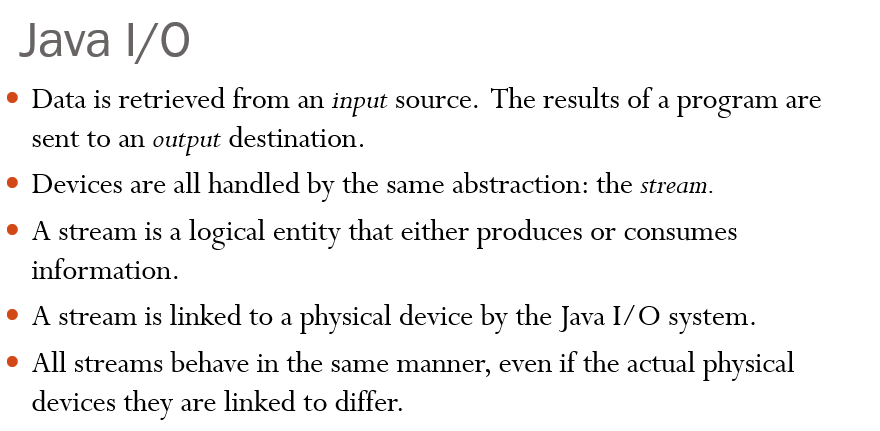
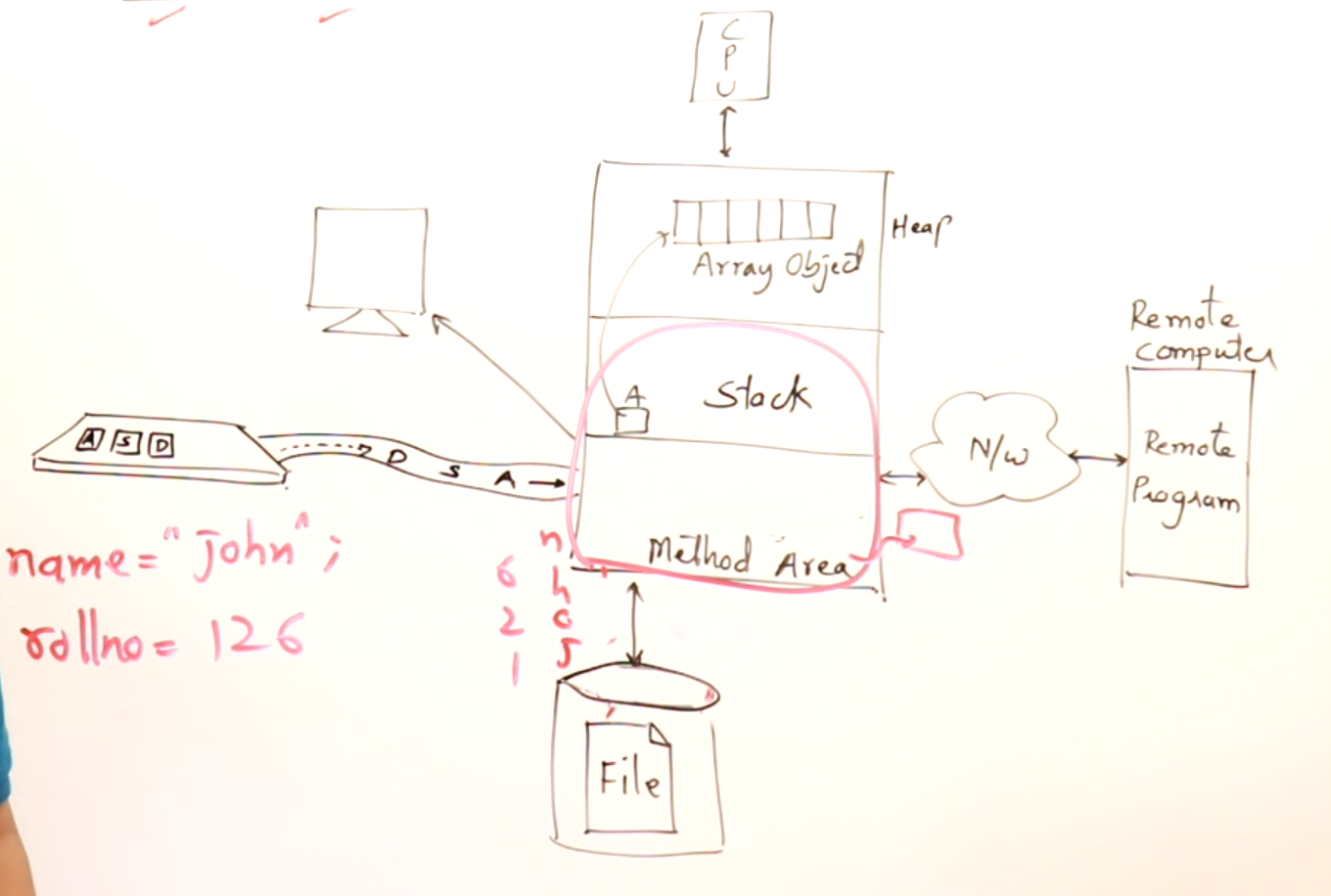
# **Input Output Streams**





## **Input Output**

The things which are external to a program 🡪 resource of that program

For any JAVA program,  
Context of the program 🡪 Stack and its Method Area  
Everything inside stack and method area are the part of a program and all other are the resource (external to the program)

Resource-1 🡪 Monitor (Output)  
Resource-2 🡪 Keyboard (Input)  
Resource-3 🡪 File in a disk  
Resource-4 🡪 Network and its related programs  
Resource-5 🡪 Heap memory

JAVA program will interact with the available resources by transferring the data.

## **Streams**

Stream 🡪 flow of data.  
The stream of data can be from program to resource/resource to program.

Why the flow of data is called streams ???  
Generally water pipes are used to transfer water from source to destination.  
After connecting the pipes if I on the tap, water will flow like a stream without any disturbance.  
Similarly, if I type something stream of character/stream of bytes will be flowing from keyboard to the program.

## **Issues related to streams**

Buffer 🡪It is the memory object which is used for holding the data for some time to bring in the compatibility between the devices.

So this JAVA program can be put inside the buffer and slowly that Remote program will consume it.

**Eg:2**Buffering in playing videos 🡪The rate at which the videos are coming from the server is different with respect to the rate at which that video is played on our device.   
So a common buffer is maintained to compensate this mis-match in speed.

**Eg:3**

Taking bath.  
We can’t able to take bath in the tap water, So we will fill some certain quantity of water in bucket and then take bath.   
For bathing 🡪 We need not fill the entire bucket and we

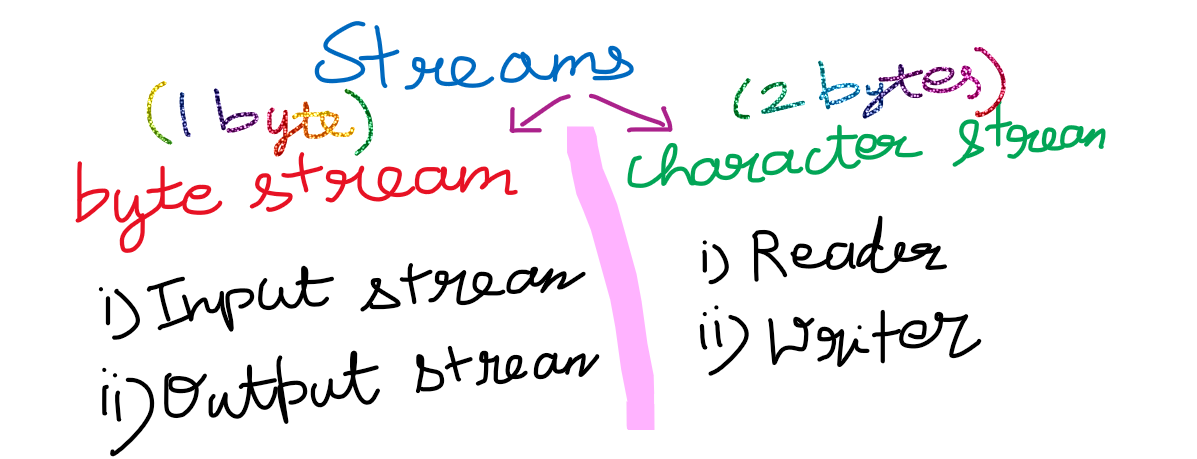
But we require some sufficient water to bath, So we will use the **bucket as a buffer** to store sufficient quantity of water to bath.

**Very very important issues in streams 🡪 We use buffer at many places.**

## **How the data flows in and out ??**

Data is sent not as a single big unit, it is broken and sent byte by byte.

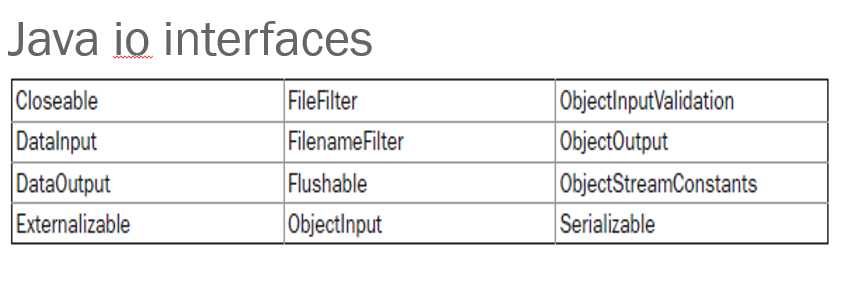
## **Classification of streams**



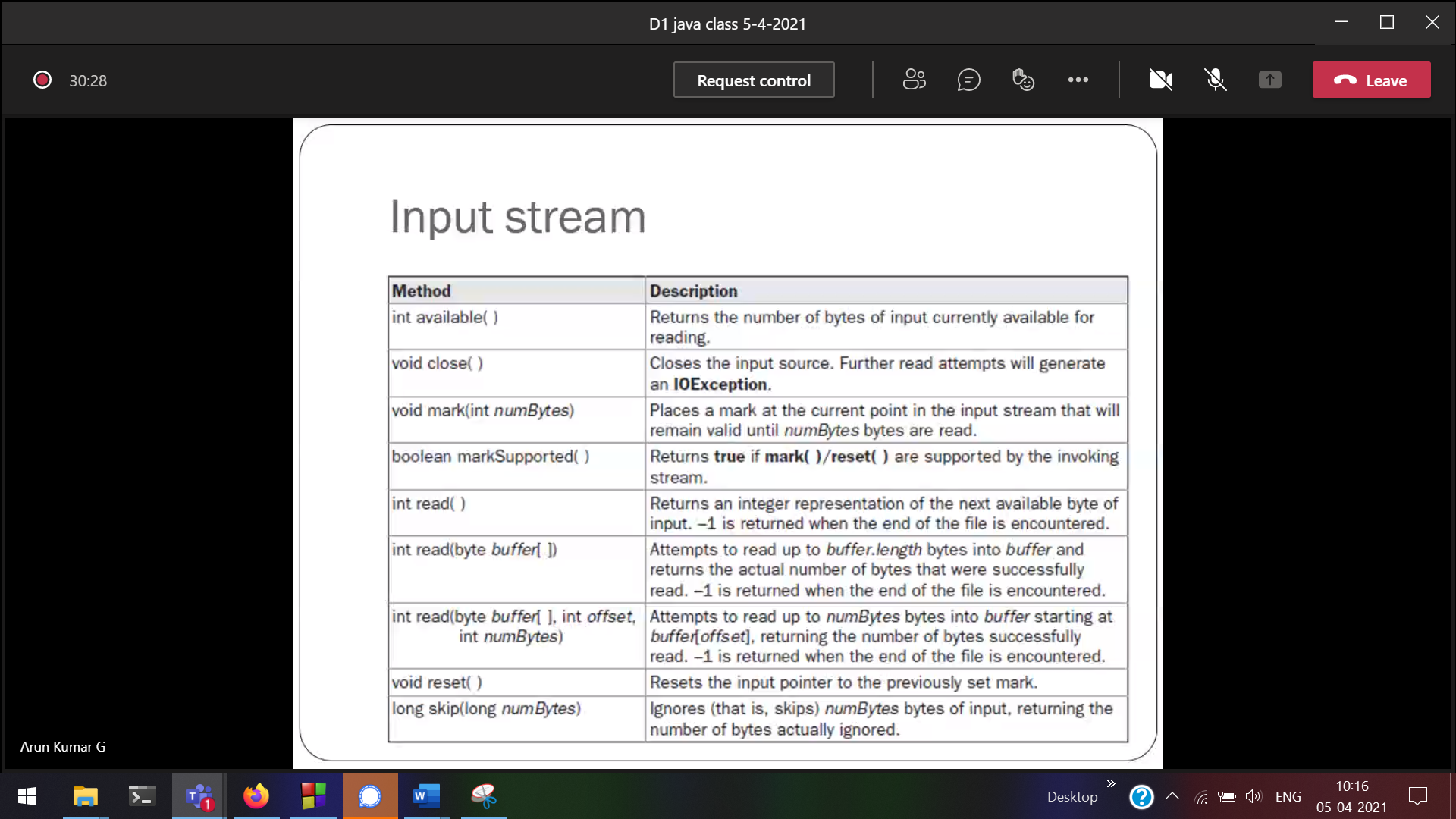
# **JAVA IO classes**



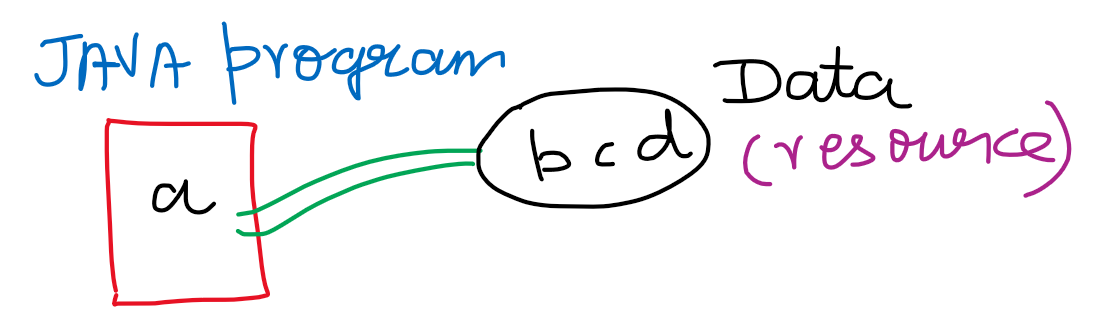
# **JAVA IO interfaces**



# **Input Output Streams Classes**

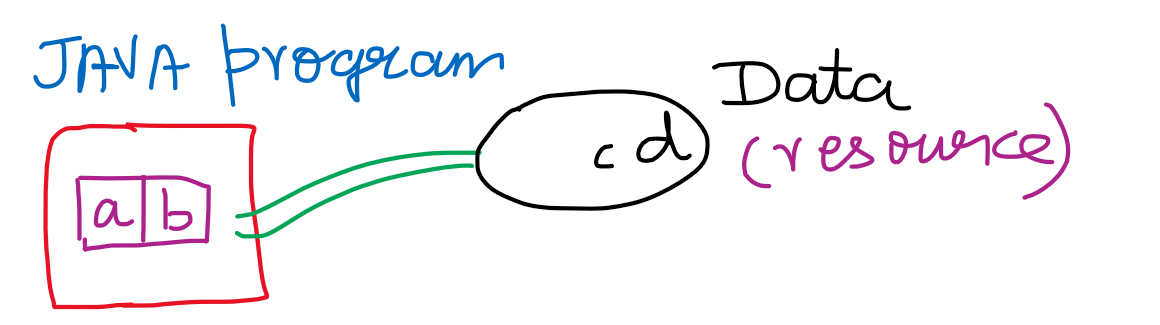


int read() (reading 1 byte at a 🡪 reading the data from Resources into our JAVA program.  
If there are no items available in the resource, int read() will return -1.

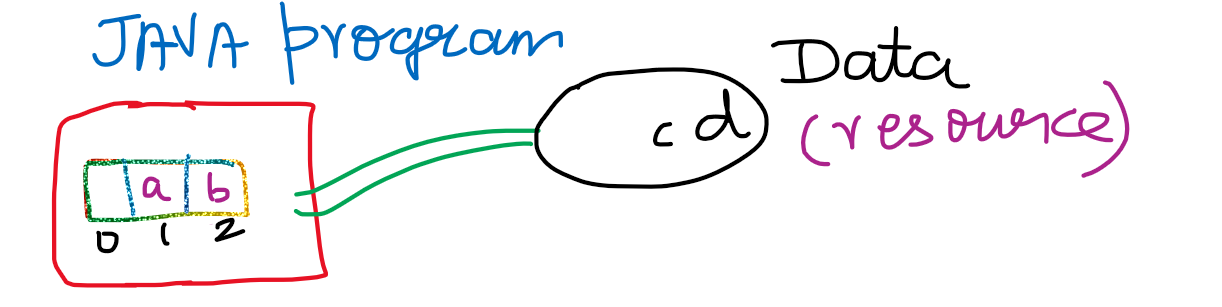


Here a is read, so a will be no longer available in Data.

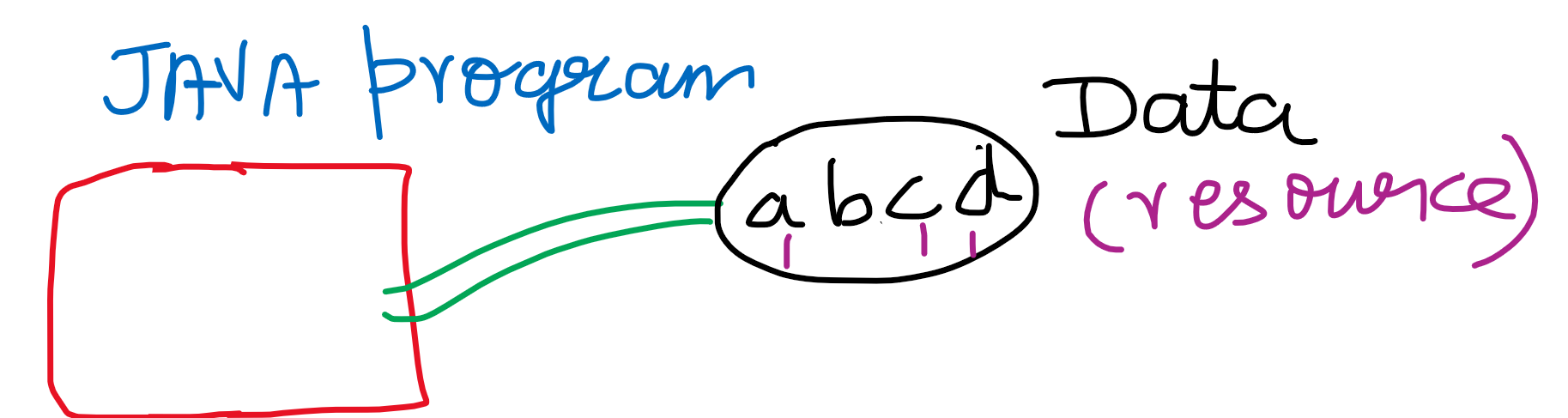
int read(byte[] b) 🡪(reading collection of bytes at a time)



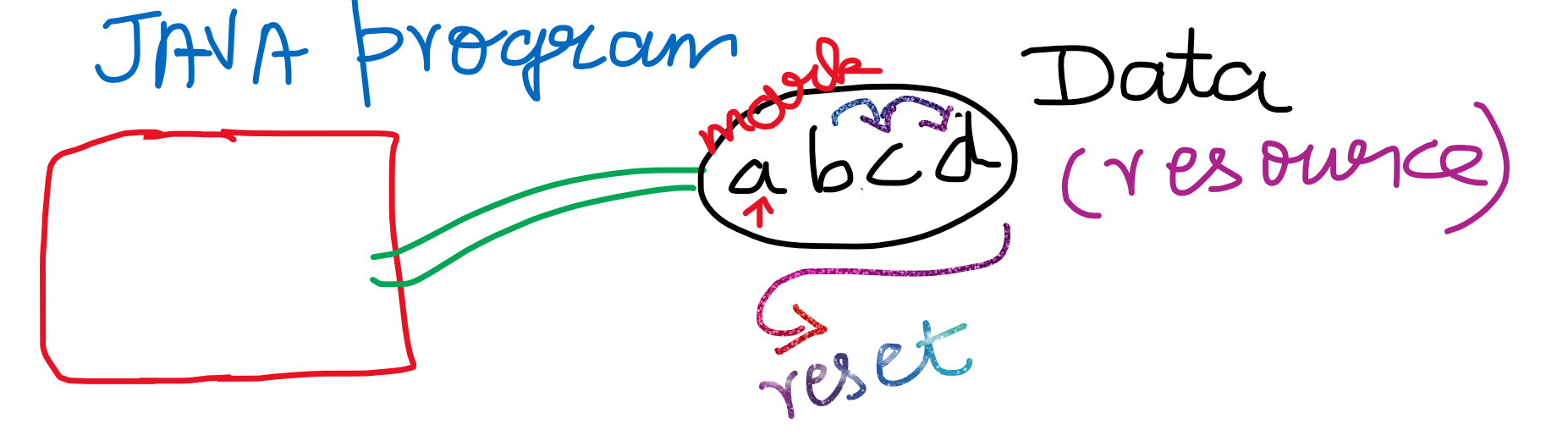
int read(byte[] b , int offset , int len)



int available() 🡪 Number of bytes available in the resource.

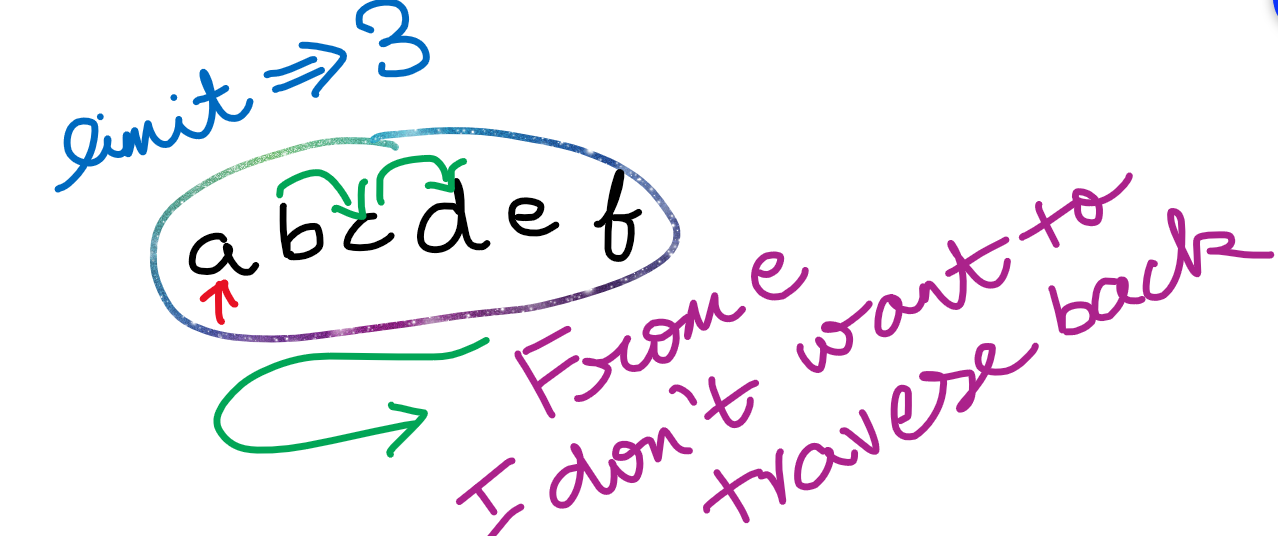
long skip(long n) 🡪b will not be brought into JAVA program itself. b will not be read, it will just discard and proceed further.

void mark(int limit) and void reset()



With mark, we can put up a mark and continue reading.  
With reset, we can traverse back and start from the marked character.

In limit we can mention how long the mark should be there.  
After some time, if we don’t want to traverse back then we can remove that mark.



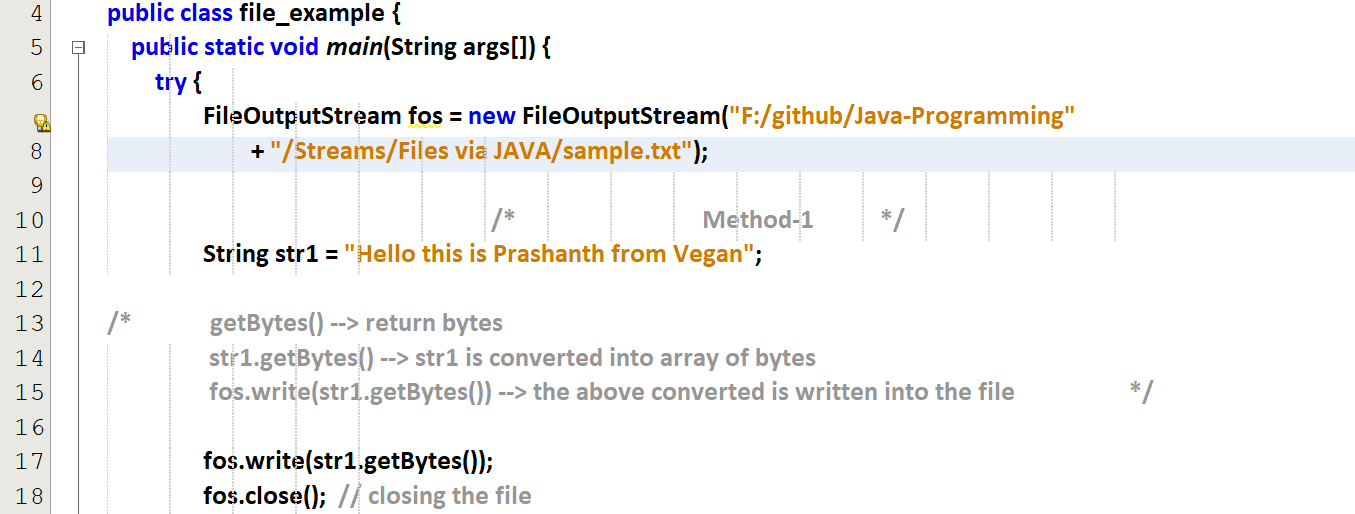
boolean markSupported(int limit) 🡪 Will tell whether mark and reset can be used in the resource.

void close() 🡪 after using a resource/stream we should close that resource/stream.

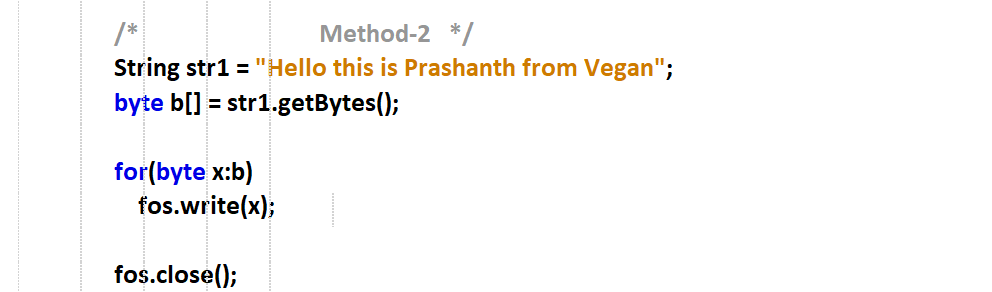
# **File Output Streams**

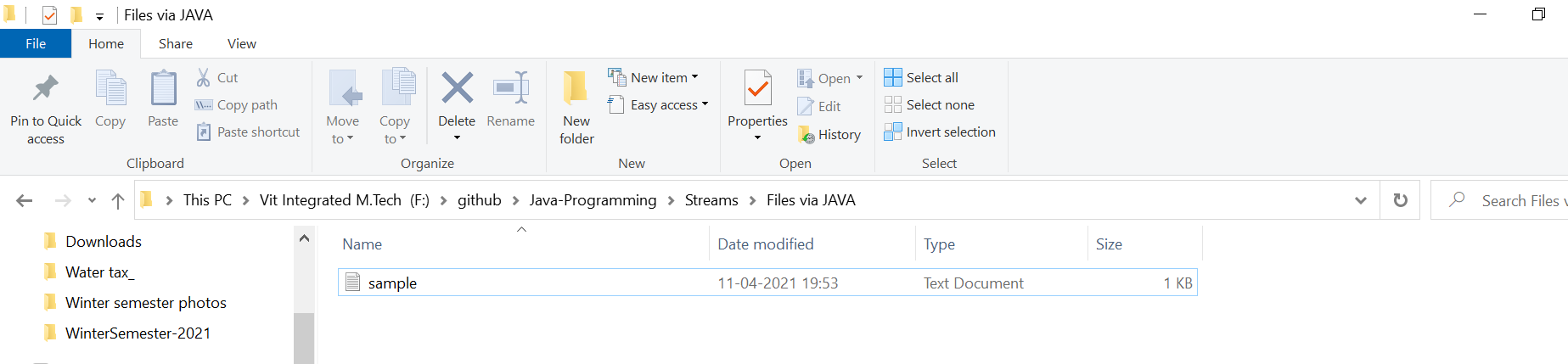
* Create a Test.txt file.
* Using fileOutputStream/FileReader write the contents into the Test.txt
* Using fileInputStream/FileWriter read the contents from Test.txt

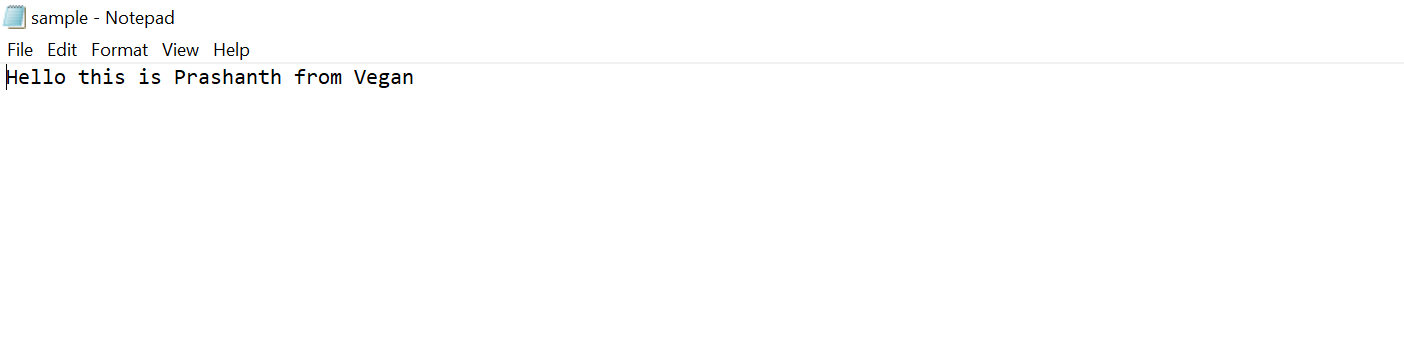
## **Method-1**



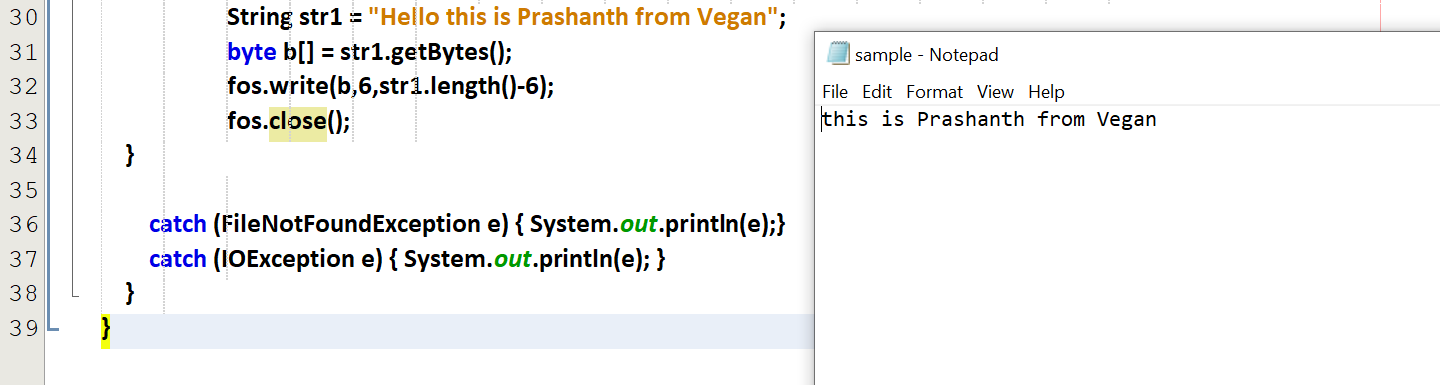
## **Method-2**



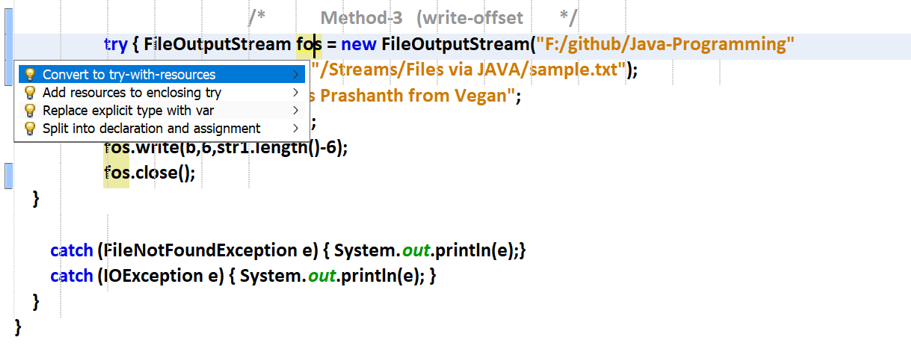
Output 



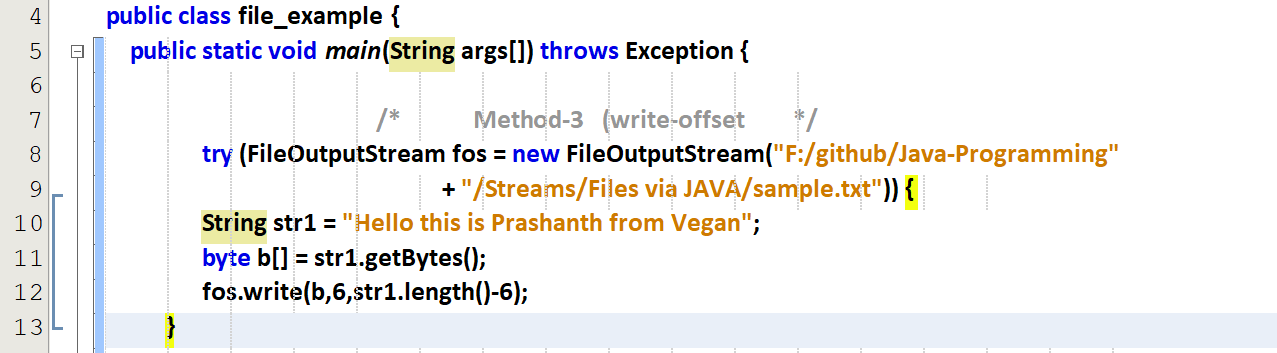
## **Method-3 (write offset)**



Press alt+enter on fos



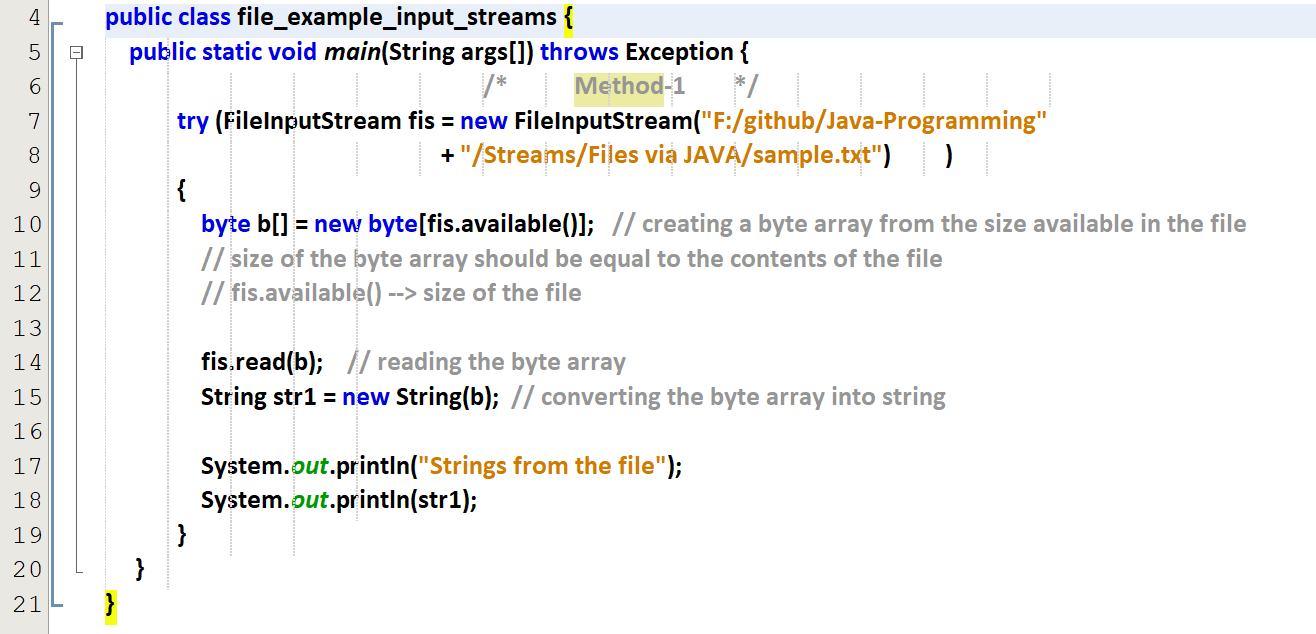
## **Method-4 (with throws Exception)**



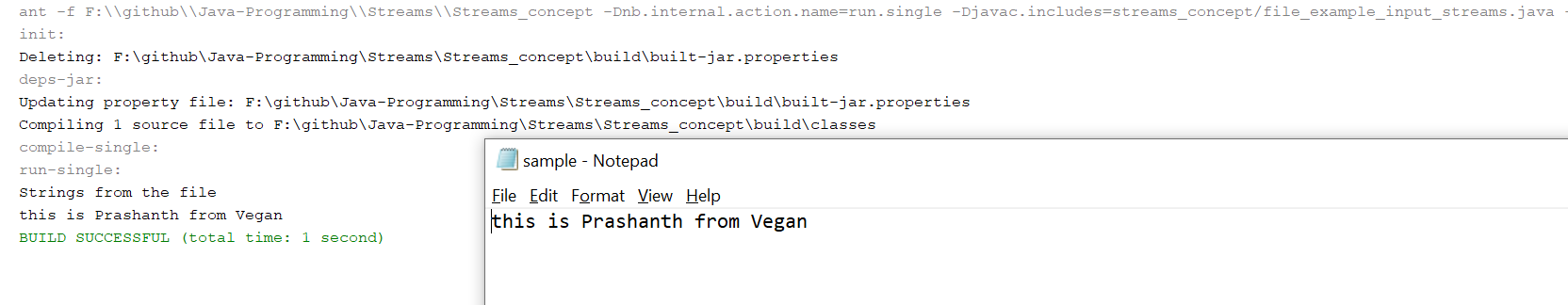
Who will handle this exception 🡪 JVM  
throws in Main will be handled by main()

# **File Input Streams**

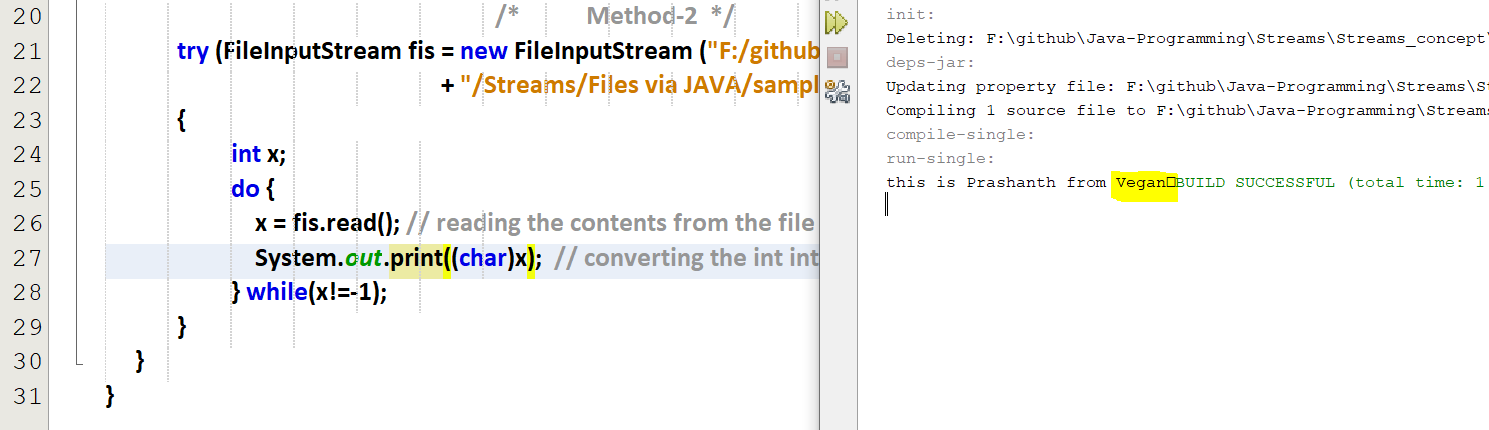
## **Method-1**



Output



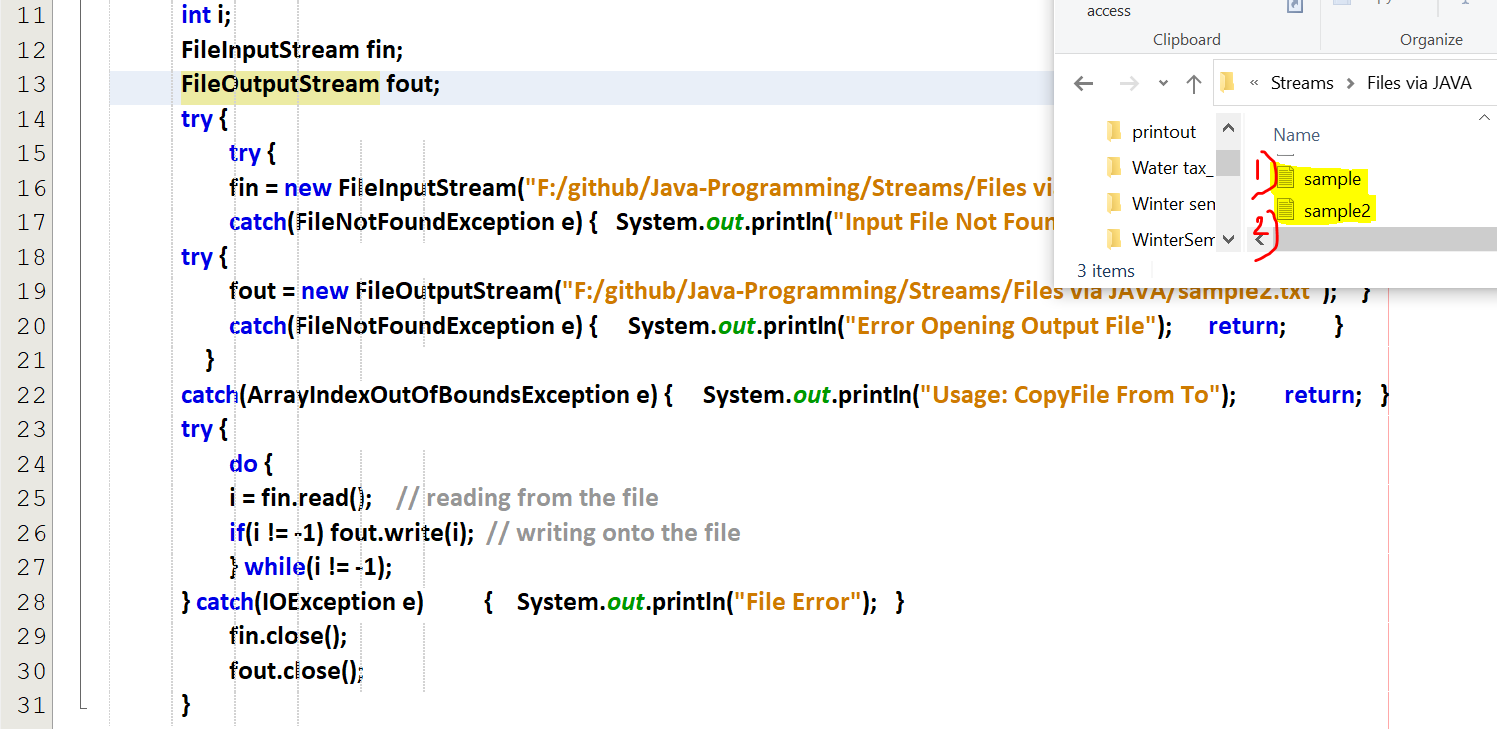
## **Method-2 (reading the last character also)**



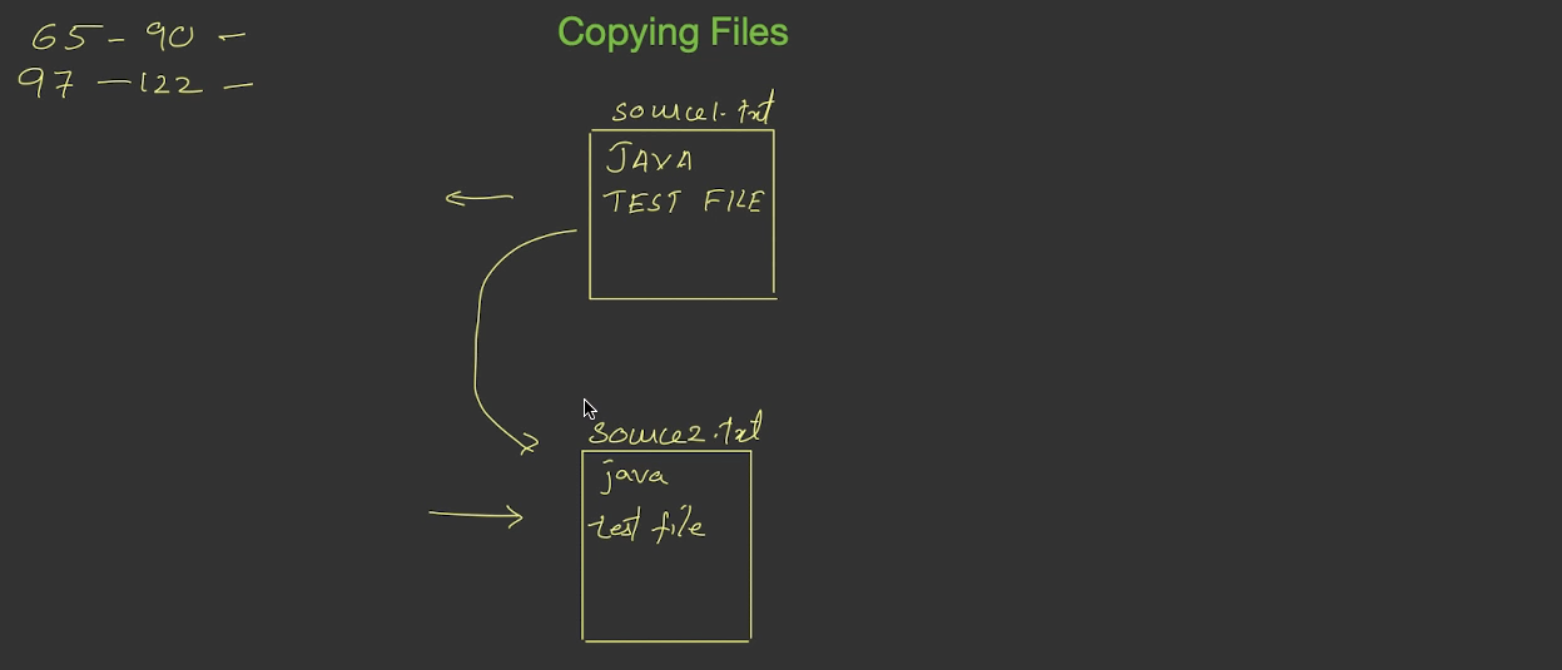
## **Method-3**

# **Copy the contents from one file to another**

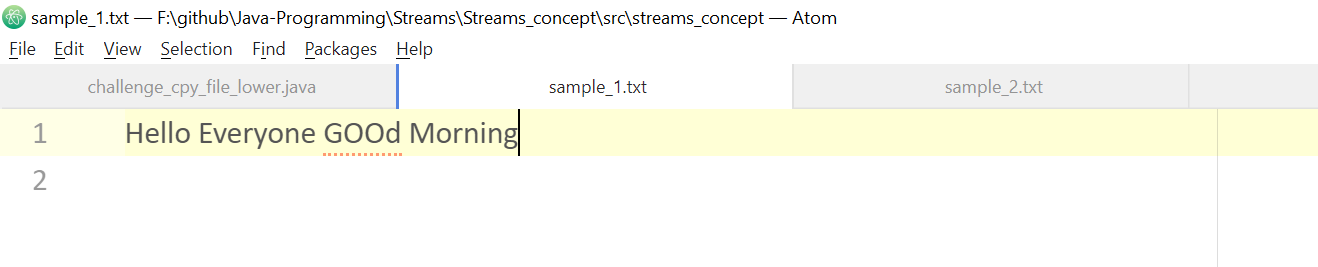
## **Refer challenge\_cpy\_file.java (simple copy)**

****

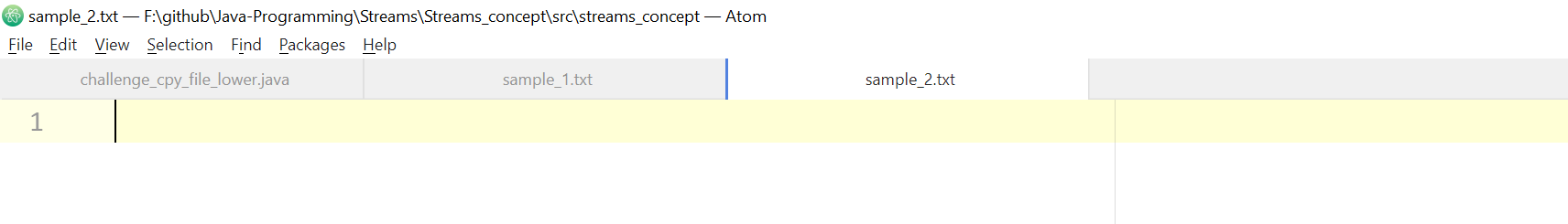
## **Refer challenge\_cpy\_file\_lower.java (from normal case to lower case copy)**

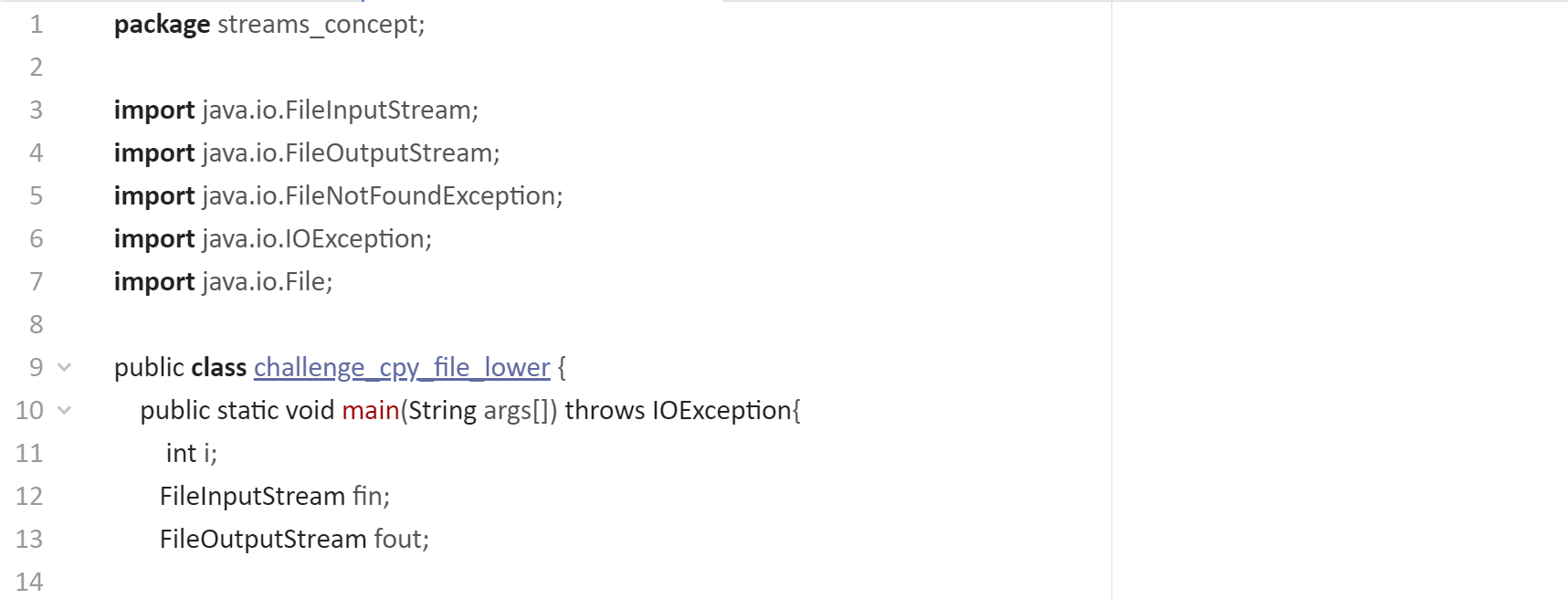
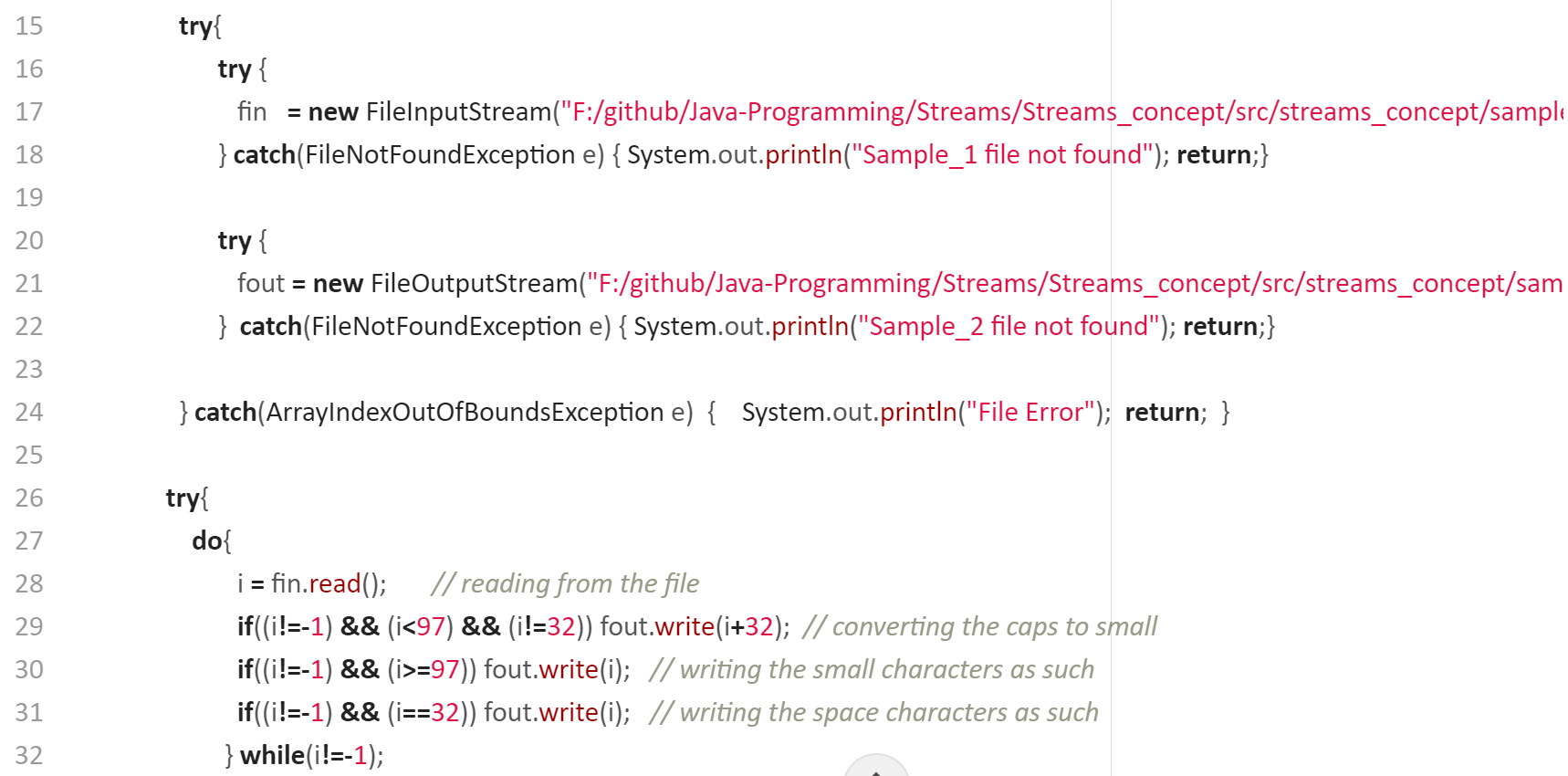


Sample\_1

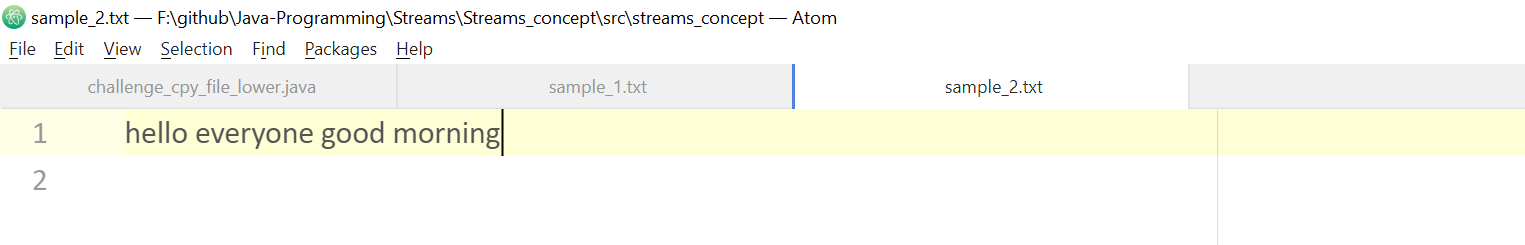


Sample\_2

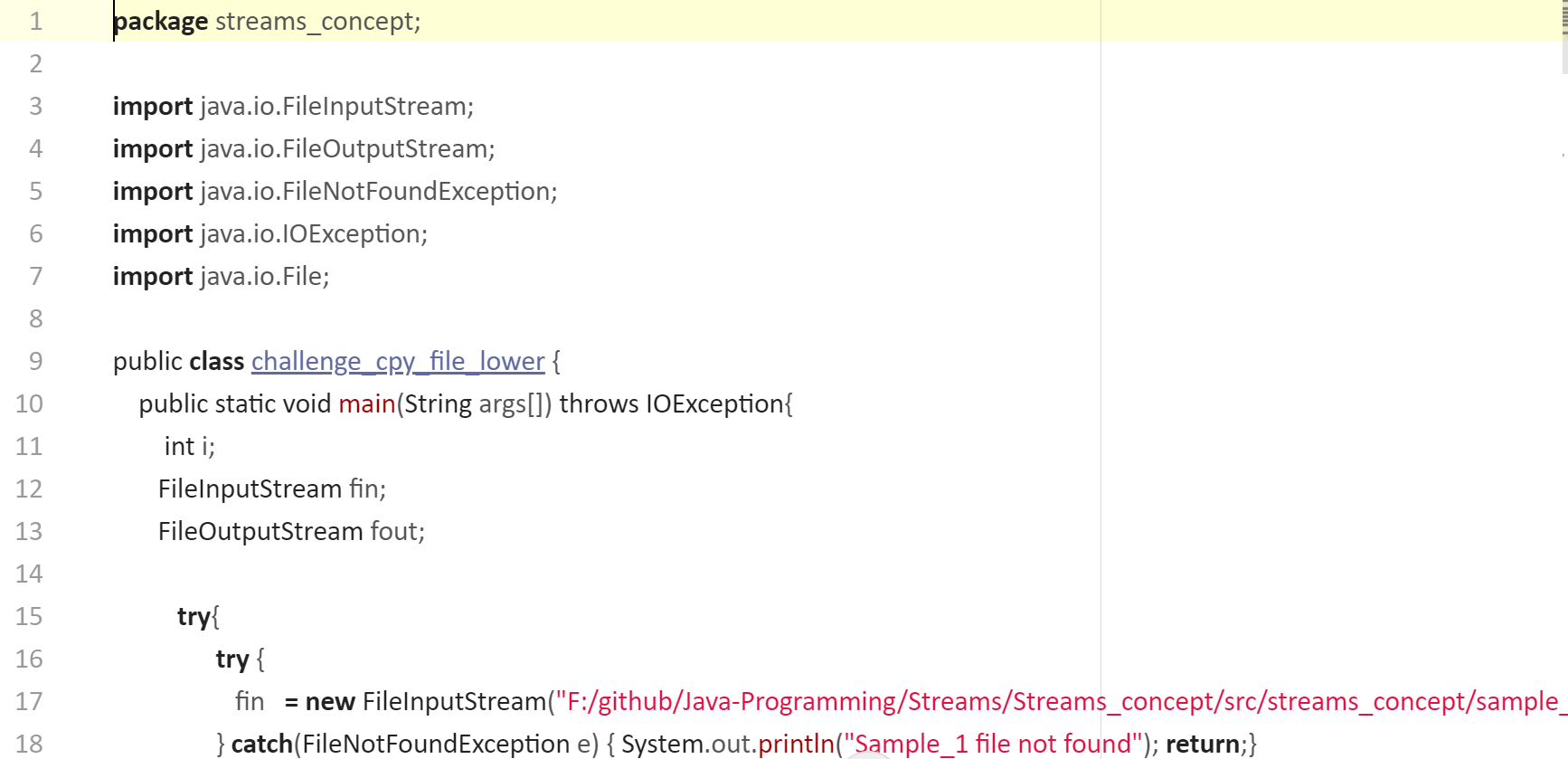
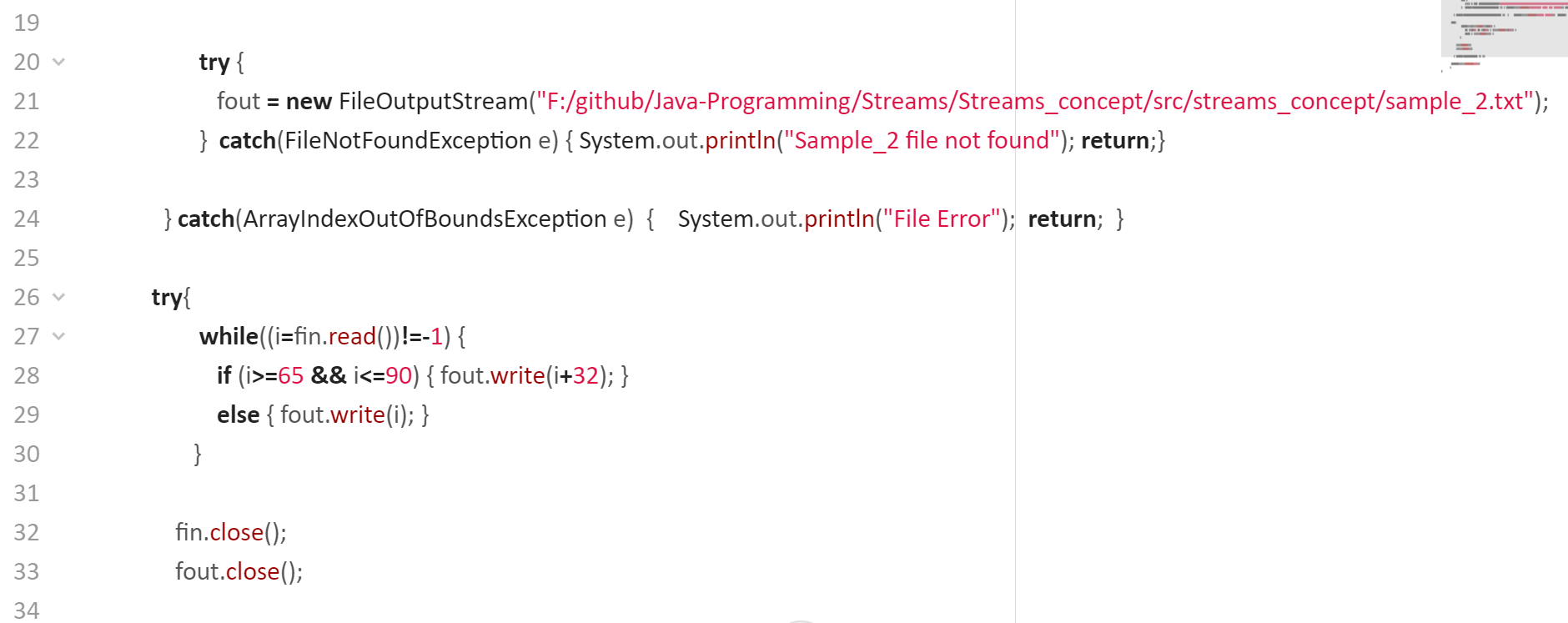


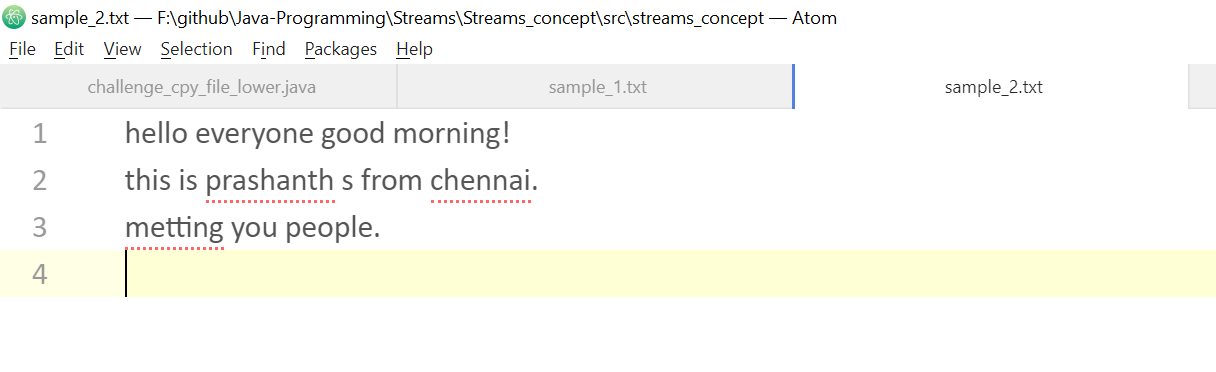

After execution sample-2



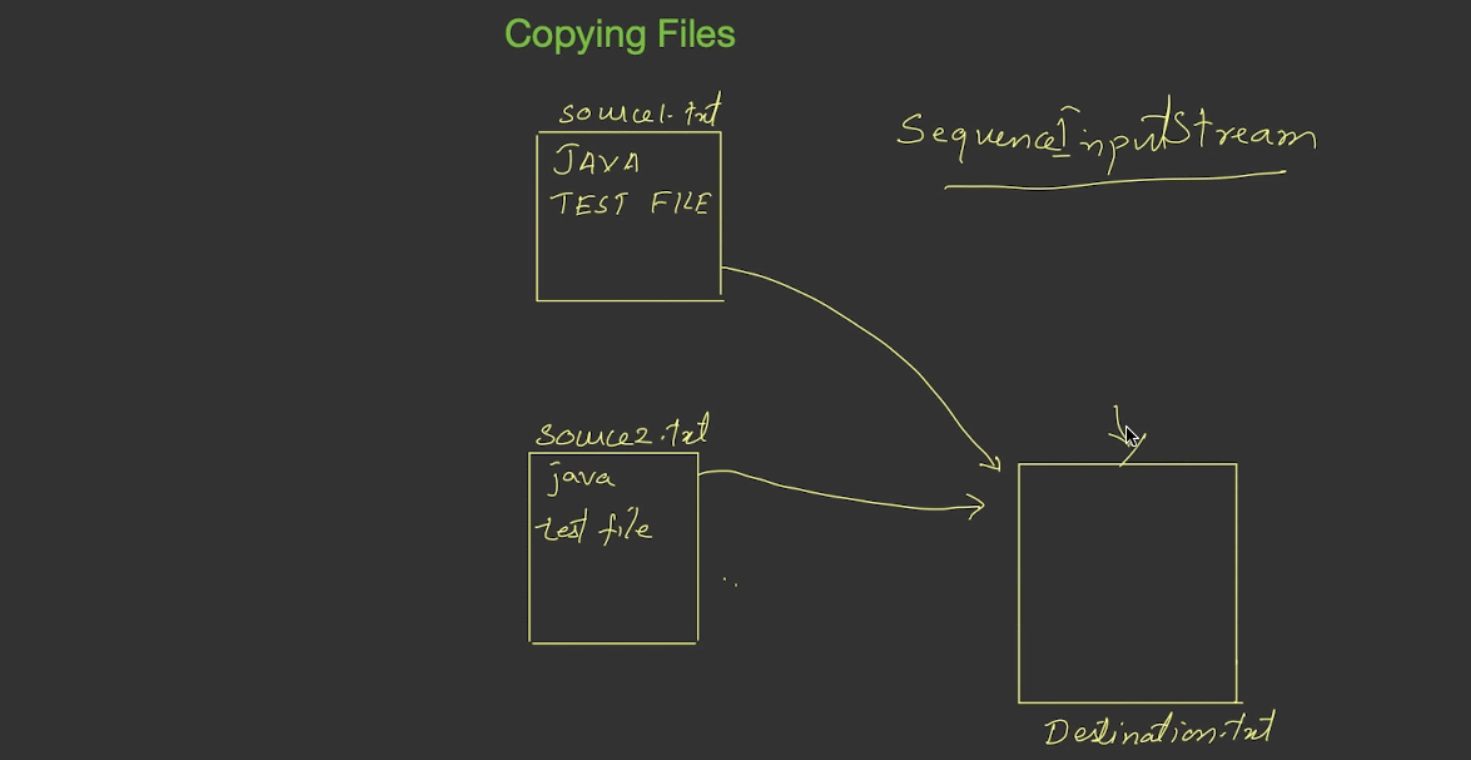
After considering the spaces and indentation. Considering only the lower case characters.

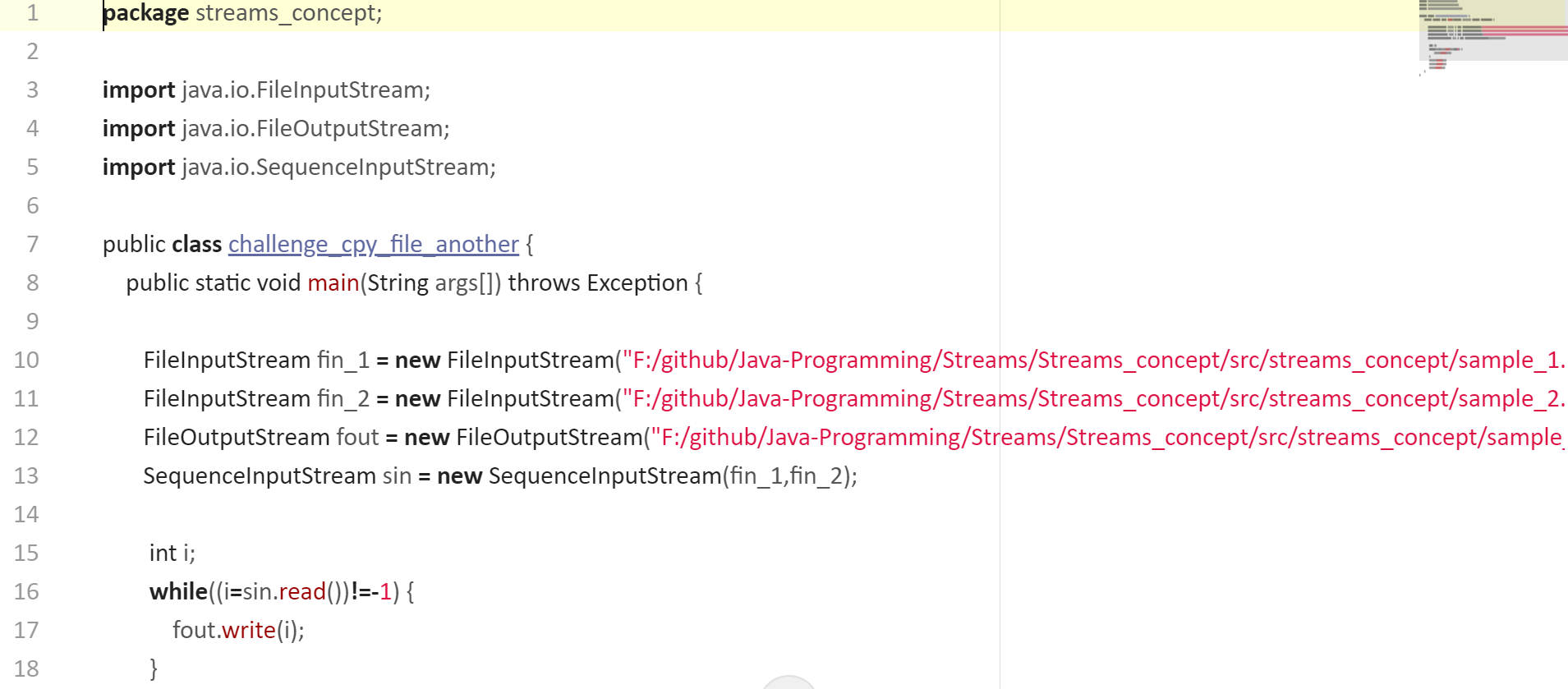
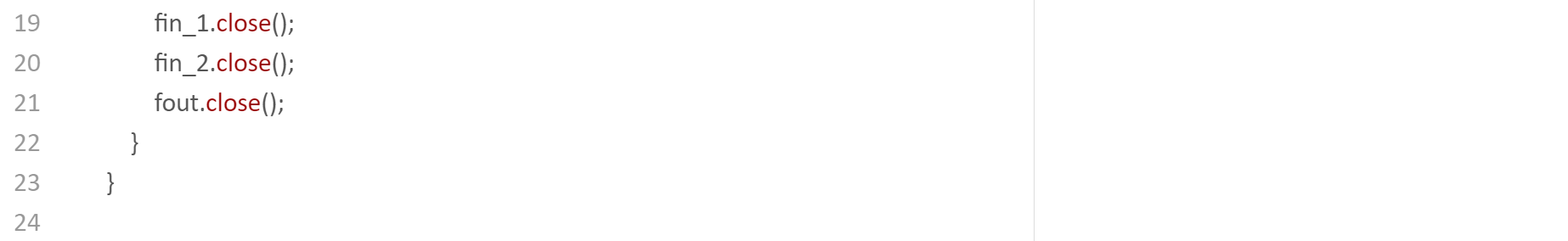
  
  

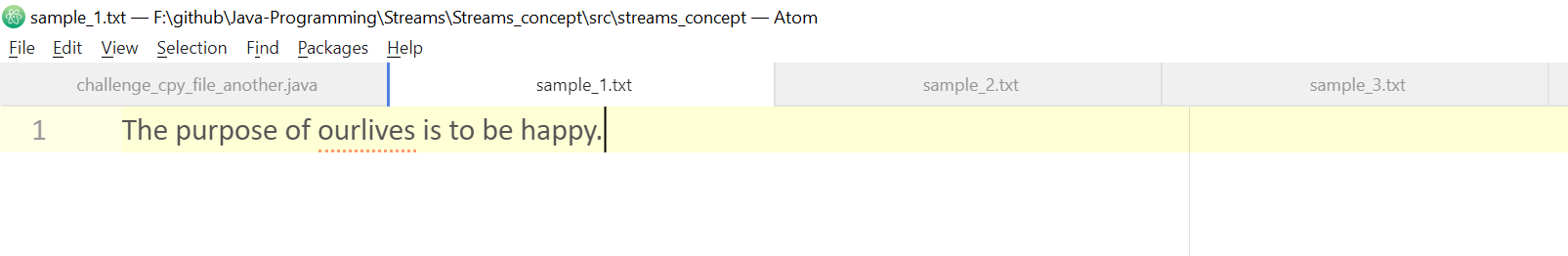
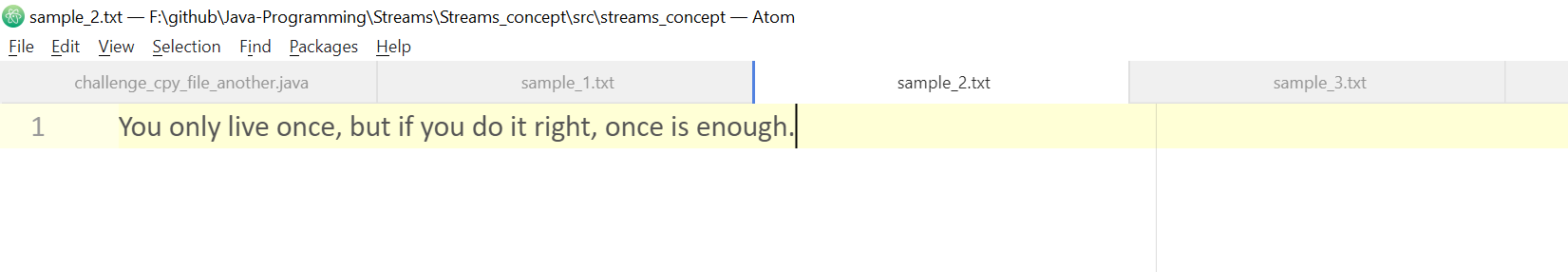
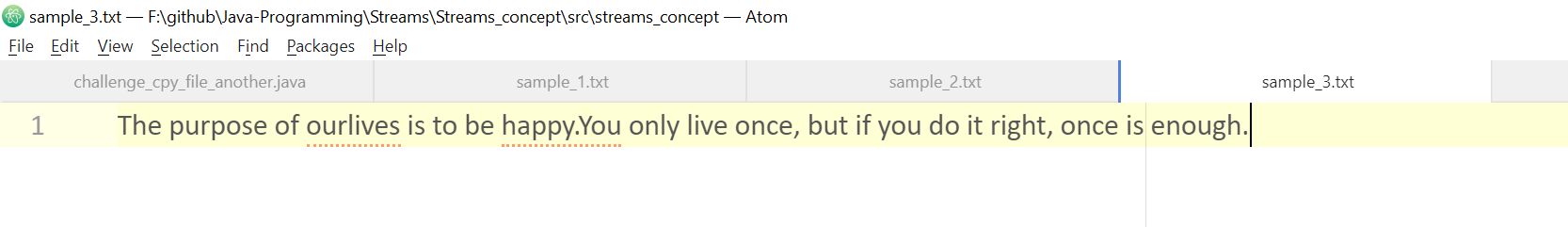

After execution sample-2



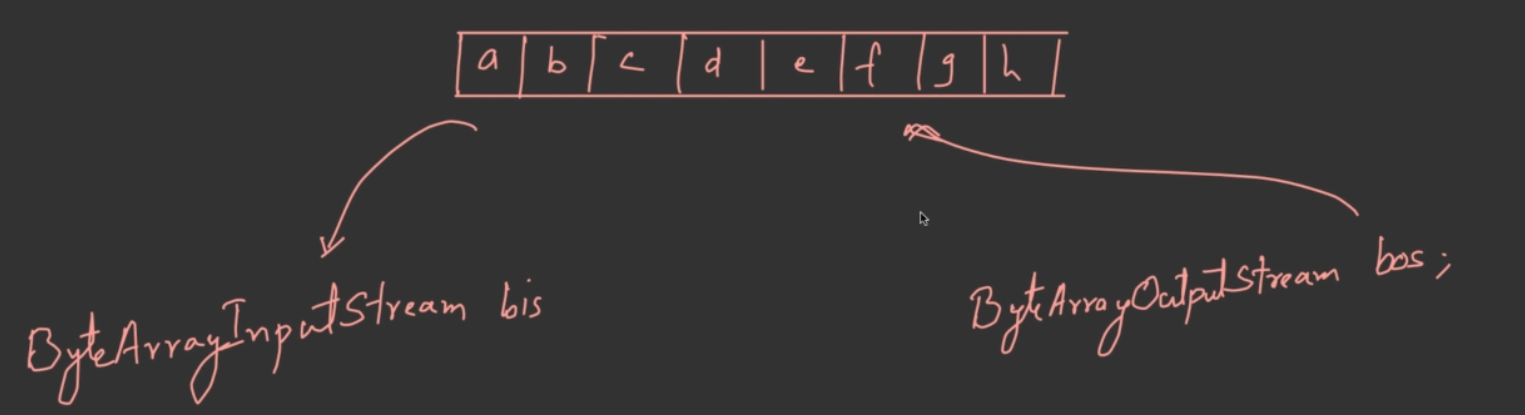
## **Refer challenge\_cpy\_file\_another.java (file1,file2 to file3)**



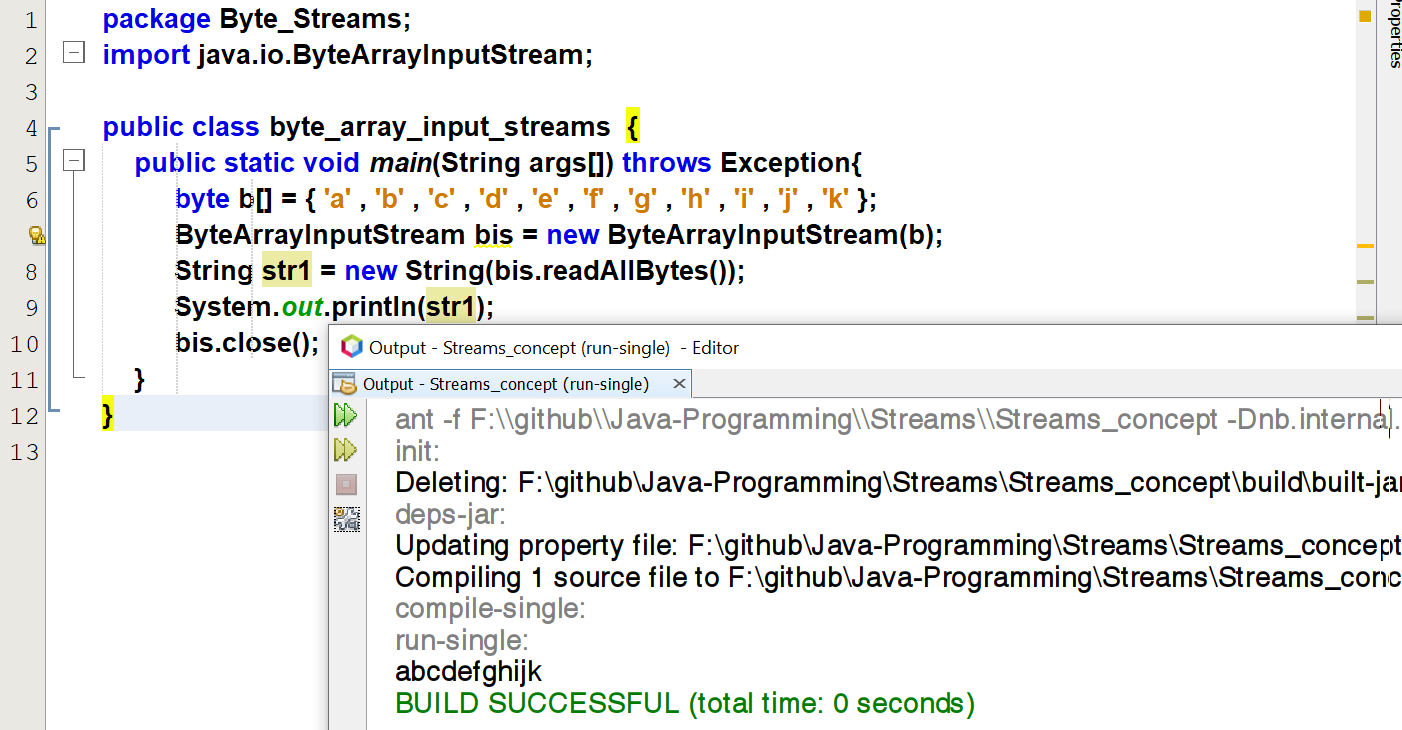
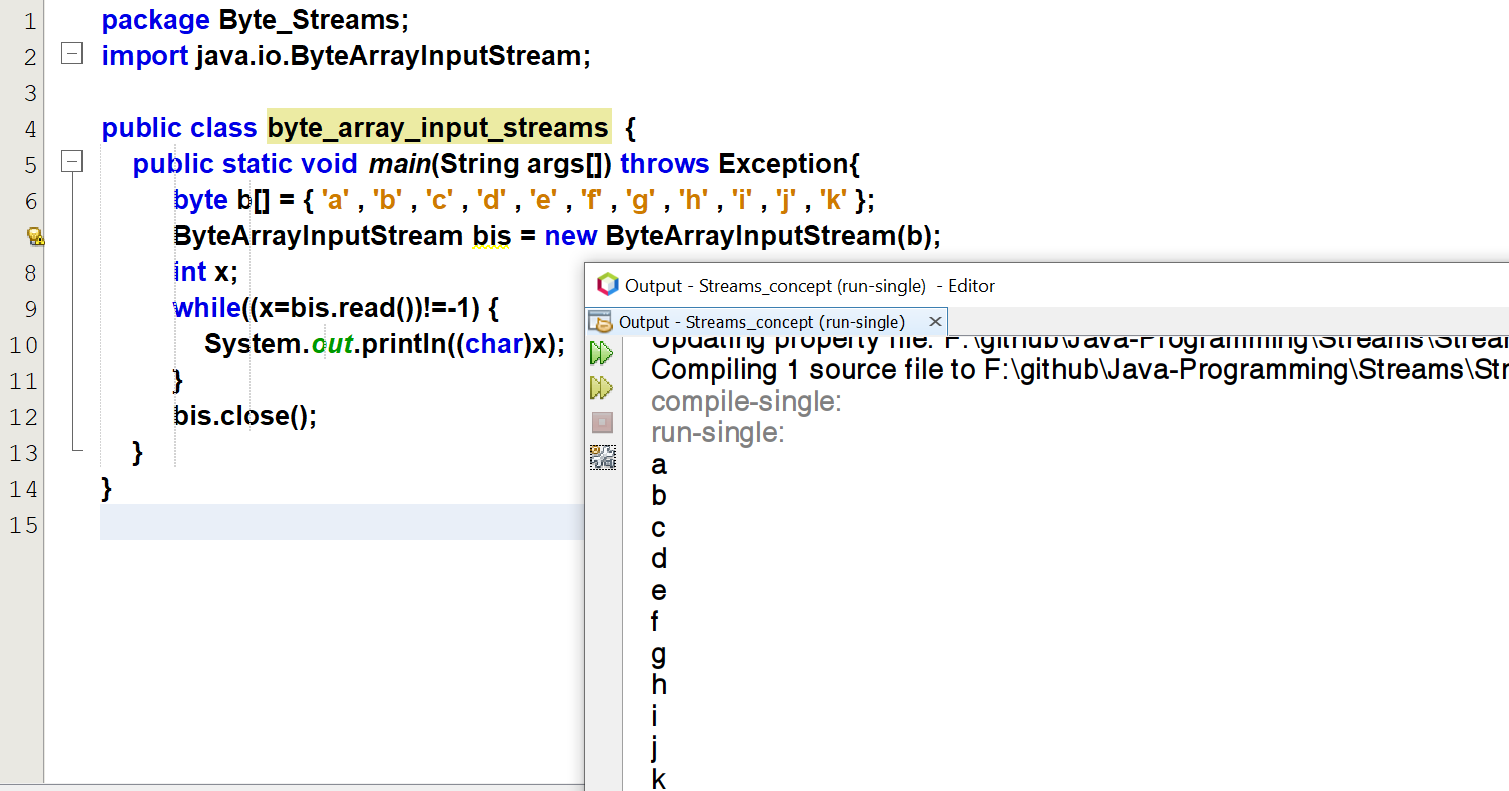
  
  


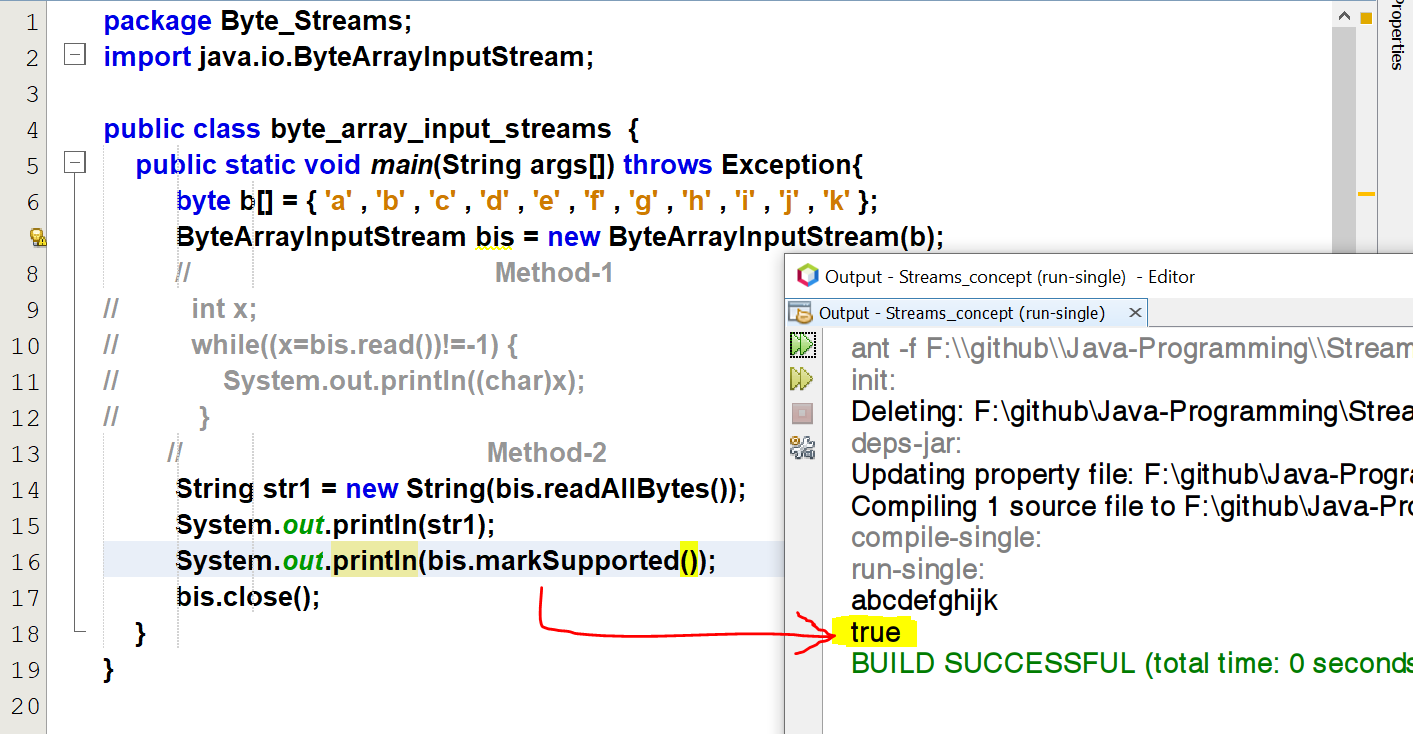
# **Byte array streams**

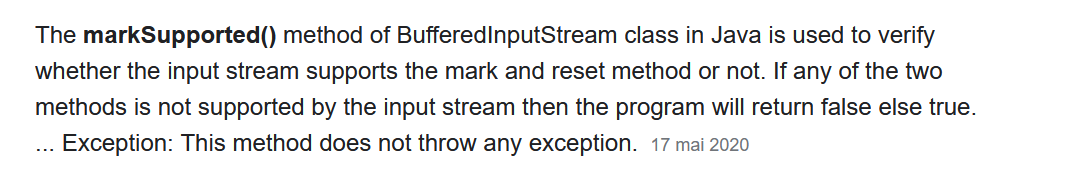


## **Byte array input streams**

import java.io.ByteArrayInputStream



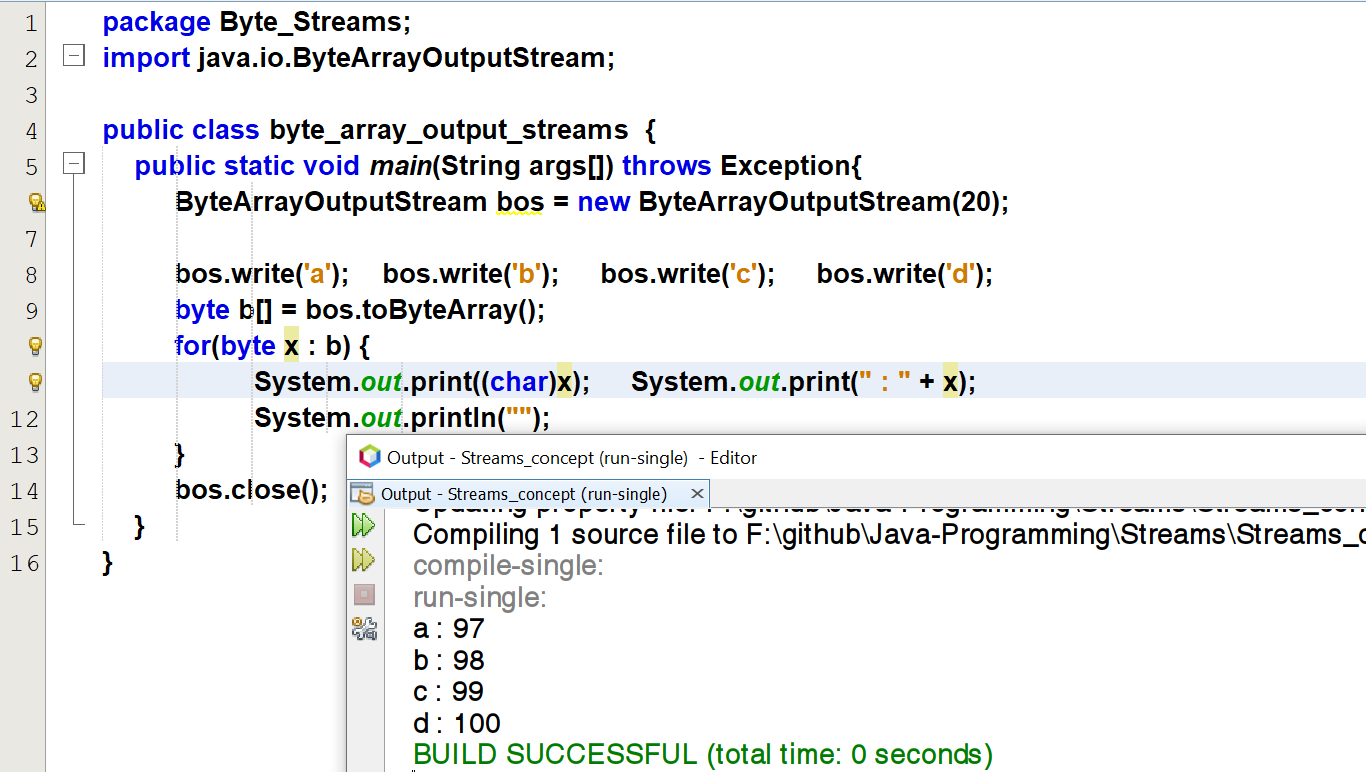
**Also supports markSupported()**

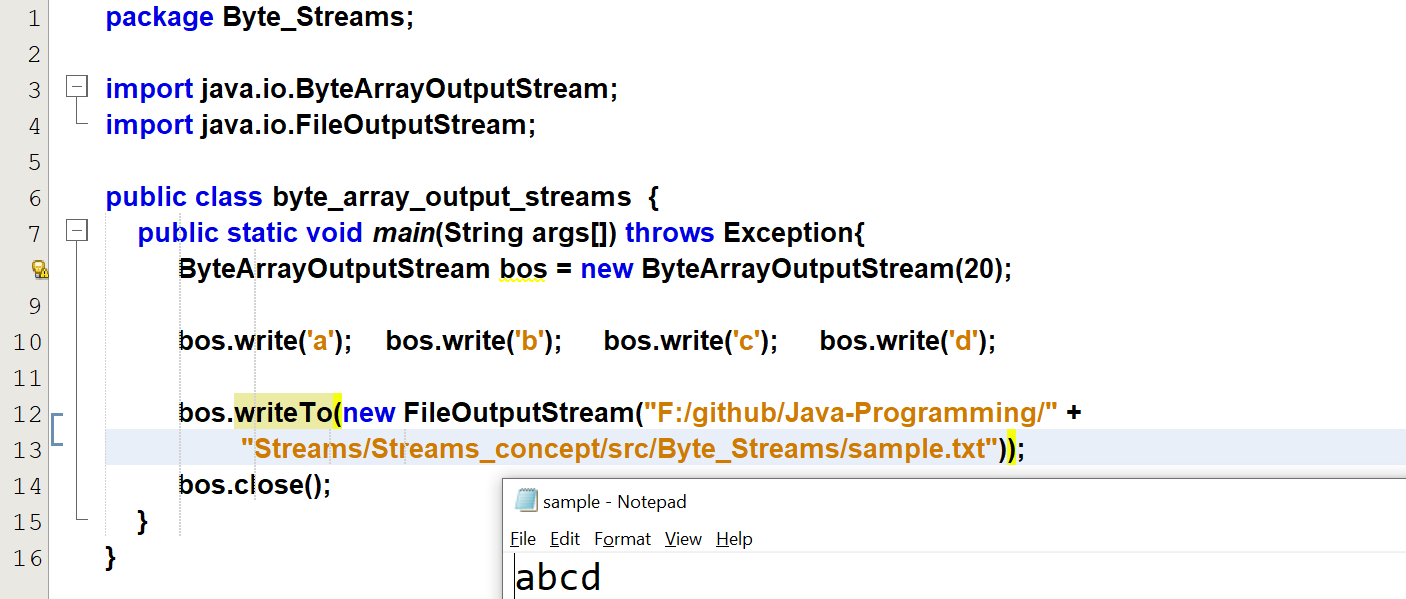


Since it is an array

## **Byte array output streams**

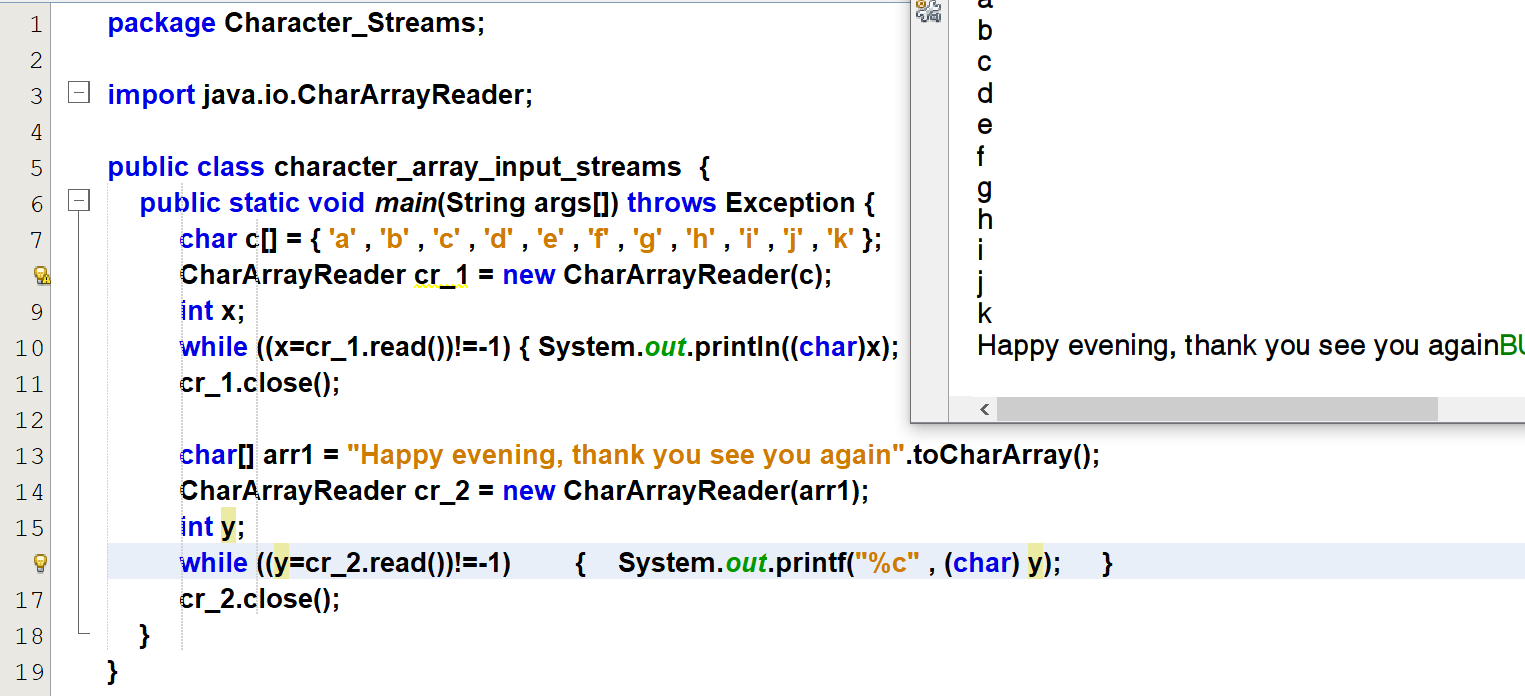
import java.io.ByteArrayOutputStream



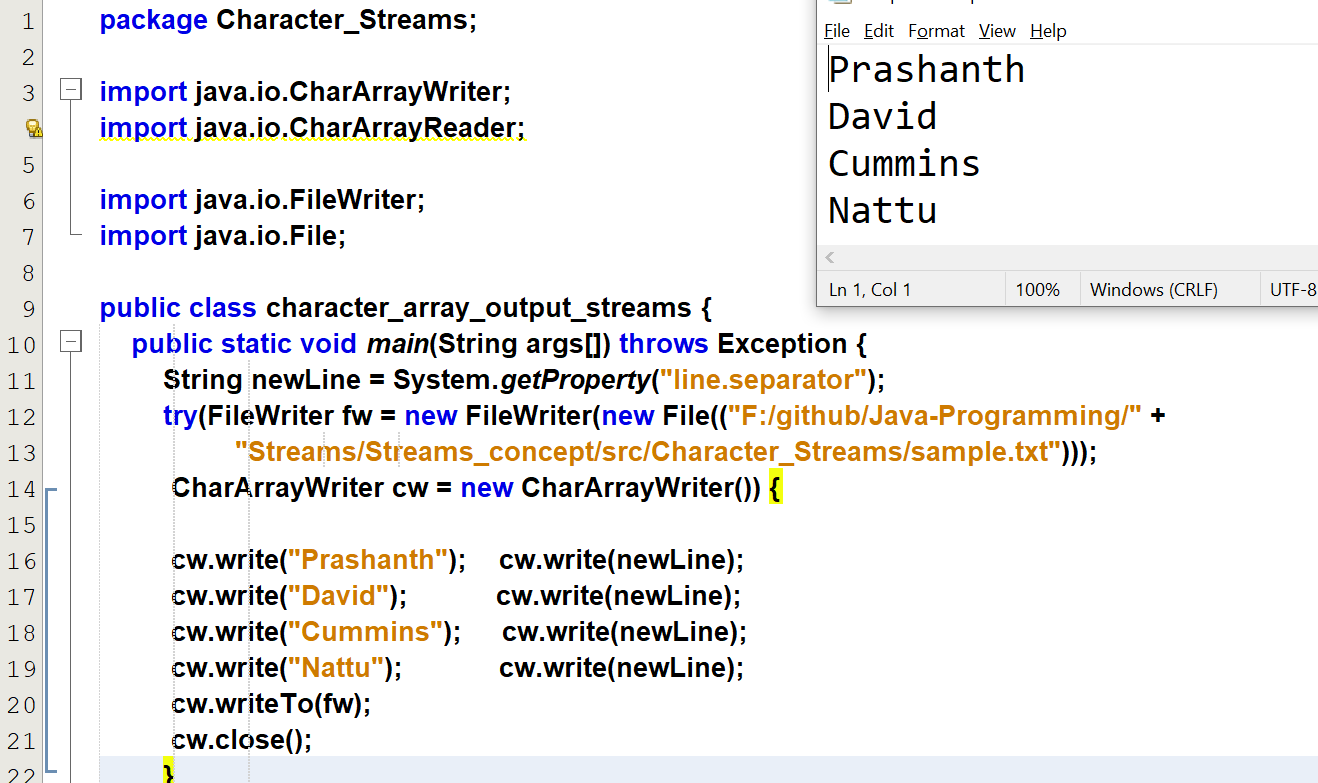


# **Character Array streams**

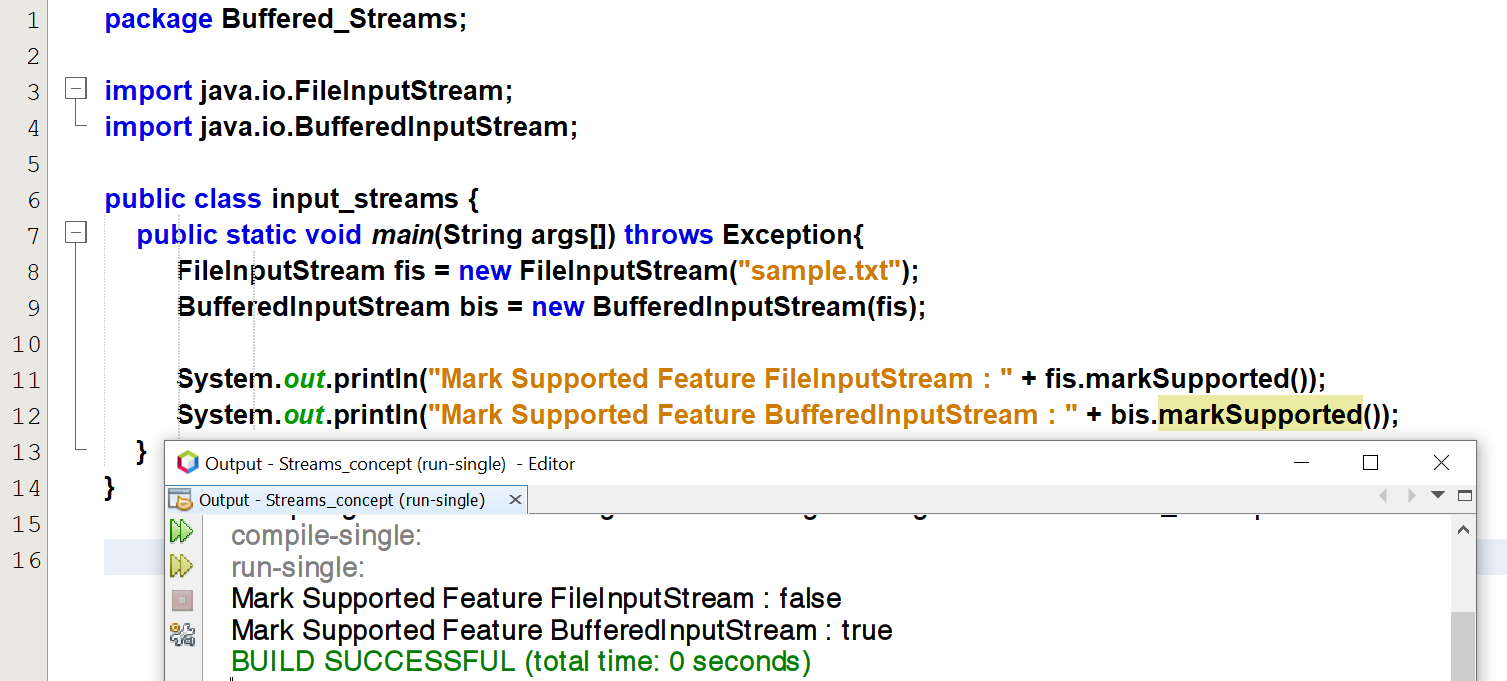
## **Character array input stream**

import java.io.CharArrayReader;

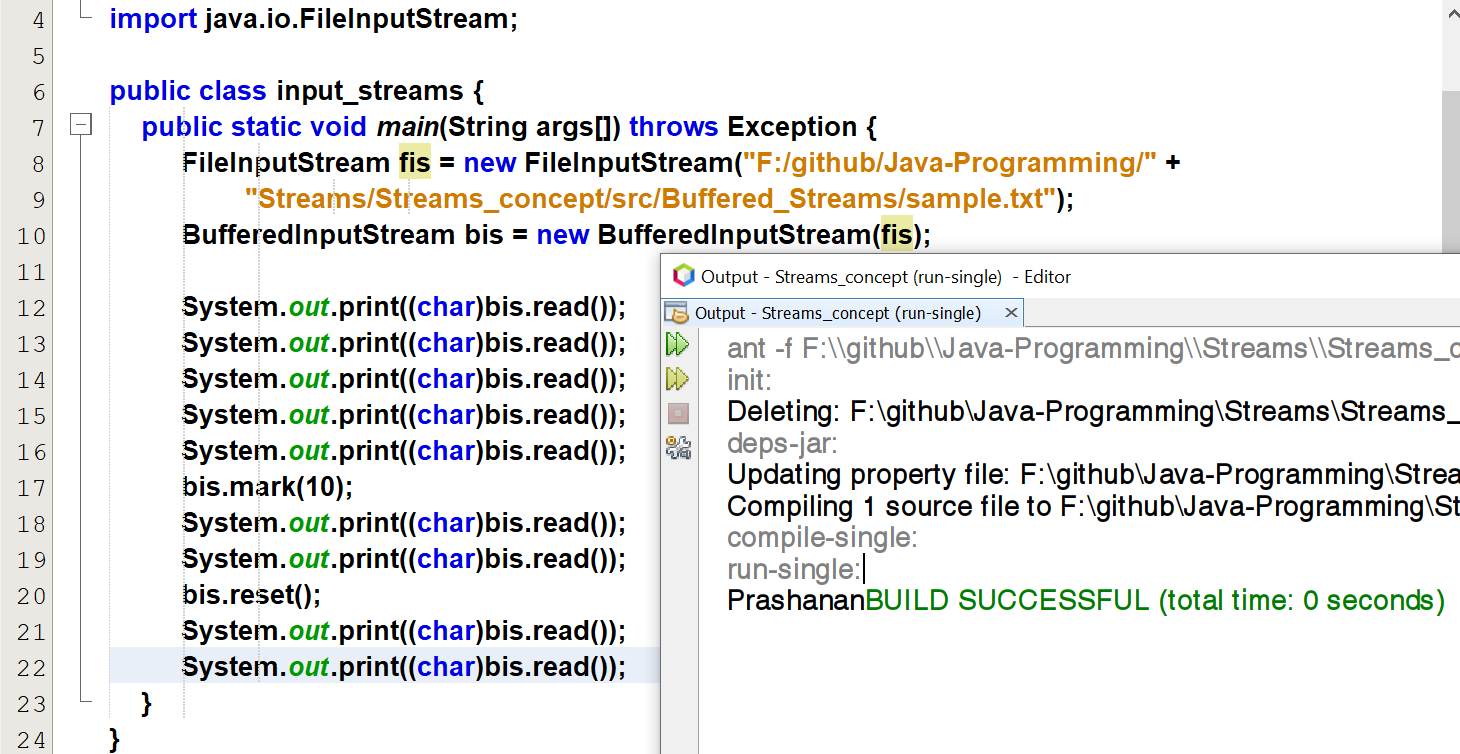
## **Character array output streams**

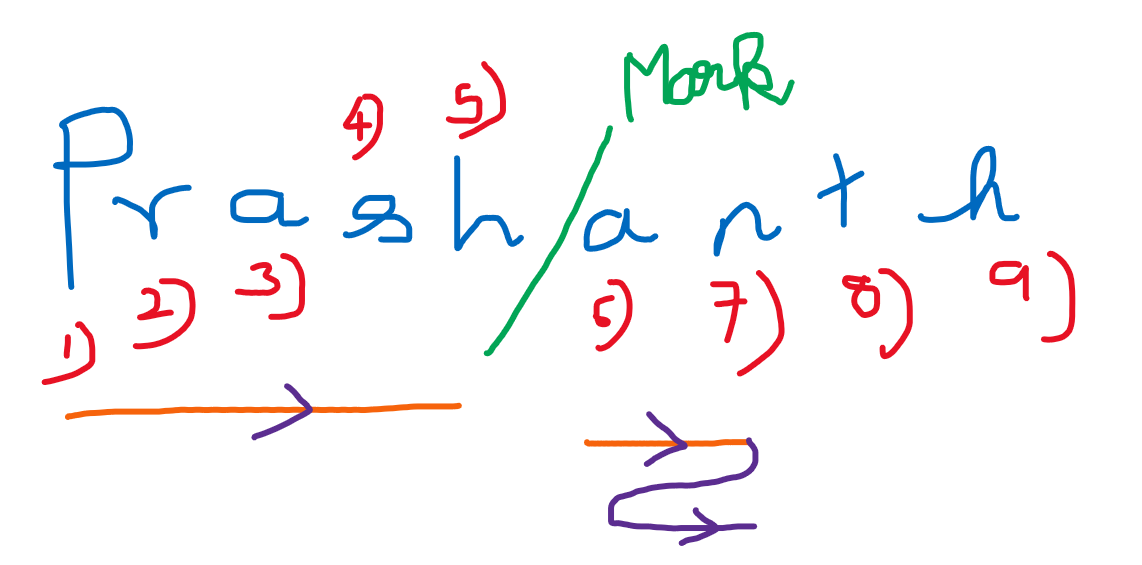
import java.io.CharArrayWriter;

# **Buffered Streams**



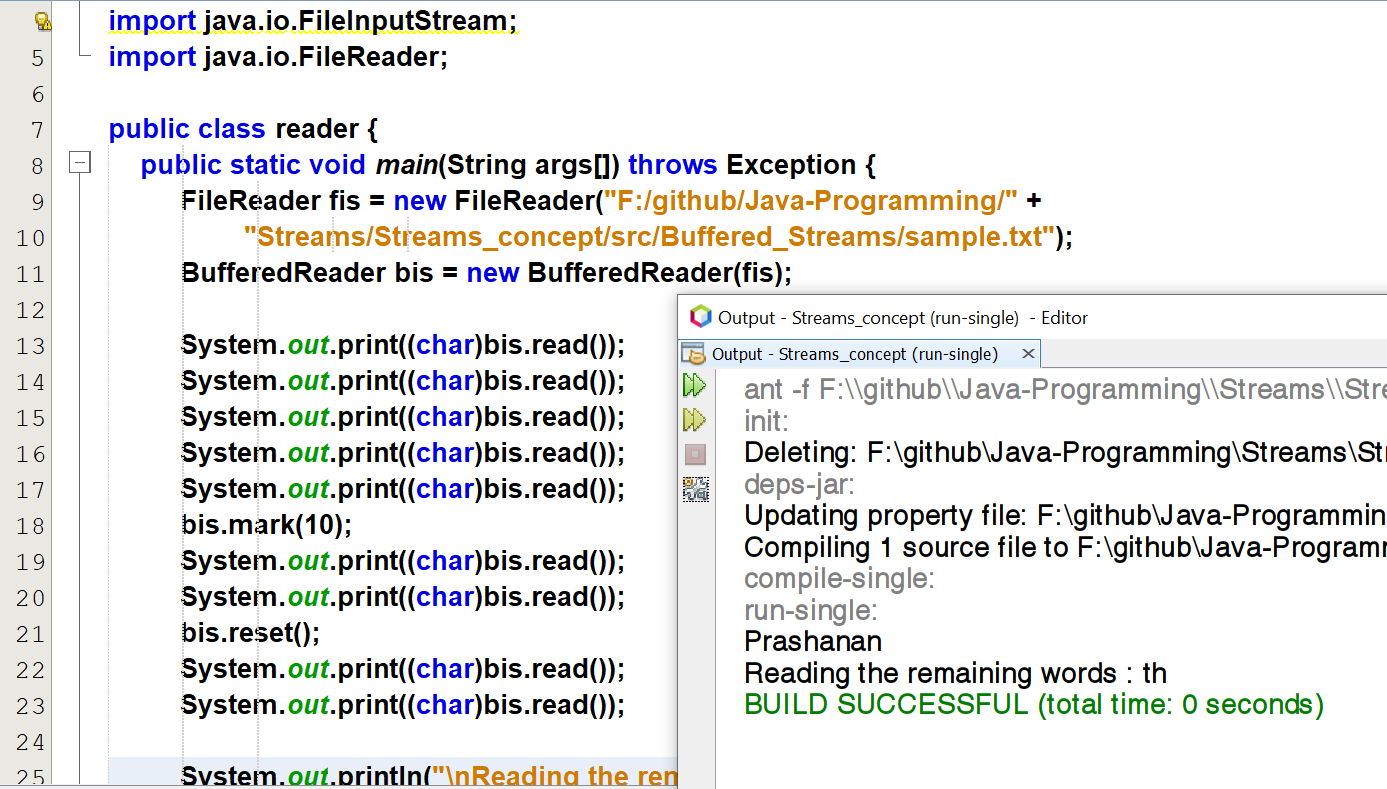
## **Buffered input stream**





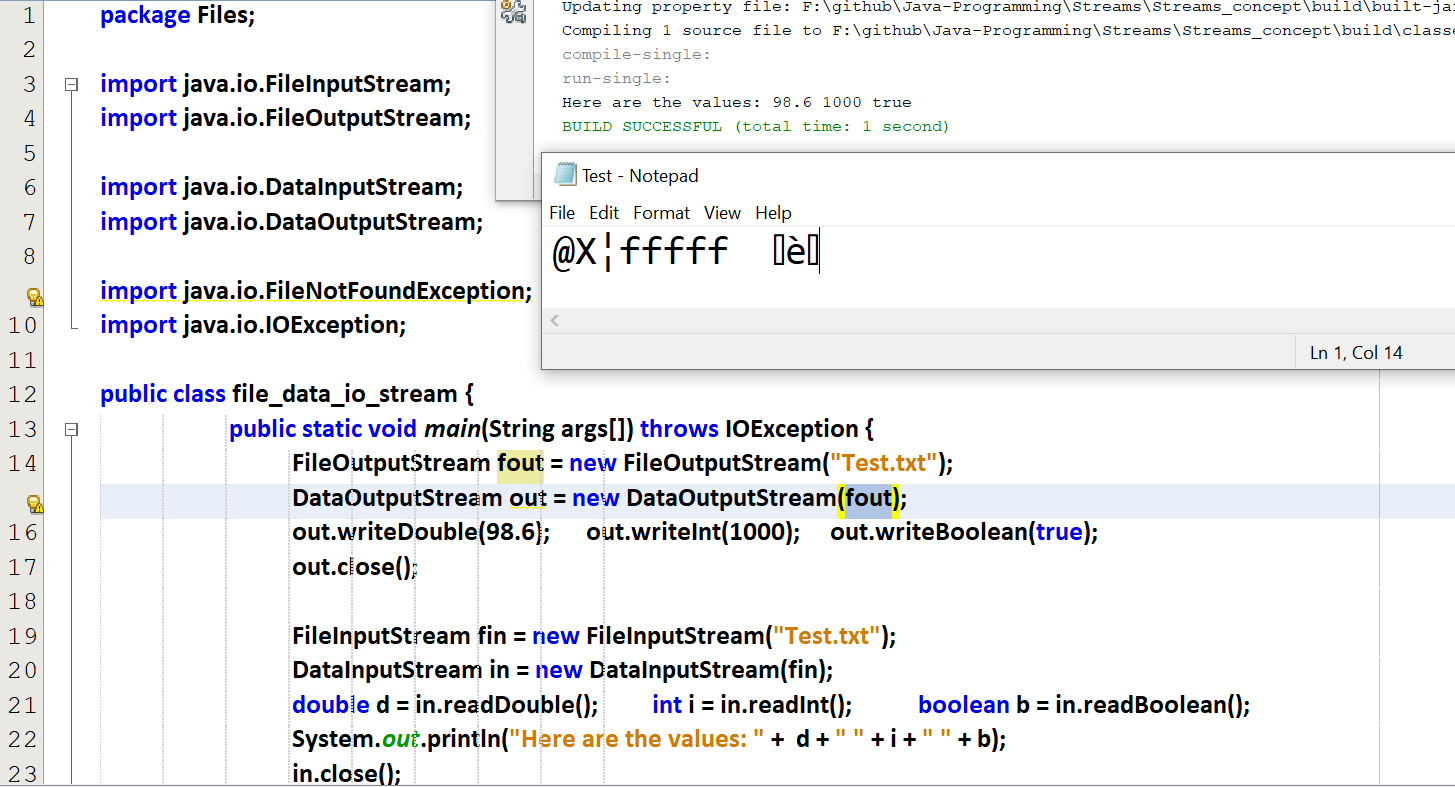
## **Buffered output stream**

## **Buffered reader stream**



## **Buffered writer stream**

# **File\_Data\_IO streams**



# **File Reader and File Writer**