Faculty of
Information Technology
King Mongkut's Institute of Technology Ladkrabang

บทที่ 12: คลาสนำเข้าและส่งออกข้อมูล (Input and Output Classes)

บรรยายโดย ผศ.ดร.ธราวิเชษฐ์ ธิติจรูญโรจน์

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อ

- คลาส File
- โครงสร้างช่องทางการสื่อสาร
  - stream
    - คลาส Byte stream
    - คลาส Char stream
  - Node
    - โหนด Input Source
    - โหนด Output Sink
- การเชื่อมต่อ stream และ stream ระดับสูง
- คลาส Object stream
- อินเตอร์เฟส Serializable และ คีย์เวิร์ด Transient

หัวข้อ

- คลาส File
- โครงสร้างช่องทางการสื่อสาร
  - stream
    - คลาส Byte stream
    - คลาส Char stream
  - Node
    - โหนด Input Source
    - โหนด Output Sink
- การเชื่อมต่อ stream และ stream ระดับสูง
- คลาส Object stream
- อินเตอร์เฟส Serializable และ คีย์เวิร์ด Transient

คลาส File

คลาส File เป็นคลาสที่อยู่ในแพคเก็จ java.io ที่ใช้ในการสร้างอ็อบเจ็คที่เป็นไฟล์หรือไดเร็กทอรี่ ซึ่งจะมีเมธอดในการจัดการกับไฟล์หรือไดเร็กทอรี่ และเมธอดในการสืบค้นข้อมูลต่าง ๆ อยู่หลายเมธอด โดยที่อ็อบเจ็คของคลาส File จะสร้างมาจาก constructor ที่มีรูปแบบดังนี้

```
public File(String name)
public File(String dir, String name)
public File(File dir, String name)
```

เมธอดของคลาส File

เมธอดของคลาส File ที่ใช้ในการสืบค้นข้อมูลหรือจัดการกับไฟล์ที่สำคัญมีดังนี้

```
public boolean exists()
public boolean isFile()
public boolean isDirectory()
public String getName()
public String getParent()
public String[] list()
public boolean canWrite()
public boolean mkdir()
public boolean renameTo(File newName)
```

การสร้างอ็อบเจ็คของคลาส File

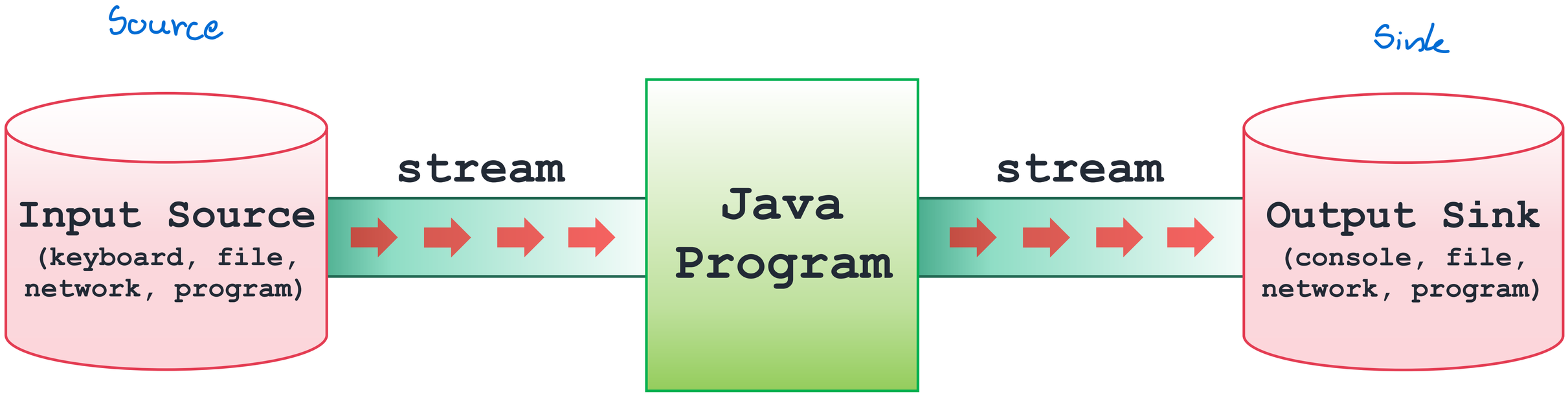
นักศึกษาสามารถสร้างอ็อบเจ็คชนิดคลาส File เพื่อตรวจสอบว่ามีไฟล์ดังกล่าวหรือไม่ โดยอาศัย Constructor ของ FileInputStream หรือ FileOutputStream หรือ FileReader หรือ FileWriter มี argument ที่เป็นอ็อบเจ็คชนิดคลาส File

```
File f = new File("test.dat");
if (f.exists()) {
      FileInputStream fir = new FileInputStream(f);
}
```

หัวข้อ

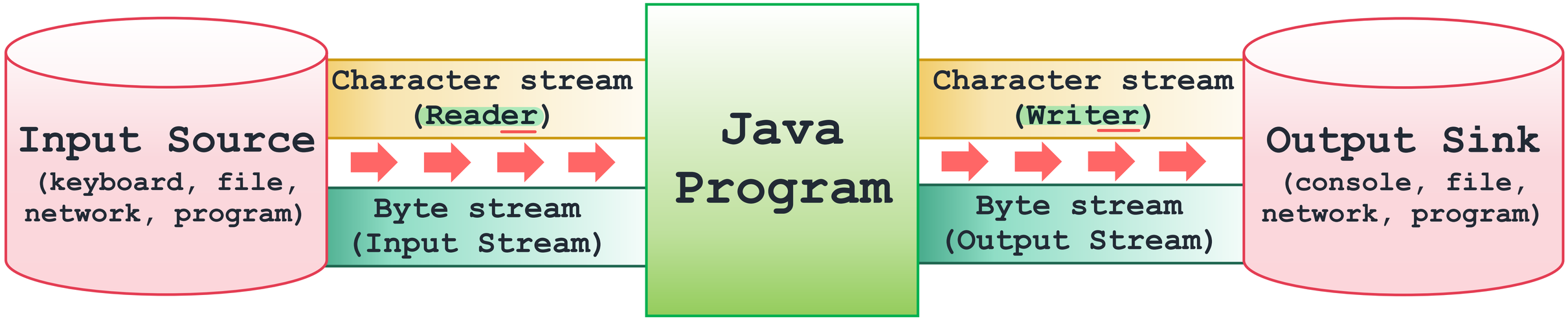
- คลาส File
- โครงสร้างช่องทางการสื่อสาร
  - stream
    - คลาส Byte stream
    - คลาส Char stream
  - Node
    - โหนด Input Source
    - โหนด Output Sink
- การเชื่อมต่อ stream และ stream ระดับสูง
- คลาส Object stream
- อินเตอร์เฟส Serializable และ คีย์เวิร์ด Transient

โครงสร้างช่องทางการสื่อสาร



ประกอบด้วย 3 ส่วนหลัก ได้แก่ ต้นทาง (input source) ช่องทางการแลกเปลี่ยนข้อมูล (stream) และปลายทาง (output sink)

ช่องทางการสื่อสาร (stream)

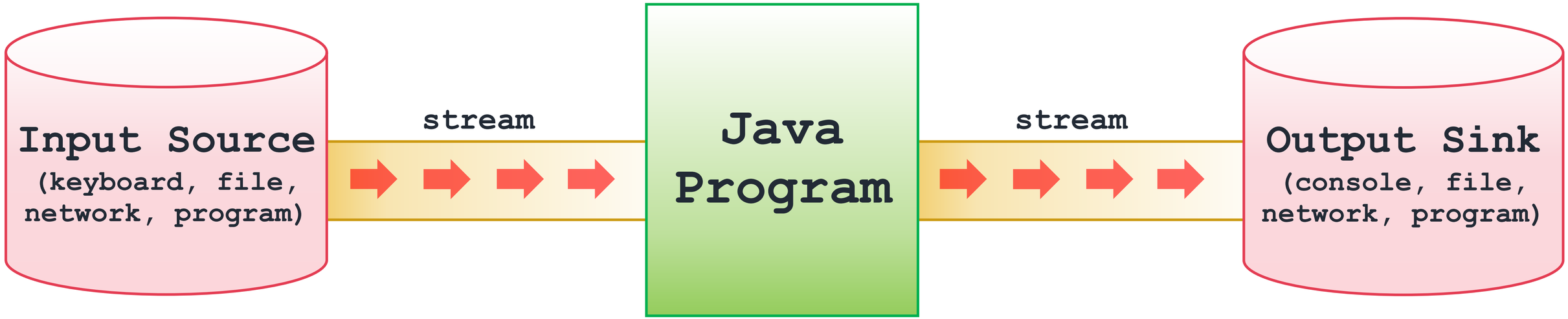


stream เปรียบเสมือนช่องทางการสื่อสารสำหรับการส่งข้อมูลจาก ต้นทาง (source) ไปสู่ปลายทาง (sink) ซึ่งต้นทางและปลายทางอาจจะเป็นฮาร์ดแวร์หรือซอฟต์แวร์ เช่น ไฟล์ หน่วยความจำ หรือ socket สำหรับภาษาจาวา stream มีทั้งหมด 2 ประเภท ได้แก่ byte stream และ character stream ตามลำดับ ซึ่ง byte stream อาจจะเรียกอีกอย่างว่า stream หรือ input stream หรือ output stream ขณะที่ character stream อาจจะเรียกว่า reader หรือ writer

หัวข้อ

- คลาส File
- โครงสร้างช่องทางการสื่อสาร
  - stream
    - คลาส Byte stream
    - คลาส Char stream
  - Node
    - โหนด Input Source
    - โหนด Output Sink
- การเชื่อมต่อ stream และ stream ระดับสูง
- คลาส Object stream
- อินเตอร์เฟส Serializable และ คีย์เวิร์ด Transient

โหนดสำหรับ Stream



ภาษาจาวากำหนดโหนดที่เป็น*ต้นทาง*และ*ปลายทาง*ของ stream ไว้ 3 แบบคือ ไฟล์ หน่วยความจำ และไปป์ (pipe)

- ไฟล์ คือ โหนดสำหรับ stream ที่เป็นไฟล์สำหรับอ่านหรือเขียนข้อมูลชนิด byte
- หน่วยความจำ คือ โหนดสำหรับ stream ที่ใช้สำหรับอ่านหรือเขียนข้อมูลที่เป็น*อะเรย์*หรือ
- ไปป์ คือ โหนดสำหรับ stream ที่จะส่งหรือรับข้อมูลระหว่าง process หรือโปรแกรมเธรด*String*

หัวข้อ

- คลาส File
- โครงสร้างช่องทางการสื่อสาร
  - stream
    - คลาส Byte stream
    - คลาส Char stream
  - Node
    - โหนด Input Source
    - โหนด Output Sink
- การเชื่อมต่อ stream และ stream ระดับสูง
- คลาส Object stream
- อินเตอร์เฟส Serializable และ คีย์เวิร์ด Transient

ช่องทางการสื่อสาร (stream)

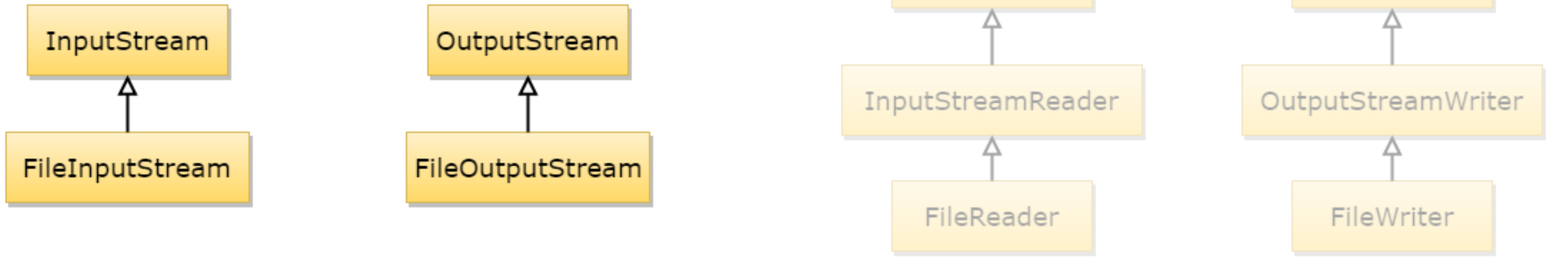
- Source หรือ Sink อาจเป็น*ฮาร์ดแวร์*หรือ*ซอฟต์แวร์* เช่น ไฟล์ หน่วยความจำ Socket เป็นต้น
- ภาษาจาวาแบ่ง stream ออกเป็น
  - byte stream
  - character stream
- Stream โดยทั่วไป จะหมายถึง byte stream
- Reader และ Writer จะหมายถึง character stream

คลาสที่เกี่ยวข้องกับ stream

คลาสที่เกี่ยวข้องกับการนำเข้าและส่งออกข้อมูลจะอยู่ในแพคเก็จ java.io ซึ่งประกอบด้วย 4 คลาส ได้แก่

		ชื่อคลาส	ความหมาย
Byte	Input	InputStream	สร้างวัตถุที่เป็นช่องทางการสำหรับการรับข้อมูลชนิด byte
	Output	OutputStream	สร้างวัตถุที่เป็นช่องทางการสำหรับการส่งข้อมูลชนิด byte
Character	Input	Reader	สร้างวัตถุที่เป็นช่องทางการสำหรับการรับข้อมูลชนิด char
	Output	Writer	สร้างวัตถุที่เป็นช่องทางการสำหรับการส่งข้อมูลชนิด char

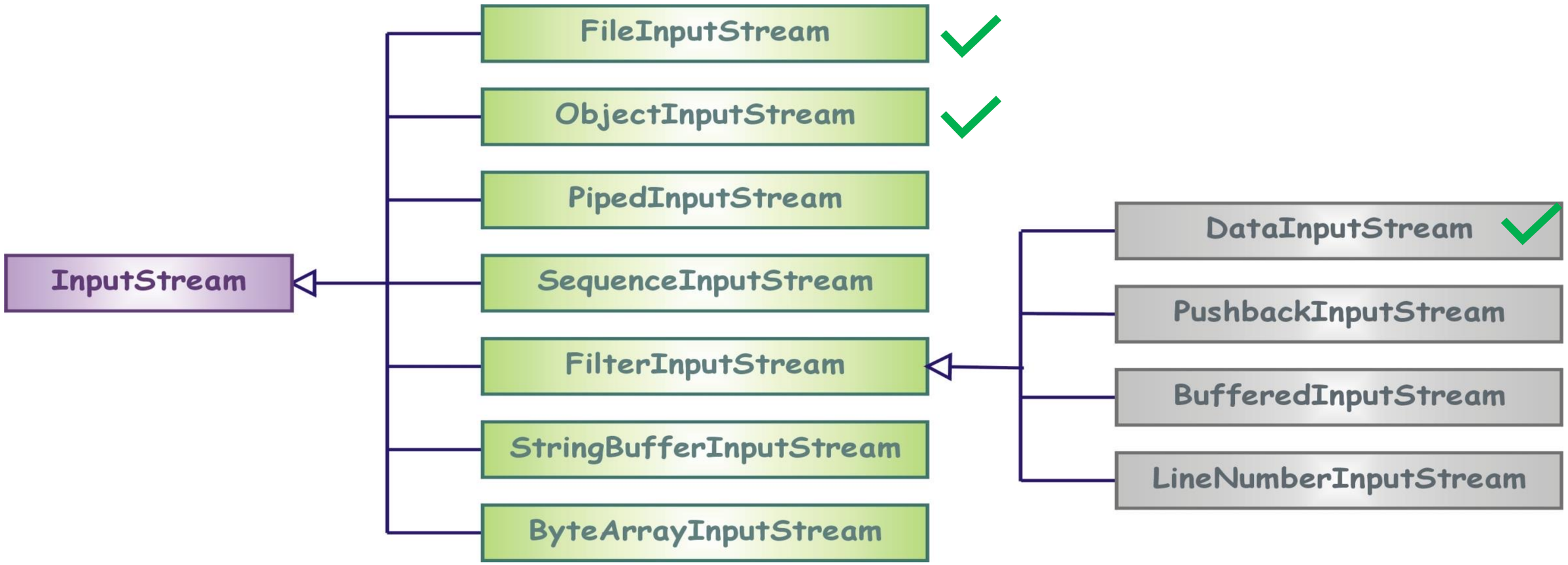
คลาสที่เกี่ยวข้องกับ stream



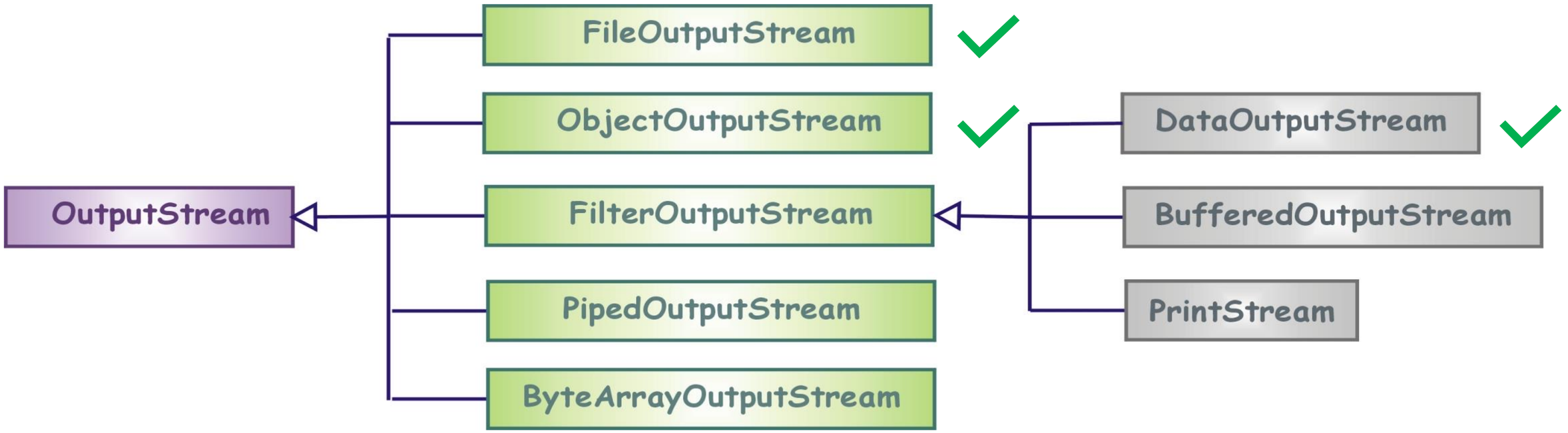
คลาสที่เกี่ยวข้องกับการนำเข้าและส่งออกข้อมูลจะอยู่ในแพคเก็จ java.io ซึ่งประกอบด้วย 4 คลาส ได้แก่

		ชื่อคลาส	ความหมาย
Byte	Input	InputStream	สร้างวัตถุที่เป็นช่องทางการสำหรับการรับข้อมูลชนิด byte
	Output	OutputStream	สร้างวัตถุที่เป็นช่องทางการสำหรับการส่งข้อมูลชนิด byte
Character	Input	Reader	สร้างวัตถุที่เป็นช่องทางการสำหรับการรับข้อมูลชนิด char
	Output	Writer	สร้างวัตถุที่เป็นช่องทางการสำหรับการส่งข้อมูลชนิด char

คลาสประเภท InputStream



คลาสประเภท **OutputStream**



คลาสประเภท Byte Stream

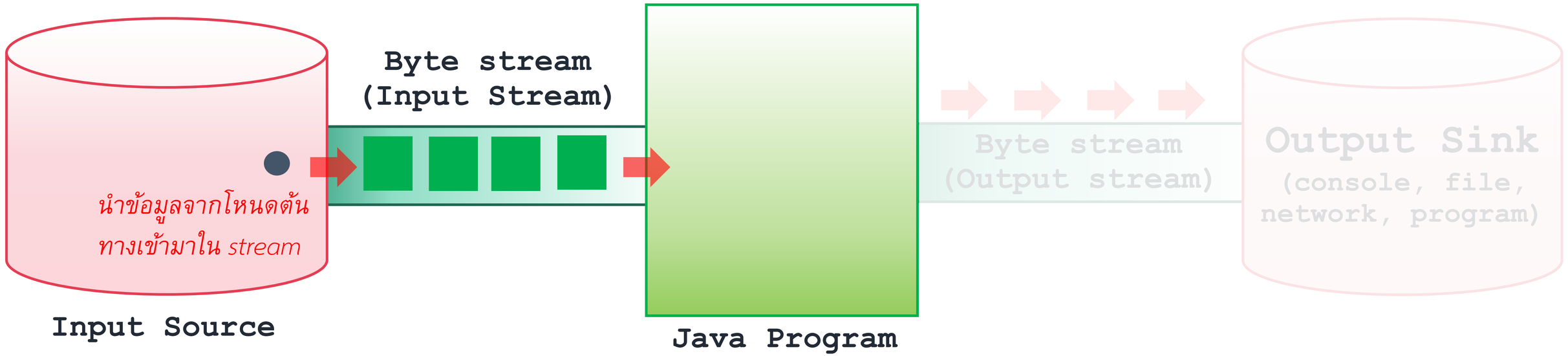
ภาษาจาวามีคลาสพื้นฐานในการจัดการกับอินพุตและเอาท์พุตที่เป็นชนิดข้อมูลแบบ byte อยู่ 2 คลาส ได้แก่ *InputStream* และ *OutputStream* (แบบ abstract) โดยมีคลาสที่เป็น subclass ซึ่งจะใช้ในการสร้างออปเจ็คสำหรับการรับและส่งข้อมูลแบบ byte ของโหนดที่มีต้นทางและปลายทางแบบต่าง ๆ อาทิเช่น

- FileInputStream และ FileOutputStream เป็นคลาสที่ใช้ในการสร้างออปเจ็คสำหรับต้นทางและปลายทางที่เป็น*ไฟล์*
- ByteArrayInputStream และ ByteArrayOutputStream เป็นคลาสที่ใช้ในการสร้างออปเจ็คสำหรับต้นทางและปลายทาง ที่เป็น*อะเรย์ของชนิดข้อมูลแบบ byte*

คลาส InputStream

คลาส InputStream จะใช้ในการ*อ่านข้อมูล*ของ stream ที่เป็นชนิดข้อมูลแบบ ***byte*** ซึ่งจะ*นำข้อมูลจากโหนดต้นทางเข้ามาใน stream* และ*การอ่านข้อมูลจาก stream จะเป็นการลบข้อมูลที่อ่านออกจาก stream* โดยมีเมธอดที่ใช้สำหรับ*การอ่านข้อมูลที่เป็น byte หรืออะเรย์ของ byte* เท่านั้นดังนี้

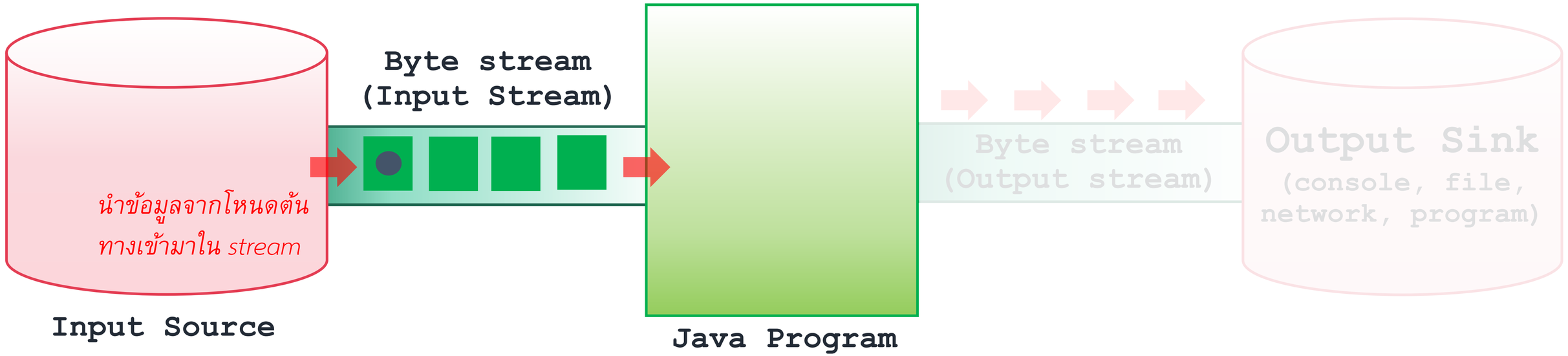
- `int read()`
- `int read(byte []b)`
- `int read(byte []b,int offset,int length)`



คลาส InputStream

คลาส InputStream จะใช้ในการ*อ่านข้อมูล*ของ stream ที่เป็นชนิดข้อมูลแบบ ***byte*** ซึ่งจะ*นำข้อมูลจากโหนดต้นทางเข้ามาใน stream* และ*การอ่านข้อมูลจาก stream จะเป็นการลบข้อมูลที่อ่านออกจาก stream* โดยมีเมธอดที่ใช้สำหรับ*การอ่านข้อมูลที่เป็น byte หรืออะเรย์ของ byte* เท่านั้นดังนี้

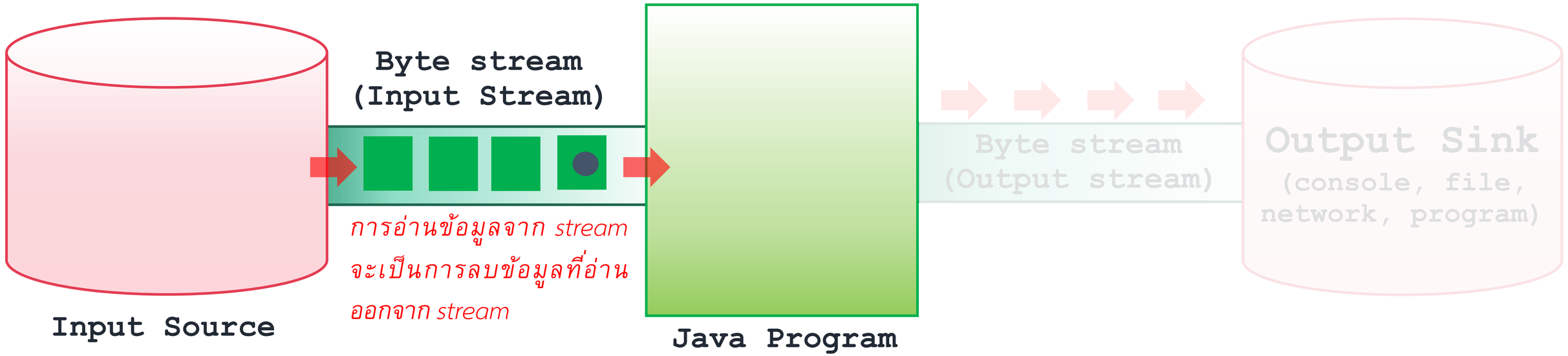
- `int read()`
- `int read(byte []b)`
- `int read(byte []b,int offset,int length)`



คลาส InputStream

คลาส InputStream จะใช้ในการ*อ่านข้อมูล*ของ stream ที่เป็นชนิดข้อมูลแบบ ***byte*** ซึ่งจะ*นำข้อมูลจากโหนดต้นทางเข้ามาใน stream* และ*การอ่านข้อมูลจาก stream จะเป็นการลบข้อมูลที่อ่านออกจาก stream* โดยมีเมธอดที่ใช้สำหรับ*การอ่านข้อมูลที่เป็น byte หรืออะเรย์ของ byte* เท่านั้นดังนี้

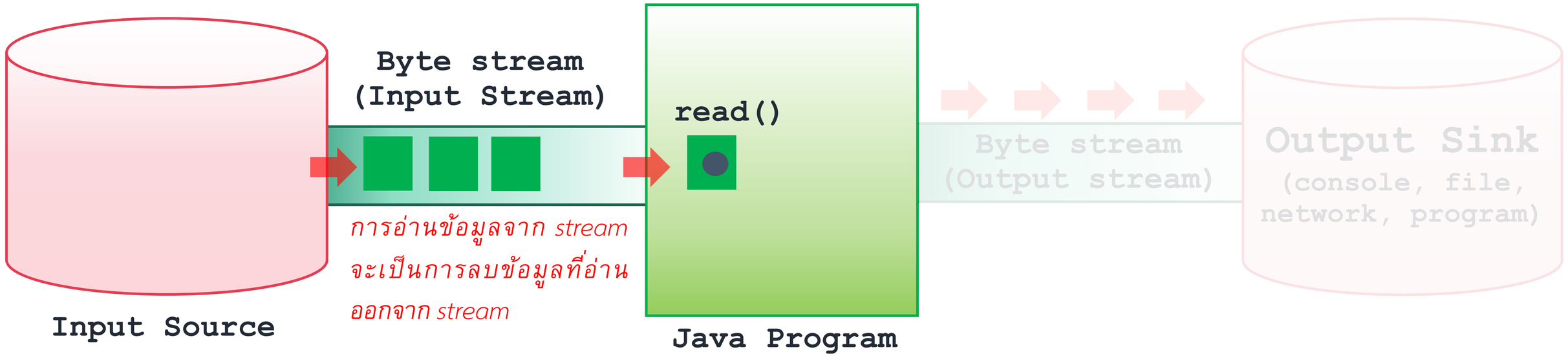
- **`int read()`**
- **`int read(byte []b)`**
- **`int read(byte []b,int offset,int length)`**



คลาส InputStream

คลาส InputStream จะใช้ในการ*อ่านข้อมูล*ของ stream ที่เป็นชนิดข้อมูลแบบ ***byte*** ซึ่งจะ*นำข้อมูลจากโหนดต้นทางเข้ามาใน stream* และ*การอ่านข้อมูลจาก stream จะเป็นการลบข้อมูลที่อ่านออกจาก stream* โดยมีเมธอดที่ใช้สำหรับ*การอ่านข้อมูลที่เป็น byte หรืออะเรย์ของ byte* เท่านั้นดังนี้

- **`int read()`**
- **`int read(byte []b)`**
- **`int read(byte []b,int offset,int length)`**



คลาส InputStream

เมธอดอื่น ๆ ที่สำคัญ

- `public void close()`
- `public int available()`
- `public void skip(long n)`
- `public boolean markSupported()`
- `public void mark(int readlimit)`
- `public void reset()`

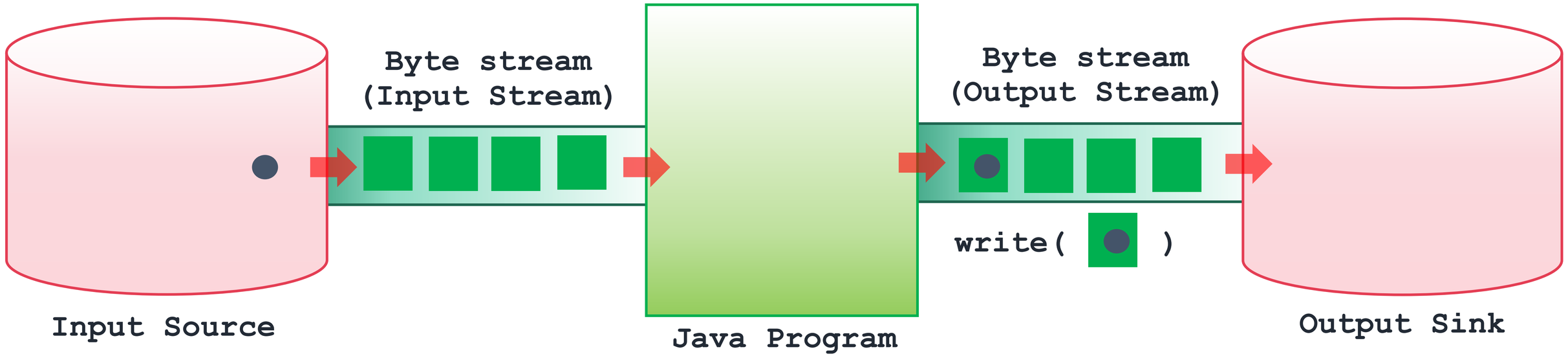
ตัวอย่างที่ 1: การอ่านไฟล์โดยอาศัย FileInputStream

```
import java.io.*;
public class InputSteamWriteDemo {
   public static void main(String args[]) {
       try (FileInputStream  fin  = new FileInputStream("dataDemo01.dat");){
            int i = fin.read();
            while (i != -1) {
                System.out.print((char)i);
                i = fin.read();
            }
        } catch (IOException e) {
           System.out.println(e);
        }
   }
}
```

คลาส OutputStream

คลาส OutputStream จะใช้การ***ส่งข้อมูลของ stream ที่เป็นชนิดข้อมูลแบบ byte*** การส่งข้อมูลของออบเจ็คชนิด Outputstream จะเป็นการเพิ่มข้อมูลลงใน stream โดยคลาสนี้จะมีเมธอด write() ที่เป็นเมธอดแบบ abstract ในรูปแบบต่างๆดังนี้

- `void write(int c)`
- `void write(byte []b)`
- `void write(byte []b, int offset,int length)`



คลาส OutputStream

- เมธอดอื่นๆ ที่สำคัญ
  - `public void close()`
  - `public void flush()`

ตัวอย่างที่ 2: การเขียนไฟล์โดยอาศัย FileOutputStream

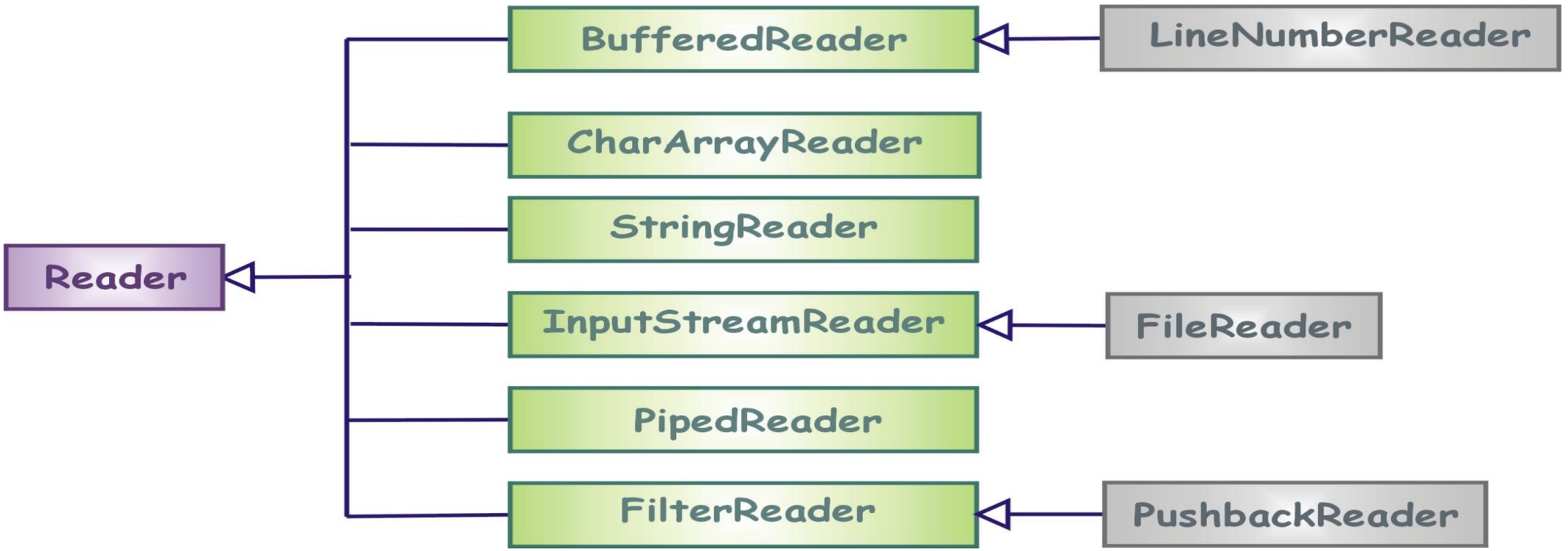
```
import java.io.*;
public class OutputSteamReadDemo1 {
    public static void main(String args[]) {
        String str = "Hi, Sara. \n My name is John.";
        try(FileOutputStream fout = new FileOutputStream("dataDemo01.dat");) {
            for(int i = 0; i < str.length(); i++)
                fout.write(str.charAt(i));
             System.out.println("Done it");
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

คลาสที่เกี่ยวข้องกับ stream

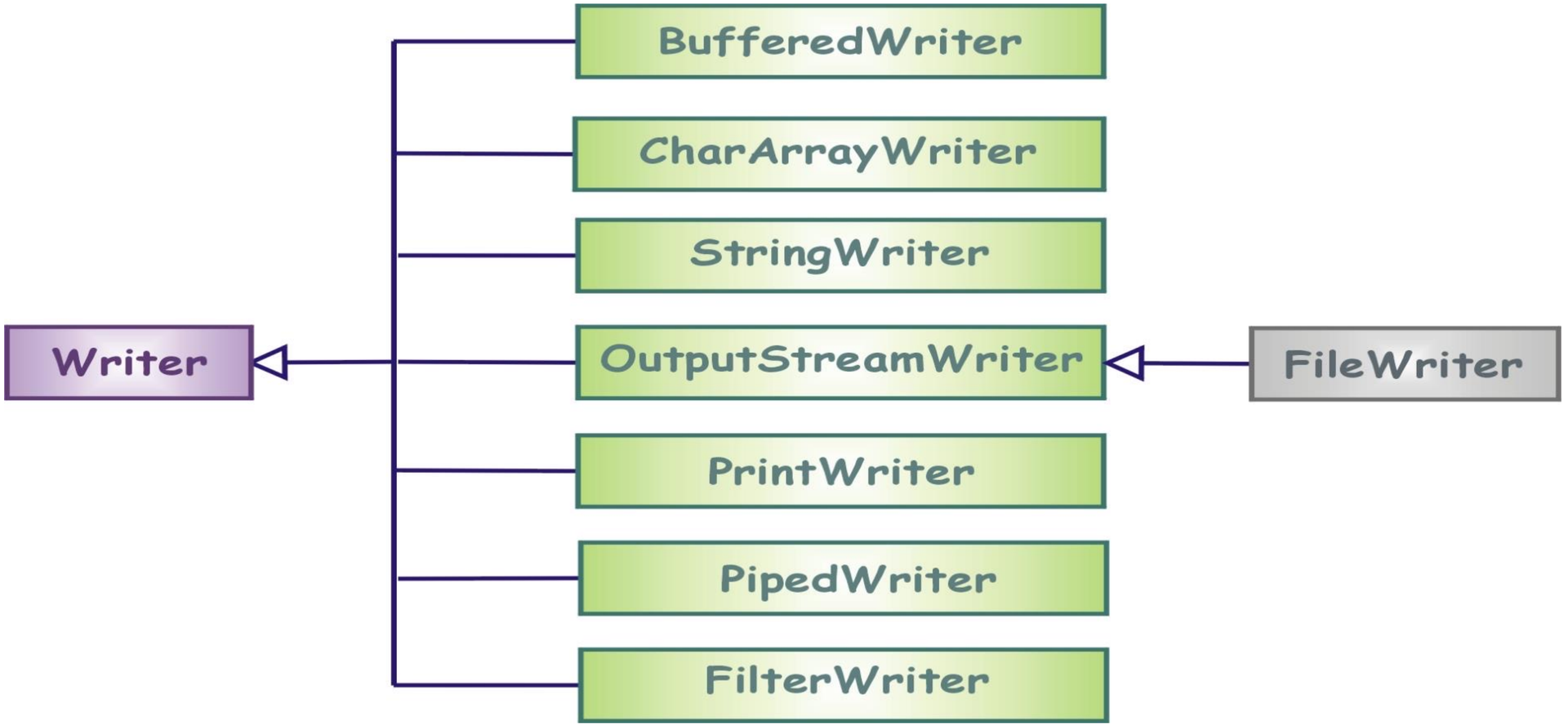
คลาสที่เกี่ยวข้องกับการนำเข้าและส่งออกข้อมูลจะอยู่ในแพคเก็จ java.io ซึ่งประกอบด้วย 4 คลาส ได้แก่

		ชื่อคลาส	ความหมาย
Byte	Input	InputStream	สร้างวัตถุที่เป็นช่องทางการสำหรับการรับข้อมูลชนิด byte
	Output	OutputStream	สร้างวัตถุที่เป็นช่องทางการสำหรับการส่งข้อมูลชนิด byte
Character	Input	Reader	สร้างวัตถุที่เป็นช่องทางการสำหรับการรับข้อมูลชนิด char
	Output	Writer	สร้างวัตถุที่เป็นช่องทางการสำหรับการส่งข้อมูลชนิด char

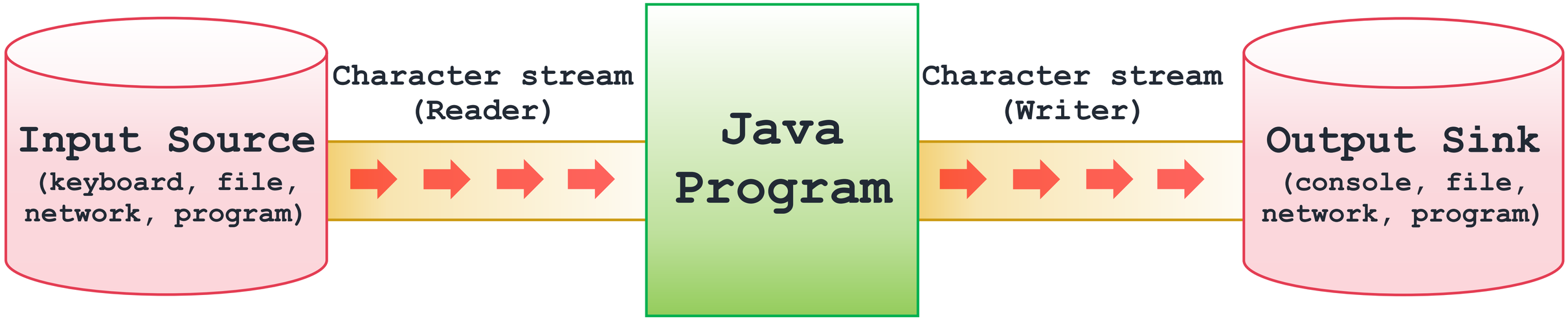
คลาสประเภท **Reader**



คลาสประเภท **Writer**



Character stream (Reader)



stream เปรียบเสมือนช่องทางการสื่อสารสำหรับการส่งข้อมูลจาก ต้นทาง (source) ไปสู่ปลายทาง (sink) ซึ่งต้นทางและปลายทางอาจจะเป็นฮาร์ดแวร์หรือซอฟต์แวร์ เช่น ไฟล์ หน่วยความจำ หรือ socket สำหรับภาษาจาวา stream มีทั้งหมด 2 ประเภท ได้แก่ byte stream และ character stream ตามลำดับ ซึ่ง byte stream อาจจะเรียกอีกอย่างว่า stream หรือ input stream หรือ output stream ขณะที่ character stream อาจจะเรียกว่า reader หรือ writer

FileWriter

คลาส FileWriter เป็นที่นิยมสำหรับการสร้างหรือเขียนไฟล์เอกสารต่าง ๆ ในรูปแบบชนิด character โดยที่คลาส FileWriter ได้สืบทอดมาจากคลาส OutputStreamWriter ซึ่งประกอบด้วย Constructors ดังต่อไปนี้

```
public FileWriter(File file)
public FileWriter (File file, boolean append)
public FileWriter (FileDescriptor fd)
public FileWriter (String fileName)
public FileWriter (String fileName, Boolean append)
```

FileWriter

และเมธอดที่สำคัญของคลาส FileWriter ได้แก่

เมธอด	ความหมาย
`public void write (int c)`	เขียนตัวอักษรหนึ่งตัวอักขระ
`public void write (char [] s)`	เขียนตัวอักษรทุกตัวในอาร์เรย์
`public void write(String str)`	เขียนหนึ่งข้อความ
`public void write (String str, int off, int len)`	เขียนบางส่วนของข้อความเริ่มจาก off ยาวไปอีก len ตัว
`public void flush()`	ล้างข้อมูลใน stream (หรือ ข้อมูลในไฟล์) และสั่งให้เขียนข้อมูลจาก buffer ลงในไฟล์
`public void close()`	ปิดช่องทางการติดต่อ และต้องเปิดช่องทางการสื่อสารใหม่ กรณีต้องการจะเขียนไฟล์อีกครั้ง

ตัวอย่างที่ 3: การเขียนไฟล์โดยอาศัย FileWriter

```
import java.io.*;
public class MyWFile {
    public static void main(String[] args){
        String str = "Hi, Bank";
        try(FileWriter fw = new FileWriter("test01.txt"); ){
            for (int i = 0; i < str.length(); i++) {
                fw.write(str.charAt(i));
            }
            System.out.println("Writing successful");
        }catch(IOException e){
            System.out.print(e);
        }
    }
}

// แนะนำให้เขียนโดยอาศัย try-with-resources Statement
```

ตัวอย่างที่ 3: การเขียนไฟล์โดยอาศัย FileWriter

```
import java.io.*;
public class MyWFile {
    public static void main(String[] args)
    {
        try{
            String str = "Hi, Bank";
            // attach a file to FileWriter
            FileWriter fw = new FileWriter("test01.txt");

            // read character wise from string and write into FileWriter
            for (int i = 0; i < str.length(); i++) {
                fw.write(str.charAt(i));
            }
            System.out.println("Writing successful");
            //close the file
            fw.close();
        }catch(IOException e){
            System.out.print(e);
        }
    }
}
```

ตัวอย่างที่ 4: การเขียนไฟล์โดยอาศัย FileWriter

```
import java.io.*;
public class MyWFile {
    public static void main(String[] args)
    {
        try{
            String str = "Hi, Bank";
            // attach a file to FileWriter
            FileWriter fw = new FileWriter("test01.txt");

            // write string into FileWriter
            fw.write(str);

            System.out.println("Writing successful");
            //close the file
            fw.close();
        }catch(IOException e){
            System.out.print(e);
        }
    }
}
```

ตัวอย่างที่ 4: การเขียนไฟล์โดยอาศัย FileWriter

```
import java.io.*;
public class MyWFile {
    public static void main(String[] args)
    {
        try{
            String str = "Hi, Bank";
            FileWriter fw = new FileWriter("test01.txt");
            fw.write(str);
            System.out.println("Writing successful");
            fw.close();
        }catch(IOException e){
            System.out.print(e);
        }
    }
}
```

```
import java.io.*;
public class Main {
    public static void main(String[] args)
    {
        String str = "Hi, Bank";
        try(FileWriter fw = new FileWriter("test01.txt"); ){
            fw.write(str);
            System.out.println("Writing successful");
        }catch(IOException e){
            System.out.print(e);
        }
    }
}
```

FileReader

คลาส FileReader เป็นที่นิยมสำหรับการอ่านไฟล์เอกสารต่าง ๆ ในรูปแบบชนิด character โดยที่คลาส FileReader ได้สืบทอดมาจากคลาส InputStreamReader ซึ่งประกอบด้วย Constructors ดังต่อไปนี้

```
public FileReader(File file)

public FileReader(FileDescripter fd)

public FileReader(String fileName)
```

FileReader

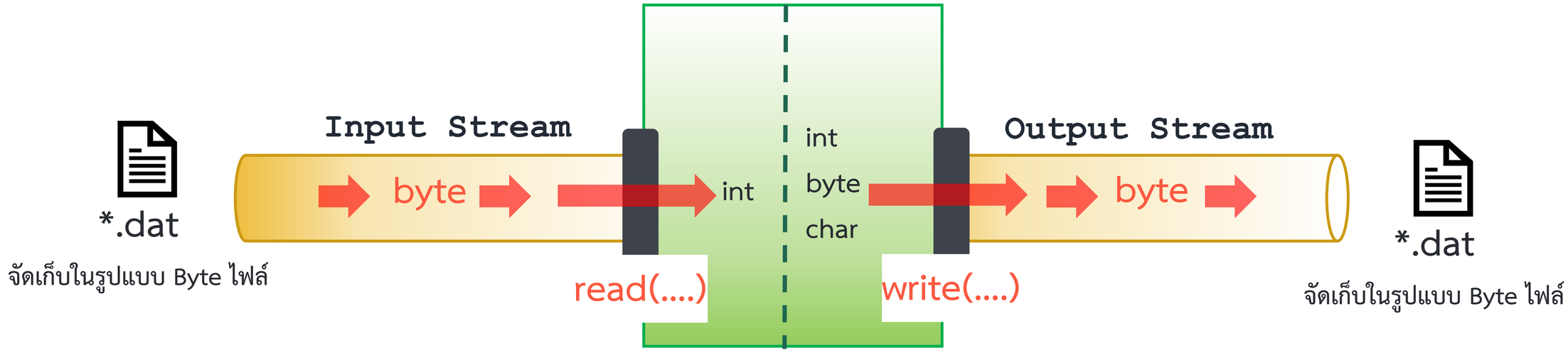
และเมธอดที่สำคัญของคลาส FileReader ได้แก่

เมธอด	ความหมาย
`public int read ()`	อ่านตัวอักษรหนึ่งตัวอักขระ
`public int read(char[] cbuff)`	อ่านตัวอักษรทุกตัวในอาร์เรย์
`public abstract int read (char[] buff, int off, int len)`	อ่านบางส่วนของข้อความเริ่มจาก off ยาวไปอีก len ตัว
`public void close()`	ปิดช่องทางการติดต่อ และต้องเปิดช่องทางการสื่อสารใหม่ กรณีต้องการจะอ่านไฟล์อีกครั้ง
`public long skip(long n)`	ข้ามการอ่านตัวอักษรไป n ตัวอักขระ
`public void reset()`	reset ช่องทางการสื่อสารใหม่
`public boolean ready()`	ตรวจสอบความพร้อมสำหรับการอ่านข้อมูลของช่องทางการสื่อสาร

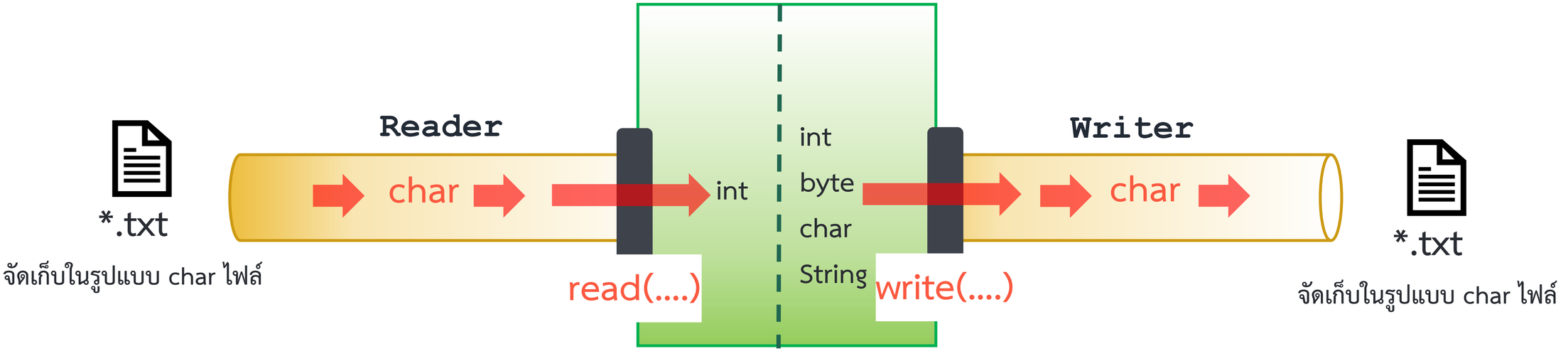
ตัวอย่างที่ 5

```
import java.io.*;
public class MyReadFile {
    public static void main(String[] args) {
        try {
            // variable declaration
            int ch;
            // check if File exists or not
            FileReader fr = new FileReader("test02.txt");

            // read from FileReader till the end of file
            while ((ch=fr.read())!=-1){
                System.out.print((char)ch);
            }
            // close the file
            fr.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

ภาพรวม



Java Program (Byte Stream)



Java Program (Character Stream)

คลาสที่เป็นโหนดสำหรับ stream ต่างๆ

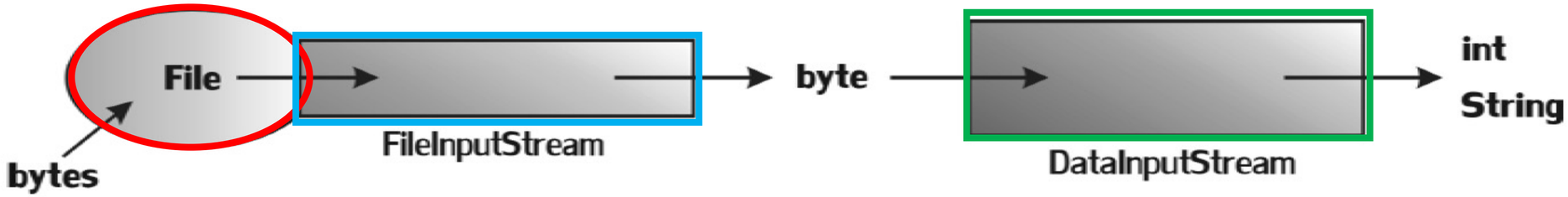
ชนิด	Byte Stream	Character Stream
File	FileInputStream FileOutputStream	FileReader FileWriter
Memory: Array	ByteArrayInputStream ByteArrayOutputStream	CharArrayReader CharArrayWriter
Memory: String	-	StringReader StringWriter
Pipe	PipedInputStream PipedOutputStream	PipedReader PipedWriter

หัวข้อ

- คลาส File
- โครงสร้างช่องทางการสื่อสาร
  - stream
    - คลาส Byte stream
    - คลาส Char stream
  - Node
    - โหนด Input Source
    - โหนด Output Sink
- **การเชื่อมต่อ stream และ stream ระดับสูง**
- คลาส Object stream
- อินเตอร์เฟส Serializable และ คีย์เวิร์ด Transient

การเชื่อมต่ออ็อบเจ็คของคลาส

โดยทั่วไปโปรแกรมภาษาจาวาจะใช้ออปเจ็คประเภท stream มากกว่าหนึ่งอ็อบเจ็คโดยจะเชื่อมอ็อบเจ็คของ stream ต่างๆ เข้าด้วยกัน*เพื่อใช้ในการแปลงชนิดข้อมูลประเภทต่าง ๆ*



ตัวอย่าง แสดงการเชื่อมต่ออ็อบเจ็คของคลาส FileInputStream ที่อ่านข้อมูลเข้ามาโดยมีชนิดข้อมูลเป็น byte เข้ากับอ็อบเจ็คของคลาส DataInputStream เพื่อใช้อ่านข้อมูลชนิดอื่นๆ ได้มากขึ้น เช่น

```
FileInputStream fin = new FileInputStream("test.dat");
DataInputStream din = new DataInputStream(fin);
```

คลาสประเภท **stream** ระดับสูง

อ็อบเจ็คที่ใช้ในการเชื่อมต่อ stream จะเป็นอ็อบเจ็คของคลาสประเภท stream ระดับสูง (high-level stream) ซึ่งสามารถที่จะ*อ่านหรือเขียนข้อมูลที่เป็นชนิดข้อมูลอื่นๆ* แล้ว*แปลงข้อมูลให้เป็นชนิดข้อมูลแบบ byte* หรือมีบัฟเฟอร์ในการเพิ่มประสิทธิภาพในการอ่านหรือเขียนข้อมูล

คลาสเหล่านี้จะ*ไม่สามารถอ่านหรือเขียนข้อมูลไปยังโหนดต้นทางหรือปลายทาง*ที่เป็นไฟล์ หน่วยความจำ หรือ Pipe *ได้โดยตรง* แต่จะ*รับข้อมูลมาจาก stream อื่นๆ* ที่เป็นคลาสพื้นฐานในการอ่านหรือเขียนข้อมูล

คลาสประเภท stream ระดับสูง

ชนิด	Byte Stream	Character Stream
Buffering	BufferedInputStream BufferedOutputStream	BufferedReader BufferedWriter
Filtering	FilterInputStream FilterOutputStream	FilterReader FilterWriter
Data conversion	DataInputStream DataOutputStream	-
Printing	PrintStream	PrintWriter
Peeking ahead	PushbackInputStream	PushbackReader

คลาสประเภท byte stream ระดับสูงที่สำคัญ

DataInputStream และ DataOutputStream

- เป็นคลาสที่ใช้ในการ*แปลงชนิดข้อมูล*ระหว่างชนิดข้อมูล*แบบ byte กับชนิดข้อมูลแบบอื่นๆ*

BufferedInputStream และ BufferedOutputStream

- เป็นคลาสที่มีบัฟเฟอร์สำหรับชนิดข้อมูล byte อยู่ภายใน เพื่อให้สามารถ*อ่านหรือเขียนข้อมูลขนาดใหญ่* ซึ่งจะช่วยเพิ่มประสิทธิภาพในการอ่านหรือเขียนข้อมูล

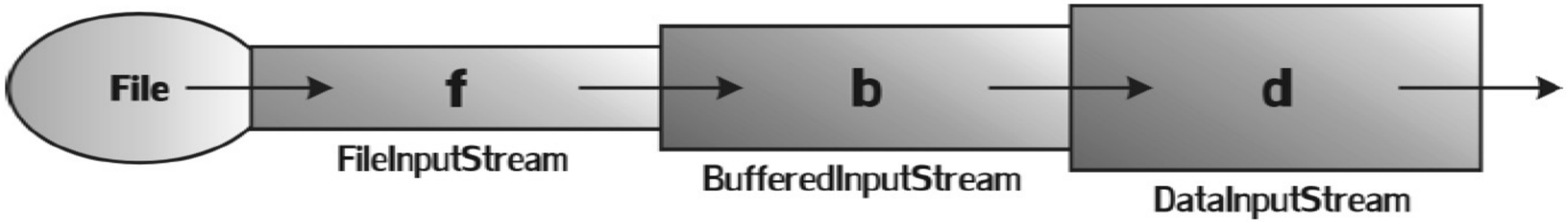
PrintStream

- เป็นคลาสที่ใช้ในการ*เขียนข้อความที่เป็น String ที่แปลงมาจากชนิดข้อมูลแบบ byte* อ็อบเจ็คที่ชื่อ out และ err ที่อยู่ในคลาสที่ชื่อ System เป็นตัวอย่างของอ็อบเจ็คที่ใช้คลาสนี้

PushbackInputStream

- คลาสนี้อนุญาตให้ส่งข้อมูลชนิด byte ที่เพิ่งอ่านมากลับไปยัง stream ได้

การเชื่อมต่อ Stream หลายชั้น



เราสามารถ*เชื่อมต่อ Stream ได้หลายชั้น* เช่น

```
FileInputStream f = new FileInputStream("text.dat");
BufferedInputStream b = new BufferedInputStream(f);
DataInputStream d = new DataInputStream(b);
```

คลาส DataInputStream

คลาส DataInputStream มีเมธอดในการอ่านข้อมูลชนิดต่าง ๆ ดังนี้

- `boolean readBoolean()`
- `byte readByte()`
- `char readChar()`
- `double readDouble()`
- `float readFloat()`
- `int readInt()`
- `long readLong()`
- `short readShort()`
- `String readUTF()`

คลาส DataOutputStream

DateOutputStream เป็น high-level Stream ซึ่งมีเมธอดในการเขียนข้อมูลชนิดต่างๆ ดังนี้

- `void writeBoolean(boolean b)`
- `void writeByte(int b)`
- `void writeByte(String s)`
- `void writeChar(int c)`
- `void writeDouble(double d)`
- `void writeFloat(float f)`
- `void writeInt(int i)`
- `void writeLong(long l)`
- `void writeShort(int s)`
- `void writeUTF(String s)`

ตัวอย่างที่ 6: การใช้คลาส DataOutputStream

```
import java.io.*;
public class DataWriterDemo {
        public static void main (String args[]) {
            try {
                    FileOutputStream fout;
                    DataOutputStream dout;
                    fout = new FileOutputStream("data.dat");
                    dout = new DataOutputStream(fout);

                    dout.writeInt(124);
                    dout.writeDouble(2.45);
                    dout.writeChar('a');
                    dout.writeUTF("test");

                    dout.close();
                    fout.close();
            } catch (IOException ex) {
                    System.out.println(ex.toString());
            }
        }
}
```

ขั้นตอนที่ 1 สร้างและต่อท่อ

ขั้นตอนที่ 2 เขียนข้อมูล

ขั้นตอนที่ 3 ปิดไฟล์และปิดท่อ

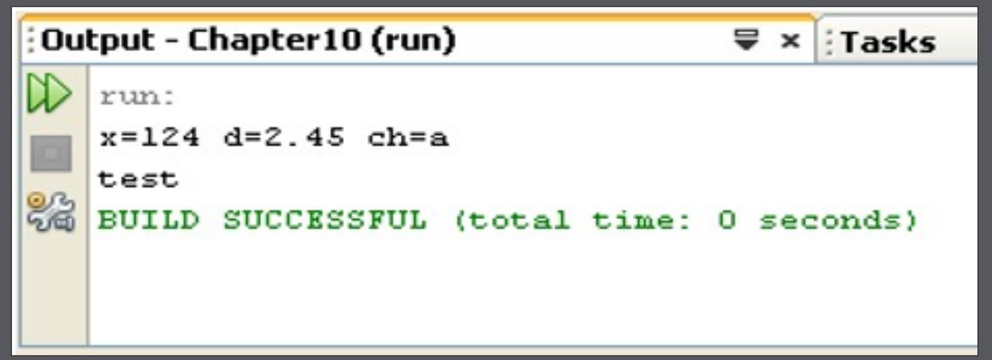
ตัวอย่างที่ 7: การใช้คลาส DataInputStream

```
import java.io.*;
public class DataReaderDemo {
   public static void main (String args[]) {
        try {
            FileInputStream fin;
            DataInputStream din;
            fin = new FileInputStream("data.dat");
            din = new DataInputStream(fin);

            int x = din.readInt();
            double d = din.readDouble();
            char ch = din.readChar();
            String line = din.readUTF();

            System.out.println("x= " + x + " d= "+ d + " ch= " + ch);
            System.out.println(line);

            din.close();
            fin.close();
        } catch (IOException ex) {
            System.out.println(ex.toString());
        }
   }
}
```

ขั้นตอนที่ 1 สร้างและต่อท่อ

ขั้นตอนที่ 2 อ่านข้อมูล

ขั้นตอนที่ 3 ปิดไฟล์และปิดท่อ



คลาสประเภท char stream ระดับสูงที่สำคัญ

BufferedReader และ BufferedWriter

- เป็นคลาสที่มีบัฟเฟอร์สำหรับชนิดข้อมูลแบบ char เพื่อให้สามารถ*อ่านหรือเขียนข้อมูลได้ใหญ่ขึ้น*

InputStreamReader และ OutputStreamWriter

- เป็นคลาสที่ใช้ในการ*แปลงชนิดข้อมูล*ระหว่างชนิดข้อมูล*แบบ char กับชนิดข้อมูลแบบอื่นๆ*

PrintWriter

- เป็นคลาสที่ใช้ในการ*เขียนข้อความที่เป็น String ที่แปลงมาจากชนิดข้อมูลแบบ char*

PushbackReader

- เป็นคลาสที่อนุญาตให้ส่งข้อมูลชนิด char ที่เพิ่งอ่านมากลับไปยัง stream ได้

การแปลงข้อมูลระหว่าง byte และ char

เราสามารถที่จะแปลง stream ระหว่างชนิดข้อมูลแบบ byte และชนิดข้อมูลแบบ char ได้โดยใช้

- คลาส InputStreamReader ซึ่งจะอ่านชนิดข้อมูลแบบ char แล้วแปลงเป็นชนิดข้อมูลแบบ byte
- คลาส OutputStreamWriter ซึ่งจะอ่านชนิดข้อมูลแบบ byte แล้วแปลงเป็นชนิดข้อมูลแบบ char

ตัวอย่างที่ 8: การเขียนไฟล์โดยอาศัย PrintWriter

```
import java.io.*;
public class FileWriter {
            public static void main(String args[]) {
                FileOutputStream fout;
                OutputStreamWriter oout;
                PrintWriter p;
                String line1 = "This is a test message";
                String line2 = "This is another line";
                try {
                    fout = new FileOutputStream("data.dat");
                    oout = new OutputStreamWriter(fout);
                    p = new PrintWriter(oout);
                    p.println(line1);
                    p.println(line2);
                    p.close();
                    oout.close();
                    fout.close();
                } catch (IOException ex) {
                    System.out.println(ex.toString());
                }
            }
}
```

ตัวอย่างที่ 8: การเขียนไฟล์โดยอาศัย PrintWriter

```
import java.io.*;
public class FileWriter {
    public static void main(String args[]) {
        String line1 = "This is a test message";
        String line2 = "This is another line";

        try (FileOutputStream fout = new FileOutputStream("data.dat");
        OutputStreamWriter oout = new OutputStreamWriter(fout);
        PrintWriter p = new PrintWriter(oout);) {

            p.println(line1);
            p.println(line2);

        } catch (IOException ex) {
            System.out.println(ex.toString());
        }
    }
}
```

// แนะนำให้เขียนโดยอาศัย try-with-resources Statement

ตัวอย่างที่ 8: การเขียนไฟล์โดยไม่อาศัย PrintWriter

```
import java.io.*;
public class FileWriter {
    public static void main(String args[]) {
        FileOutputStream fout;
        OutputStreamWriter oout;
        String line1 = "This is a test message";
        String line2 = "This is another line";
        try {
            fout = new FileOutputStream("data1.dat");
            oout = new OutputStreamWriter(fout);
            for (int i = 0; i < line1.length(); i++) {
               oout.write(line1.charAt(i));
            } oout.write('\n');
            for (int i = 0; i < line2.length(); i++) {
               oout.write(line2.charAt(i));
            }
            oout.close();
            fout.close();
        } catch (IOException ex) {
            System.out.println(ex.toString());
        }
    }
}
```

ตัวอย่างที่ 9: การอ่านไฟล์โดยอาศัย OutputStreamWriter

```
import java.io.*;

public class FileReader {
    public static void main(String args[]) {
        try {
            FileInputStream fin = new FileInputStream("data1.dat");
            InputStreamReader in = new InputStreamReader(fin);
            BufferedReader bin = new BufferedReader(in);

            System.out.println(bin.readLine());
            System.out.println(bin.readLine());

            bin.close();
            in.close();
            fin.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

หัวข้อ

- คลาส File
- โครงสร้างช่องทางการสื่อสาร
  - stream
    - คลาส Byte stream
    - คลาส Char stream
  - Node
    - โหนด Input Source
    - โหนด Output Sink
- การเชื่อมต่อ stream และ stream ระดับสูง
- คลาส Object stream
- อินเตอร์เฟส Serializable และ คีย์เวิร์ด Transient

ObjectStream

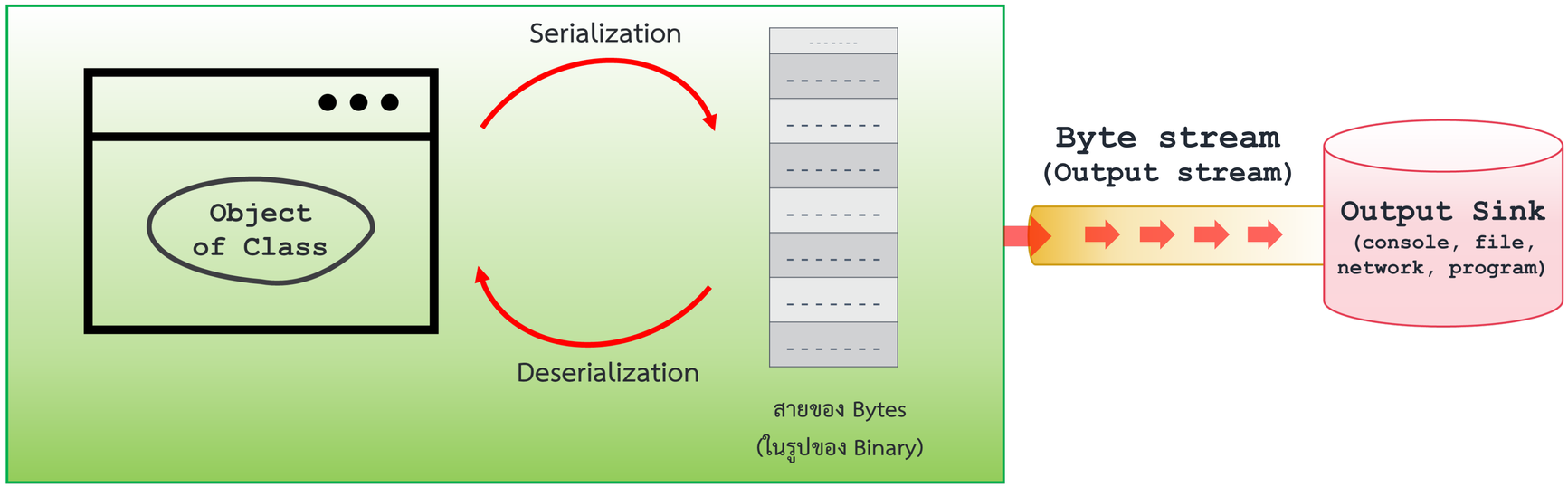
ภาษาจาวามีคลาสที่ใช้ในการรับและส่งข้อมูลของอ็อบเจ็ค ได้แก่ ObjectInputStream และ ObjectOutputStream โดยที่

- คลาส ObjectOutputStream เป็นคลาสที่ใช้ในการเขียนออปเจ็คใด ๆ ลงใน stream โดยมีเมธอด writeObject() ที่ใช้ในการเขียนข้อมูล
- ส่วนคลาส ObjectInputStream เป็นคลาสที่ใช้ในการอ่านออปเจ็คใดๆ มาจาก stream โดยมีเมธอด readObject() ที่ใช้ในการอ่านข้อมูล

ซึ่งมีเงื่อนไขดังต่อไปนี้

- อ็อบเจ็คที่จะสามารถส่งผ่าน stream ได้จะต้องเป็นอ็อบเจ็คของคลาสที่ implements อินเตอร์เฟส Serializable ซึ่งอยู่ในแพคเก็จ java.io
- อ็อบเจ็คบางประเภทจะไม่สามารถที่จะ implements อินเตอร์เฟส Serializable ได้ เนื่องจากข้อมูลของอ็อบเจ็คอาจเปลี่ยนแปลงได้ตลอดเวลา อาทิเช่น อ็อบเจ็คของคลาส Thread หรือ FileInputStream

อินเตอร์เฟส Serializable



Serialization คือ กระบวนการนำข้อมูล Object มาแปลงเป็นสายของ Bytes (ในรูปของ Binary) เพื่อให้สามารถส่งข้ามระบบเครือข่ายหรือบันทึกลงบนฐานข้อมูลหรือไฟล์ได้ ในขณะที่ Deserialization คือ กระบวนแปลงการย้อนกลับจากสาย Bytes ให้มีรูปแบบ Object จากที่ได้กล่าวไว้ว่าอ็อบเจ็คที่จะสามารถส่งผ่าน stream ได้จะต้องเป็นอ็อบเจ็คของคลาสที่ implements อินเตอร์เฟส Serializable ซึ่งมีรายละเอียดดังนี้

- Serializable คือ อินเตอร์เฟสที่ไม่มีเมธอด ซึ่งจะดำเนินการ Serialization ผ่านทาง JVM แบบอัตโนมัติ
- การทำ Serialization สามารถทำได้โดย implements อินเตอร์เฟส Serializable กับคลาสที่ต้องการเท่านั้น ซึ่งถือว่าง่ายและสะดวกต่อการใช้งาน แต่จะใช้ต้นทุนการการะประมวลผลค่อนข้างมาก
- กระบวนการ Serialization จะไม่มีการเรียก Constructor ของคลาสใดเลย

อินเตอร์เฟส Serializable

java.io

Interface Serializable

All Known Subinterfaces:

AdapterActivator, Attribute, Attribute, Attributes, BindingIterator, CertPathValidatorException.Reason, ClientRequestInfo, ClientRequestInterceptor, Codec, CodecFactory, Control, Current, Current, Current, CustomValue, DataInputStream, DataOutputStream, Descriptor, DHPrivateKey, DHPublicKey, DocAttribute, DomainManager, DSAPrivateKey, DSAPublicKey, DynAny, DynAnyFactory, DynArray, DynEnum, DynFixed, DynSequence, DynStruct, DynUnion, DynValue, DynValueBox, DynValueCommon, ECPrivateKey, ECPublicKey, ExtendedRequest, ExtendedResponse, Externalizable, IdAssignmentPolicy, IDLEntity, IDLType, IdUniquenessPolicy, ImplicitActivationPolicy, Interceptor, IORInfo, IORInterceptor, IORInterceptor_3_0, IRObject, Key, LifespanPolicy, Name, NamingContext, NamingContextExt, NotificationFilter, ObjectReferenceFactory, ObjectReferenceTemplate, ORBInitializer, ORBInitInfo, PBEKey, POA, POAManager, Policy, PolicyFactory, PrintJobAttribute, PrintRequestAttribute, PrintServiceAttribute, PrivateKey, PublicKey, QueryExp, RelationType, RemoteRef, RequestInfo, RequestProcessingPolicy, RSAMultiPrimePrivateCrtKey, RSAPrivateCrtKey, RSAPrivateKey, RSAPublicKey, RunTime, SecretKey, ServantActivator, ServantLocator, ServantManager, ServantRetentionPolicy, ServerRef, ServerRequestInfo, ServerRequestInterceptor, StreamableValue, SupportedValuesAttribute, ThreadPolicy, UnsolicitedNotification, ValueBase, ValueExp

---

```
public interface Serializable
```

Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.

To allow subtypes of non-serializable classes to be serialized, the subtype may assume responsibility for saving and restoring the state of the supertype's public, protected, and (if accessible) package fields. The subtype may assume this responsibility only if the class it extends has an accessible no-arg constructor to initialize the class's state. It is an error to declare a class Serializable if this is not the case. The error will be detected at runtime.

ตัวอย่างที่ 7: การเขียนออปเจ็คใดๆ ลงใน stream

```
import java.io.*;
public class Student implements Serializable {// object ที่ต้องการเขียนลงบน memory ต้อง Serialize
    public String name;
    public int id;
    public Student() { }
}
public class SerializeDemo {
   public static void main(String [] args) {
      Student s = new Student();
      s.name = "Taravichet Ti";    s.id = 60070050;
      try {
         FileOutputStream fOut = new FileOutputStream("dataStudent.dat");
         ObjectOutputStream oout = new ObjectOutputStream(fOut);
         oout.writeObject(s);
         oout.close();          fOut.close();
         System.out.printf("Serialized data is saved");
      } catch (IOException i) {
         i.printStackTrace();
      }
   }
}
```

ตัวอย่างที่ 8: การอ่านออปเจ็คใดๆ จาก stream

```
import java.io.*;
public class DeserializeDemo {
   public static void main(String [] args) {
      Student s = null;
      try {
         FileInputStream fin = new FileInputStream("dataStudent.dat");
         ObjectInputStream in = new ObjectInputStream(fin);
         s = (Student) in.readObject();     // แปลงชนิดข้อมูลจาก Object ไปเป็น Student
         in.close();
         fin.close();
      } catch (IOException i) {
         i.printStackTrace();
      } catch (ClassNotFoundException c) { // จะเกิดขึ้นในกรณีที่ตอนอ่านไฟล์แล้ว ไม่พบคลาส Student
         c.printStackTrace();
      }

      System.out.println("Deserialized Student.");
      System.out.println("Name: " + s.name);
      System.out.println("ID: " + s.id);
   }
}
```

```
Output:
Deserialized Student...
Name: Taravichet Ti
ID: 60070050
```

คีย์เวิร์ด Transient

อย่างไรก็ตาม นักศึกษาสามารถจะระบุคุณลักษณะของอ็อบเจ็คหรือคุณลักษณะของคลาสที่ไม่ต้องการจะเก็บหรือเรียกดูข้อมูลผ่าน stream ได้ โดยกำหนดให้มี modifier เป็นคีย์เวิร์ดที่ชื่อ transient ตัวอย่างเช่น

```
import java.io.*;
public class Student implements Serializable {
    public String name;
    public transient int id;
    public Student() { }
}
```

ตัวอย่างที่ 9: การเขียนออปเจ็คแบบ transient

```
import java.io.*;
public class Student implements Serializable {   // object ที่ต้องการเขียนลงบน memory ต้อง Serialize
    public String name;
    public transient int id;
    public Student() { }
}
public class SerializeDemo {
   public static void main(String [] args) {
      Student s = new Student();
      s.name = "Taravichet Ti";    s.id = 60070050;
      try {
         FileOutputStream fOut = new FileOutputStream("dataStudent.dat");
         ObjectOutputStream oout = new ObjectOutputStream(fOut);
         oout.writeObject(s);
         oout.close();          fOut.close();
         System.out.printf("Serialized data is saved");
      } catch (IOException i) {
         i.printStackTrace();
      }
   }
}
```

ตัวอย่างที่ 8: การอ่านออปเจ็คแบบ transient

```
import java.io.*;
public class DeserializeDemo {
   public static void main(String [] args) {
      Student s = null;
      try {
         FileInputStream fin = new FileInputStream("dataStudent.dat");
         ObjectInputStream in = new ObjectInputStream(fin);
         s = (Student) in.readObject();     // แปลงชนิดข้อมูลจาก Object ไปเป็น Student
         in.close();
         fin.close();
      } catch (IOException i) {
         i.printStackTrace();
      } catch (ClassNotFoundException c) { // จะเกิดขึ้นในกรณีที่ตอนอ่านไฟล์แล้ว ไม่พบคลาส Student
         c.printStackTrace();
      }

      System.out.println("Deserialized Student.");
      System.out.println("Name: " + s.name);
      System.out.println("ID: " + s.id);
   }
}
```

```
Output:
Deserialized Student...
Name: Taravichet Ti
ID: 0
```