1. Đọc nghi file trong java với byte stream

Các chương trình sử dụng Byte Stream để đọc ghi dữ liệu theo từng byte(8bit). Tất cả các class Byte Stream có nguồn gốc từ InputStream và OutputStream.

Sử dụng Byte Stream

Có rất nhiều class Byte Stream, để hình dung Byte Stream hoạt động như thế nào, chúng ta sẽ tập trung vào FileInputStream và FileOutputStream, ví dụ:

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public class CopyFileByte {      public static void main(String [] args) throws IOException {        FileInputStream inputStream = null;        FileOutputStream outputStream = null;          try {           inputStream = new FileInputStream("inStream.txt");           outputStream = new FileOutputStream("outStream.txt");             int c;           while ((c = inputStream.read()) != -1) {              outputStream.write(c);           }        } finally {           if (inputStream != null) {              inputStream.close();           }           if (outputStream != null) {              outputStream.close();           }        }      }  } |

Việc đóng một Stream khi mà không có nhu cầu sử dụng nó nữa là một việc rất quan trong - tránh bị leak tài nguyên. Ví dụ trên sử dụng khối finally để đảm bảo cả 2 Streams (input, output) đều được đóng ngay cả khi có lỗi xảy ra.

2. Đọc ghi file trong java với character stream

Byte Stream trong Java được sử dụng để thực hiện input và output của các byte (8 bit), trong khi đó, Character Stream trong Java được sử dụng để thực hiện input và output cho Unicode 16 bit. Tất cả các class Character Stream có nguồn gốc từ Reader và Writer.

Sử dụng Character Stream

Mặc dù có nhiều lớp liên quan tới Character Stream nhưng các lớp thường dùng nhất là FileReader và FileWriter, ví dụ:

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | ublic class CopyFileCharacter {      public static void main(String [] args) throws IOException {        FileReader in = null;        FileWriter out = null;          try {           in = new FileReader("input.txt");           out = new FileWriter("output.txt");             int c;           while ((c = in.read()) != -1) {              out.write(c);           }        }finally {           if (in != null) {              in.close();           }           if (out != null) {              out.close();           }        }      }  } |

3. Đọc ghi file trong java với buffered stream

Các ví dụ trên không sử dụng Buffered Streams, điều này có nghĩa là việc đọc và xuất dữ liệu được thực hiện trực tiếp dưới quyền điều khiển của hệ điều hành, gây lãng phí thời gian và tài nguyên. Để giảm thiểu những trên, Buffered Streams đã được sinh ra. Buffered Streams được sử dụng để tăng tốc độ hoạt động I/O, bằng cách đơn giản là tạo ra một khoảng nhớ đệm với kích thước cụ thể nào đó. Vì vậy chúng ta không cần phải truy cập vào ổ đĩa cứng khi thực hiện I/O. Một chương trình có thể chuyển đổi từ không sử dụng buffered stream (Byte Stream và Chracter Stream sang sử dụng buffered stream bằng việc sử dụng ý tưởng "Wrapping"

Lớp wrapper cho byte stream

1. [BufferedInputStream](https://viettuts.vn/java-io/lop-bufferedinputstream-trong-java)
2. [BufferedOutputStream](https://viettuts.vn/java-io/lop-bufferedoutputstream-trong-java)

Lớp wrapper cho character stream

1. [BufferedReader](https://viettuts.vn/java-io/lop-bufferedreader-trong-java)
2. [BufferedWriter](https://viettuts.vn/java-io/lop-bufferedwriter-trong-java)

Sử dụng buffered stream để đọc ghi file

Ví dụ: Sử dụng Wrapper cho byte stream

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | public class CopyFileBuffer1 {      public static void main(String [] args) throws IOException {          BufferedInputStream bufferIn = null;          BufferedOutputStream bufferOut = null;            try {              InputStream inputStream = new FileInputStream("input.txt");              OutputStream outputStream = new FileOutputStream("output.txt");                bufferIn = new BufferedInputStream(inputStream);              bufferOut = new BufferedOutputStream(outputStream);                int c;              while ((c = bufferIn.read()) != -1) {                  bufferOut.write(c);              }          } finally {              if (bufferIn != null) {                  bufferIn.close();              }              if (bufferOut != null) {                  bufferOut.close();              }          }      }  } |

Ví dụ: Sử dụng Wrapper cho character stream

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | public class CopyFileBuffer2 {      public static void main(String [] args) throws IOException {          BufferedReader bufferedReader = null;          BufferedWriter bufferedWriter = null;            try {              Reader reader = new FileReader("input.txt");              Writer writer = new FileWriter("output.txt");                bufferedReader = new BufferedReader(reader);              bufferedWriter = new BufferedWriter(writer);                int c;              while ((c = bufferedReader.read()) != -1) {                  bufferedWriter.write(c);              }          } finally {              if (bufferedReader != null) {                  bufferedReader.close();              }              if (bufferedWriter != null) {                  bufferedWriter.close();              }          }      }  } |

4. Đọc ghi file trong java với Scanning và Formatting

Scanning

Lớp Scanner có thể phân tích được các kiểu dữ liệu nguyên thủy và kiểu String bằng việc sử dụng biểu thức chính quy (regular expressions). Mặc định các khoảng trắng được Scanner dùng để phân biệt các ký tự trong một chuỗi.

Sử dụng Scanner

Scanner có thể được sử dụng để nhập dữ liệu từ bàn phím, phân tích chuỗi ký tự, nhập dữ liêu từ file.

Ví dụ 1: Đoạn code này cho phép người dùng đọc ký tự số từ bàn phím bằng cách khởi tạo Scanner sử dụng System.in:

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2 | Scanner sc = new Scanner(System.in);  int i = sc.nextInt(); |

Ví dụ 2: Đoạn code này lấy tất cả các số có kiểu long từ file myNumbers.txt:

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2  3  4 | Scanner sc = new Scanner(new File("myNumbers.txt"));  while (sc.hasNextLong()) {      long aLong = sc.nextLong();  } |

Ví dụ 3: Scanner cũng có thể sử dụng dụng bộ phân cách (Delimiters) khác do người dùng tự định nghĩa, đoạn code sau đọc và phân tích thành các mục từ một chuỗi.

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | String input = "1 fish 2 fish red fish blue fish";  Scanner s = new Scanner(input).useDelimiter("\\s\*fish\\s\*");  System.out.println(s.nextInt());  System.out.println(s.nextInt());  System.out.println(s.next());  System.out.println(s.next());  s.close(); |

Một số phương thức của lớp Scanner

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| nextInt() | trả về kiểu int |
| nextFloat() | trả về kiểu float |
| nextBoolean() | trả về kiểu boolean |
| nextByte() | trả về kiểu byte |
| nextLine() | trả về kiểu String |

Formatting

Ngoài 2 phương thức print và println. Java cung cấp 2 phương thức định dạng in cho chúng ta là printf và format và 2 phương thức này có chức năng tương tự nhau. Bạn có thể gọi phương thức này ở bất kỳ nơi đâu trong code của bạn.

Ví dụ: định dạng in cho các biến số:

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2  3 | System.out.format("The value of " + "the float variable is " +       "%f, while the value of the " + "integer variable is %d, " +       "and the string is %s", floatVar, intVar, stringVar); |

Một điều nữa là ta có thể in theo hệ thống định dạng Pháp, ngăn cách phần nguyên và phần thập phân của số thực thập phân bằng định dạng Locale.FRANCE, ví dụ:

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2  3  4 | System.out.format(Locale.FRANCE,      "The value of the float " + "variable is %f, while the " +      "value of the integer variable " + "is %d, and the string is %s%n",      floatVar, intVar, stringVar); |

Ví dụ: một chương trình đơn giản:

[?](https://viettuts.vn/vi-du-java-io/doc-ghi-file-trong-java)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | public class TestFormat {        public static void main(String[] args) {        long n = 461012;        System.out.format("%d%n", n);      //  -->  "461012"        System.out.format("%08d%n", n);    //  -->  "00461012"        System.out.format("%+8d%n", n);    //  -->  " +461012"        System.out.format("%,8d%n", n);    // -->  " 461,012"        System.out.format("%+,8d%n%n", n); //  -->  "+461,012"          double pi = Math.PI;          System.out.format("%f%n", pi);       // -->  "3.141593"        System.out.format("%.3f%n", pi);     // -->  "3.142"        System.out.format("%10.3f%n", pi);   // -->  "     3.142"        System.out.format("%-10.3f%n", pi);  // -->  "3.142"        System.out.format(Locale.FRANCE,                          "%-10.4f%n%n", pi); // -->  "3,1416"          Calendar c = Calendar.getInstance();        System.out.format("%tB %te, %tY%n", c, c, c); // -->  "May 29, 2006"          System.out.format("%tl:%tM %tp%n", c, c, c);  // -->  "2:34 am"          System.out.format("%tD%n", c);    // -->  "05/29/06"      }  } |

Bảng sau thông kê một vài "Converters" và "Flags" được sử dụng trong ví dụ trên:

|  |  |  |
| --- | --- | --- |
| **Converter** | **Flag** | **Giải thích** |
| d |  | Một số nguyên thập phân. |
| f |  | Một số float. |
| n |  | Một ký tự dòng mới phù hợp với nền tảng chạy ứng dụng. Bạn nên luôn luôn sử dụng %n, hơn là \n. |
| tB |  | In ra tên tháng trong biến date & time, ví dụ: April. |
| td, te |  | In ra ngày trong biến date & time, 2 số của ngày trong tháng. td sẽ in ra cả số 0 chẳng hạn như ngày 07, te thì không. |
| ty, tY |  | In ra năm trong biến date & time, ty = 2-số cuối của năm, tY = 4-số của năm. |
| tl |  | In ra giờ trong biến date & time, theo định dạng 12-giờ. |
| tM |  | In ra phút trong biến date & time, bao gồm 2 số, cả số 0 nếu cần thiết. |
| tp |  | In ra date & time dưới dạng am/pm (chữ thường). |
| tm |  | In ra tháng của biến date & time, gồm 2 số, cả số 0 nếu cần thiết. |
| tD |  | In ra ngày của biến date & time như định dạng %tm%td%ty |
|  | 08 | Chỉ định chiều rộng là 8 ký tự, bao gồm cả số 0 nếu cần thiết. |
|  | + | In ra ký tự +. |
|  | , | Nhóm các số bằng dấu ',' thay vì '.', ví dụ 10,000,900 thay vì 10.000.900 . |
|  | - | Căn trái. |
|  | .3 | Ba số sau dấu thập phân. |
|  | 10.3 | Rộng 10 ký tự, căn phải, với 3 số sau dấu thập phân. |

4. Đọc ghi file trong java theo ObJect

– Đặt 1 tình huống thực tế: khi ta muốn quản lý 1 danh sách sinh viên, tất nhiên sẽ là 1 dãy rất nhiều đối tượng, mỗi đối tượng sinh viên lại có nhiều thuộc tính: tên, tuổi, điểm, .v..v.. và chúng ta cần lưu toàn bộ thông tin này vào file, ngoài ra còn phải xử lý, chỉnh sửa thông tin nếu cần. Như vậy, nếu sử dụng kiến thức ở những bài trước làm điều này thì là rất khó khăn. Nhưng với phần kiến thức ở bài này thì điều đó lại trở lên cực kỳ dễ dàng. Cụ thể ở bài tập giống như tình huống giả thiết trên, chúng ta sẽ sử dụng kiểu đọc ghi object để thao thác, lưu trữ từng đối tượng lên file!

– Mình sẽ viết 2 ví dụ cụ thể, với các mức độ khác nhau,từ đơn giản nhất. Nếu bạn thấy khó hiểu chỗ nào có thể bình luận phía dưới. Mọi người cùng trao đổi.

\*\**Chú ý: Để đọc ghi theo Object trong Java thì tại class Object đó ta cần đặt giao tiếp java.io.Serializable (implements Serializable)*

Ví dụ 1: Đọc ghi 1 đối tượng lên file theo từng object:  
– Khai báo 1 class SinhVien gồm 2 thuộc tính họ tên, tuổi. Tạo 1 đối tượng cụ thể sinh viên a sau đó  
gán thông tin cho sinh viên a rồi ghi đối tượng a này xuống file “E:\\sinhvien.dat”  
– Mở file trên ra. Đọc file đó theo kiểu object để lấy đối tượng đó ra, (cần ép kiểu về kiểu dữ liệu của đối tượng), rồi gán nó vào đối tượng a1. In ra màn hình thông tin đối tượng a1. Code như sau:

class SinhVien implements Serializable {

public String hoTen;

public int tuoi;

}

public class JavaAndroidVn {

public static void main(String[] args) throws IOException {

SinhVien a = new SinhVien();

a.hoTen = "Vũ Văn Tường";

a.tuoi = 21;

try {

FileOutputStream f = new FileOutputStream("E:\\sinhvien.dat");

ObjectOutputStream oOT = new ObjectOutputStream(f); // Sử dụng để ghi file theo từng Object

oOT.writeObject(a); // Ghi Object là đối tượng a xuống file

oOT.close();

f.close();

} catch (IOException e) {

System.out.println("Có lỗi xảy ra!");

}

SinhVien a1 = new SinhVien(); //Tạo đối tượng a1 mới, để phía dưới gán bằng đối tượng ta lấy được ra từ file

try {

FileInputStream f = new FileInputStream("E:\\sinhvien.dat");

ObjectInputStream oIT = new ObjectInputStream(f); // Sử

dụng để đọc file theo từng Object

a1 = (SinhVien) oIT.readObject(); //Đọc Object đầu tiên

ép kiểu về kiểu SinhVien sau đó gán bằng đối tượng a1

oIT.close();

f.close();

} catch (IOException io) {

System.out.println("Có lỗi xảy ra!");

} catch (ClassNotFoundException ex) {

System.out.println("Không tìm thấy class");

}

System.out.println("Thông tin sinh vien a1 đọc ra là: \nTên: "+a1.hoTen+"\nTuổi: "+a1.tuoi);

}

}

Ví dụ 2: Đọc ghi 2 đối tượng lên file theo từng object. Các thuộc tính bài này để private:  
(Qua ví dụ này, và kết hợp bài sau, mình sẽ giới thiệu cách đọc ghi 1 danh sách các đối tượng lên file, giúp việc quản lý danh sách các đối tượng dễ dàng hơn!)

class SinhVien implements Serializable {

private String hoTen;

private int tuoi;

public String getHoTen() {

return hoTen;

}

public void setHoTen(String hoTen) {

this.hoTen = hoTen;

}

public int getTuoi() {

return tuoi;

}

public void setTuoi(int tuoi) {

this.tuoi = tuoi;

}

}

public class JavaAndroidVn {

public static void main(String[] args) throws IOException {

SinhVien a = new SinhVien();

a.setHoTen("Vũ Văn T");

a.setTuoi(21);

SinhVien b = new SinhVien();

b.setHoTen("Nguyễn Văn A");

b.setTuoi(22);

try {

FileOutputStream f = new FileOutputStream("E:\\sinhvien.dat");

ObjectOutputStream oOT = new ObjectOutputStream(f); // Sử dụng để ghi file theo từng Object

oOT.writeObject(a); // Ghi Object là đối tượng a xuống file

oOT.writeObject(b);

oOT.close();

f.close();

} catch (IOException e) {

System.out.println("Có lỗi xảy ra!");

}

SinhVien a1 = new SinhVien(); //Tạo đối tượng a1 mới, để phía dưới gán bằng đối tượng ta lấy được ra từ file

SinhVien b1 = new SinhVien(); //Tạo đối tượng b1 mới, để phía dưới gán bằng đối tượng ta lấy được ra từ file

try {

FileInputStream f = new FileInputStream("E:\\sinhvien.dat");

ObjectInputStream oIT = new ObjectInputStream(f); // Sử dụng để đọc file theo từng Object

a1 = (SinhVien) oIT.readObject(); //Đọc Object đầu tiên ép kiểu về kiểu SinhVien sau đó gán bằng đối tượng a1

b1 = (SinhVien) oIT.readObject(); //Đọc Object tiếp theo, ép kiểu về kiểu SinhVien sau đó gán bằng đối tượng b1

oIT.close();

f.close();

} catch (IOException io) {

System.out.println("Có lỗi xảy ra!");

} catch (ClassNotFoundException ex) {

System.out.println("Không tìm thấy class");

}

System.out.println("Thông tin sinh vien a1 đọc ra là: \nTên: " + a1.getHoTen() + "\nTuổi: " + a1.getTuoi());

System.out.println("Thông tin sinh vien b1 đọc ra là: \nTên: " + b1.getHoTen() + "\nTuổi: " + b1.getTuoi());

}

}

## ****1. Lớp BufferedReader trong Java****

Lớp **BufferedReader** nằm trong package **java.io** được sử dụng với các lớp **Reader** khác để giúp đọc dữ liệu là các ký tự (character) hiệu quả hơn. Lớp **BufferedReader** kế thừa từ lớp **abstract** **Reader**.

Lớp **BufferedReader**gồm có một bộ nhớ đệm (internal buffer) lưu trữ **8192 ký tự**. Khi sử dụng **BufferedReader**, một số lượng nhất định các ký tự được đọc từ nơi lưu trữ (data source) và lưu trữ trong bộ nhớ đệm này. Do đó, số lần truy xuất dữ liệu đến data source giảm đi. Đó là lý do tại sao đọc dữ liệu với **BufferedReader** hiệu quả hơn..

#### ****Tạo một BufferedReader****

Để sử dụng **BufferedReader**, chúng ta phải **import** **java.io.BuferedReader**.

//Tạo ra một FileReader

FileReader file = new FileReader(String file);

//Tạo ra một BufferedReader

BufferedReader buffer = new BufferedReader(file);

Trong ví dụ trên, chúng ta tạo ra một đối tượng **BufferedReader** tên là **buffer** với một đối tượng **FileReader** tên là **file** để giúp đọc dữ liệu hiệu quả hơn.

Bộ nhớ đệm (internal buffer) mặc định của **BufferedReader** có thể lưu trữ **8192 ký tự**. Chúng ta cũng có thể thay đổi kích thước lưu trữ của bộ nhớ đệm với tham số **int size**.

//Tạo một BufferdReader với tham số int size

BufferedReader buffer = new BufferedReader(file, int size);

#### ****Các phương thức của BufferedReader****

**Phương thức read()**

* **read()** đọc một ký tự từ internal buffer của reader
* **read(char[] array)** đọc các ký tự từ reader và lưu các ký tự này vào mảng array
* **read(char[] array, int start, int length)** đọc **length** ký tự từ reader và lưu trữ vào mảng array tại vị trí **start** của mảng

Giả sử có một file **input.txt** trong phân vùng **D** với nội dung bên dưới.

**D:\\input.txt**

This is a line of text inside the file.

Chúng ta có thể đọc dữ liệu là các ký tự trong file với **BufferedReader**.

**Main.java**

import java.io.FileReader;

import java.io.BufferedReader;

class Main {

public static void main(String[] args) {

//Tạo một array lưu trữ các ký tự

char[] array = new char[100];

try {

//Tạo một FileReader

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

//Tạo một BufferedReader

BufferedReader input = new BufferedReader(file);

//Đọc các ký tự và lưu vào array

input.read(array);

System.out.println("Data in the file: ");

System.out.println(array);

//Đóng reader

input.close();

}

catch(Exception e) {

System.err.println(e.getStackTrace());

}

}

}

##### **Kết quả**

Data in the file:

This is a line of text inside the file.

**Phương thức skip()**

Để bỏ qua một số ký tự không cần đọc, chúng ta có thể sử dụng hàm **skip()**.

**Main.java**

import java.io.FileReader;

import java.io.BufferedReader;

public class Main {

public static void main(String args[]) {

char[] array = new char[100];

try {

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

BufferedReader input = new BufferedReader(file);

//Bỏ qua 5 ký tự

input.skip(5);

//Đọc các ký tự và lưu vào array

input.read(array);

System.out.println("Data after skipping 5 characters:");

System.out.println(array);

input.close();

}

catch (Exception e) {

System.err.println(e.getStackTrace());

}

}

}

##### **Kết quả**

Data after skipping 5 characters:

is a line of text inside the file.

Trong ví dụ trên, chúng ta đã bỏ qua 5 ký tự ‘T’, ‘h’, ‘i’, ‘s’ và ‘ ‘ không cần đọc.

## ****2. Lớp BufferedWriter trong Java****

Lớp **BufferedWriter**nằm trong package **java.io** được sử dụng với các lớp **Writer** khác để giúp ghi dữ liệu là các ký tự (character) hiệu quả hơn. Lớp **BufferedWriter**kế thừa từ lớp **abstract** **Writer**.

Lớp **BufferedWriter** có sử dụng bộ nhớ đệm (internal buffer) có khả năng lưu trữ **8192 ký tự**. Khi sử dụng lớp **BufferedWriter** các ký tự sẽ được ghi vào internal buffer thay vì ghi ngay lập tức vào nơi cần lưu trữ (data destination). Khi internal buffer đã đầy hoặc Writer đóng thì tất cả các ký tự sẽ được ghi vào data destination. Vì thế, số lần ghi vào data destination sẽ giảm. Điều đó giải thích vì sao **BufferedWriter** giúp ghi dữ liệu hiệu quả hơn.

#### ****Tạo một BufferedWriter****

Để sử dụng **BufferedWriter**, chúng ta phải **import java.io.BufferedWriter**.

//Tạo một FileWriter

FileWriter file = new FileWriter(String name);

//Tạo một BufferedWriter

BufferedWriter buffer = new BufferedWriter(file);

Trong ví dụ trên, chúng ta tạo ra một đối tượng **BufferedWriter** tên là **buffer** với một đối tượng **FileWriter** tên là **file** để giúp ghi dữ liệu hiệu quả hơn. Bộ nhớ đệm (internal buffer) mặc định của **BufferedWriter** có thể lưu trữ **8192 ký tự**. Chúng ta cũng có thể thay đổi kích thước lưu trữ của bộ nhớ đệm với tham số **int size**.

//Tạo một BufferedWriter với tham số int size

BufferedWriter buffer = new BufferedWriter(file, int size);

#### ****Các phương thức của BufferedWriter****

**Phương thức write()**

* **write()** ghi một ký tự vào internal buffer của writer
* **write(char[] array)** ghi các ký tự của mảng array vào writer
* **write(String data)** ghi chuỗi **String data** vào writer

**Main.java**

import java.io.FileWriter;

import java.io.BufferedWriter;

public class Main {

public static void main(String args[]) {

String data = "This is the data in the output file";

try {

//Tạo một FileWriter

FileWriter file = new FileWriter("D:\\output.txt");

//Tạo một BufferedWriter

BufferedWriter output = new BufferedWriter(file);

//Ghi string vào file

output.write(data);

//Đóng writer

output.close();

}

catch (Exception e) {

System.err.println(e.getStackTrace());

}

}

}

Chương trình trên sẽ ghi dữ liệu **String data = "This is the data in the output file.";** vào file **D:\\output.txt**.

**Phương thức flush()**

Để xóa internal buffer, chúng ta có thể sử dụng hamf **flush()**. Hàm này bắt buộc ghi tất cả dữ liệu trong buffer và data destination.

**Main.java**

import java.io.FileWriter;

import java.io.BufferedWriter;

public class Main {

public static void main(String[] args) {

String data = "This is a demo of the flush method";

try {

FileWriter file = new FileWriter("D:\\flush.txt");

BufferedWriter output = new BufferedWriter(file);

output.write(data);

//ghi tất cả dữ liệu vào data destination

output.flush();

System.out.println("Data is flushed to the file.");

output.close();

}

catch(Exception e) {

System.err.println(e.getStackTrace());

}

}

}

Khi không sử dụng **BufferedWriter** hoặc **BufferedReader** nữa thì các bạn nhớ sử dụng hàm **close()** để đóng kết nối với output stream và input stream.