OOP with Java

Yuanbin Wu
cs@ecnu

OOP with Java

- 通知
  - Project 6: 12 月 21 日晚 9 点
  - Project 7 :1 月 11 日晚 9 点
  - 12 月 28 日复习
  - 1 月 4 日考试（暂定，地点另行通知）
  - 答疑
    - 12 月 30 日 13： 00 – 15： 00
    - 理科楼 B911

- 复习
  - 异常处理
- 语法
  - 抛出异常 : throw
  - 处理异常 : try, catch
  - 异常对象 : Exception 类的子类
- 从方法中抛出异常
  - 方法的异常说明 :throws
  - 中断当前方法的执行，返回抛出的异常对象，在该方法的调用路径上寻找合适的 catch.

```
class SimpleException extends Exception { }

public class InheritingExceptions {

    public static void main(String[] args) {
        try {
            System.out.println("Throw SimpleException from f()");
            throw new SimpleException();
        } catch(Exception e) {
            System.out.println("Caught it!");
            System.out.println(e);
            System.out.println(e.printStackTrace(System.out));
        }
    }
}
```

```
bar() throws Type1Exception, Type2Exception{
    …
    throw new Type1Exception ();
    …
    throw new Type2Exception ();
}
```

```
foo() {
    try{
        …
        bar();
    }
    catch (Type1Exception e){
        ...
    }
    catch (Type2Exception e){
        ...
    }
}
```

I/O

- Introduction
- Path Object
- InputStream/OutputStream
- Reader/Writer
- Typical usage of I/O streams
- Object Serialization

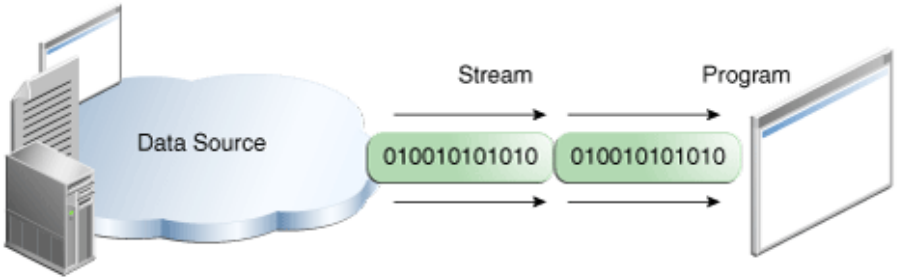
Introduction

- I/O
  - Input/output
  - 哪些行为有 I/O 操作？
- I/O 操作是复杂的
  - 多种交互的设备

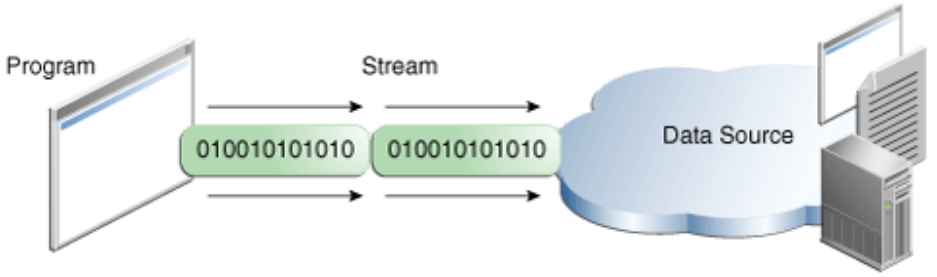
Introduction

- I/O 过程的抽象
  - 流 (Stream)

Input



Output



Introduction

- I/O 流
  - 数据的流向
    - 外界数据进入程序 (InputStream)
    - 程序数据进入外界 (OutputStream)
  - 数据的内容
    - 字节 (01 串): ByteArrayStream
    - 文件 : FileStream
    - 字符串 : StringStream
    - 对象 : ObjectStream

Introduction

- 核心 I/O Stream 操作
  - InputStream: read()
  - OutputStream: write()
  - close()
- 为生活带来便利的 I/O Stream 操作
  - 从文件读取一行 : readLine()
  - 读取一个基本类型 : readInt(), readDouble()…
  - 读取对象

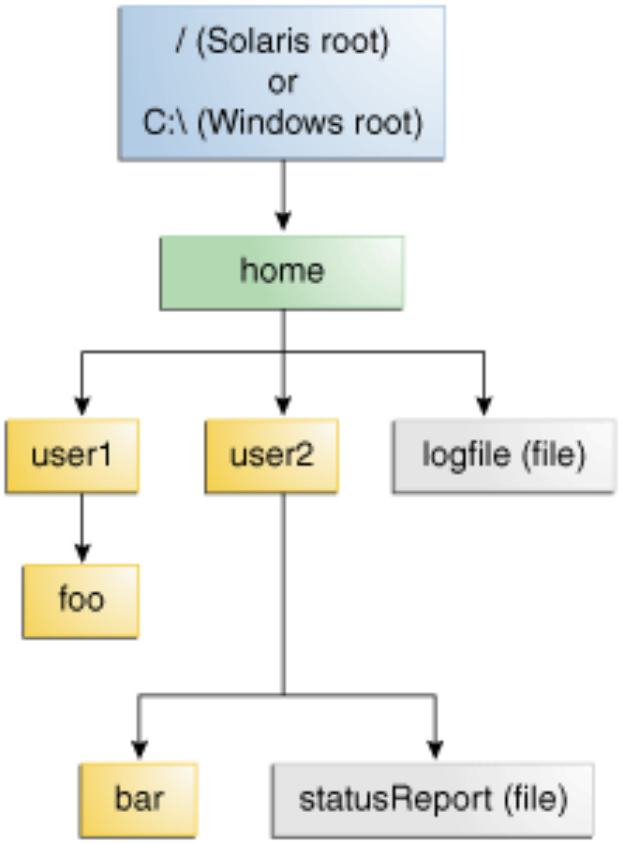
```
public static void fileOutput() throws IOException{
	String str = "hello world!";
	File file = new File("d:\\test2.txt");  // 创建文件
	if(!file.exists()){
		file.createNewFile();    // 如果文件不存在，则进行创建
	}
	FileOutputStream fOutput = new FileOutputStream(file);
	BufferedOutputStream bOutput = new BufferedOutputStream(fOutput);
	byte[] buffer = str.getBytes(); // 将字符串文本转换成字节数组
	bOutput.write(buffer);
	bOutput.close();
	fOutput.close();
}

public static void fileInput() throws IOException{
	File file = new File("d:\\test2.txt");   // 创建文件
	FileInputStream finput = new FileInputStream(file);
	BufferedInputStream bfinput = new BufferedInputStream(finput);
	int temp = 0;
	while((temp = bfinput.read())!= -1){    // 当 temp 为 -1 时，数据读取完毕
		System.out.print((char)temp);
	}
	bfinput.close();
	finput.close();
}
```

Path 接口

- File 类
  - Java 7 之前
- Path 接口
  - Java 7 引入，用于替代 File 类
  - File 类对象可通过 toPath() 得到对应的 Path 对象

Path 接口

- 文件路径
  - 绝对路径
    - C:/Documents/tmp/Hello.java
  - 相对路径
    - ../../tmp/Hello.java



Path 接口

- Path 接口
  - java.nio.file
  - 提供对文件路径字符串的操作

Path 接口

- 创建 Path 类型的对象

```
Path p1 = Paths.get("C:/Document/tmp/Hello.java");

Path p2 = FileSystems.getDefault().getPath("C:/Document/tmp/Hello.java");
```

Path 接口

```
// Microsoft Windows syntax
Path path = Paths.get("C:\\home\\joe\\foo");

// Solaris syntax
// Path path = Paths.get("/home/joe/foo");

System.out.format("toString: %s%n", path.toString());          // C:\home\joe\foo
System.out.format("getFileName: %s%n", path.getFileName());    // foo
System.out.format("getName(0): %s%n", path.getName(0));        // home
System.out.format("getNameCount: %d%n", path.getNameCount());  // 3
System.out.format("subpath(0,2): %s%n", path.subpath(0,2));    // home\joe
System.out.format("getParent: %s%n", path.getParent());        // home\joe\
System.out.format("getRoot: %s%n", path.getRoot());            // C:\
```

Files 类

- 包含文件操作的静态方法
  - 文件是否存在
  - 创建文件？
  - 删除文件
  - 拷贝 / 移动
  - 重命名
  - 列出目录下所有文件
  - ...

Files 类

- 判断文件是否存在

```
Path p = Paths.get("C:/Document/tmp/Hello.java");
System.out.println(Files.exists(p));
```

- 判断文件是否可读 / 可写 / 可执行

```
Path p = Paths.get("C:/Document/tmp/Hello.java");
System.out.println(Files.isReadable(p));
System.out.println(Files.isWritable(p));
System.out.println(Files.isExecutable(p));
```

Files 类

- 删除文件

```
try {
    Files.delete(path);
} catch (NoSuchFileException x) {
    System.out.format("%s: no such" + " file or directory%n", path);
} catch (DirectoryNotEmptyException x) {
    System.out.format("%s not empty%n", path);
} catch (IOException x) {
    // File permission problems are caught here.
    System.out.println(x);
}
```

Files 类

- 获得文件相关的信息

```
size(Path)
isDirectory(Path, LinkOption)
isRegularFile(Path, LinkOption...)
isSymbolicLink(Path)
getLastModifiedTime(Path, LinkOption...)
setLastModifiedTime(Path, FileTime)
getOwner(Path, LinkOption...)
setOwner(Path, UserPrincipal)
```

Stream

- I/O 流
  - InputStream 抽象类
    - read(), close()
  - OutputStream 抽象类
    - write(), close()
  - 每次读入 / 写出一个字节

InputStream

- 从不同的源头产生输入
  - Byte arrays
  - String objects
  - files
  - …
- 对应于不同的 InputStream 子类
  - ByteArrayInputStream
  - StringBufferInputStream
  - FileInputStream
  - ...

OutputStream

- 输出到不同的目的地
  - Byte arrays
  - String objects
  - files
  - …
- 对应于不同的 OutputStream 子类
  - ByteArrayOutputStream
  - FileOutputStream

Stream

InputStream

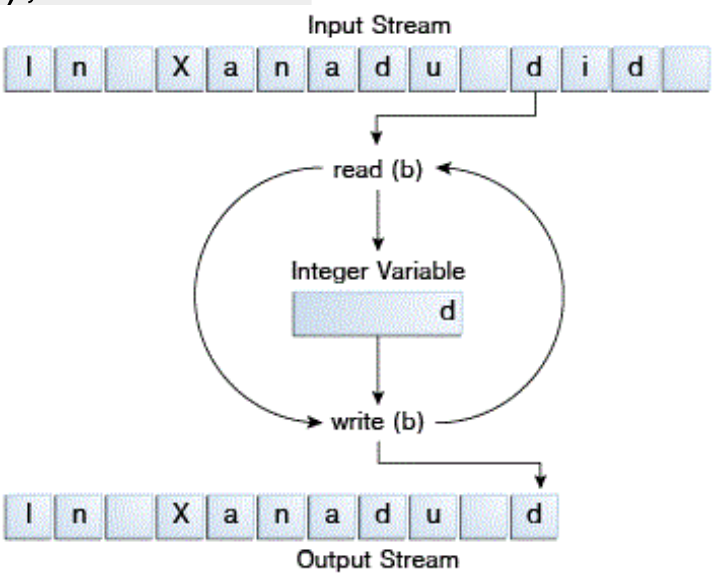
ByteArrayInputStream
StringBufferInputStream
FileInputStream
...

OutputStream

ByteArrayOutputStream

FileOutputStream
...

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            int c;
            while ((c = in.read()) != -1)
                out.write(c);
        } finally {
            if (in != null)
                in.close();
            if (out != null)
                out.close();
        }
    }
}
```



Stream

- 是否有效
  - InputStream/OutputStream 每次读 / 写一个字节
  - 缓冲区 (buffer) 可以帮助提高效率
- 是否易用
  - 有时需要读入 / 写出一个基本类型，而不是字节
  - readInt(), readDouble()
- 是否易读
  - 需要格式化的输出
  - println(), print(), printf(),..

Stream

- 希望：对每一种输入输出流都可以
  - 选择是否带缓冲
  - 选择读入字节还是基本类型
  - 提供格式化的输出
- How?

Stream

- 如何实现带缓冲的输入 / 输出流？
- 方案 1:
  - read() 方法增加参数
  - read(byte[ ], int off, int len)
- 缺点
  - buffer 长度需要用户指定
  - 用户需要知道更多实现细节

Stream

- 如何实现带缓冲的输入 / 输出流？
- 方案 2
  - 继承
  - BufferedByteArrayInputStream, BufferedByteArrayOutputStream
  - BufferedStringBufferInputStream, BufferedStringBufferOutputStream
  - BufferedFileInputStream, BufferedFileOutputStream
  - …
- 缺点
  - 类过多
  - 无法动态加载

Stream

- 如何实现带缓冲的输入 / 输出流？
- 方案 3
  - 组合
  - 定义 BufferedInputStream 类
  - 包含一个 InputStream 对象作为成员
  - 调用该 InputStream 对象的 read(byte[], int off, int len) 实现缓冲
  - 该调用在不同系统上有不同的优化，并且被封装

```
FileInputStream fin = new FileInputStream("xanadu.txt");
BufferedInputStream bf = new BufferedInputStream(fin);
bf.read(); // buffered read
```

```
FileOutputStream fout = new FileOutputStream("xanadu.txt");
BufferedOutputStream bf = new BufferedOutputStream(fout);
bf.write(1); // buffered write
```

```
ByteArrayInputStream bin = new ByteArrayInputStream("xanadu.txt".getBytes());
BufferedInputStream bf = new BufferedInputStream(bin);
bf.read(); // buffered read
```

```
ByteArrayOutputStream bout = new ByteArrayOutputStream("xanadu.txt".getBytes());
BufferedOutputStream bf = new BufferedOutputStream(bout);
bf.write(1); // buffered read
```

Stream

- 添加缓冲功能
  - BufferedInputStream
- 添加读取基本类型功能
  - DataInputStream
    - readInt(), readDouble(), readFloat(), ...

```
FileInputStream fin = new FileInputStream("xanadu.txt");
DataInputStream din = new DataInputStream(fin);
din.read();  din.readInt();  din.readDouble();
```

```
FileOutputStream fout = new FileOutputStream("xanadu.txt");
DataOutputStream dout = new DataOutputStream(fout);
dout.write(1); dout.writeInt(10), dout.writeDouble(3.14);
```

```
ByteArrayInputStream bin = new ByteArrayInputStream("xanadu.txt".getBytes());
DataInputStream din = new DataInputStream(bin);
din.read(); din.readInt();  din.readDouble();
```

```
ByteArrayOutputStream bout = new ByteArrayOutputStream("xanadu.txt".getBytes());
DataOutputStream bout = new DataOutputStream(bout);
bout.write(1); bout.writeInt(10), bout.writeDouble(3.14);
```

Stream

- 添加缓冲功能
  - BufferedInputStream
- 添加读取基本类型功能
  - DataInputStream
    - readInt(), readDouble(), readFloat(), ...
- 添加格式化输出功能
  - PrintStream
    - println(), print(),

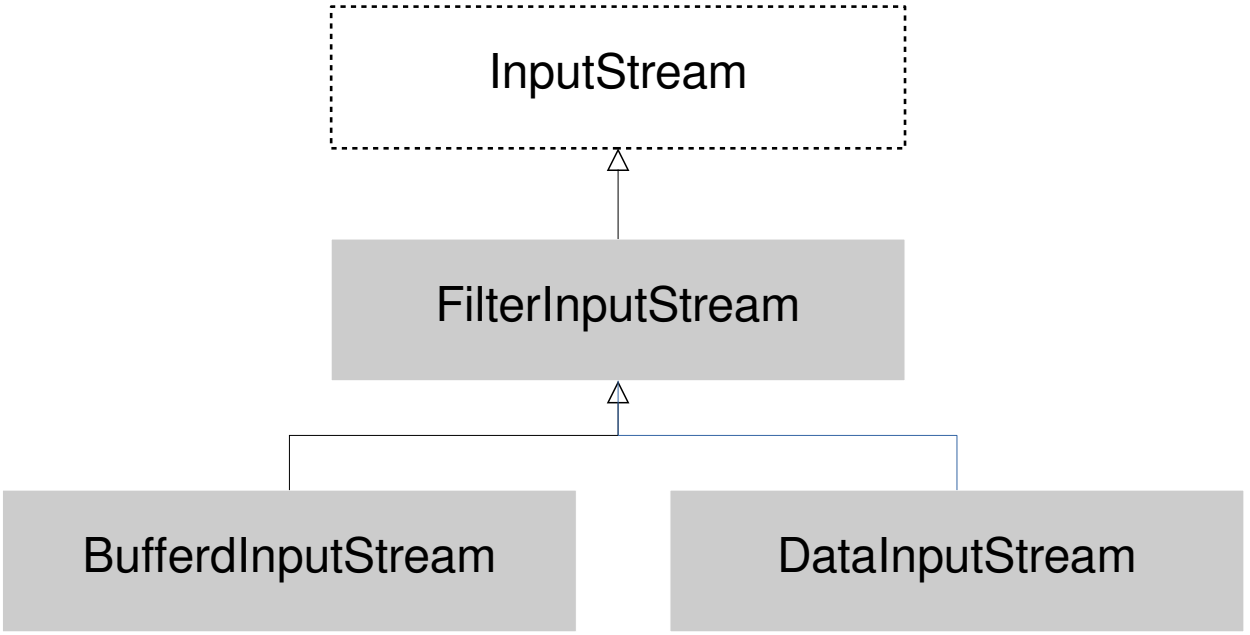
```
FileOutputStream fout = new FileOutputStream("xanadu.txt");
PrintStream ps = new PrintStream(fout);
ps.write(1); ps.println(“hello”); ps.print(“world”);
```

```
ByteArrayOutputStream bout = new ByteArrayOutputStream("xanadu.txt".getBytes());
PrintStream ps = new PrintStream(bout);
ps.write(1); ps.println(“hello”); ps.print(“world”);
```

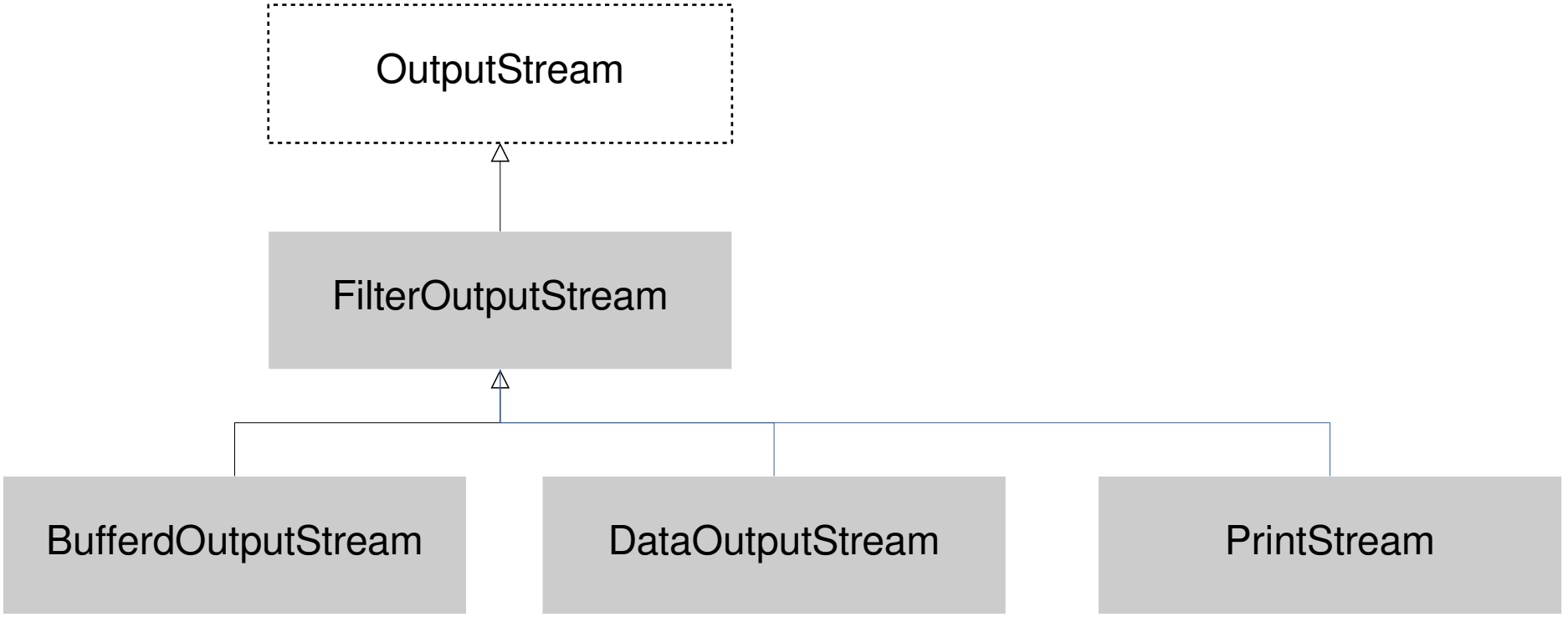
FilterInputStream

- 特点
  - InputStream 的子类
  - 构造函数：
    - protected FilterInputStream(InputStream in)
  - 包含一个 InputStream 类型的数据成员
  - 子类 :BufferedInputStream, DataInputStream

FilterInputStream



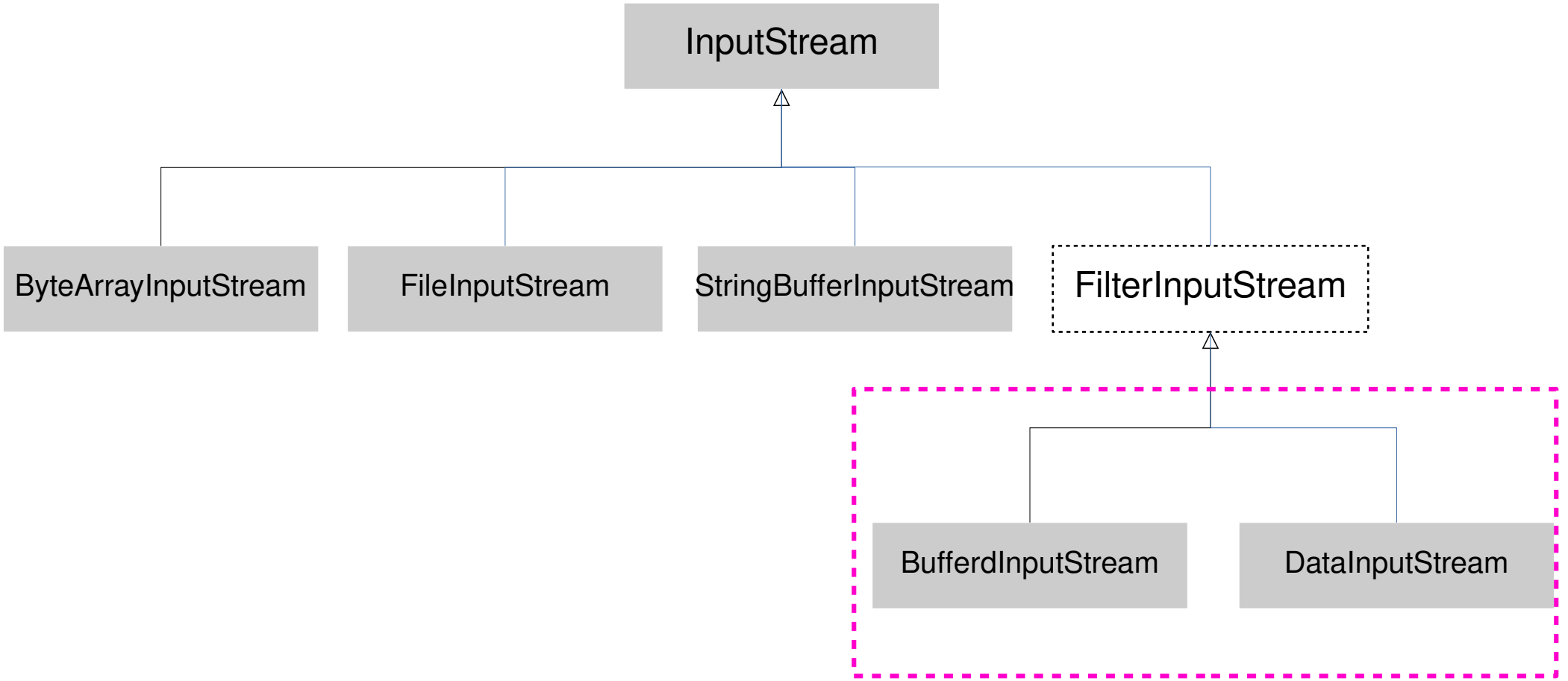
FilterOutputStream



```
FileInputStream fin = new FileInputStream("xanadu.txt");
BufferedInputStream bf = new BufferedInputStream(fin);
DataInputStream din = new DataInputStream(bf);

din.read();  din.readInt();  din.readDouble();
```

```
FileOutputStream fout = new FileOutputStream("xanadu.txt");
BufferedOutputStream bf = new BufferedOutputStream(fout);
DataOutputStream dout = new DataOutputStream(bf);

dout.write(1);  dout.writeInt(10);  dout.writeDouble(3.14);
```

Decorator pattern



总结

- Stream
  - InputStream, OutputStream
- 根据数据源的不同，分为
  - FileInputStream, ByteArrayInputStream …
  - FileOutputStream, ByteArrayOutputStream…
- 可以有不同的装饰器
  - BufferedInputStream, DataInputStream
  - PrintStream

```
public static void fileOutput() throws IOException{
	String str = "hello world!";
	File file = new File("d:\\test2.txt");  // 创建文件
	if(!file.exists()){
		file.createNewFile();    // 如果文件不存在，则进行创建
	}
	FileOutputStream fOutput = new FileOutputStream(file);
	BufferedOutputStream bOutput = new BufferedOutputStream(fOutput);
	byte[] buffer = str.getBytes(); // 将字符串文本转换成字节数组
	bOutput.write(buffer);
	bOutput.close();
	fOutput.close();
}

public static void fileInput() throws IOException{
	File file = new File("d:\\test2.txt");   // 创建文件
	FileInputStream flutput = new FileInputStream(file);
	BufferedInputStream blutput = new BufferedInputStream(flutput);
	int temp = 0;
	while((temp = blutput.read())!= -1){    // 当 temp 为 -1 时，数据读取完毕
		System.out.print((char)temp);
	}
	blutput.close();
	flutput.close();
}
```

Reader / Writer

- InputStream, OutputStream
  - 每次读入 / 写出一个字节
- Reader, Writer
  - 每次读入 / 写出一个字符
    - Utf-16
    - 每次读入 / 写出 16bit，或者 32bit

Reader / Writer

读写字节	读写字符
InputStream	Reader
OutputStream	Writer
FileInputStream	FileReader
FileOutputStream	FileWriter
StringBufferInputStream	StringReader
(no corresponding class)	StringWriter
ByteArrayInputStream	CharArrayReader
ByteArrayOutputStream	CharArrayWriter

Reader / Writer decorators

读写字节	读写字符
FilterInputStream	FilterReader
FilterOutputStream	FilterWriter
BufferedInputStream	BufferedReader
BufferedOutputStream	BufferedWriter
DataInputStream	DataInputStream
DataOutputStream	DataOutputStream
PrintStream	PrintWriter

Typical uses of I/O streams

- 例子

```
import java.io.*;
public class BufferedInputFile {
	// Throw exceptions to console:
	public static String read(String filename) throws IOException {
		// Reading input by lines:
		BufferedReader in = new BufferedReader(new FileReader(filename));
		String s;
		StringBuilder sb = new StringBuilder();
		while((s = in.readLine())!= null)
			sb.append(s + "\n");
		in.close();
		return sb.toString();
	}
	public static void main(String[] args) throws IOException {
		System.out.print(read("BufferedInputFile.java"));
	}
}
```

```
import java.io.*;
public class MemoryInput {
	public static void main(String[] args) throws IOException {
		StringReader in = new StringReader(
				BufferedInputFile.read("MemoryInput.java"));
		int c;
		while((c = in.read()) != -1)
			System.out.print((char)c);
	}
}
```

```
import java.io.*;
public class FormattedMemoryInput {
	public static void main(String[] args) throws IOException {
		try {
			DataInputStream in = new DataInputStream(
			new ByteArrayInputStream(
					BufferedInputFile.read(
						"FormattedMemoryInput.java").getBytes()));
			while(true)
				System.out.print((char)in.readByte());
		} catch(EOFException e) {
			System.err.println("End of stream");
		}
	}
}
```

```
import java.io.*;
public class TestEOF {
	public static void main(String[] args) throws IOException {
		DataInputStream in = new DataInputStream(
			new BufferedInputStream(
			new FileInputStream("TestEOF.java")));
		while(in.available() != 0)
			System.out.print((char)in.readByte());
	}
}
```

```
import java.io.*;
public class BasicFileOutput {
	static String file = "BasicFileOutput.out";
	public static void main(String[] args) throws IOException {
		BufferedReader in = new BufferedReader(
			new StringReader(
			BufferedInputFile.read("BasicFileOutput.java")));
		PrintWriter out = new PrintWriter(
			new BufferedWriter(new FileWriter(file)));
		int lineCount = 1;
		String s;
		while((s = in.readLine()) != null )
			out.println(lineCount++ + ": " + s);
		out.close();
		// Show the stored file:
		System.out.println(BufferedInputFile.read(file));
	}
}
```

Short cut:
PrintWriter out = new PrinterWriter(file);

```
import java.io.*;
public class StoringAndRecoveringData {
	public static void main(String[] args) throws IOException {
		DataOutputStream out = new DataOutputStream(
			new BufferedOutputStream(new FileOutputStream("Data.txt")));
		out.writeDouble(3.14159);
		out.writeUTF("That was pi");
		out.writeDouble(1.41413);
		out.writeUTF("Square root of 2");
		out.close();
		DataInputStream in = new DataInputStream(
			new BufferedInputStream(new FileInputStream("Data.txt")));
		System.out.println(in.readDouble());
		// Only readUTF() will recover the
		// Java-UTF String properly:
		System.out.println(in.readUTF());
		System.out.println(in.readDouble());
		System.out.println(in.readUTF());
	}
}
```

问题： Why not overloading?

Standard I/O

- System.out
  - PrintStream
- System.err
  - PrintStream
- System.in
  - 需要一些预处理

```
import java.io.*;
public class Echo {
	public static void main(String[] args) throws IOException {
		BufferedReader stdin = new BufferedReader(
			new InputStreamReader(System.in));
		String s;
		while((s = stdin.readLine()) != null && s.length()!= 0)
			System.out.println(s);
	}
}
```