Java Programming Theory Assignment

Prashanth.S
(19MID0020)

Question → Streams and I/O
My assignment complete work-flow

- File Output Streams
  - Method-1
  - Method-2
  - Method-3 (write offset)
  - Method-4 (with throws Exception)
- File Input Streams
  - Method-1
  - Method-2 (reading the last character...
  - Method-3
- Byte Array Output Stream
  - Method-1
  - Method-2
- Byte Array Input Stream
  - Method-1
  - Method-2
  - Also supports markSupported()
- Character Array input Streams
- Character Array output Streams
- Buffered Streams
  - Buffered input stream
  - Buffered reader
- Piped Streams
  - PipedInputStreams
  - PipedOutputStreams
  - Producer Consumer problem

File Output Streams

Java FileOutputStream is an output stream used for writing data to a file.

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

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

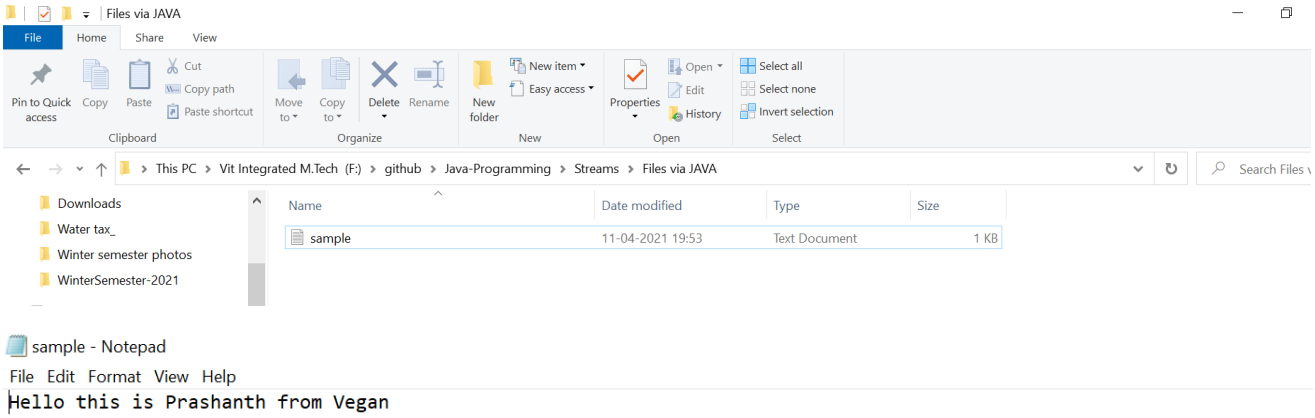
Method-1

```
4   public class file_example {
5     public static void main(String args[]) {
6       try {
            FileOutputStream fos = new FileOutputStream("F:/github/Java-Programming"
8               + "/Streams/Files via JAVA/sample.txt");
9
10                                  /*              Method-1      */
11          String str1 = "Hello this is Prashanth from Vegan";
12
13  /*          getBytes() --> return bytes
14              str1.getBytes() --> str1 is converted into array of bytes
15              fos.write(str1.getBytes()) --> the above converted is written into the file          */
16
17          fos.write(str1.getBytes());
18          fos.close();  // closing the file
```

Method-2

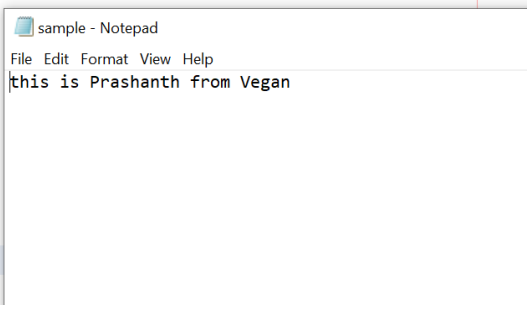
```
/*                  Method-2   */
String str1 = "Hello this is Prashanth from Vegan";
byte b[] = str1.getBytes();

for(byte x:b)
   fos.write(x);

fos.close();
```

Output



Method-3 (write offset)

```
30                  String str1 = "Hello this is Prashanth from Vegan";
31                  byte b[] = str1.getBytes();
32                  fos.write(b,6,str1.length()-6);
33                  fos.close();
34          }
35
36          catch (FileNotFoundException e) { System.out.println(e);}
37          catch (IOException e) { System.out.println(e); }
38      }
39  }
```



Method-4 (with throws Exception)

```
4   public class file_example {
5       public static void main(String args[]) throws Exception {
6
7                               /*      Method-3  (write-offset      */
8           try (FileOutputStream fos = new FileOutputStream("F:/github/Java-Programming"
9                                   + "/Streams/Files via JAVA/sample.txt")) {
10          String str1 = "Hello this is Prashanth from Vegan";
11          byte b[] = str1.getBytes();
12          fos.write(b,6,str1.length()-6);
13      }
```

Who will handle this exception → JVM

throws in Main will be handled by main()

File Input Streams

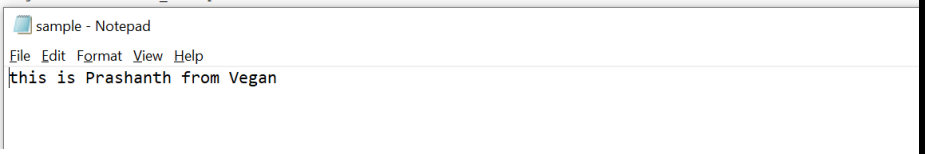
Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

Method-1

```
4   public class file_example_input_streams {
5     public static void main(String args[]) throws Exception {
6                                         /*      Method-1     */
7           try (FileInputStream fis = new FileInputStream("F:/github/Java-Programming"
8                                       + "/Streams/Files via JAVA/sample.txt")      )
9           {
10            byte b[] = new byte[fis.available()];   // creating a byte array from the size available in the file
11            // size of the byte array should be equal to the contents of the file
12            // fis.available() --> size of the file
13
14            fis.read(b);   // reading the byte array
15            String str1 = new String(b);  // converting the byte array into string
16
17            System.out.println("Strings from the file");
18            System.out.println(str1);
19          }
20      }
21  }
```

Output

```
ant -f F:\\github\\Java-Programming\\Streams\\Streams_concept -Dnb.internal.action.name=run.single -Djavac.includes=streams_concept/file_example_input_streams.java
init:
Deleting: F:\github\Java-Programming\Streams\Streams_concept\build\built-jar.properties
deps-jar:
Updating property file: F:\github\Java-Programming\Streams\Streams_concept\build\built-jar.properties
Compiling 1 source file to F:\github\Java-Programming\Streams\Streams_concept\build\classes
compile-single:
run-single:
Strings from the file
this is Prashanth from Vegan
BUILD SUCCESSFUL (total time: 1 second)
```



Method-2 (reading the last character also)

```
20                                        /*      Method-2 */
21          try (FileInputStream fis = new FileInputStream ("F:/github
22                                      + "/Streams/Files via JAVA/sampl
23          {
24              int x;
25              do {
26                x = fis.read(); // reading the contents from the file
27                System.out.print((char)x);  // converting the int int
28              } while(x!=-1);
29          }
30      }
31  }
```

```
init:
Deleting: F:\github\Java-Programming\Streams\Streams_concept
deps-jar:
Updating property file: F:\github\Java-Programming\Streams\S
Compiling 1 source file to F:\github\Java-Programming\Stream
compile-single:
run-single:
this is Prashanth from VeganBUILD SUCCESSFUL (total time: 1
```

Method-3

```
31                                  /*      Method-3    */
32          try (FileInputStream fis = new FileInputStream ("F:/github/Java-Programming"
33                                  + "/Streams/Files via JAVA/sample.txt")     )
34          {
35              int x;
36              while((x = fis.read())!=-1) {   System.out.print((char)x);      }
37          }
38      }
39  }
```

```
Debugger Console ×  Streams_concept (run-single) ×
ant -f F:\\github\\Java-Programming\\Streams\\Stream
init:
Deleting: F:\github\Java-Programming\Streams\Streams
deps-jar:
Updating property file: F:\github\Java-Programming\S
Compiling 1 source file to F:\github\Java-Programmin
compile-single:
run-single:
this is Prashanth from VeganBUILD SUCCESSFUL (total
```

Byte Array Output Stream

Java ByteArrayOutputStream class is used to **write common data** into multiple files. In this stream, the data is written into a byte array which can be written to multiple streams later.

The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams.

The buffer of ByteArrayOutputStream automatically grows according to data.

Method-1

```
1   package Byte_Streams;
2   import java.io.ByteArrayInputStream;
3
4   public class byte_array_input_streams  {
5       public static void main(String args[]) throws Exception{
6           byte b[] = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' , 'h' , 'i' , 'j' , 'k' };
            ByteArrayInputStream bis = new ByteArrayInputStream(b);
8           int x;
9           while((x=bis.read())!=-1) {
10              System.out.println((char)x);
11          }
12          bis.close();
13      }
14  }
15
```

```
Output - Streams_concept (run-single) - Editor
Output - Streams_concept (run-single) ×
Updating property file: F:\github\Java-Programming\Streams\Strea
Compiling 1 source file to F:\github\Java-Programming\Streams\Str
compile-single:
run-single:
a
b
c
d
e
f
g
h
i
j
k
```

Method-2

```java
package Byte_Streams;
import java.io.ByteArrayInputStream;

public class byte_array_input_streams {
    public static void main(String args[]) throws Exception{
        byte b[] = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' , 'h' , 'i' , 'j' , 'k' };
        ByteArrayInputStream bis = new ByteArrayInputStream(b);
        String str1 = new String(bis.readAllBytes());
        System.out.println(str1);
        bis.close();
    }
}
```

```
Output - Streams_concept (run-single) - Editor
Output - Streams_concept (run-single)
ant -f F:\\github\\Java-Programming\\Streams\\Streams_concept -Dnb.interna
init:
Deleting: F:\github\Java-Programming\Streams\Streams_concept\build\built-ja
deps-jar:
Updating property file: F:\github\Java-Programming\Streams\Streams_concept
Compiling 1 source file to F:\github\Java-Programming\Streams\Streams_conc
compile-single:
run-single:
abcdefghijk
BUILD SUCCESSFUL (total time: 0 seconds)
```

Byte Array Input Stream

The ByteArrayInputStream is composed of two words: ByteArray and InputStream. As the name suggests, it can be used to read byte array as input stream.

Java ByteArrayInputStream class contains an internal buffer which is used to **read byte array** as stream. In this stream, the data is read from a byte array.

The buffer of ByteArrayInputStream automatically grows according to data.

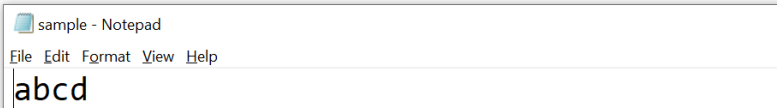
Method-1

```java
package Byte_Streams;
import java.io.ByteArrayOutputStream;

public class byte_array_output_streams {
    public static void main(String args[]) throws Exception{
        ByteArrayOutputStream bos = new ByteArrayOutputStream(20);

        bos.write('a');   bos.write('b');   bos.write('c');   bos.write('d');
        byte b[] = bos.toByteArray();
        for(byte x : b) {
                System.out.print((char)x);   System.out.print(" : " + x);
                System.out.println("");
        }
        bos.close();
    }
}
```

```
Output - Streams_concept (run-single) - Editor
Output - Streams_concept (run-single)
Compiling 1 source file to F:\github\Java-Programming\Streams\Streams_c
compile-single:
run-single:
a : 97
b : 98
c : 99
d : 100
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method-2

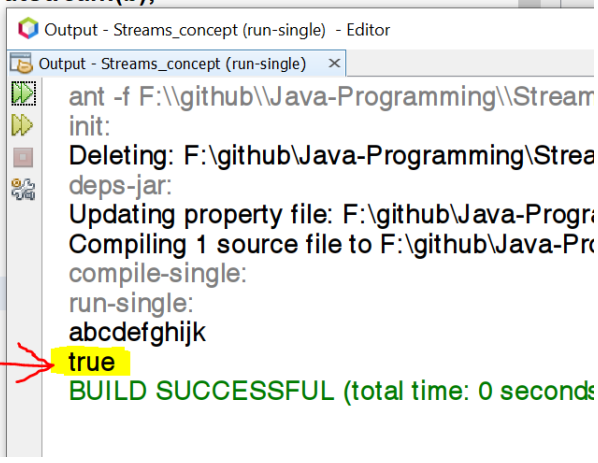
```
1  package Byte_Streams;
2
3  import java.io.ByteArrayOutputStream;
4  import java.io.FileOutputStream;
5
6  public class byte_array_output_streams  {
7      public static void main(String args[]) throws Exception{
8          ByteArrayOutputStream bos = new ByteArrayOutputStream(20);
9
10         bos.write('a');    bos.write('b');    bos.write('c');    bos.write('d');
11
12         bos.writeTo(new FileOutputStream("F:/github/Java-Programming/" +
13             "Streams/Streams_concept/src/Byte_Streams/sample.txt"));
14         bos.close();
15     }
16 }
```



Also supports markSupported()

The **markSupported()** method of BufferedInputStream class in Java is used to verify whether the input stream supports the mark and reset method or not. If any of the two methods is not supported by the input stream then the program will return false else true. ... Exception: This method does not throw any exception. 17 mai 2020

```
1  package Byte_Streams;
2  import java.io.ByteArrayInputStream;
3
4  public class byte_array_input_streams  {
5      public static void main(String args[]) throws Exception{
6          byte b[] = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' , 'h' , 'i' , 'j' , 'k' };
7          ByteArrayInputStream bis = new ByteArrayInputStream(b);
8          //                          Method-1
9  //          int x;
10 //          while((x=bis.read())!=-1) {
11 //              System.out.println((char)x);
12 //          }
13         //                          Method-2
14         String str1 = new String(bis.readAllBytes());
15         System.out.println(str1);
16         System.out.println(bis.markSupported());
17         bis.close();
18     }
19 }
20
```



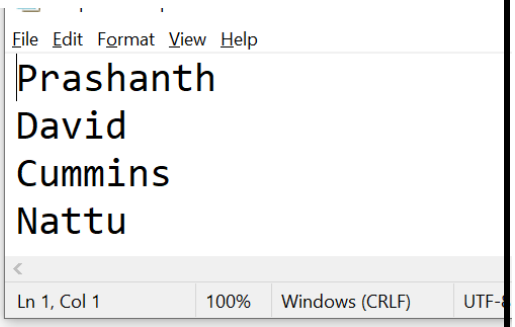
Character Array input Streams

```
1  package Character_Streams;
2
3  import java.io.CharArrayReader;
4
5  public class character_array_input_streams  {
6      public static void main(String args[]) throws Exception {
7          char c[] = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' , 'h' , 'i' , 'j' , 'k' };
8          CharArrayReader cr_1 = new CharArrayReader(c);
9          int x;
10         while ((x=cr_1.read())!=-1) { System.out.println((char)x);
11         cr_1.close();
12
13         char[] arr1 = "Happy evening, thank you see you again".toCharArray();
14         CharArrayReader cr_2 = new CharArrayReader(arr1);
15         int y;
16         while ((y=cr_2.read())!=-1)       {    System.out.printf("%c" , (char) y);    }
17         cr_2.close();
18     }
19 }
```

```
a
b
c
d
e
f
g
h
i
j
k
Happy evening, thank you see you againB
```

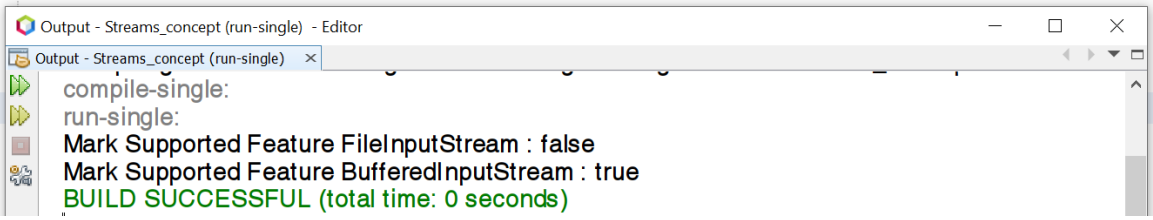
Character Array output Streams

```java
package Character_Streams;

import java.io.CharArrayWriter;
import java.io.CharArrayReader;

import java.io.FileWriter;
import java.io.File;

public class character_array_output_streams {
    public static void main(String args[]) throws Exception {
        String newLine = System.getProperty("line.separator");
        try(FileWriter fw = new FileWriter(new File(("F:/github/Java-Programming/" +
                "Streams/Streams_concept/src/Character_Streams/sample.txt")));
        CharArrayWriter cw = new CharArrayWriter()) {

        cw.write("Prashanth");    cw.write(newLine);
        cw.write("David");        cw.write(newLine);
        cw.write("Cummins");      cw.write(newLine);
        cw.write("Nattu");        cw.write(newLine);
        cw.writeTo(fw);
        cw.close();
        }
```



Buffered Streams

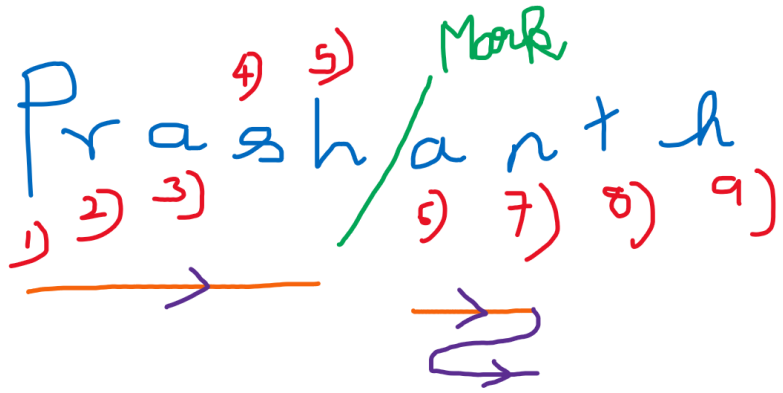
```java
package Buffered_Streams;

import java.io.FileInputStream;
import java.io.BufferedInputStream;

public class input_streams {
    public static void main(String args[]) throws Exception{
        FileInputStream fis = new FileInputStream("sample.txt");
        BufferedInputStream bis = new BufferedInputStream(fis);

        System.out.println("Mark Supported Feature FileInputStream : " + fis.markSupported());
        System.out.println("Mark Supported Feature BufferedInputStream : " + bis.markSupported());
    }
}
```



Buffered input stream

```
4   import java.io.FileInputStream;
5
6   public class input_streams {
7       public static void main(String args[]) throws Exception {
8           FileInputStream fis = new FileInputStream("F:/github/Java-Programming/" +
9                   "Streams/Streams_concept/src/Buffered_Streams/sample.txt");
10          BufferedInputStream bis = new BufferedInputStream(fis);
11
12          System.out.print((char)bis.read());
13          System.out.print((char)bis.read());
14          System.out.print((char)bis.read());
15          System.out.print((char)bis.read());
16          System.out.print((char)bis.read());
17          bis.mark(10);
18          System.out.print((char)bis.read());
19          System.out.print((char)bis.read());
20          bis.reset();
21          System.out.print((char)bis.read());
22          System.out.print((char)bis.read());
23      }
24  }
```

```
Output - Streams_concept (run-single) - Editor
Output - Streams_concept (run-single)
ant -f F:\\github\\Java-Programming\\Streams\\Streams_c
init:
Deleting: F:\github\Java-Programming\Streams\Streams_
deps-jar:
Updating property file: F:\github\Java-Programming\Strea
Compiling 1 source file to F:\github\Java-Programming\St
compile-single:
run-single:
PrashananBUILD SUCCESSFUL (total time: 0 seconds)
```

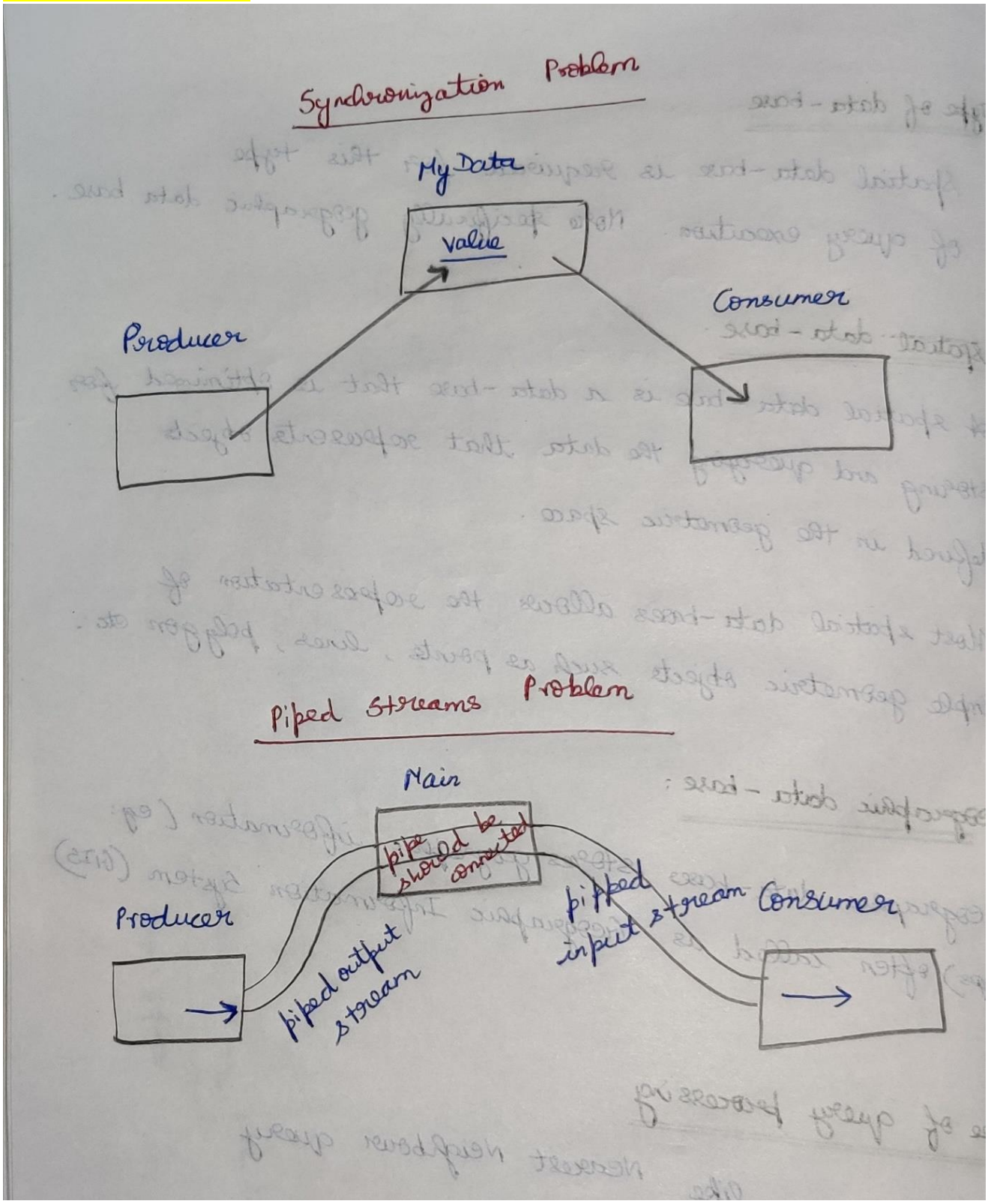


Buffered reader

```
1   import java.io.FileInputStream;
5   import java.io.FileReader;
6
7   public class reader {
8       public static void main(String args[]) throws Exception {
9           FileReader fis = new FileReader("F:/github/Java-Programming/" +
10                  "Streams/Streams_concept/src/Buffered_Streams/sample.txt");
11          BufferedReader bis = new BufferedReader(fis);
12
13          System.out.print((char)bis.read());
14          System.out.print((char)bis.read());
15          System.out.print((char)bis.read());
16          System.out.print((char)bis.read());
17          System.out.print((char)bis.read());
18          bis.mark(10);
19          System.out.print((char)bis.read());
20          System.out.print((char)bis.read());
21          bis.reset();
22          System.out.print((char)bis.read());
23          System.out.print((char)bis.read());
24
25          System.out.println("\nReading the ren
```

```
Output - Streams_concept (run-single) - Editor
Output - Streams_concept (run-single)
ant -f F:\\github\\Java-Programming\\Streams\\Stre
init:
Deleting: F:\github\Java-Programming\Streams\St
deps-jar:
Updating property file: F:\github\Java-Programmin
Compiling 1 source file to F:\github\Java-Programm
compile-single:
run-single:
Prashanan
Reading the remaining words : th
BUILD SUCCESSFUL (total time: 0 seconds)
```

Piped Streams



Pipes in IO provides a link between two threads running in JVM at the same time. So, Pipes are used both as source or destination.

- PipedInputStream is also piped with PipedOutputStream. So, data can be written using PipedOutputStream and can be written using PipedInputStream.But, using both threads at the same time will create a deadlock for the threads.
- PipedOutputStream is sending end of the pipe. Data is written to the PipedOutputStream. The pipe is said to be broken if the PipedInputStream, that was reading the data is no more.

PipedInputStreams

Method	Syntax	Description
read()	public int read()	reads next byte of the Piped Input Stream. The value is returned as integer in the range of 0 - 255. The method blocks if end of Stream is reached or exception is thrown.
read(byte[] buffer, int offset, int maxlen)	public int read(byte[] buffer, int offset, int maxlen)	reads upto maxlen bytes of the data from Piped Input Stream to the array of buffers. The method blocks if end of Stream is reached or exception is thrown.
receive(int byte)	protected void receive(int byte)	receives byte of the data. If no input is available, then the method blocks.
close()	public void close()	closes the Piped Input Stream and releases the allocated resources.
connect(PipedOutput Stream source)	public void connect(PipedOutputStream source)	connects the Piped Input Stream to the 'source' Piped Output Stream
available()	public int available()	returns no. of bytes that can be read from Input Stream without actually being blocked.

Declaration:

```
public class PipedInputStream
  extends InputStream
```

Constructor :

- **PipedInputStream() :** creates a PipedInputStream, that it is not connected.
- **PipedInputStream(int pSize) :** creates a PipedInputStream, that it is not connected with specified pipe size.
- **PipedInputStream(PipedOutputStream outStream) :** creates a PipedInputStream, that it is connected to PipedOutputStream – 'outStream'.
- **PipedInputStream(PipedOutputStream outStream, int pSize) :** creates a Piped Input Stream that is connected to Piped Output Stream with the specified pipe size.

```java
package Piped_Streams;

import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.io.IOException;

public class Piped_Input_Streams {
    public static void main(String args[]) throws IOException{
        PipedInputStream pis  = new PipedInputStream();
        PipedOutputStream pos = new PipedOutputStream();

        pis.connect(pos);  // connecting the input and output streams

        // Writing and Reading the character one-by-one
        pos.write(77);   System.out.println("Using read() : " + (char)pis.read());
        pos.write(80);   System.out.println("Using read() : " + (char)pis.read());
        pos.write(79);   System.out.println("Using read() : " + (char)pis.read());

        // Wrting and Reading the characters together
        pos.write(70);   pos.write(71);   pos.write(72);   pos.write(73);   pos.write(74);

        System.out.println("Available contents : " + pis.available());

        byte buffer [] = new byte[5];
        pis.read(buffer,0,5);                  // pis.read(buffer , offset , maxlen)
        String str = new String(buffer);   // converting the buffer to string

        System.out.println("String read : " + str);

    }
}
```

```
Using read() : M
Using read() : P
Using read() : O
Available contents : 5
String read : FGHIJ
BUILD SUCCESSFUL (total time: 0 seconds)
```

PipedOutputStreams

Method	Syntax	Description
write()	public void write(int byte)	writes specified byte to the Piped Output Stream
write(byte[] buffer, int offset, int maxlen)	public void write(byte[] buffer, int offset, int maxlen)	writes maxlen bytes of the data from buffer to the Piped Output Stream. The method blocks if no bytes are written to the Stream.
close()	public void close()	closes the Piped Output Stream and releases the allocated resources.
connect(PipedInput Stream destination)	public void connect(PipedInputStream destination)	connects the Piped Output Stream to the 'destination' Piped Input Stream
flush()	public void flush()	flushes the Output Stream.

```
1   package Piped_Streams;
2
3   import java.io.PipedInputStream;
4   import java.io.PipedOutputStream;
5   import java.io.IOException;
6
7   public class Piped_Output_Streams {
8       public static void main(String args[]) throws IOException{
9           PipedInputStream pis     = new PipedInputStream();
10          PipedOutputStream pos = new PipedOutputStream();
11
12          pis.connect(pos);  // connecting the input and output str
13
14          byte[] buffer = { 'P' , 'r' , 'a' , 's' , 'h' , 'a' , 'n' , 't' , 'h' };
15          pos.write(buffer,0,9);    // pos.write(buffer , offset , maxle
16          int a = 5;
17          while(a>0) {   System.out.print(" " + (char)pis.read());
18      }
19  }
20
```

```
Output - Streams_concept (run-single)
ant -f F:\\github\\Java-Programming\\S
init:
Deleting: F:\github\Java-Programming
deps-jar:
Updating property file: F:\github\Java-
Compiling 1 source file to F:\github\Ja
compile-single:
run-single:
 P r a s h a n t h
```

Producer Consumer problem

```
1   package Piped_Streams;
2
3   import java.io.PipedInputStream;
4   import java.io.PipedOutputStream;
5   import java.io.IOException;
6
7   class Producer extends Thread {
8       PipedOutputStream pos;
9       Producer(PipedOutputStream pos_main) {  this.pos = pos_main;    }
10      synchronized public void run() {
11          int count = 1;
12          while(true) {
13              try{
14                  pos.write(count);
15                  System.out.println("Producer produced : " + count);
16                  count+=1;
17                  pos.flush();
18                  Thread.sleep(10);
19              } catch(Exception e) {}
20          }
21      }
22  }
23
24  class Consumer extends Thread {
25      PipedInputStream pis;
26      Consumer(PipedInputStream pis_main) { this.pis = pis_main; }
27      synchronized public void run() {
28      int x;
29      try {
30          while(true) {
31              Thread.sleep(10);
32              x = pis.read();
33              System.out.println("Consumer consumed : " + x);
34          }
35      }  catch(Exception e) {}
36    }
37  }
38
39  public class Producer_Consumer {
40    public static void main(String args[]) throws IOException {
41      PipedInputStream pis     = new PipedInputStream();
42      PipedOutputStream pos = new PipedOutputStream();
43
44      pis.connect(pos);      // or pos.connect(pis)
45
46      Producer p = new Producer(pos);
47      Consumer c = new Consumer(pis);
48
49      p.start();
50      c.start();
51    }
52  }
53
```

```
compile-single:
run-single:
Producer produced : 1
Consumer consumed : 1
Producer produced : 2
Consumer consumed : 2
Producer produced : 3
Consumer consumed : 3
Producer produced : 4
Consumer consumed : 4
Producer produced : 5
Consumer consumed : 5
Producer produced : 6
Consumer consumed : 6
Producer produced : 7
Consumer consumed : 7
Producer produced : 8
Consumer consumed : 8
Producer produced : 9
Consumer consumed : 9
Producer produced : 10
Consumer consumed : 10
Producer produced : 11
Consumer consumed : 11
Producer produced : 12
Consumer consumed : 12
Producer produced : 13
Consumer consumed : 13
```