viettel
software services

CHƯƠNG TRÌNH ĐÀO TẠO JAVA NÂNG CAO

Java IO

Đơn vị tổ chức: Phòng đào tạo và phát triển nguồn nhân lực

viettel
software services

NỘI DUNG HỌC PHẦN

BÀI 1

Java IO

BÀI 2

Java NIO

viettel
software services

BÀI 1

Java IO

viettel
software services

MỤC LỤC

viettel
software services

01

Các khai niệm trong Java IO

viettel
software services

Java IO là gì?

- Java IO hay còn gọi là Java Input và Output. Gói Java.io cung cấp các lớp đầu vào (input) và đầu ra (output) của hệ thống thông qua các luồng dữ liệu.

viettel
software services

Khái niệm Stream

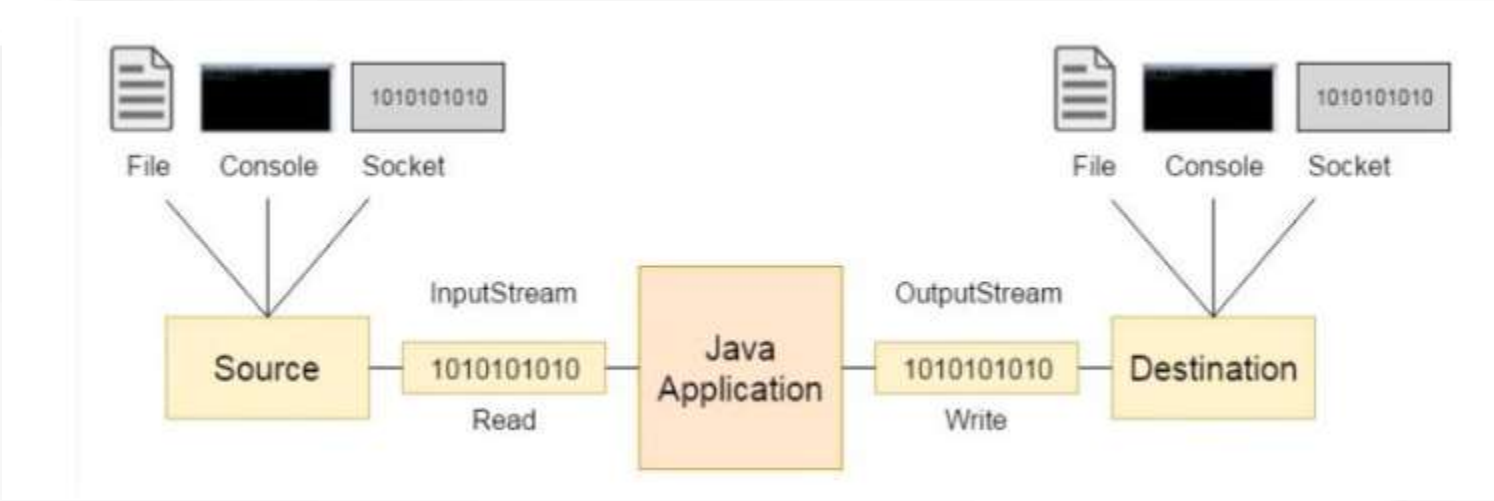
- Stream là một dòng liên tục, có thứ tự các bytes dữ liệu chảy giữa chương trình và thiết bị ngoại vi. Nó là khái niệm trừu tượng giúp giảm bớt các thao tác vào ra phức tạp đối với người lập trình. Nó cho phép kết nối nhiều thiết bị ngoại vi khác nhau với chương trình.

Các loại Stream trong Java

- Ứng dụng Java sử dụng **InputStream** để đọc dữ liệu từ một nguồn (nó có thể là một tập tin, một mảng, thiết bị ngoại vi hoặc socket)
- Ứng dụng Java sử dụng **OutputStream** để ghi dữ liệu đến đích (nó có thể là một tập tin, một mảng, thiết bị ngoại vi hoặc socket).

viettel
software services

Công việc của Java OutputStream và InputStream được mô tả bằng hình dưới đây.



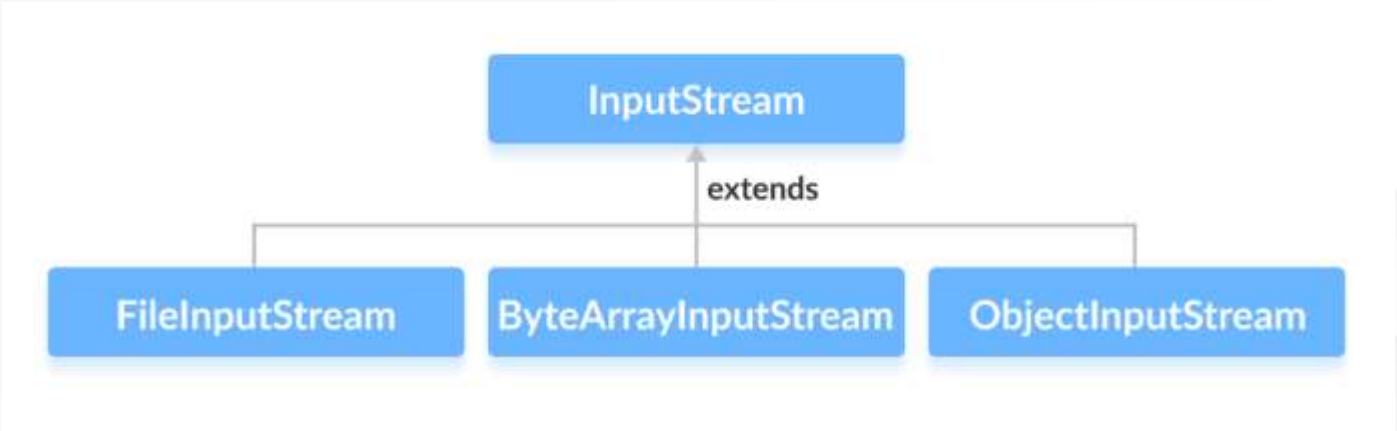
02

Input Stream

viettel
software services

Input Stream

- Class **InputStream** của gói **java.io** là một abstract class đại diện cho một dòng đầu vào của byte.
- Vì **InputStream** là một lớp trừu tượng nên bản thân nó không hữu ích. Tuy nhiên, các lớp con của nó có thể được sử dụng để đọc dữ liệu.
- Để sử dụng chức năng của **InputStream**, chúng ta có thể sử dụng các lớp con của nó. Một số trong số đó là: **FileInputStream, ByteArrayInputStream, ObjectInputStream**



viettel
software services

Các phương thức của InputStream

- Các lớp InputStream cung cấp phương pháp khác nhau được thực hiện bởi lớp con của nó. Dưới đây là một số phương pháp thường được sử dụng:
- read() – đọc một byte dữ liệu từ luồng đầu vào
- read(byte[] array) – đọc các byte từ luồng và lưu trữ trong mảng được chỉ định
- available() – trả về số byte có sẵn trong luồng đầu vào
- mark() – đánh dấu vị trí trong luồng đầu vào mà dữ liệu đã được đọc
- reset() – trả lại điều khiển đến điểm trong luồng nơi đánh dấu được đặt
- markSupported()- kiểm tra xem phương thức mark()và reset()có được hỗ trợ trong luồng không
- skips() – bỏ qua và loại bỏ số byte được chỉ định khỏi luồng đầu vào
- close() – đóng luồng đầu vào

viettel
software services

Ví dụ: InputStream Sử dụng FileInputStream

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

        byte[] array = new byte[100];

        try {
            InputStream input = new FileInputStream( name: "input.txt");

            System.out.println("Available bytes in the file: " + input.available());

            // Read byte from the input stream
            input.read(array);
            System.out.println("Data read from the file: ");

            // Convert byte array into string
            String data = new String(array);
            System.out.println(data);

            // Close the input stream
            input.close();
        }
        catch (Exception e) {
            e.getStackTrace();
        }
    }
}
```

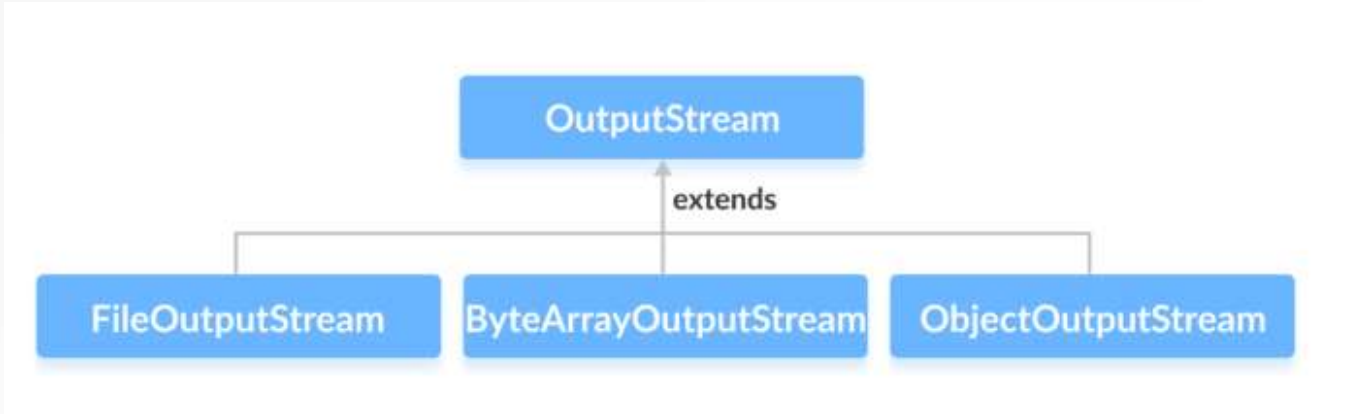
03

Output Stream

viettel
software services

Output Stream

- OutputStream là một abstract class đại diện cho một luồng dữ liệu đầu ra dạng byte có thứ tự. Hay nói cách khác bạn có thể truyền dữ liệu từ OutputStream theo một chuỗi byte có thứ tự. Điều này sẽ giúp ích khi chúng ta ghi dữ liệu vào file hoặc qua network.
- Vì **OutputStream** là một lớp trừu tượng nên bản thân nó không sử dụng trực tiếp được. Tuy nhiên, các lớp con của nó có thể được sử dụng để ghi dữ liệu.
- Để sử dụng chức năng của **OutputStream**, chúng ta có thể sử dụng các lớp con của nó. Một số trong số đó là: **FileOutputStream, ByteArrayOutputStream, ObjectOutputStream**



viettel
software services

Các phương thức của Output Stream

- OutputStream là một abstract class chứa các method giúp thao tác ghi dữ liệu như:
- **void close():** Đóng output stream, giải phóng tất cả các tài nguyên đang được kết nối với luồng này.
- **void flush():** Xoá dữ liệu đang được lưu trong output stream và buộc nó ghi dữ liệu xuống điểm đích.
- **void write(byte[] b):** Ghi toàn bộ byte từ mảng byte tham số vào output stream.
- **void write(byte[], int offset, int length):** Ghi length byte từ mảng byte tham số vào output stream bắt đầu tại vị trí offset trong mảng.
- **abstract void write(int b):** Ghi byte b được chỉ định trong tham số truyền vào xuống outputstream.
- Việc ghi dữ liệu với mảng byte vào outputstream sẽ nhanh hơn so với bạn ghi từng byte đơn vào outputstream. Vì vậy cố gắng sử dụng write(byte[]) hoặc write(byte[] b, int off, int len) thay write(int b).

viettel
software services

Heap trong xử lý đa luồng

```
class Main {

    public static void main(String args[]) {
        String data = "This is a line of text inside the file.";

        try {
            OutputStream out = new FileOutputStream( name: "output.txt");

            // Converts the string into bytes
            byte[] dataBytes = data.getBytes();

            // Writes data to the output stream
            out.write(dataBytes);
            System.out.println("Data is written to the file.");

            // Closes the output stream
            out.close();
        }

        catch (Exception e) {
            e.getStackTrace();
        }
    }
}
```

viettel
software services

04

Serializalbe

viettel
software services

Serializable

- **Serializable** trong Java hay tuần tự hóa trong Java là một cơ chế giúp lưu trữ và chuyển đổi trạng thái của 1 đối tượng (Object) vào 1 byte stream sao cho byte stream này có thể chuyển đổi ngược trở lại thành một Object.
- Quá trình chuyển đổi byte stream trở thành 1 Object được gọi là **DeSerialization**.
- Để một Object có thể thực hiện **Serialization** hay gọi tắt là **Serializable**, class của Object cần phải thực hiện implements interface **java.io.Serializable**.
- **java.io.Serializable** là một Interface (giao diện) đánh dấu không có các dữ liệu và phương thức.
- Kỹ thuật của **Serialization** trong Java thường được sử dụng chủ yếu trong công nghệ như RMI, EJB, JPA và Hibernate

viettel
software services

Ví dụ với Serializable

- Tạo class Employee như sau

```
class Employee implements java.io.Serializable {
    public String name;
    public String address;
    public transient int SSN;
    public int number;

    public void mailCheck() {
        System.out.println("Mailing a check to " + name + " " + address);
    }
}
```

viettel
software services

Ví dụ với Serializable

- Thực hiện hàm main

```
public static void main(String[] args) {
    Employee e = new Employee();
    e.name = "Nguyen Trung Duc";
    e.address = "80 Dich Vong Hau - Cau Giay - Ha Noi";
    e.SSN = 11122333;
    e.number = 101;

    try {
        FileOutputStream fileOut = new FileOutputStream( name: "C:\\employee.data");
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        out.writeObject(e);
        out.close();
        fileOut.close();
        System.out.printf("Serialized data is saved in C:\\employee.data");
    } catch (IOException i) {
        i.printStackTrace();
    }
}
```

viettel
software services

Ví dụ với DeSerializable

- Thực hiện hàm main

```
public static void main(String[] args) {
    Employee e = null;
    try {
        FileInputStream fileIn = new FileInputStream( name: "C:\\employee.data");
        ObjectInputStream in = new ObjectInputStream(fileIn);
        e = (Employee) in.readObject();
        in.close();
        fileIn.close();
    } catch (IOException i) {
        i.printStackTrace();
        return;
    } catch (ClassNotFoundException c) {
        System.out.println("Employee class not found");
        c.printStackTrace();
        return;
    }

    System.out.println("Deserialized Employee...");
    System.out.println("Name: " + e.name);
    System.out.println("Address: " + e.address);
    System.out.println("SSN: " + e.SSN);
    System.out.println("Number: " + e.number);
}
```

viettel
software services

BÀI 2

Java NIO

viettel
software services

MỤC LỤC

viettel
software services

01

Java NIO

viettel
software services

Java NIO

- **Java IO** trong Java được sử dụng để thực các thao tác đọc ghi dữ liệu từ các nguồn khác nhau. Package java.io.package chứa tất cả các class cần để thực thi các thao tác IO.
- **Về sau, Java NIO (New input/output)** được giới thiệu trong phiên bản JDK 4 cho phép thực thi các thao tác IO với tốc độ nhanh và cung cấp nhiều tính năng hỗ trợ tối ưu hóa hiệu suất.
- **Java NIO (New input/output)** được xem là một công cụ thay thế Java IO, giúp các thao tác đọc ghi dữ liệu từ network hoặc các File đạt hiệu suất cao hơn. Cũng giống như java.io package, java.nio package định nghĩa các buffer class được sử dụng xuyên suốt trong các NIO API.

viettel
software services

Tính năng quan trọng Java NIO

- **Non-blocking IO operation:** có nghĩa là nó sẽ đọc dữ liệu bất cứ khi nào chúng sẵn sàng, trong thời gian chờ thì thread có thể thực thi các nhiệm vụ khác làm tăng hiệu suất của toàn ứng dụng.
- **Buffer oriented approach:** Java NIO tiếp cận dữ liệu bằng sử dụng bộ nhớ đệm. Dữ liệu được đọc và lưu lại trong đó, bất cứ khi nào cần chúng sẽ được lấy ra và xử lý tại đây.

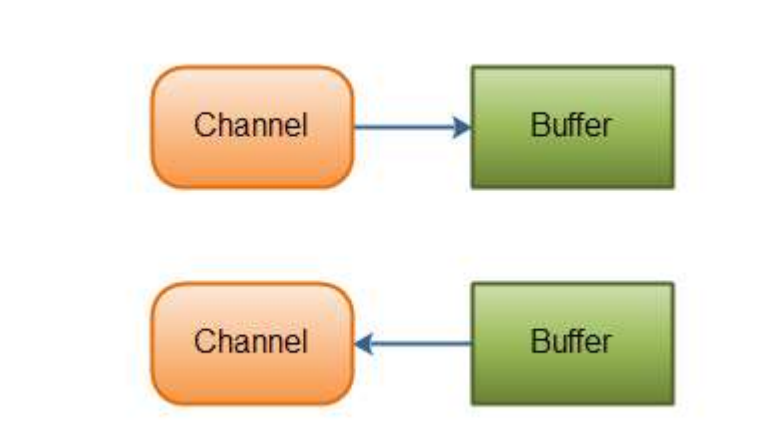
viettel
software services

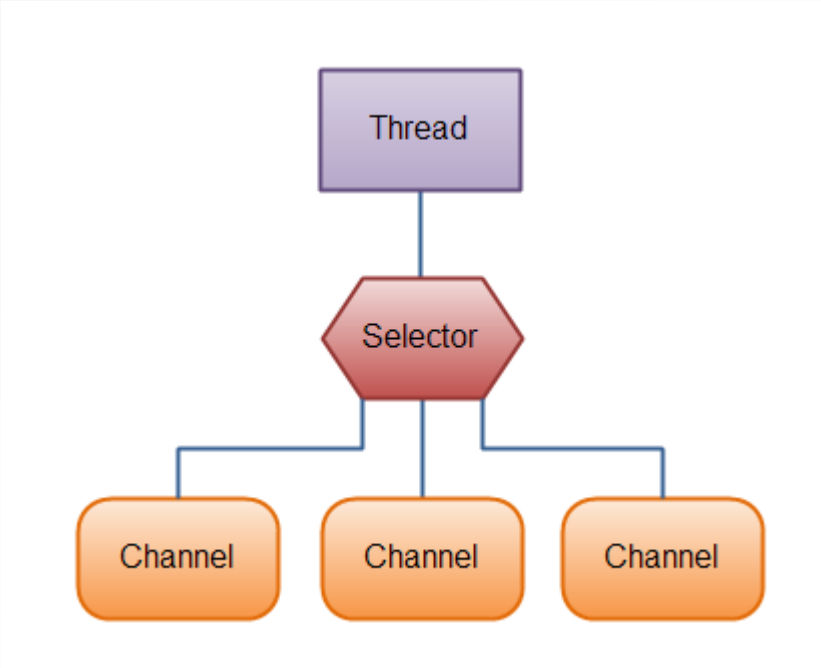
Cách thức hoạt động Java NIO thông qua các thành phần:

- **Buffers:** Mỗi buffer thật chất là một vùng nhớ mà bạn có thể đọc và ghi dữ liệu từ đó. Vùng nhớ này được bao bọc trong một NIO Buffer object, cung cấp các method hỗ trợ thao tác với dữ liệu trong vùng nhớ này.
- **Channel:** Có một chút tương đồng với Stream API, được dùng để giao tác giữa nội bộ Java NIO với bên ngòai. Từ một channel chúng ta có thể đọc dữ liệu từ một buffer hoặc ghi dữ liệu vào một buffer.
- **Selectors:** Một selector là một object quản lý nhiều channel. Khi Java NIO thực thi các hoạt động non-block IO, selector sẽ có nhiệm vụ chọn ra các channel đang sẵn sàng để thực thi nhiệm vụ.

viettel
software services

Cách thức hoạt động Java NIO





viettel
software services

Java NIO Channel

- Tương tự như stream trong IO, là cầu nối giữa buffer và nguồn dữ liệu (file, socket,…), và khác với IO vài điểm sau:
- Có thể đọc và ghi 2 chiều trên cùng 1 channel, stream chỉ một chiều.
- Channel có thể đọc ghi bất đồng bộ (vừa đọc vừa ghi).
- Cuối cùng, cơ bản nhất, channel phải được đọc và ghi từ 1 buffer.
- Lưu ý: đọc(read) là data đi từ channel vào buffer (nguồn dữ liệu -> buffer), ghi(write) ngược lại (buffer -> nguồn dữ liệu)

viettel
software services

Java NIO Buffer

- **Java NIO Buffer** đại diện cho một bộ chứa với sức chứa (capacity) cố định để lưu trữ các dữ liệu nguyên thuỷ. Nó thường được sử dụng cùng với **các Java NIO Channel(s**). Cụ thể, dữ liệu sẽ được đọc từ **Channel** vào **Buffer** hoặc ghi dữ liệu từ **Buffer** vào **Channel**.
- Mối quan hệ giữa **Channel** và **Buffer** cũng giống như mối quan hệ giữa cái bát và cái thìa. Cái thìa có thể sử dụng như một bộ chứa nhỏ để lấy đường ra từ bát và nó cũng được sử dụng như một bộ chứa nhỏ để cho đường từ bên ngoài vào trong bát. Như vậy, cái thìa hoạt động như một **Buffer** còn cái bát hoạt động như một **Channel**.

viettel
software services

02

Ví dụ

viettel
software services

Ví dụ với Java NIO

```
public static void readFileChannel() throws IOException {
    RandomAccessFile randomAccessFile = new RandomAccessFile( name: "C:/Test/temp.txt",
            mode: "rw");
    FileChannel fileChannel = randomAccessFile.getChannel();
    ByteBuffer byteBuffer = ByteBuffer.allocate(512);
    Charset charset = Charset.forName("US-ASCII");
    while (fileChannel.read(byteBuffer) > 0) {
        byteBuffer.rewind();
        System.out.print(charset.decode(byteBuffer));
        byteBuffer.flip();
    }
    fileChannel.close();
    randomAccessFile.close();
}
```

viettel
software services

Ví dụ với Java NIO

```
public static void writeFileChannel(ByteBuffer byteBuffer) throws IOException {
    Set<StandardOpenOption> options = new HashSet<>();
    options.add(StandardOpenOption.CREATE);
    options.add(StandardOpenOption.APPEND);
    Path path = Paths.get( first: "C:/Test/temp.txt");
    FileChannel fileChannel = FileChannel.open(path, options);
    fileChannel.write(byteBuffer);
    fileChannel.close();
}
```

viettel
software services

Ví dụ với Java NIO

```
public static void main(String args[]) throws IOException {
    //append the content to existing file
    writeFileChannel(ByteBuffer.wrap("Welcome to TutorialsPoint".getBytes()));
    //read the file
    readFileChannel();
}
```

viettel
software services

Ví dụ với Java NIO

```
public static void main(String arr[])
        throws FileNotFoundException
{
    // Creating a File object that
    // represents the disk file
    PrintStream o = new PrintStream(new File( pathname: "demo.txt"));
    // Store current System.out
    // before assigning a new value
    PrintStream console = System.out;
    // Assign o to output stream
    // using setOut() method
    System.setOut(o);
    // Display message only
    System.out.println("This will be written to the text file");
    // Use stored value for output stream
    System.setOut(console);
    // Display message only
    System.out.println( "This will be written on the console!");
}
```

viettel
software services

XIN CẢM ƠN