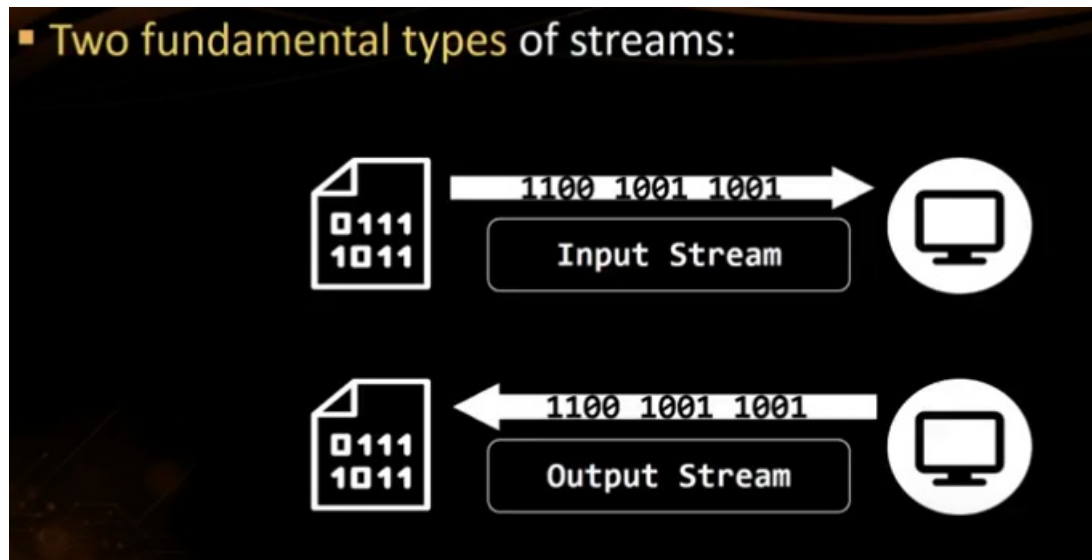


Потоци(Streams) - начин за пренос на данни. Позволява ни да пренасяме информация от нашата програма към някакъв външен за нея източник. Независимо от езика на който пишем, съществуват два вида потоци - входящи и изходящи.



В единия случай имам входящ поток(stream) или четене на информация от файла, а във втория случай потокът е изходящ или записване на информация върху файла.

Потоците(Streams) са винаги еднопосочни. Няма такова нещо като двупосочен стрийм.

Как се отваря поток в Java ?

Използва се класа `FileInputStream`.

```
String path = "C:\\input.txt";  
  
FileInputStream fileStream =  
    new FileInputStream(path);
```

За да четем от файл трябва да използваме метода `read()` от класа `FileInputStream`, както е показано на следващата картинка. (`fileStream` идва от горната картката, а този ред с код е нейно продължение).

```
int oneByte = fileStream.read();
```

Стриймовете работят с байтове. Те не се интересуват каква е информацията от долу , а че има такава.

Opening a File Stream

```
String path = "C:\\input.txt";

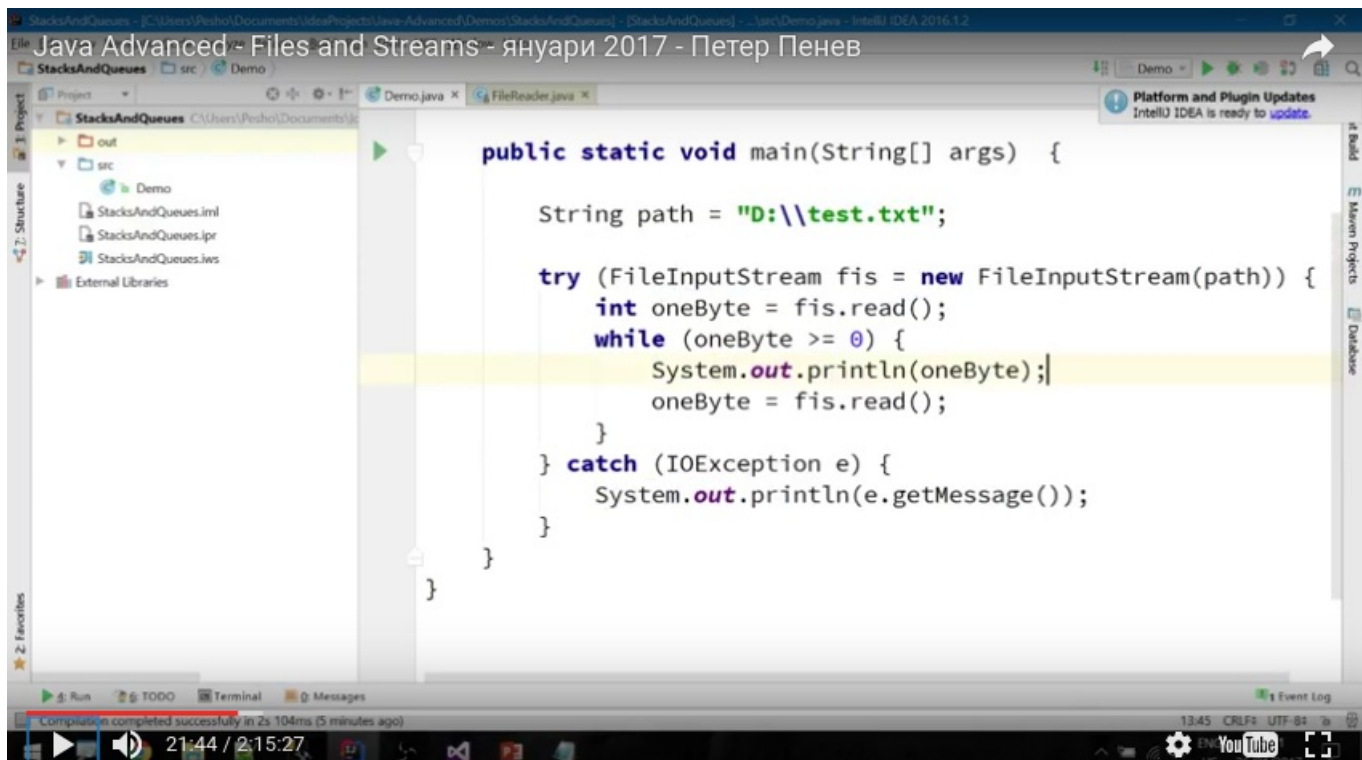
FileInputStream fileStream =
    new FileInputStream(path);

int oneByte = fileStream.read();
while (oneByte >= 0) {
    System.out.print(oneByte);
    oneByte = fileStream.read();
}
```

Returns -1 if empty

КОГАТО СЕ РАБОТИ СЪС СТРИЙМОВЕ ТРЯБВА ВИНАГИ ДА СИ ОСВОБОЖДАВАМЕ РЕСУРСИТЕ.

Втори начин да се чете от файл, който е по-лесен и по-умен е следният:



Тук когато се излезе от try автоматично ще затвори ресурсите и не се налага да се изписва fis.close();

Какво се случва ако не затворим ресурсите ?

Ако програмката, която пишем е малка , операционната система е достатъчно умна да се усети, че след известно време файлът не се използва и ще затвори потока автоматично. Обаче ако имаме някакво по-голямо приложение, което всяка секунда отваря хиляди връзки към нещо. Примерно youtube - всеки един момент получават заявки към сървърите си, отварят се потоци, гледат се видеа и ако не се затворят потоците... сървърите ще забият.

Потоци в Java

Байтови потоци и такива, които работят с букви. - byte streams , character streams.

Байтове потоци. (Byte Streams)

Java Advanced - Files and Streams - януари 2017 - Петер Пенев

Byte Stream

SOFTWARE UNIVERSITY FOUNDATION

- Byte streams are the **lowest level streams**
 - Byte streams can read or write **one byte at a time**
 - All byte streams **descend from InputStream and OutputStream**

InputStream

100101 111111 100011 -1

OutputStream

Character streams

Java Advanced - Files and Streams - януари 2017 - Петер Пенев

Character Streams

SOFTWARE UNIVERSITY FOUNDATION

- All character streams descend from **FileReader** and **FileWriter**

```
String path = "D:\\input.txt";  
FileReader reader = new FileReader(path);
```

А Б В Г
Д Е Ж З
И К Л М Н
О П Р С Т
У Ф Х Ц Ч Ш Щ
Ъ Ъ
Ы
10101

Разликата между byte и character streams , е че вторите ни дават възможност да работим с различни енкодинг много по-лесно отколкото ако го правим ръчно. Общото между тях е, че отдолу character streams работят с bytes streams.

Combining streams (Комбинирани стриймове) - В Java стриймовете се комбинират и почти не се използват самостоятелно. - wrap-ване на stream-ве.

Combining Streams

- Character streams are often "wrappers" for byte streams.

```
String path = "D:\\input.txt";
```

```
Scanner reader =  
    new Scanner(new FileInputStream(path));
```



Combining Streams

- Character streams are often "wrappers" for byte streams.
 - FileReader uses FileInputStream
 - FileWriter uses FileOutputStream

```
String path = "D:\\input.txt";
```

```
Scanner reader =  
    new Scanner(new FileInputStream(path));
```



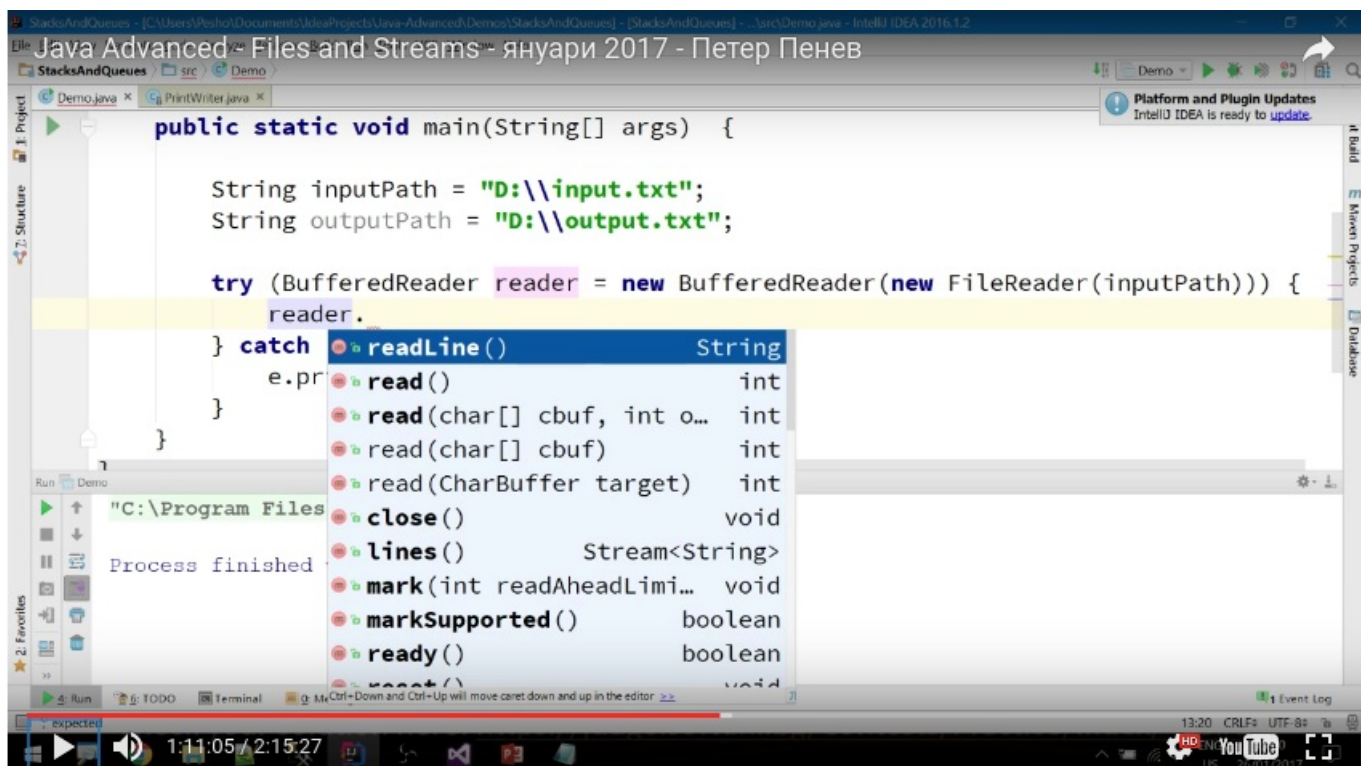
- Reading information in **chunks**
- Significantly **boost performance**

F	i	l	e	s	a	n	d	
46	69	6c	65	73	20	61	6e	64

Position ↑

Buffer 46 69

Подобни са на другите stream-ve , само че отдолу работи с buffer. Помага ни да си оптимизираме работата с файлове, защото когато имаме някакъв буфер, който помни на части се намалява интеракцията с файла. А тя е бавна, защото се чете от харддиска. Същото е и ако информацията идва от мрежата. На картинката горе се вижда как работи. (Имаме буфер с размер 2 и в началото запазват първите два елемента, след това продължава със следващите. Запазва и тях и така до края). И така можем да зададем на един `FileInputStream`, buffer с големина примерно 1000 байта. И той ги прочита тези 1000 байта и ги изпраща като голямо парче информация.



```

Java Advanced - Files and Streams - януари 2017 - Петер Пенев

public static void main(String[] args) {

    String inputPath = "D:\\input.txt";
    String outputPath = "D:\\output.txt";

    try (BufferedReader reader = new BufferedReader(new FileReader(inputPath))) {
        reader.
    } catch (IOException e) {
    }
}
  
```

Method tooltip for `readLine()`:

- `readLine()` String
- `read()` int
- `read(char[] cbuf, int off, int len)` int
- `read(char[] cbuf)` int
- `read(CharBuffer target)` int
- `close()` void
- `lines()` Stream<String>
- `mark(int readAheadLimit)` void
- `markSupported()` boolean
- `ready()` boolean

System in , System out - те също са стриймове.

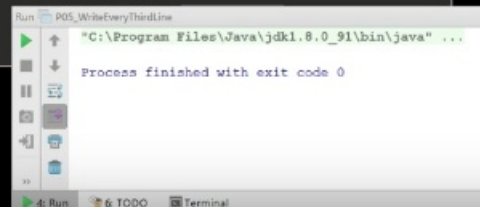
Command Line I/O

- Standard Input – **System.in**
- Standard Output – **System.out**
- Standard Error – **System.err**

```
Scanner scanner = new Scanner(System.in);
String line = scanner.nextLine();
System.out.println(line);
```

Input Stream

Output Stream



1:13:58 / 2:15:27



YouTube



Файлове и пътища (Files and paths)

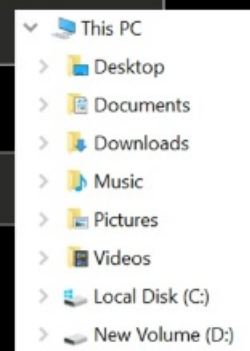
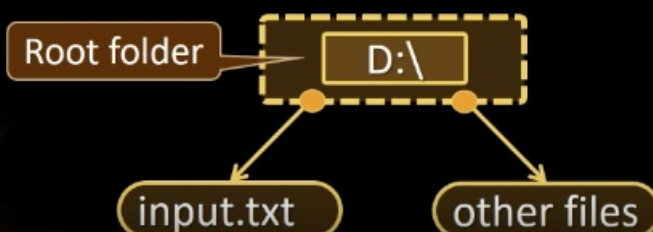
Пътища(Paths) - Описание на мястото, където се намира даден файл. Модерните файлови системи работят с дървовидна файлова структура.

- The location of a file in the file system

D:\input.txt

- Represented in Java by the Path class

```
Path path = Paths.get("D:\\input.txt");
```



1:15:18 / 2:15:27



YouTube



Файлове(Files)

Files

- Provides static methods for creating streams

```
Path path = Paths.get("D:\\input.txt");

try (BufferedReader reader =
    Files.newBufferedReader(path)) {
    // TODO: work with file
} catch (IOException e) {
    // TODO: handle exception
}
```

Този клас ни позволява да си създаваме стриймове по малко по-различен начин. Това е от Java 7 насам. От Oracle казват че `BufferedReader reader = Files.newBufferedReader(path)` е по-бърз и по-лесен от

```
BufferedReader reader = new BufferedReader(new FileReader(path))
```

Следващия код, прочита всички редове от даден файл и ги запазва в лист от стрингове. Удобно е само за малки файлове, ако се чете по този начин файл голям ГБ, ще стане много бавно, защото всичко се пази в рама. Става единствено за по-малки файлчета.

```
List<String> lines = Files.readAllLines(path);
```

File Class in Java

- Provides methods for quick and easy manipulation of files

```
import java.io.File;

File file = new File("D:\\input.txt");

boolean isExisting = file.exists();
long length = file.length();
boolean isDirectory = file.isDirectory();
File[] files = file.listFiles();
```



има левел, опит, и има някакви предмети и когато искаме да излезем от играта, искаме примерно тези неща да се запишат някъде, за да може при следващото пускане на играта, нашият герой да започне от това ниво, с този опит и тези предмети. Ако не запишем тези неща в някакъв външен файл, когато се спре играта те ще се загубят и таква при всяко стартиране на играта героят ще започва от Ота, което не е много яко :D За това се използва сериализация.

Или накратко казано отваряме поток към даден файл и записваме състоянието на обект.

Java Advanced - Files and Streams - януари 2017 - Петер Пенев

Serialization

SOFTWARE UNIVERSITY FOUNDATION

- Save objects to a file


```
List<String> names = new ArrayList<>();
Collections.addAll(names, "Mimi", "Gosho");

FileOutputStream fos = new FileOutputStream(path);
ObjectOutputStream oos =
    new ObjectOutputStream(fos);

oos.writeObject(names);

// TODO: handle exceptions
```

Save objects to .ser file



1:57:32 / 2:15:27

HD YouTube

Deserialization - Десериализиране на обекти.

Java Advanced - Files and Streams - януари 2017 - Петер Пенев

Deserialization


SOFTWARE UNIVERSITY FOUNDATION

- Load objects from a file

```
FileInputStream fis =
    new FileInputStream(path);
ObjectInputStream oos =
    new ObjectInputStream(fis);

List<String> names =
    (List<String>) oos.readObject();

// TODO: handle exceptions
```



1:58:19 / 2:15:27

HD YouTube

Както е показано на предната снимка резултатът трябва да се кастане, защото веднъж записан във файл компилаторът няма как да знае какъв тип са обектите или обекта.


```
List<String> names =  
→ (List<String>) oos.readObject();
```

Друго важно нещо когато искаме да сериализираме обекти, е че трябва да се имплементира интерфейса `Serializable`. (както е показано на следващата снимка). Интерфейси ще учим малко по-нататък в ООП.

Java Advanced - Files and Streams - януари 2017 - Петер Пенев

Serialization of Custom Objects



- Custom objects should implement the `Serializable` interface

```
class Cube implements Serializable {  
    String color;  
    double width;  
    double height;  
    double depth;  
}
```

