DÒNG DỮ LIỆU VÀO/RA

NỘI DUNG TRONG CHƯƠNG

- *Dòng dữ liệu*
- *Vào ra với dữ liệu nhị phân*
- *Vào ra với dữ liệu văn bản*
- *Làm việc với dữ liệu đối tượng*

Vào/ra là những thao tác quan trọng trong chương trình. Java hỗ trợ mãnh mẽ việc này bằng cách cung cấp gói java.io trong đó có chứa các lớp để làm việc với các nguồn dữ liệu khác nhau. Từ đó, lập trình viên dễ dàng tạo ra các chương trình để làm việc tập tin, và các nguồn dữu liệu khác như bàn phím, màn hình, máy in, dữ liệu trên mạng...

DÒNG DỮ LIỆU

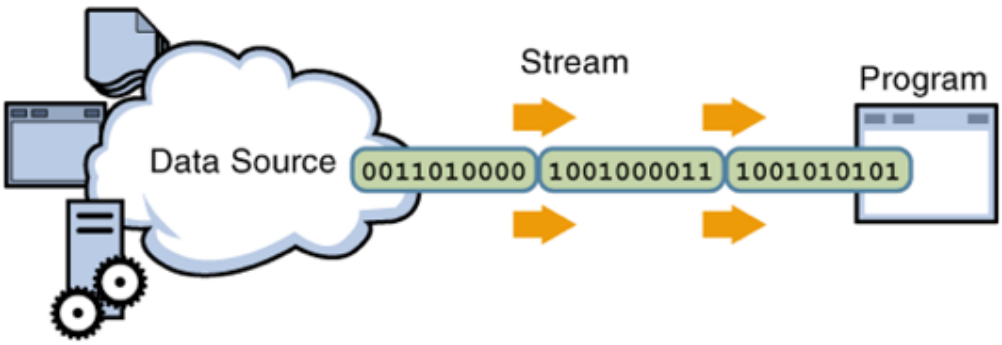
Dòng dữ liệu là gì?

Việc vào ra trong Java dựa trên dòng (stream)

- Một đối tượng mà từ đó chúng ta có thể **ĐỌC** được gọi là dòng đầu vào (input stream)
- Một đối tượng mà từ đó chúng ta có thể **GHI** được gọi là dòng đầu ra (output stream)

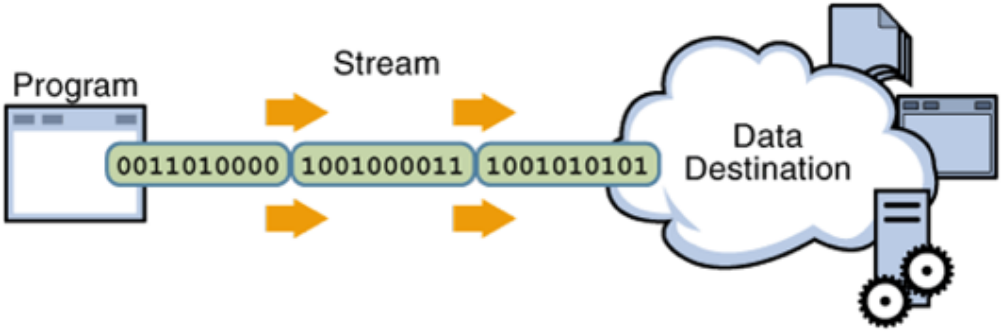
Dòng thể hiện luồng dữ liệu, hoặc một kênh giao tiếp với thành phần ghi (writer) ở một đầu và thành phần đọc (reader) ở đầu kia.

Một chương trình có thể đọc thông tin từ nhiều nguồn như từ một hệ thống tập tin, một máy chủ, hoặc một chương trình khác cần một dòng dữ liệu đầu vào:



Hình 7-1 Dòng dữ liệu vào

Ngược lại, một chương trình phát một dòng dữ liệu ra sẽ tạo thành một nguồn dữ liệu để lưu trữ vào hệ thống tập tin, máy chủ lưu trữ hoặc một ứng dụng khác:



Hình 7-2 Dòng dữ liệu ra

Dòng ký tự và dòng nhị phân

Có hai loại dữ liệu sử dụng phổ biến khi vào ra: dữ liệu **nhị phân** (máy tính quy định và đọc được) và **văn bản** (có thể đọc được bởi con người).

Dữ liệu nhị phân là các dữ liệu được mã hóa thành các bit nhị phân, nó có thể biểu diễn một số, ký tự, cấu trúc dữ liệu hoặc các dữ liệu đa phương tiện như hình ảnh, âm thanh. Về cơ bản, dữ liệu nhị phân có thể biểu diễn mọi loại dữ liệu trong máy tính. Dữ liệu con người có thể đọc được ở dạng ký tự. Khi bạn đọc một số chẳng hạn như 3,141592654, bạn đang đọc một chuỗi các ký tự và diễn giải chúng thành một số. Số trên cũng có thể biểu diễn dưới dạng nhị phân mà con người không thể (hoặc rất khó khăn) đọc được. Để làm việc với hai loại biểu diễn dữ liệu trên, Java có hai loại dòng (streams) tương ứng: dòng ký tự và dòng nhị phân (luồng đối tượng có thể coi là một trường hợp riêng của dòng nhị phân). Có một số lớp được xây dựng sẵn trong thư viện Java đại diện cho các dòng của mỗi loại.

Luồng byte có thể hữu ích cho giao tiếp trực tiếp giữa các máy tính và hữu ích trong việc lưu trữ hầu hết các định dạng dữ liệu. Tuy nhiên, dữ liệu nhị phân thường rất khó hay nói cách khác là không thể đọc bởi con người. Khi đối mặt với một chuỗi dài các số 0 và 1, người đọc sẽ không biết thông tin đó đại diện cho thông tin gì và thông tin đó được mã hóa như thế nào nếu không biết cấu trúc của nó. Xu hướng hiện tại dường như hướng tới việc tăng cường sử dụng dữ liệu ký tự, được thể hiện theo cách làm cho ý nghĩa của nó càng rõ ràng càng tốt.

Ví dụ số 127:

Bảng 7-1 Bảng so sánh dữ liệu văn bản và dữ liệu nhị phân

Dữ liệu văn bản: '1', '2', '7'	Dữ liệu nhị phân: 127
ASCII (Hex): 31, 32, 37 Unicode (Hex): 0031, 0032, 0037 UTF8 (Hex): 31, 32, 37	1 byte (byte): 01111110 2 bytes (short): 00000000 01111110 4 bytes (int): 00000000 00000000 00000000 01111110

JAVA.IO PACKAGE

Java định nghĩa các lớp để thực hiện vào/ra trong gói **java.io**. Trong gói **java.io** có đầy đủ các lớp để tạo ra các dòng vào ra.

Một đối tượng để đọc dữ liệu nhị phân sẽ là một lớp con của lớp trừu tượng **InputStream**, trong khi mội đối tượng đối tượng xuất dữ liệu ra dòng nhị phân thuộc một trong các lớp con của lớp trừu tượng **OutputStream**. Để đọc và ghi dữ liệu ký tự mà con người có thể đọc được, các lớp chính là các lớp trừu tượng **Reader** và **Writer**. Tất cả các lớp dòng ký tự đều là lớp con của một trong những lớp này. Nếu một

số được ghi vào dòng Writer, máy tính phải dịch nó thành một chuỗi ký tự có thể đọc được của con người đại diện cho số đó. Các ký tự được lưu trữ trong máy tính dưới dạng giá trị Unicode 16 bit. Đối với những người sử dụng Bảng chữ cái tiếng Anh, dữ liệu ký tự thường được lưu trữ trong các tệp bằng mã ASCII, chỉ sử dụng 8 bit cho mỗi ký tự. Các lớp Reader và Writer đảm nhiệm việc dịch này và cũng có thể xử lý các bảng chữ cái và ký tự của nhiều quốc gia như tiếng Trung).
Khi sử dụng các lớp trong **gói java.io** để đọc và ghi dữ liệu, các bước chính cần thực hiện như sau:

- **import java.io.*;**
- **Mở một dòng**
- **Sử dụng stream đó để đọc, ghi (có thể thực hiện cả 2 đồng thời)**
- **Đóng dòng đó lại**

Các lớp **Input/Output Stream** và **Reader/Writer** cung cấp các phương thức giúp làm việc với các dòng. Tùy thuộc vào dữ liệu mốn thao tác mà sẽ sử dụng các lớp khác nhau, trong nội dung chương này sẽ đề cập tới các lớp sau:

Các **Input Stream** và **Output Stream** dùng để đọc và ghi dữ liệu nhị phân:

- FileInputStream và FileOutputStream
- DataInputStream và DataOutputStream
- BufferedInputStream và BufferedOutputStream

Các **Reader** và **Writer** dùng để đọc và ghi dữ liệu văn bản:

- InputStreamReader và OutputStreamWriter
- InputStreamReader và OutputStreamWriter
- FileReader và FileWriter

DÒNG DỮ LIỆU NHỊ PHÂN

FileInputStream và FileOutputStream

Đọc và ghi tập tin là 2 thao tác được sử dụng nhiều trong khi lập trình ứng dụng. Hai thao tác đó có thể được thực hiện một cách đơn giản bằng cách sử dụng FileInputStream và FileOutputStream:

- **FileOutputStream** để ghi dữ liệu từng byte ra tập tin

Bảng 7-2 Các phương thức đọc ghi của DataInputStream và DataOutputStream

Phương thức ghi	Phương thức đọc	Mô tả
write()	read()	Đọc và ghi 1 hoặc nhiều byte

Ví dụ, để sao chép một tập tin ảnh, có thể đọc từng byte của tập tin gốc và copy sang tập tin mới. Hàm tạo của FileInputStream là đường dẫn của tập tin:

```
try {
    FileInputStream  fin = new FileInputStream("SourcePics.jpg");
    FileOutputStream fout = new FileOutputStream("DestPics.jpg");
    int c;
    while ((c = fin.read()) != -1)
        fout.write(c);
    if (fin != null)
        fin.close();
    if (fout != null)
        fout.close();
} catch (IOException e) {
   System.out.println("Có lỗi xảy ra trong quá trình đọc ghi file");
}
```

Lưu ý khi làm việc với các Stream, các phương thức khởi tạo cũng như các phương thức đọc ghi thường bắn ra các ngoại lệ thuộc loại "checked", vì vậy luôn cần try catch khi viết mã. Để bắt được từng lỗi trong các trường hợp cụ thể, cần quan tâm tới các loại ngoại lệ sau:

- **FileNotFoundException:** Ngoại lệ bắn ra bởi các hàm tạo khi không tạo được dòng tới tập tin
- **IOException:** Ngoại lệ bắn ra trong quá trình đọc ghi dữ liệu từ dòng.

Tuy nhiên, có thể thấy đối tượng được tạo ra từ FileInputStream và FileOutputStream chỉ cung cấp các hàm đơn giản cho phép đọc 1 hoặc nhiều byte. Điều này hữu ích đối với dữ liệu thô gồm các byte, khi đó các byte sẽ được đọc tuần tự. Tuy nhiên đối với những tập tin có cấu trúc, việc thao tác với dữu liệu sử dụng các lớp này sẽ gặp khó khăn.

DataInputStream và DataOutputStream

Lớp **DataInputStream** cung cấp các phương thức đọc tương ứng với các phương thức ghi trong **DataOutputStream** để đọc và ghi dữ liệu cơ sở.

Các phương thức của lớp này thể hiện trong bảng dưới đây:

Bảng 7-3 Các phương thức đọc ghi của DataInputStream và DataOutputStream

Phương thức ghi	Phương thứ đọc	Mô tả
writeByte(), writeShort(), writeInt(), writeLong()	readByte(), readShort(), readInt(), readLong()	Đọc và ghi số byte, short, int, long
writeFloat(), writeDouble()	readFloat(), readDouble()	Đọc và ghi số float, double
writeChar()	readChar()	Đọc và ghi 1 ký tự char
writeBoolean()	readBoolean()	Đọc và ghi một dữ liệu boolean

writeUTF()	readBytes(), readUTF()	Đọc và ghi một chuỗi ký tự
writeChars()	-	Ghi nhiều ký tự
writeBytes()	-	Ghi nhiều byte

Hàm tạo của **DataOutputStream** yêu cầu truyền vào các **OutputStream**. Ví dụ, muốn ghi dữ liệu là một số nguyên vào tập tin:

```
1  DataOutputStream dos = new DataOutputStream(new FileOutputStream("Int.txt"));
2  int i = 100;
3  dos.writeInt(i);
4  dos.close();
```

Viết code để đọc dữ liệu trên từ tập tin:

```
1  FileInputStream fis = new FileInputStream("Int.txt");
2  DataInputStream dis = new DataInputStream(fis);
3  int i = dis.readInt();
4  System.out.println(i);
5  dis.close();
```

Kết quả in ra màn hình:

```
100
```

BufferedInputStream và BufferedOutputStream

Các dòng "Buffered" bổ sung thêm cơ chế làm việc với "bộ đệm" cho phép việc vào ra hiệu quả hơn. Khi một dòng "Buffered" được tạo, một bộ đệm đồng thời cũng được tạo ra. Bộ đệm này giúp trung chuyển dữ liệu khiến cho việc thao tác trên dữ liệu được xử lý với tốc độ nhanh hơn.

Bảng 7-4 Các phương thức đọc ghi của DataInputStream và DataOutputStream

Phương thức ghi	Phương thức đọc	Mô tả
write()	read()	Đọc và ghi 1 hoặc nhiều byte

Tương tự như DataInputStream và DataOutputStream các hàm tạo của BufferedOutputStream và BuffredInputStream là các Input/Ouput Stream. Ví dụ:

```
1  BufferedInputStream bis = new BufferedInputStream(new FileInputStream("a.pdf"));
2  BufferedOutputStream  bos = new BufferedOutputStream(new FileOutputStream("b.pdf"));
```

DÒNG DỮ LIỆU VĂN BẢN

InputStreamReader và OutputStreamWriter

InputStreamReader và OutputStreamWriter hoạt động tương tự như các InputStream khác nhưng nó giúp chuyển dữ liệu từ nhị phân sang dữ liệu dạng ký tự để thuận tiện làm việc với các văn bản. Các ký tự khi chuyển có thể được mã hóa theo một bảng mã nào đó, phổ biến nhất hiện nay vẫn là các bảng mã hóa ký tự UTF.

Các lớp OutputStreamWriter và InputStreamReader cung cấp các phương thức sau:

Bảng 7-5 Các phương thức đọc ghi của OutputStreamWriter và InputStreamReader

Phương thức ghi	Phương thức đọc	Mô tả
write()	read()	Đọc và ghi 1 hoặc nhiều ký tự vào dòng

Hàm tạo của cả 2 lớp này sẽ yêu cầu các InputStream và OutputStream tương ứng.

Ví dụ sau thực hiện việc viết mã lệnh ghi nội dung văn bản (mã hóa ký tự UTF8) vào một tập tin và sau đó đọc lại tập tin này, hiển thị nội dung ra màn hình:

```
try {
    FileOutputStream fos = new FileOutputStream("TestUTF8.txt");
    OutputStreamWriter out = new OutputStreamWriter(fos, "UTF8");
    out.write("Hello, Việt Nam");
    out.close();
    FileInputStream fis = new FileInputStream("TestUTF8.txt");
    Reader in = new InputStreamReader(fis, "UTF8");
    int charRead;
    while ((charRead = in.read()) != -1) {
        System.out.printf("%c(%02X) ", (char) charRead, charRead);
    }
    in.close();
} catch (IOException ex) {
    ex.printStackTrace();
}
```

Kết quả hiển thị ra màn hình:

```
H(48) e(65) l(6C) l(6C) o(6F) ,(2C)  (20) V(56) i(69) ệ(1EC7) t(74)  (20) N(4E) a(61) m
(6D)
```

FileReader và FileWriter

FileReader và FileWriter là những lớp giúp làm việc với tập tin văn bản được dễ dàng hơn, phương thức khởi tạo của FileReader và FileWriter cho phép truyền thẳng tên của tập tin để tạo ra dòng văn bản

FileReader và FileWriter cũng cung cấp các phương thức write() và read() giúp đọc và ghi các ký tự vào tập tin tương tự như InputStreamReader và OutputStreamWriter

Bảng 7-6 Các phương thức đọc ghi của FileWriter và FileReader

Các phương thức ghi	Các phương thức đọc	Mô tả

của FileWriter	của FileReader	
write()	read()	Đọc và ghi 1 hoặc nhiều ký tự vào dòng

Ví dụ ghi dữ liệu văn bản vào file text.txt bằng cách sử dụng FileWriter

```
1  FileWriter out = new FileWriter("text.txt");
2  out.write("FileReader");
3  out.flush();
4  out.close();
```

Đọc dữ liệu văn bản từ tập tin văn bản nói trên bằng cách sử dụng FileReader

```
1  FileReader in = new FileReader("text.txt");
2  int data = in.read();
3  while (data != -1) {
4    System.out.printf("%c", data);
5    data = in.read();
6  }
7  in.close();
```

BufferedReader và BufferedWriter

Sử dụng các Reader và Writer đã đề cập ở phần trên có thể thao tác với dòng dữ liệu văn bản một cách dễ dàng, tuy nhiên, để cải thiện hiệu năng trong việc đọc và ghi, có thể sử dụng đến hai lớp BufferedReader và BufferedWriter. Hai lớp này cho phép đọc dữ liệu văn bản có sử dụng buffer để việc vào ra được hiệu quả hơn. Ngoài ra BufferedReader cũng bổ sung thêm phương thức readLine() để đọc cả dòng (line) trong dữ liệu văn bản nguồn.

Bảng 7-7 Các phương thức đọc ghi của BufferedWriter và BufferedReader

Các phương thức ghi của BufferedWriter	Các phương thức đọc của FileReader	Mô tả
write()	read()	Đọc và ghi 1 hoặc nhiều ký tự
-	readLine()	Đọc cả một dòng (line) ký tự từ dữ liệu

Ví dụ ghi một file văn bản với nội dung "BufferedWriter"

```
1  FileWriter out = new FileWriter("text.txt");
2  BufferedWriter bw = new BufferedWriter(out);
3  bw.write("BufferedWriter");
4  bw.flush();
5  bw.close();
```

Đọc tập tin văn bản nói trên bằng cách sử dụng BufferedReader. Tốc độ đọc sẽ không có khác biệt với nội dung của tập tin nhỏ như trên nhưng sẽ rất khác biệt khi dữ liệu gốc lớn.

```
1  FileReader in = new FileReader("text.txt");
2  BufferedReader br = new BufferedReader(in);
```

```
String str = br.readLine();
while (str != null) {
   System.out.printf(str);
   str = br.readLine();
}
br.close();
```

ĐỌC VÀ GHI ĐỐI TƯỢNG

Trong Java, để đọc và ghi đối tượng từ một file, ta có thể sử dụng đối tượng ObjectOutputStream và ObjectInputStream. Để sử dụng các đối tượng này, ta cần truyền vào một đối tượng của lớp FileOutputStream hoặc FileInputStream để thực hiện ghi hoặc đọc file. Tuy nhiên đối tượng muốn ghi vào file phải được triển khai từ giao diện Serializable. Ví dụ để ghi đối tượng sinh viên vào file:

Tạo lớp Student triển khai từ giao diện Serializable (Chỉ khi triển khai từ giao diện này thì đối tượng tạo ra từ lớp Student mới có thể được ghi vào tập tin)

```
class Student implements Serializable {
   private int studentID;
   private String studentName;
   public Student(int studentID, String studentName) {
       this.studentID = studentID;
       this.studentName = studentName;
   }
   public int getStudentID() {
       return studentID;
   }
   public void setStudentID(int studentID) {
       this.studentID = studentID;
   }
   public String getStudentName() {
       return studentName;
   }
   public void setStudentName(String studentName) {
       this.studentName = studentName;
   }
}
```

Viết mã lệnh ghi đối tượng vào file:

```
public static void main(String[] args) {
   ObjectInputStream in = null;
   ObjectOutputStream out = null;
   try {
       out = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream("o
bject.dat")));
       // Create an array of 10 SerializedObjects with ascending numbers
       Student[] list = new Student[3];
       list[0] =  new Student(1, "Peter");
       list[1] =  new Student(2, "David");
       list[2] =  new Student(3, "Tommy");
       // Write the entire array in one go.
```

```
12          out.writeObject(list);
13          out.close();
14      } catch (Exception ex) {
15          ex.printStackTrace();
16      }
17  }
```

Cần lưu ý, nếu muốn thuộc tính không ghi ra tập tin, sử dụng từ khóa **transient** trước mỗi biến. Thuộc tính này sẽ khiến cho Java bỏ qua biến này khi "**Serialization**" một đối tượng.

LỚP FILE

Các Tập tin và Thư mục trong máy tính được quản lý thông qua lớp File. Để quản lý file hay thư mục cần tạo ra đối tượng File, ví dụ:

```
File file = new File("myfile.txt");
```

Lớp file cung cấp một số phương thức

- Boolean **exist**(): Kiểm tra xem có tồn tại hay không
- Boolean **canWrite**(): Kiểm tra xem có quyền ghi hay không
- Boolean **canRead**(): Kiểm tra xem có quyền đọc hay không
- Boolean **isFile**(): Kiểm tra xem có phải là Tập tin hay không
- String **getName**(): Lấy ra tên File
- String **getAbsolutePath**(): Lấy ra đường dẫn tuyệt đối tới File.

THỰC HÀNH

Thực hành copy file

Bài thực hành này sẽ viết mã copy file sử dụng các cách sau:

- Cách 1: Không sử dụng Buffering
- Cách 2: Sử dụng BufferedInputStream
- Cách 3: Sử dụng Buffer với kích thước 4K
- Cách 4: Sử dụng Buffer với kích thước khác nhau

Cách 1: Không sử dụng Buffering

Tạo và viết mã cho lớp FileCopyNoBuffer

```
1  import java.io.*;
2  public class FileCopyNoBuffer {
3      public static void main(String[] args) {
4          File fileIn;
```

```
        FileInputStream in = null;
        FileOutputStream out = null;
        long startTime, elapsedTime;  // for speed benchmarking
        try {
            fileIn = new File("a.jpg");
            System.out.println("File size is " + fileIn.length() + " bytes");
            in = new FileInputStream(fileIn);
            out = new FileOutputStream("b.jpg");
            startTime = System.nanoTime();
            int byteRead;
            // Read a unsigned byte (0-255) and padded to 32-bit int
            while ((byteRead = in.read()) != -1) {
                // Write the least significant byte, drop the upper 3 bytes
                out.write(byteRead);
            }
            elapsedTime = System.nanoTime() - startTime;
            System.out.println("Elapsed Time is " + (elapsedTime / 1000000.0) + " m
sec");
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {    // always close the streams
            try {
                if (in != null) in.close();
                if (out != null) out.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Hiển thị kích thước và thời gian để copy file:

```
File size is 2906316 bytes
Elapsed Time is 10061.436241 msec
```

Cách 2: Sử dụng Buffered Streams

Tạo và viết mã cho lớp **FileCopyBuffered**

```
import java.io.*;
public class FileCopyBuffered {
   public static void main(String[] args) {
        File fileIn;
        BufferedInputStream in = null;
        BufferedOutputStream out = null;
        long startTime, elapsedTime;  // for speed benchmarking
        try {
            fileIn = new File("a.jpg");
            System.out.println("File size is " + fileIn.length() + " bytes");
            in  = new BufferedInputStream(new FileInputStream(fileIn));
            out = new BufferedOutputStream(new FileOutputStream("b.jpg"));
            startTime = System.nanoTime();
```

```
            int byteRead;
            while ((byteRead = in.read()) != -1) {
                out.write(byteRead);
            }
            elapsedTime = System.nanoTime() - startTime;
            System.out.println("Elapsed Time is " + (elapsedTime / 1000000.0) + " mse
c");
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {            // always close the streams
            try {
                if (in != null) in.close();
                if (out != null) out.close();
            } catch (IOException ex) { ex.printStackTrace(); }
        }
    }
}
```

Hiển thị kích thước và thời gian để copy file:

```
File size is 2906316 bytes
Elapsed Time is 90.529071 msec
```

Cách 3: Sử dụng Buffer 4K

Tạo và viết mã cho lớp **FileCopyUserBuffer**

```
import java.io.*;
public class FileCopyUserBuffer {
   public static void main(String[] args) {
        File fileIn;
        FileInputStream in = null;
        FileOutputStream out = null;
        long startTime, elapsedTime;  // for speed benchmarking
        try {
            fileIn = new File("a.jpg");
            System.out.println("File size is " + fileIn.length() + " bytes");
            in = new FileInputStream(fileIn);
            out = new FileOutputStream("b.jpg");
            startTime = System.nanoTime();
            byte[] byteBuf = new byte[4096];   // 4K buffer
            int numBytesRead;
            while ((numBytesRead = in.read(byteBuf)) != -1) {
                out.write(byteBuf, 0, numBytesRead);
            }
            elapsedTime = System.nanoTime() - startTime;
            System.out.println("Elapsed Time is " + (elapsedTime / 1000000.0) + " m
sec");
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {            // always close the streams
            try {
                if (in != null) in.close();
```

```
                if (out != null) out.close();
            } catch (IOException ex) { ex.printStackTrace(); }
        }
    }
}
```

Hiển thị kích thước và thời gian để copy file:

```
File size is 2906316 bytes
Elapsed Time is 4.243062 msec
```

Hãy viết lại chương trình vào class **FileCopyUserBuffer**, trong đó sử dụng các kích thước buffer khác nhau (Cách 4):

```
try {
    int[] bufSizeKB = {1, 2, 4, 8, 16, 32, 64, 256, 1024};  // in KB
    fileIn = new File("a.jpg");
    System.out.println("File size is " + fileIn.length() + " bytes");
    for (int i = 0; i < bufSizeKB.length; i++) {
        in = new FileInputStream(fileIn);
        out = new FileOutputStream("b.jpg");
        startTime = System.nanoTime();
        int bufSize = bufSizeKB[i] * 1024;
        byte[] byteBuf = new byte[bufSize]; // buffer
        int numBytesRead;
        while ((numBytesRead = in.read(byteBuf)) != -1) {
            out.write(byteBuf, 0, numBytesRead);
        }
        elapsedTime = System.nanoTime() - startTime;
        System.out.println(bufSizeKB[i] + "KB Elapsed Time is " + (elapsedTime / 1000
000.0) + " msec");
    }
} catch (IOException ex) {
    ex.printStackTrace();
} finally {            // always close the streams
    try {
        if (in != null) in.close();
        if (out != null) out.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Hãy cho biết kết quả chạy chương trình và cho biết kích thước Buffer bao nhiêu thì thời gian copy file là nhanh nhất.

Duyệt nội dung thư mục

Tạo lớp ListDirectoryRecusive. Viết phương thức listRecursive() method

```
public static void listRecursive(File dir) {
    if (dir.isDirectory()) {
```

```
        File[] items = dir.listFiles();
        if (items != null) {
            for (File item : items) {
                System.out.println(item.getAbsoluteFile());
                if (item.isDirectory())
                    listRecursive(item);
            }
        }
    }
}
```

Chương trình chính để liệt kê nội dung của một thư mục

```
public static void main(String[] args) throws IOException {
    File dir = new File("C:\\Windows");
    listRecursive(dir);
}
```

Thực hiện chương trình

```
C:\Windows\actsetup.log
C:\Windows\addins\
C:\Windows\addins\FXSEXT.ecf
C:\Windows\AppCompat
...
```

Đọc dữ liệu bằng Scanner

Tạo ra file dữ liệu **test.txt** có nội dung như sau:

```
1 fish 2 fish red fish blue fish
```

Dữ liệu như trên có thể được đọc bằng cách sử dụng một Scanner như sau:

```
File text = new File("test.txt");
Scanner scanner = new Scanner(text);
while (scanner.hasNextLine()) {
    String line = scanner.nextLine();
    System.out.println(line);
}
```

Kết quả khi chạy:

```
1 fish 2 fish red fish blue fish
```

Đối tượng Scanner cung cấp một phương thức có tên useDelimiter giúp phân tách dữ liệu nguồn một cách dễ dàng, ví dụ phân tách dựa theo dấu cách (space)

```
File text = new File("test.txt");
Scanner scanner = new Scanner(text).useDelimiter("\\s"); //space;;

```

```
System.out.println(scanner.nextInt());
System.out.println(scanner.next());
System.out.println(scanner.nextInt());
System.out.println(scanner.next());
scanner.close();
```

Kết quả khi chạy

```
1
fish
2
fish
```

Đọc dữ liệu trên mạng

Đọc thông tin Covid từ API và lưu vào 1 file có tên Covid Summary.txt

```
String url = "https://api.covid19api.com/summary";
String fileName = "Covid Summary.txt";
BufferedInputStream in = new BufferedInputStream(new URL(url).openStream());
FileOutputStream fileOutputStream = new FileOutputStream(fileName);
byte dataBuffer[] = new byte[1024];
int bytesRead;
while ((bytesRead = in.read(dataBuffer, 0, 1024)) != -1) {
    fileOutputStream.write(dataBuffer, 0, bytesRead);
}
```

Dữ liệu lưu về sẽ như sau:

```
{"Message":"Caching in progress","Global":{"NewConfirmed":0,"TotalConfirmed":0,"NewDeaths":0,"TotalDeaths":0,"NewRecovered":0,"TotalRecovered":0,"Date":"0001-01-01T00:00:00Z"},"Date":"0001-01-01T00:00:00Z"}
```

Một ví dụ khác, đọc file văn bản lưu trữ trên mạng:

```
String url = "https://www.w3schools.com/js/json_demo.txt";
BufferedReader br = new BufferedReader(new InputStreamReader(new URL(url).openStream()));
String str = br.readLine();
while (str != null) {
    System.out.printf(str);
    str = br.readLine();
}
br.close();
```

Dữ liệu in ra màn hình:

```
{    "name":"John",    "age":31,    "pets":[        { "animal":"dog", "name":"Fido" },
{ "animal":"cat", "name":"Felix" },        { "animal":"hamster", "name":"Lightning" }
]}
```

CÂU HỎI ÔN TẬP LÝ THUYẾT

1. Các lớp để thực hiện vào/ra được khai báo trong gói:

- A. java.io
- B. java.stream
- C. java.file
- D. java.in và java.out

2. Một InputStream được sử dụng để:

- A. Đọc dữ liệu nhị phân
- B. Ghi dữ liệu nhị phân
- C. Đọc dữ liệu văn bản
- D. Ghi dữ liệu văn bản

3. BufferReader được sử dụng để:

- A. Đọc dữ liệu nhị phân
- B. Ghi dữ liệu nhị phân
- C. Đọc dữ liệu văn bản
- D. Ghi dữ liệu văn bản

4. FileOutputStream được dùng để:

- A. Đọc dữ liệu trong dòng (stream)
- B. Đọc dữ liệu nhị phân
- C. Ghi dữ liệu văn bản
- D. Ghi dữ liệu ra file

5. Để ghi một số double vào file, nên dùng phương thức của lớp gì:

- A. DataOutputStream
- B. BufferedInputStream
- C. BufferDataStream
- D. NumberStream

6. Muốn nâng cao hiệu năng, tốc độ vào/ra:

- A. Nên sử dụng kiểu dữ liệu nhị phân
- B. Nên sử dụng các lớp dùng bộ đệm (Buffered class)
- C. Nên dùng dữ liệu dạng nguyên thủy
- D. Nên tạo ra nhiều tiến trình để vào ra

7. Một đối tượng muốn ghi ra file thì:

- A. Lớp tạo ra đối tượng phải kế thừa từ lớp Serializable
- B. Lớp tạo ra đối tượng phải kế thừa từ lớp Writeable
- C. Lớp tạo ra đối tượng phải triển khai từ giao diện Serializable
- D. Lớp tạo ra đối tượng phải triển khai từ giao diện Writeable

8. Lớp File có thể được dùng để:

- A. Đọc dữ liệu trong file
- B. Lấy thông tin về File và thư mục
- C. Ghi dữ liệu ra File
- D. Quét và tìm kiếm dữ liệu từ File

9. Để vào ra dữ liệu trên mạng:

- A. Bắt buộc phải dùng các lớp trong java.nio
- B. Phải dùng đối tượng Scanner
- C. Cần phải dùng tới lớp OnlineStream
- D. Cần phải mở một InputStream liên kết đến nguồn dữ liệu trên mạng

10. Hãy cho biết kết quả thực hiện đoạn mã sau:

```
File someFile = new File("test.txt");
int aCount = 0;
try (FileReader fr = new FileReader(someFile);
     BufferedReader br = new BufferedReader(fr)) {
    int ch;
    while ((ch = br.read()) != -1) {
        if (ch == 'i') {
            aCount++;
        }
    }
    System.out.println(aCount);
}
```

- A. In ra số chữ i trong file test
- B. In ra chữ i
- C. Copy file test.txt vào bộ nhớ buffe
- D. Ghi chữ i vào file test.txt

11. Hãy cho biết kết quả thực hiện đoạn mã sau:

```
try {
    Scanner scanner = new Scanner(new File("test.txt"));
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        System.out.println(line);
    }
```

```
7  } catch (FileNotFoundException e) {
8      e.printStackTrace();
9  }
```

- A. Đọc file test.txt và in ra màn hình
- B. Đếm số dòng trong file test.txt
- C. Kiểm tra sự tồn tại của file test.txt
- D. Quét và đếm số ký tự trong file test.txt

BÀI TẬP TỰ THỰC HÀNH

Ghi chuỗi vào file bằng OutputStream

Viết chương trình Java để ghi một chuỗi dữ liệu vào một file bằng cách sử dụng lớp OutputStream trong package java.io.

Sử dụng ObjectInputStream và ObjectOutputStream

Tạo một đối tượng User với các thuộc tính: id (kiểu int), name (kiểu String), age (kiểu int).

Lưu trữ thông tin của User vào file user.dat bằng cách sử dụng ObjectOutputStream. Đọc thông tin User từ file user.dat bằng cách sử dụng ObjectInputStream và hiển thị thông tin lên màn hình console.

Đếm từ trong file văn bản

Viết chương trình Java để đọc nội dung của một file text và đếm số lượng từ trong file đó bằng cách sử dụng lớp Reader và Writer.

Tạo file và thư mục

Viết chương trình Java để tạo một thư mục mới và tạo một tệp mới trong thư mục đó. Tên thư mục đặt là Thư mục của tôi, còn tên tệp đặt theo ngày giờ hiện tại.

TÀI LIỆU THAM KHẢO

[1] Core Java: Fundamentals (2021) , Cay Horstmann (Oracle Press Java)

[2] Head First Java: A Brain-Friendly Guide (2022), Kathy Sierra, O'Reilly Media

[3] Java OOP Done Right: Create object oriented code you can be proud of with modern Java Paperback (2019), Mr Alan Mellor, Mellor Books

[4] Murach's Java Programming (5th Edition) (2017), Joe Murach, Mike Murach & Associates

[5]. Java for Absolute Beginners Learn to Program the Fundamentals the Java 9+ Way

[6]. Modern Java Recipes: Simple Solutions to Difficult Problems in Java 8 and 9 (2017), by Ken Kousen, O'Reilly Media

[7] Effective Java (2018), Joshua Bloch, Addison-Wesley Professional