File và I/O trong Java

Gói **java.io** chứa gần như tất cả các lớp bạn cần để thực hiện input và output (I/O) trong Java. Tất cả những stream này biểu diễn một nguồn input và một output đích đến. Stream trong java.io package hỗ trợ nhiều như liệu như các kiểu gốc, Object, các ký tự nội bộ, …

Một stream có thể được định nghĩa như là một dãy liên tục dữ liệu. **InputStream** được sử dụng để đọc dữ liệu từ một nguồn và **OutputStream** được sử dụng để ghi dữ liệu tới một đích đến.

Java cung cấp sự hỗ trợ mạnh mẽ nhưng linh hoạt cho I/O liên quan tới các File và các mạng nhưng trong phần hướng dẫn này, chúng tôi chỉ bàn luận tính năng cơ bản liên quan tới các stream và I/O. Chúng ta sẽ xem xét từng ví dụ được sử dụng phổ biến nhất.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài: **Ví dụ về File trong Java**.

Byte Stream trong Java

Byte Stream trong Java được sử dụng để thực hiện input và output của các byte (8 bit). Mặc dù có nhiều lớp liên quan tới byte stream nhưng các lớp thường được sử dụng nhất là:**FileInputStream** và **FileOutputStream**. Sau đây là một ví dụ sử dụng hai lớp này để sao chép một input file vào trong một output file:

```
import java.io.*;


public class CopyFile {
   public static void main(String args[]) throws IOException
   {
      FileInputStream in = null;
      FileOutputStream out = null;


      try {
         in = new FileInputStream("input.txt");
         out = new FileOutputStream("output.txt");


         int c;
         while ((c = in.read()) != -1) {
```

```
            out.write(c);
         }
      }finally {
         if (in != null) {
            in.close();
         }
         if (out != null) {
            out.close();
         }
      }
   }
}
```

Bây giờ giả sử chúng ta có một file là *input.txt* có nội dung sau:

```
This is test for copy file.
```

Trong bước tiếp theo, biên dịch chương trình trên và thực thi nó, sẽ cho kết quả là tạo một file là *output.txt* có cùng nội dung như chúng ta có trong input.txt. Vì thế, bạn đặt code trên vào trong CopyFile.java file và làm như sau:

```
$javac CopyFile.java
$java CopyFile
```

Character Stream trong Java

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. Mặc dù có nhiều lớp liên quan tới character stream nhưng các lớp thường dùng nhất là **FileReader** và **FileWriter**… Mặc dù trong nội tại, FileReader sử dụng FileInputStream và FileWriter sử dụng FileOutputStream, nhưng có một điểm khác biệt lớn ở đây là FileReader đọc hai byte cùng một thời điểm và FileWriter ghi 2 byte cùng một lúc.

Chúng ta có thể viết lại ví dụ trên mà sử dụng hai lớp này để sao chép một input file (có các ký tự Unicode) vào trong một output file.

```
import java.io.*;
```

```
public class CopyFile {

   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();
         }
      }
   }
}
```

Giả sử chúng ta có **input.txt** có nội dung sau:

```
This is test for copy file.
```

Trong bước tiếp theo, biên dịch chương trình trên và thực thi nó, sẽ cho kết quả là tạo một file là output.txt có cùng nội dung như chúng ta có trong input.txt. Vì thế, bạn đặt code trên vào trong CopyFile.java file và làm như sau:

```
$javac CopyFile.java
$java CopyFile
```

Standard Stream trong Java

Tất cả các Ngôn ngữ lập trình cung cấp sự hỗ trợ cho I/O chuẩn, tại đây chương trình của người sử dụng có thể nhận đầu vào từ một bàn phím và sau đó tạo kết quả trên màn hình máy tính. Nếu bạn đã biết về các ngôn ngữ C/C++, thì bạn phải biết về 3 thiết bị chuẩn là STDIN, STDOUT, và STDERR. Theo cách tương tự, Java cung cấp 3 Standard Stream sau:

- **Đầu vào chuẩn (Standard Input):** Nó được sử dụng để truyền dữ liệu tới chương trình của người dùng và thường thì một bàn phím được sử dụng như là đầu vào chuẩn và được biểu diễn như **System.in**.
- **Đầu ra chuẩn (Standard Output):** Nó được sử dụng để hiển thị kết quả đầu ra từ chương trình của người dùng và thường thì một màn hình máy tính được sử dụng như là đầu ra chuẩn và được biểu diễn như là **System.out**.
- **Lỗi chuẩn (Standard Error):** Được sử dụng để hiển thị các lỗi trong chương trình của người dùng và thường thì một màn hình máy tính được sử dụng như là lỗi chuẩn và được biểu diễn như là **System.err**.

Sau đây là một chương trình đơn giản tạo **InputStreamReader** để đọc luồng đầu vào chuẩn tới khi người sử dụng gõ một "q".

```
import java.io.*;

public class ReadConsole {
   public static void main(String args[]) throws IOException
   {
      InputStreamReader cin = null;

      try {
         cin = new InputStreamReader(System.in);
         System.out.println("Enter characters, 'q' to quit.");
         char c;
         do {
            c = (char) cin.read();
            System.out.print(c);
         } while(c != 'q');
```

```
      }finally {
         if (cin != null) {
            cin.close();
         }
      }
   }
}
```

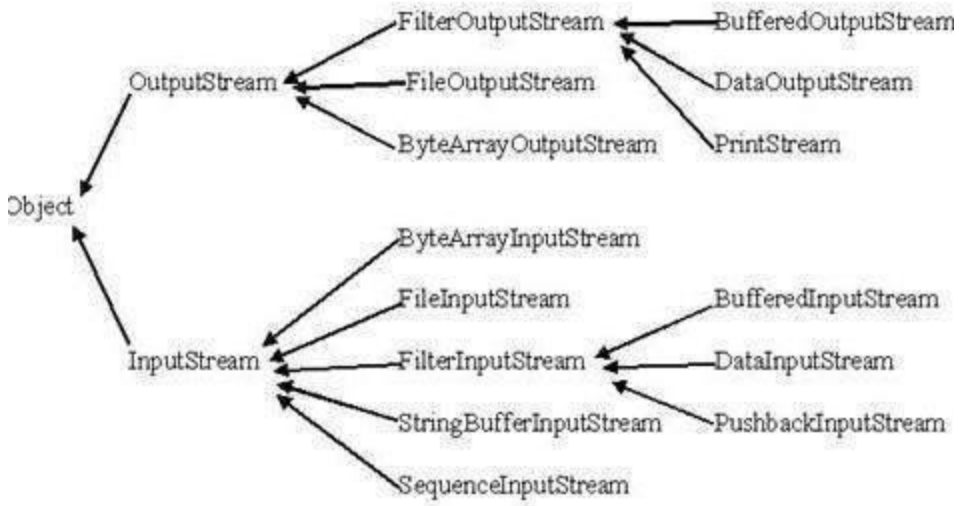
Giữ code trên trong ReadConsole.java file và thực thi và biên dịch nó như dưới đây. Chương trình này tiếp tục đọc và hiển thị kết quả tới khi chúng ta nhấn phím “q”.

```
$javac ReadConsole.java
$java ReadConsole
Enter characters, 'q' to quit.
1
1
e
e
q
q
```

Đọc và Ghi File trong Java

Như đã miêu tả trước đó, một Stream có thể được định nghĩa như là một dãy liên tục của dữ liệu. **InputStream** được sử dụng để đọc dữ liệu từ một nguồn và **OutputStream** được sử dụng để ghi dữ liệu tới một đích.

Dưới đây là một cấu trúc có thứ tự của các lớp để xử lý các luồng Input và Output.



Hai luồng quan trọng nhất là **FileInputStream** và **FileOutputStream**, sẽ được bàn luận sau đây:

FileInputStream trong Java:

Luồng này được sử dụng để đọc dữ liệu từ các file. Các đối tượng có thể được tạo bởi sử dụng từ khóa new và có một số kiểu constructor có sẵn.

Constructor sau đây nhận tên file như là một chuỗi để tạo một đối tượng Input Stream để đọc file:

```
InputStream f = new FileInputStream("C:/java/hello");
```

Constructor sau nhận một đối tượng File để tạo một đối tượng Input Stream để đọc file. Đầu tiên chúng ta tạo một đối tượng file bởi sử dụng phương thức File() như sau:

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Khi chúng ta có đối tượng **InputStream**, thì khi đó có một danh sách các phương thức có thể được sử dụng để đọc stream hoặc để thực hiện hoạt động nào khác trên stream này.

STT	Phương thức và Miêu tả
1	**public void close() throws IOException{}** Phương thức này đóng output stream. Giải phóng bất kỳ nguồn hệ thống nào liên kết với file. Ném một IOException

2	**protected void finalize()throws IOException {}** Phương thức này xóa sự kết nối tới File đó. Bảo đảm rằng phương thức close của output stream này được gọi khi không có tham chiếu nào nữa tới stream này. Ném một IOException
3	**public int read(int r)throws IOException{}** Phương thức này đọc byte dữ liệu đã xác định từ InputStream. Trả về một int. Trả về byte dữ liệu tiếp theo và -1 sẽ được trả về nếu kết thúc file.
4	**public int read(byte[] r) throws IOException{}** Phương thức này đọc r byte từ input stream vào trong một mảng. Trả về tổng số byte đã đọc. Nếu kết thúc file, -1 được trả về.
5	**public int available() throws IOException{}** Cung cấp số byte mà được đọc từ input stream này. Trả về một int

Có một số input stream quan trọng khác có sẵn, để biết thêm chi tiết, bạn tham khảo theo link sau:

- ByteArrayInputStream
- DataInputStream

FileOutputStream trong Java

FileOutputStream được sử dụng để tạo một file và ghi dữ liệu vào trong nó. Luồng này sẽ tạo một file, nếu nó đã không tồn tại, trước khi mở nó để ghi output.

Dưới đây là hai constructor mà có thể được sử dụng để tạo một đối tượng FileOutputStream trong Java.

Constructor sau nhận một tên file như là một chuỗi để tạo một đối tượng input stream để ghi file.

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

Constructor sau nhận một đối tượng file để tạo một đối tượng output stream để ghi file.

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Khi bạn có đối tượng **OutputStream** này, thì sau đây có các phương thức có thể được sử dụng để ghi stream hoặc để thực hiện các hoạt động khác trên stream này:

STT	Phương thức và Miêu tả
1	**public void close() throws IOException{}** Phương thức này đóng output stream. Giải phóng bất kỳ nguồn hệ thống nào liên kết với file. Ném một IOException
2	**protected void finalize()throws IOException {}** Phương thức này xóa sự kết nối tới File đó. Bảo đảm rằng phương thức close của output stream này được gọi khi không có tham chiếu nào nữa tới stream này. Ném một IOException
3	**public void write(int w)throws IOException{}** Phương thức này ghi byte đã xác định tới output stream
4	**public void write(byte[] w)** Ghi w byte từ mảng byte đã đề cập tới OutputStream.

Ngoài ra cũng có một số output stream quan trọng khác, bạn tham khảo theo link sau:

- ByteArrayOutputStream
- DataOutputStream

Ví dụ:

Ví dụ sau minh họa InputStream và OutputStream:

```
import java.io.*;
```

```
public class fileStreamTest{


   public static void main(String args[]){


   try{
      byte bWrite [] = {11,21,3,40,5};
      OutputStream os = new FileOutputStream("test.txt");
      for(int x=0; x < bWrite.length ; x++){
         os.write( bWrite[x] ); // writes the bytes
      }
      os.close();


      InputStream is = new FileInputStream("test.txt");
      int size = is.available();


      for(int i=0; i< size; i++){
         System.out.print((char)is.read() + "  ");
      }
      is.close();
   }catch(IOException e){
      System.out.print("Exception");
   }
   }
}
```

Code trên tạo test.txt file và sẽ ghi các số đã cho trong định dạng nhị phân. Kết quả tương tự trên màn hình stdout.

Điều hướng file và I/O trong Java

Có một số lớp khác chúng ta cần tìm hiểu để biết các khái niệm cơ bản về Điều hướng file và I/O.

- Lớp File trong Java
- Lớp FileReader trong Java

- Lớp FileWriter trong Java

Thư mục trong Java

Một thư mục là File mà có thể giữ một danh sách các file và thư mục khác. Bạn sử dụng đối tượng **File** để tạo các thư mục, để liệt kê các file có sẵn trong một thư mục. Để hiểu chi tiết, bạn kiểm tra các phương thức mà bạn có thể gọi trên đối tượng File và những gì liên quan tới thư mục.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài: **Ví dụ về Thư mục trong Java**.

Tạo thư mục trong Java

Có hai phương thức tiện ích **File** hữu ích, mà có thể được sử dụng để tạo các thư mục:

- Phương thức **mkdir()** tạo một thư mục, trả về true nếu thành công và false nếu thất bại. Việc thất bại là do đường truyền đã xác định trong đối tượng File đã tồn tại, hoặc là do thư mục không thể được tạo vì cả đường truyền không tồn tại.

- Phương thức **mkdirs()** tạo cả một thư mục và tất cả các thư mục cha của nó.

Ví dụ sau tạo thư mục /tmp/user/java/bin

```
import java.io.File;


public class CreateDir {
   public static void main(String args[]) {
      String dirname = "/tmp/user/java/bin";
      File d = new File(dirname);
      // Create directory now.
      d.mkdirs();
  }
}
```

Biên dịch và thực thi code trên sẽ tạo thư mục /tmp/user/java/bin

Ghi chú: Java tự động chăm sóc các bộ tách đường truyền trên UNIX và Windows theo qui ước tương ứng của từng hệ thống. Nếu bạn sử dụng một dấu gạch chéo (/) trên phiên bản Windows của Java, đường truyền sẽ vẫn giải quyết một cách chính xác.

Liệt kê thư mục trong Java

Bạn có thể sử dụng phương thức **list()** được cung cấp bởi đối tượng File để liệt kê tất cả các file và thư mục có sẵn trong một thư mục như sau:

```
import java.io.File;


public class ReadDir {
   public static void main(String[] args) {


      File file = null;
      String[] paths;


      try{
         // create new file object
         file = new File("/tmp");


         // array of files and directory
         paths = file.list();


         // for each name in the path array
         for(String path:paths)
         {
            // prints filename and directory name
            System.out.println(path);
         }
      }catch(Exception e){
         // if any error occurs
         e.printStackTrace();
      }
   }
}
```

Nó sẽ cho kết quả sau dựa trên các thư mục và file có sẵn trong thư mục **/tmp** của bạn:

```
test1.txt
test2.txt
ReadDir.java
ReadDir.class
```