некоторый другой входной поток, который он использует в качестве своего основного источника данных, возможно, преобразуя данные по пути или предоставляя дополнительную функциональность.

**ObjectInputStream** - это класс который десериализует примитивные данные и объекты, ранее записанные с помощью ObjectOutputStream.

**ObjectInputStream.GetField** - это класс который предоставьте доступ к постоянным полям, считываемым из входного потока.

**PipedInputStream** - это класс который предоставляет любые байты данных, записанные в конвейерный выходной поток. Если конвейерный входной поток подключен к конвейерному выходному потоку.

**PushbackInputStream** - это класс который добавляет функциональность к другому входному потоку, а именно возможность "отбрасывать" или "непрочитанные" байты, сохраняя отброшенные байты во внутреннем буфере.

**SequenceInputStream** - это класс который представляет собой логическую конкатенацию других входных потоков.

Потоки для работы с символами

Reader

**Reader –** это абстрактный класс для чтения потоков символов. Единственными методами, которые должен реализовать подкласс, являются read(char[], int, int) и close() .

**Методы:**

**close()** - закрывает поток и освобождает все системные ресурсы, связанные с ним.

**mark(int readAheadLimit)** - отмечает текущую позицию в потоке.

**markSupported()** - сообщает, поддерживает ли этот поток операцию mark().

**nullReader()** - возвращает новое, Reader которое не читает символы.

**read()** - считывает один символ.

**read(char[] cbuf)** - считывает символы в массив.

**read(char[] cbuf, int off, int len)** - считывает символы в часть массива.

**read(CharBuffer target)** - пытается прочитать символы в указанный буфер символов.

**ready()** - сообщает, готов ли этот поток к чтению.

**reset()** - сбрасывает поток.

**skip(long n)** - пропускает символы.

**transferTo(Writer out)** - считывает все символы из этого средства чтения и записывает символы в заданный элемент записи в том порядке, в котором они были прочитаны.

**Расширяющие его классы:**

**BufferedReader** – это класс который считывает текст из потока ввода символов, буферизуя символы таким образом, чтобы обеспечить эффективное чтение символов, массивов и строк.

**CharArrayReader** - это класс который реализует символьный буфер, который может использоваться в качестве потока ввода символов.

**FileReader** - это класс который считывает текст из символьных файлов, используя размер буфера по умолчанию.

**FilterReader** – это абстрактный класс который используется для чтения отфильтрованных потоков символов.

**InputStreamReader** - это класс который является мостом между потоками байтов и потоками символов. Он считывает байты и декодирует их в символы, используя указанную кодировку.

**LineNumberReader** - это класс который является буферизованным потоком ввода символов, который отслеживает номера строк.

**PipedReader** - это класс который является конвейерным потоком ввода символов.

**PushbackReader** - это класс который является средство чтения потока символов, которое позволяет вставлять символы обратно в поток.

**StringReader** - это класс который является символьным потоком, источником которого является строка.

Writer

**Writer** – это абстрактный класс для записи в потоки символов. Единственными методами, которые должен реализовать подкласс, являются write(char[], int, int), flush() и close().

**Методы:**

**append(char c)** - добавляет указанный символ к этому writer.

**append(CharSequence csq)** - добавляет указанную последовательность символов к этому средству записи.

**append(CharSequence csq, int start, int end)** - добавляет подпоследовательность указанной последовательности символов к этому writer.

**close()** - закрывает поток, сначала удаляя его.

**flush()** - сбрасывает поток.

**nullWriter()** - возвращает новый, Writer который отбрасывает все символы.

**write(char[] cbuf)** - записывает массив символов.

**write(char[] cbuf, int off, int len)** - записывает часть массива символов.

**write(int c)** - записывает один символ.

**write(String str)** - записывает строку.

**write(String str, int off, int len)** - записывает часть строки.

**Расширяющие его классы:**

**BufferedWriter** – это класс который записывает текст в поток вывода символов, буферизуя символы таким образом, чтобы обеспечить эффективную запись отдельных символов, массивов и строк.

**CharArrayWriter** - это класс который реализует символьный буфер, который можно использовать в качестве средства записи.

**FileWriter** - это класс который записывает текст в символьные файлы, используя размер буфера по умолчанию.

**FilterWriter** – это абстрактный класс для записи отфильтрованных потоков символов.

**OutputStreamWriter** - это класс который является это мостом между потоками символов и потоками байтов: символы, записанные в него, кодируются в байты с использованием заданной кодировки.

**PipedWriter** - это класс который является конвейерным потоком вывода символов.

**PrintWriter** - это класс который выводит форматированные представления объектов в текстовый поток вывода.

**StringWriter** - это класс который является символьным потоком, который собирает свои выходные данные в stringbuffer, который затем может быть использован для построения строки.

Работы с файлами

File

**File** – это абстрактный класс, который является представление имен путей к файлам и каталогам.

**Поля:**

**String pathSeparator** - зависящий от системы символ-разделитель пути, представленный в виде строки для удобства.

**сhar pathSeparatorChar -** зависящий от системы символ-разделитель пути.

**String separator -** зависящий от системы символ-разделитель имен по умолчанию, представленный в виде строки для удобства.

**сhar separatorChar -** зависящий от системы символ-разделитель имен по умолчанию.

**Методы:**

**canExecute()** - проверяет, может ли приложение выполнить файл, обозначенный этим абстрактным путем.

**canRead()** - проверяет, может ли приложение прочитать файл, обозначенный этим абстрактным путем.

**canWrite()** - проверяет, может ли приложение изменять файл, обозначенный этим абстрактным путем.

**compareTo(File pathname)** - сравнивает два абстрактных пути лексикографически.

**createNewFile()** - атомарно создает новый пустой файл с именем по этому абстрактному пути тогда и только тогда, когда файл с таким именем еще не существует.

**createTempFile(String prefix, String suffix)** - создает пустой файл в каталоге временных файлов по умолчанию, используя указанный префикс и суффикс для создания его имени.

**createTempFile(String prefix, String suffix, File directory)** - создает новый пустой файл в указанном каталоге, используя заданные строки префикса и суффикса для создания его имени.

**delete()** - удаляет файл или каталог, обозначенный этим абстрактным путем.

**deleteOnExit()** - запрашивает, чтобы файл или каталог, обозначенный этим абстрактным путем, был удален при завершении работы виртуальной машины.

**equals(Object obj)** - проверяет это абстрактное имя пути на равенство с данным объектом.

**exists()** - проверяет, существует ли файл или каталог, обозначенный этим абстрактным путем.

**getAbsoluteFile()** - возвращает абсолютную форму этого абстрактного пути.

**getAbsolutePath()** - возвращает строку абсолютного пути этого абстрактного пути.

**getCanonicalFile()** - возвращает каноническую форму этого абстрактного пути.

**getCanonicalPath()** - возвращает строку канонического пути этого абстрактного пути.

**getFreeSpace()** - возвращает количество нераспределенных байт в разделе, названном этим абстрактным именем пути.

**getName()** - возвращает имя файла или каталога, обозначаемого этим абстрактным путем.

**getParent()** - возвращает строку пути родительского элемента этого абстрактного пути, или null если это имя пути не указывает родительский каталог.

**getParentFile()** - возвращает абстрактный путь к родительскому имени этого абстрактного пути или null, если это имя пути не указывает родительский каталог.

**getPath()** - преобразует этот абстрактный путь в строку имени пути.

**getTotalSpace()** - возвращает размер раздела, названного по этому абстрактному пути.

**getUsableSpace()** - возвращает количество байт, доступных для этой виртуальной машины в разделе, названном по этому абстрактному пути.

**hashCode()** - вычисляет хэш-код для этого абстрактного пути.

**isAbsolute()** - проверяет, является ли этот абстрактный путь абсолютным.

**isDirectory()** - проверяет, является ли файл, обозначенный этим абстрактным путем, каталогом.

**isFile()** - проверяет, является ли файл, обозначенный этим абстрактным путем, обычным файлом.

**isHidden()** - проверяет, является ли файл, названный этим абстрактным путем, скрытым файлом.

**lastModified()** - возвращает время последнего изменения файла, обозначаемого этим абстрактным путем.

**length()** - возвращает длину файла, обозначенную этим абстрактным путем.

**list()** - возвращает массив строк, именующих файлы и каталоги в каталоге, обозначенном этим абстрактным именем пути.

**list(FilenameFilter filter)** - возвращает массив строк, именующих файлы и каталоги в каталоге, обозначаемом этим абстрактным именем пути, которые удовлетворяют указанному фильтру.

**listFiles()** - возвращает массив абстрактных путей, обозначающих файлы в каталоге, обозначенном этим абстрактным путем.

**listFiles(FileFilter filter)** - возвращает массив абстрактных путей, обозначающих файлы и каталоги в каталоге, обозначаемом этим абстрактным путем, которые удовлетворяют указанному фильтру.

**listFiles(FilenameFilter filter)** - возвращает массив абстрактных путей, обозначающих файлы и каталоги в каталоге, обозначаемом этим абстрактным путем, которые удовлетворяют указанному фильтру.

**listRoots()** - перечислите доступные корни файловой системы.

**mkdir()** - создает каталог с именем по этому абстрактному пути.

**mkdirs()** - создает каталог с именем по этому абстрактному пути, включая любые необходимые, но несуществующие родительские каталоги.

**renameTo(File dest)** - переименовывает файл, обозначенный этим абстрактным путем.

**setExecutable(boolean executable)** - удобный метод для установки разрешения владельца на выполнение для этого абстрактного пути.

**setExecutable(boolean executable, boolean ownerOnly)** - задает разрешение владельца или всех пользователей на выполнение для этого абстрактного пути.

**setLastModified(long time)** - задает время последнего изменения файла или каталога, названного этим абстрактным путем.

**setReadable(boolean readable)** - удобный способ установить разрешение владельца на чтение для этого абстрактного пути.

**setReadable(boolean readable, boolean ownerOnly)** - задает разрешение владельца или всех пользователей на чтение для этого абстрактного пути.

**setReadOnly()** - помечает файл или каталог, названный этим абстрактным путем, так что разрешены только операции чтения.

**setWritable(boolean writable)** - удобный способ установить разрешение владельца на запись для этого абстрактного пути.

**setWritable(boolean writable, boolean ownerOnly) - з**адает разрешение владельца или всех пользователей на запись для этого абстрактного пути.

**toPath()** - возвращает java.nio.file.Path объект, созданный на основе этого абстрактного пути.

**toString()** - возвращает строку пути этого абстрактного пути.

**toURI()** - создает file: URI, который представляет этот абстрактный путь.

**toURL()** - не рекомендуется. Этот метод автоматически не экранирует символы, которые являются недопустимыми в URL-адресах.

**Классы и интерфейсы, которые наследуют его:**

**FileDescriptor** – это класс который представляет экземпляры класса файлового дескриптора которые служат непрозрачным дескриптором для базовой структуры, специфичной для конкретной машины, представляющей открытый файл, открытый сокет или другой источник или приемник байтов.

**FileFilter** - это класс который представляет фильтр для абстрактных имен путей.

**FilenameFilter** - это интерфейс который используются для фильтрации имен файлов.

**FilePermission** - это класс который представляет доступ к файлу или каталогу.

**RandomAccessFile** - это класс который представляет как чтение, так и запись в файл произвольного доступа.

**FileStore** – это класс который является хранилище для файлов.

**FileSystem** - это класс который является фабрикой объектов для доступа к файлам и другим объектам в файловой системе.

**FileSystems** - это класс который является фабрикой методом для файловых систем.

**FileVisitOption** - это класс который определяет параметры обхода дерева файлов.

**FileVisitor<T>** - это интерфейс который позволяет осуществлять просмотр файлов.

**FileVisitResult** - это класс который является перечислением результатов a FileVisitor.

**SimpleFileVisitor<T>** - это класс который является простым посетителем файлов с поведением по умолчанию для посещения всех файлов и повторной выдачи ошибок ввода-вывода.

Files

**Files** – это класс который состоит исключительно из статических методов, которые работают с файлами, каталогами или другими типами файлов.

**Методы:**

**copy(InputStream in, Path target, CopyOption... options)** - копирует все байты из входного потока в файл.

**copy(Path source, OutputStream out)** - копирует все байты из файла в выходной поток.

**copy(Path source, Path target, CopyOption... options)** - скопируйте файл в целевой файл.

**createDirectories(Path dir, FileAttribute<?>... attrs)** - создает каталог, сначала создавая все несуществующие родительские каталоги.

**createDirectory(Path dir, FileAttribute<?>... attrs)** - создает новый каталог.

**createFile(Path path, FileAttribute<?>... attrs)** - создает новый и пустой файл, завершающийся ошибкой, если файл уже существует.

**createLink(Path link, Path existing)** - создает новую ссылку (запись каталога) для существующего файла (необязательная операция).

**createSymbolicLink(Path link, Path target, FileAttribute<?>... attrs)** - создает символическую ссылку на целевой объект (необязательная операция).

**createTempDirectory(String prefix, FileAttribute<?>... attrs)** - создает новый каталог в каталоге временных файлов по умолчанию, используя указанный префикс для создания его имени.

**createTempDirectory(Path dir, String prefix, FileAttribute<?>... attrs) - с**оздает новый каталог в указанном каталоге, используя указанный префикс для создания его имени.

**createTempFile(String prefix, String suffix, FileAttribute<?>... attrs)** - создает пустой файл в каталоге временных файлов по умолчанию, используя указанный префикс и суффикс для создания его имени.

**createTempFile(Path dir, String prefix, String suffix, FileAttribute<?>... attrs)** - создает новый пустой файл в указанном каталоге, используя заданные строки префикса и суффикса для создания его имени.

**delete(Path path)** - удаляет файл.

**deleteIfExists(Path path)** - удаляет файл, если он существует.

**exists(Path path, LinkOption... options)** - проверяет, существует ли файл.

**find(Path start, int maxDepth, BiPredicate<Path,BasicFileAttributes> matcher, FileVisitOption... options)** - возвращают Stream которые лениво заполняются Path путем поиска файлов в файловом дереве с корнем в заданном стартовом файле.

**getAttribute(Path path, String attribute, LinkOption... options)** - считывает значение атрибута файла.

**getFileAttributeView(Path path, Class<V> type, LinkOption... options)** - возвращает представление атрибута файла заданного типа.

**getFileStore(Path path)** - возвращает FileStore, представляющее файловое хранилище, в котором находится файл.

**getLastModifiedTime(Path path, LinkOption... options)** - возвращает время последнего изменения файла.

**getOwner(Path path, LinkOption... options)** - возвращает владельца файла.

**getPosixFilePermissions(Path path, LinkOption... options)** - возвращает права доступа к файлу в формате POSIX.

**isDirectory(Path path, LinkOption... options)** - проверяет, является ли файл каталогом.

**isExecutable(Path path)** - проверяет, является ли файл исполняемым.

**isHidden(Path path)** - указывает, считается ли файл скрытым или нет.

**isReadable(Path path)** - проверяет, доступен ли файл для чтения.

**isRegularFile(Path path, LinkOption... options)** - проверяет, является ли файл обычным файлом с непрозрачным содержимым.

**isSameFile(Path path, Path path2)** - проверяет, находятся ли два пути к одному и тому же файлу.

**isSymbolicLink(Path path)** - проверяет, является ли файл символической ссылкой.

**isWritable(Path path)** - проверяет, доступен ли файл для записи.

**lines(Path path)** - считывать все строки из файла как Stream.

**lines(Path path, Charset cs)** - считывать все строки из файла как Stream.

**list(Path dir)** - возвращают лениво заполненный каталог Stream, элементами которого являются записи в каталоге.

**mismatch(Path path, Path path2)** - находит и возвращает позицию первого несовпадающего байта в содержимом двух файлов, или -1L если несоответствия нет.

**move(Path source, Path target, CopyOption... options)** - переместите или переименуйте файл в целевой файл.

**newBufferedReader(Path path)** - открывает файл для чтения, возвращая BufferedReader для эффективного чтения текста из файла.

**newBufferedReader(Path path, Charset cs)** - открывает файл для чтения, возвращая BufferedReader который может быть использован для эффективного чтения текста из файла.

**newBufferedWriter(Path path, Charset cs, OpenOption... options)** - открывает или создает файл для записи, возвращая BufferedWriter который может быть использован для эффективной записи текста в файл.

**newBufferedWriter(Path path, OpenOption... options)** - открывает или создает файл для записи, возвращая BufferedWriter для эффективной записи текста в файл.

**newByteChannel(Path path, OpenOption... options)** - открывает или создает файл, возвращая доступный для поиска байтовый канал для доступа к файлу.

**newByteChannel(Path path, Set<? extends OpenOption> options, FileAttribute<?>... attrs)** - открывает или создает файл, возвращая доступный для поиска байтовый канал для доступа к файлу.

**newDirectoryStream(Path dir)** - открывает каталог, возвращая DirectoryStream для перебора всех записей в каталоге.

**newDirectoryStream(Path dir, String glob)** - открывает каталог, возвращая DirectoryStream для перебора записей в каталоге.

**newDirectoryStream(Path dir, DirectoryStream.Filter<? super Path> filter)** - открывает каталог, возвращая DirectoryStream для перебора записей в каталоге.

**newInputStream(Path path, OpenOption... options)** - открывает файл, возвращая входной поток для чтения из файла.

**newOutputStream(Path path, OpenOption... options)** - открывает или создает файл, возвращая выходной поток, который может использоваться для записи байтов в файл.

**notExists(Path path, LinkOption... options)** - проверяет, не существует ли файл, расположенный по этому пути.

**probeContentType(Path path)** - проверяет тип содержимого файла.

**readAllBytes(Path path)** - считывает все байты из файла.

**readAllLines(Path path)** - прочитайте все строки из файла.

**readAllLines(Path path, Charset cs)** - прочитайте все строки из файла.

**readAttributes(Path path, Class<A> type, LinkOption... options)** - считывает атрибуты файла как массовую операцию.

**readAttributes(Path path, String attributes, LinkOption... options)** - считывает набор атрибутов файла как массовую операцию.

**readString(Path path)** - считывает все содержимое из файла в строку, декодируя из байтов в символы с помощью UTF-8 charset.

**readString(Path path, Charset cs)** - считывает все символы из файла в строку, декодируя из байтов в символы с использованием указанной кодировки.

**readSymbolicLink(Path link)** - считывает цель символьной ссылки (необязательная операция).

**setAttribute(Path path, String attribute, Object value, LinkOption... options)** - задает значение атрибута файла.

**setLastModifiedTime(Path path, FileTime time)** - обновляет атрибут времени последнего изменения файла.

**setOwner(Path path, UserPrincipal owner)** - обновляет владельца файла.

**setPosixFilePermissions(Path path, Set<PosixFilePermission> perms)** - устанавливает разрешения POSIX для файла.

**size(Path path)** - возвращает размер файла (в байтах).

**walk(Path start, int maxDepth, FileVisitOption... options)** - верните a, Stream который лениво заполняется Path путем обхода дерева файлов с корнем в данном стартовом файле.

**walk(Path start, FileVisitOption... options)** - верните a, Stream который лениво заполняется Path путем обхода дерева файлов с корнем в данном стартовом файле.

**walkFileTree(Path start, FileVisitor<? super Path> visitor)** - перемещается по файловому дереву.

**walkFileTree(Path start, Set<FileVisitOption> options, int maxDepth, FileVisitor<? super Path> visitor)** - перемещается по файловому дереву.

**write(Path path, byte[] bytes, OpenOption... options)** - записывает байты в файл.

**write(Path path, Iterable<? extends CharSequence> lines, Charset cs, OpenOption... options)** - запишите строки текста в файл.

**write(Path path, Iterable<? extends CharSequence> lines, OpenOption... options)** - запишите строки текста в файл.

**writeString(Path path, CharSequence csq, Charset cs, OpenOption... options)** - запишите последовательность символов в файл.

**writeString(Path path, CharSequence csq, OpenOption... options)** - запишите последовательность символов в файл.

Path

**Path** – этоинтерфейс который может использоваться для определения местоположения файла в файловой системе. Обычно он представляет путь к файлу, зависящий от системы.

**Методы:**

**compareTo(Path other)** - сравнивает два абстрактных пути лексикографически.

**endsWith(String other)** - проверяет, заканчивается ли этот путь на Path, созданный путем преобразования заданной строки пути, в точности способом, указанным endsWith(Path) методом.

**endsWith(Path other)** - проверяет, заканчивается ли этот путь указанным путем.

**equals(Object other)** - проверяет этот путь на равенство с данным объектом.

**getFileName()** - возвращает имя файла или каталога, обозначаемого этим путем как Path объект.

**getFileSystem()** - возвращает файловую систему, которая создала этот объект.

**getName(int index)** - возвращает элемент name этого пути в виде Path объекта.

**getNameCount()** - возвращает количество элементов name в пути.

**getParent()** - возвращает родительский путь, или null если у этого пути нет родительского пути.

**getRoot()** - возвращает корневой компонент этого пути в виде Path объекта или null, если у этого пути нет корневого компонента.

**hashCode()** - вычисляет хэш-код для этого пути.

**isAbsolute()** - указывает, является ли этот путь абсолютным.

**iterator()** - возвращает итератор по элементам name этого пути.

**normalize()** - возвращает путь, который является этим путем с удаленными избыточными элементами name.

**of(String first, String... more)** - возвращает Path путем преобразования строки пути или последовательности строк, которые при объединении образуют строку пути.

**of(URI uri)** - возвращает Path путем преобразования URI.

**register(WatchService watcher, WatchEvent.Kind<?>... events**) - регистрирует файл, расположенный по этому пути, в службе просмотра.

**register(WatchService watcher, WatchEvent.Kind<?>[] events, WatchEvent.Modifier... modifiers)** - регистрирует файл, расположенный по этому пути, в службе просмотра.

**relativize(Path other)** - создает относительный путь между этим путем и заданным путем.

**resolve(String other)** - преобразует заданную строку path в a Path и преобразует ее в соответствии с этим Path точно так, как указано в resolve методе.

**resolve(Path other)** - сопоставьте указанный путь с этим путем.

**resolveSibling(String other)** - преобразует заданную строку path в Path и преобразует ее в соответствие с parent path этого пути точно так, как указано в resolveSibling методе.

**resolveSibling(Path other)** - преобразует заданный путь в parent путь этого пути.

**startsWith(String other)** - проверяет, начинается ли этот путь с Path, созданного путем преобразования заданной строки пути, в точности способом, указанным startsWith(Path) методом.

**startsWith(Path other)** - проверяет, начинается ли этот путь с заданного пути.

**subpath(int beginIndex, int endIndex)** - возвращает относительное значениеPath, которое является последовательностью элементов name этого пути.

**toAbsolutePath()** - возвращает Path объект, представляющий абсолютный путь этого пути.

**toFile()** - возвращает File объект, представляющий этот путь.

**toRealPath(LinkOption... options)** - возвращает реальный путь к существующему файлу.

**toString()** - возвращает строковое представление этого пути.

**toUri()** - возвращает URI для представления этого пути.

Paths

**Paths** – это класс который состоит исключительно из статических методов, которые возвращают Path путем преобразования строки пути или URI.

**Методы:**

**get(String first, String... more)** - преобразует строку пути или последовательность строк, которые при объединении образуют строку пути, в Path.

**get(URI uri)** - преобразует данный URI в Path объект.

Работа с консолью

Console

**Console** – этокласс которыйпредоставляет доступ к консольному устройству на основе символов, если таковое имеется, связанному с текущей виртуальной машиной Java.

**Методы:**

**charset()** - возвращает Charset объект, используемый для Console.

**flush()** - очищает консоль и принудительно записывает любой буферизованный вывод немедленно.

**format(String fmt, Object... args)** - записывает форматированную строку в выходной поток этой консоли, используя указанную строку формата и аргументы.

**printf(String format, Object... args)** - удобный способ записи форматированной строки в выходной поток этой консоли с использованием указанной строки формата и аргументов.

**reader()** - извлекает уникальный Reader объект, связанный с этой консолью.

**readLine()** - считывает одну строку текста с консоли.

**readLine(String fmt, Object... args)** - предоставляет форматированное приглашение, затем считывает одну строку текста с консоли.

**readPassword()** - считывает пароль или кодовую фразу с консоли с отключенным эхом

**readPassword(String fmt, Object... args)** - выдает форматированное приглашение, затем считывает пароль или кодовую фразу с консоли с отключенным эхом.

**writer()** - извлекает уникальный PrintWriter объект, связанный с этой консолью.

Scanner

**Scanner** – это класс представляющий простой текстовый сканер, который может анализировать примитивные типы и строки с использованием регулярных выражений.

**Методы:**

**close()** - закрывает этот сканер.

**delimiter()** - возвращает Pattern это Scanner в настоящее время используется для сопоставления разделителей.

**findAll(String patString)** - возвращает поток результатов сопоставления, которые соответствуют предоставленной строке шаблона.

**findAll(Pattern pattern)** - возвращает поток результатов сопоставления с этим сканером.

**findInLine(String pattern)** - пытается найти следующее вхождение шаблона, построенного из указанной строки, игнорируя разделители.

**findInLine(Pattern pattern)** - пытается найти следующее вхождение указанного шаблона, игнорируя разделители.

**findWithinHorizon(String pattern, int horizon)** - пытается найти следующее вхождение шаблона, построенного из указанной строки, игнорируя разделители.

**findWithinHorizon(Pattern pattern, int horizon)** - пытается найти следующее вхождение указанного шаблона.

**hasNext()** - возвращает true, если у этого сканера есть другой токен на входе.

**hasNext(String pattern)** - возвращает true, если следующий токен соответствует шаблону, построенному из указанной строки.

**hasNext(Pattern pattern)** - возвращает true, если следующий завершенный маркер соответствует указанному шаблону.

**hasNextBigDecimal()** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как a BigDecimal с помощью nextBigDecimal() метода.

**hasNextBigInteger()** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как BigInteger в базе данных по умолчанию с помощью nextBigInteger() метода.

**hasNextBigInteger(int radix)** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как BigInteger в указанном радиусе с помощью nextBigInteger() метода.

**hasNextBoolean()** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как логическое значение с использованием шаблона без учета регистра, созданного из строки "true | false".

**hasNextByte()** - возвращает значение true, если следующий токен во входных данных этого сканера может быть интерпретирован как байтовое значение в базе данных по умолчанию с помощью nextByte() метода.

**hasNextByte(int radix)** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как байтовое значение в указанном радиусе с помощью nextByte() метода.

**hasNextDouble()** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как двойное значение с помощью nextDouble() метода.

**hasNextFloat()** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как значение с плавающей точкой с помощью nextFloat() метода.

**hasNextInt()** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как значение int в базе данных по умолчанию с помощью nextInt() метода.

**hasNextInt(int radix)** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как значение int в указанном радиусе с помощью nextInt() метода.

**hasNextLine()** - возвращает true, если на входе этого сканера есть другая строка.

**hasNextLong()** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как длинное значение в базе данных по умолчанию с помощью nextLong() метода.

**hasNextLong(int radix)** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как длинное значение в указанном радиусе с помощью nextLong() метода.

**hasNextShort()** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как короткое значение в базе данных по умолчанию с помощью nextShort() метода.

**hasNextShort(int radix)** - возвращает true, если следующий токен во входных данных этого сканера может быть интерпретирован как короткое значение в указанном радиусе с помощью nextShort() метода.

**ioException()** - возвращает IOException последний результат, выданный этим Scanner базовым Readable.

**locale()** - возвращает языковой стандарт этого сканера.

**match()** -возвращает результат соответствия последней операции сканирования, выполненной этим сканером.

**next()** - находит и возвращает следующий завершенный токен из этого сканера.

**next(String pattern)** - возвращает следующий токен, если он соответствует шаблону, построенному из указанной строки.

**next(Pattern pattern)** - возвращает следующий токен, если он соответствует указанному шаблону.

**nextBigDecimal()** - сканирует следующий маркер входных данных как BigDecimal.

**nextBigInteger()** - сканирует следующий маркер входных данных как BigInteger.

**nextBigInteger(int radix)** - сканирует следующий маркер входных данных как BigInteger.

**nextBoolean()** - преобразует следующий токен входных данных в логическое значение и возвращает это значение.

**nextByte()** - сканирует следующий маркер входных данных как byte.

**nextByte(int radix)** - сканирует следующий маркер входных данных как byte.

**nextDouble()** - сканирует следующий маркер входных данных как double.

**nextFloat()** - сканирует следующий маркер входных данных как float.

**nextInt()** - сканирует следующий маркер входных данных как int.

**nextInt(int radix)** - сканирует следующий маркер входных данных как int.

**nextLine()** - продвигает этот сканер дальше текущей строки и возвращает пропущенный ввод.

**nextLong()** - сканирует следующий маркер входных данных как long.

**nextLong(int radix)** - сканирует следующий маркер входных данных как long.

**nextShort()** - сканирует следующий маркер входных данных как short.

**nextShort(int radix)** - сканирует следующий маркер входных данных как short.

**radix()** - возвращает исходный код этого сканера по умолчанию.

**remove()** - операция удаления не поддерживается этой реализацией Iterator.

**reset()** - сбрасывает этот сканер.

**skip(String pattern)** - пропускает ввод, соответствующий шаблону, построенному на основе указанной строки.

**skip(Pattern pattern)** - пропускает ввод, соответствующий указанному шаблону, игнорируя разделители.

**tokens()** - возвращает поток токенов, разделенных разделителями, из этого сканера.

**toString()** - возвращает строковое представление этого Scanner.

**useDelimiter(String pattern)** - устанавливает шаблон разграничения этого сканера в шаблон, созданный на основе указанного String.

**useDelimiter(Pattern pattern)** - устанавливает шаблон разграничения этого сканера в соответствии с указанным шаблоном.

**useLocale(Locale locale)** - устанавливает языковой стандарт этого сканера в указанный языковой стандарт.

**useRadix(int radix)** - устанавливает радиус этого сканера по умолчанию в указанный радиус.