OPEN STREAM

213-04.IOストリーム

Make IT your Success

目次（1/2）

ストリームの概念
- ストリームとは
- Streamクラス図

OutputStream
- OutputStreamクラスの種類
- 単独書き込み系ストリーム
- フィルタ系ストリーム
- フィルタ系ストリームのイメージ
- OutputStreamのクラス図

オブジェクトの直列化
- フィルタ系ストリームの利用

目次（2/2）

文字の読み書き(文字ストリーム)クラス
- Writer
- OutputStream/InputStream/Writer/Reader

java.nio
- java.nioとは
- Pathオブジェクト
- Pathsクラス
- Filesクラス
- Files.writeメソッド
- StandardOpenOptionクラス

この講義について

目的：

- ストリームの概念について学び、ファイルやメモリへの読み書きの流れについて学ぶ。
- OutputStream（InputStream）の種類として何があるのかを学ぶ。
- java.nioの利点について学ぶ。

ゴール：

文字、byte単位での入出力を適切に行うことができる。

ストリームの概念

ストリームとは

- 「ファイル・バッファ・ネットワークの抽象概念」

  ファイル・・・ファイルへの読み書き

  バッファ・・・メモリ上のバイト配列への読み書き

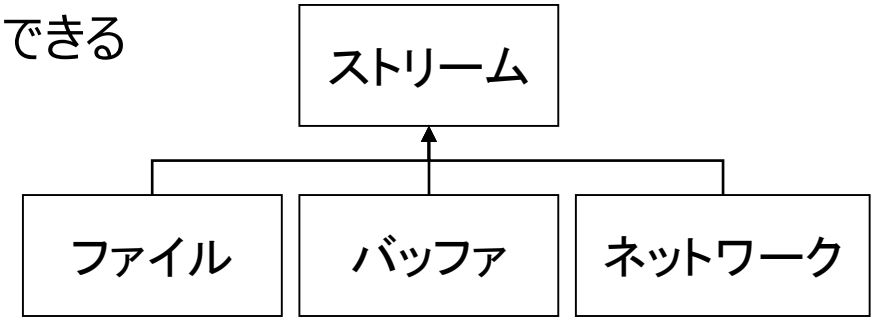
  ネットワーク・・・ネットワーク上の送受信

  ファイルへの読み書きも、メモリへの読み書きも、ネットワーク上の送受信も、一本のデータの流れ。byte単位(もっと細かく分解するとbit単位)のデータの流れで処理を行う。

  例). ファイルに一文字づつ書き込む。インターネットに一文字づつ通信を流す。

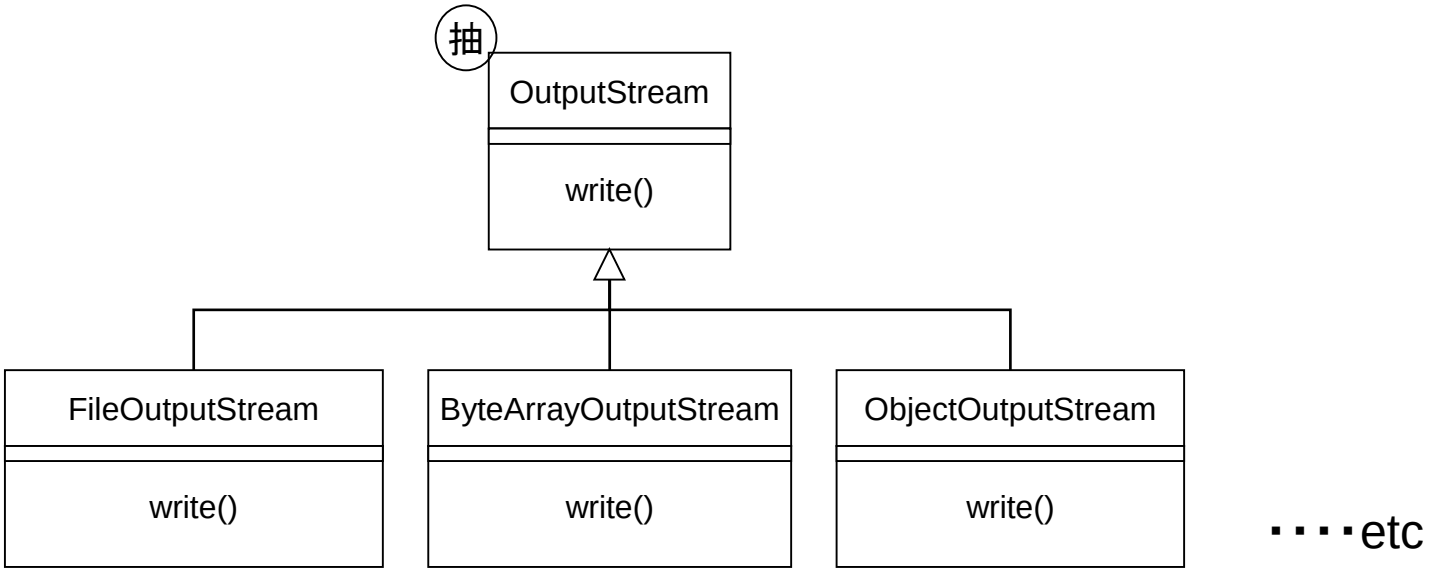
  →**ストリーム（流れ）**

- これらに対してすべて透過的に操作できる



Streamクラス図

(例：出力クラスの継承図　…入力クラスの継承図も同じ)



OutputStream

OutputStreamクラスの種類

（主に2種類に分けられる）

- 単独で出力できる系　　＝出力先を知っている
  →単独書き込み系ストリーム
- 単独で出力できない系　＝出力先を知らない
  (コンストラクタで出力先を知っているストリームを渡す)
  →フィルタ系ストリーム

※入力系(InputStream)も同じ種類(単独出力系とフィルタ系)に別けられる

単独書き込み系ストリーム

特徴

- 出力先を知っている
- write()で書けるのは、byte、byte配列
- close()・・・リソース開放
- flush()・・・OutputStreamクラスから継承したflushは何もしない

代表的なクラス

- FileOutputStream
  1byteもしくはbyte配列をファイルへ書出しするクラス。
- ByteArrayOutputStream
  1byteもしっくはbyte配列をバッファ(byte配列)へ書出しするクラス。

※入力系(InputStream)も同じ

フィルタ系ストリーム

特徴

- 出力先を知らないのでコンストラクタで何か出力先を知っているストリームを渡している
- IO発生の回数が減る
- close()・・・リソース開放
- flush()・・・バッファ上のものを強制書き込み(※書き間違い注意　×flash　○flush)

代表的なクラス

- BufferedOutputStream

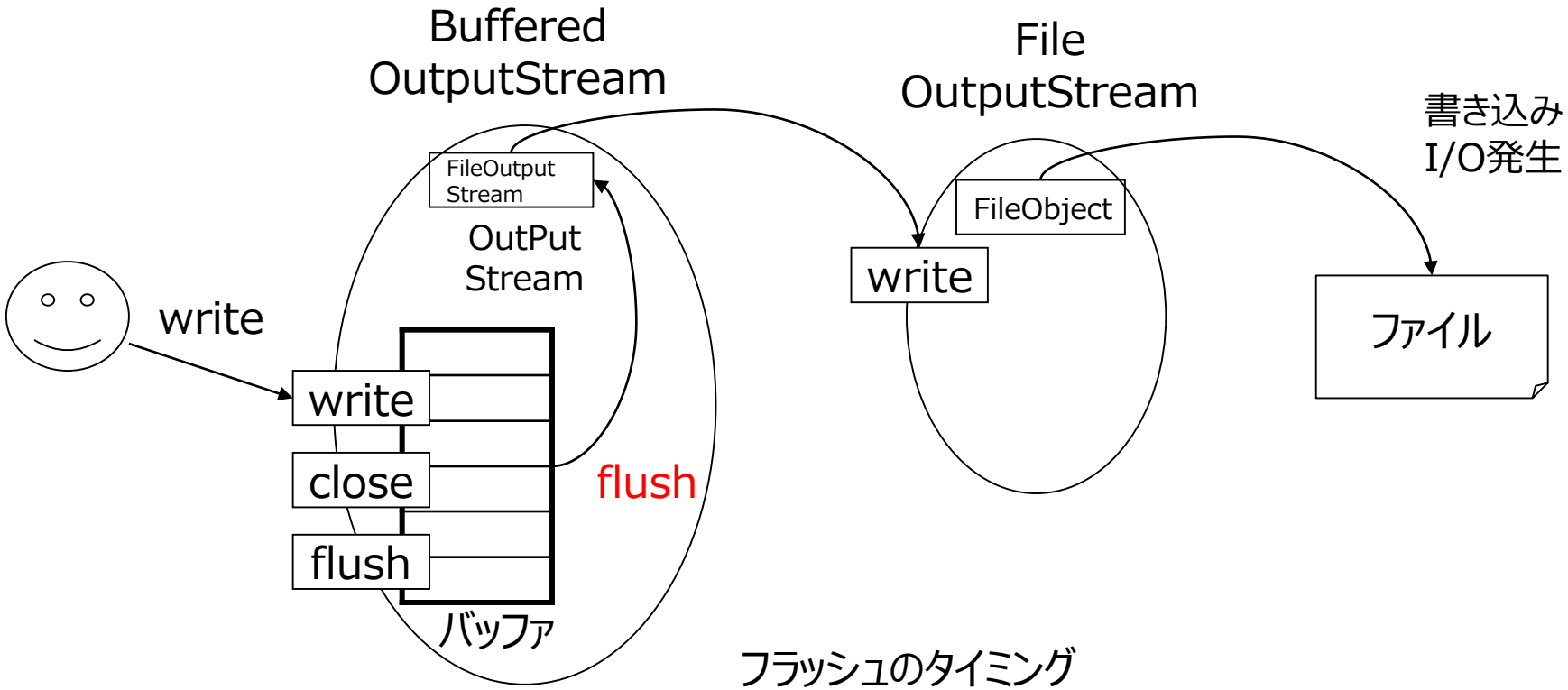
  バッファを持ち、バイト単位ではなく複数バイトをまとめて書き込むことができる。処理効率の向上が期待できる。
- PrintStream

  引数にいろいろな型を指定できる。
- ObjectOutputStream

  オブジェクトを永続化できる。

※入力系(InputStream)も同じ

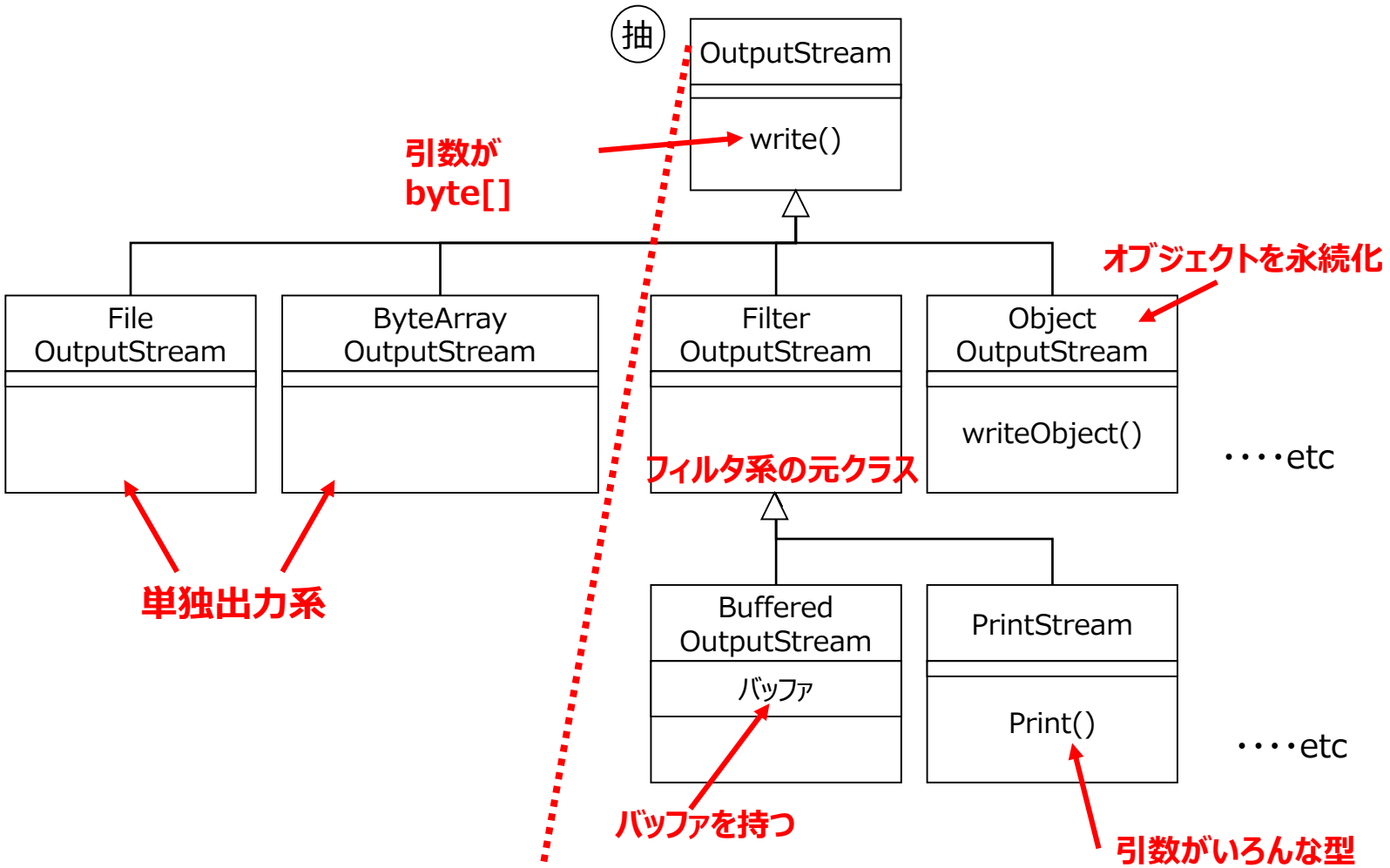
フィルタ系ストリームのイメージ



フラッシュのタイミング

1. Close()した時
2. バッファがいっぱいになったとき
3. Flush()を呼んだ時

OutputStreamのクラス図



◎Javadocでフィルタ系のストリームのコンストラクタを見てみる

オブジェクトの直列化

直列化(シリアライズ)

- インスタンスの情報をストリームで保存する
  →ファイルやバイト配列に格納、ネットワークで送ったり出来る

永続化

- プログラムが終了してもインスタンスの状態が消滅せず残っていること
- 直列化は永続化の一つ(他にDBに保存するなど)

オブジェクトの直列化

フィルタ系ストリームの利用

Dogクラスのオブジェクト(インスタンス)を直列化してファイルに保存して、また元のオブジェクトに戻す。

- 直列化　　…　シリアライズ
- 元に戻す　　…　デシリアライズ

ポイント

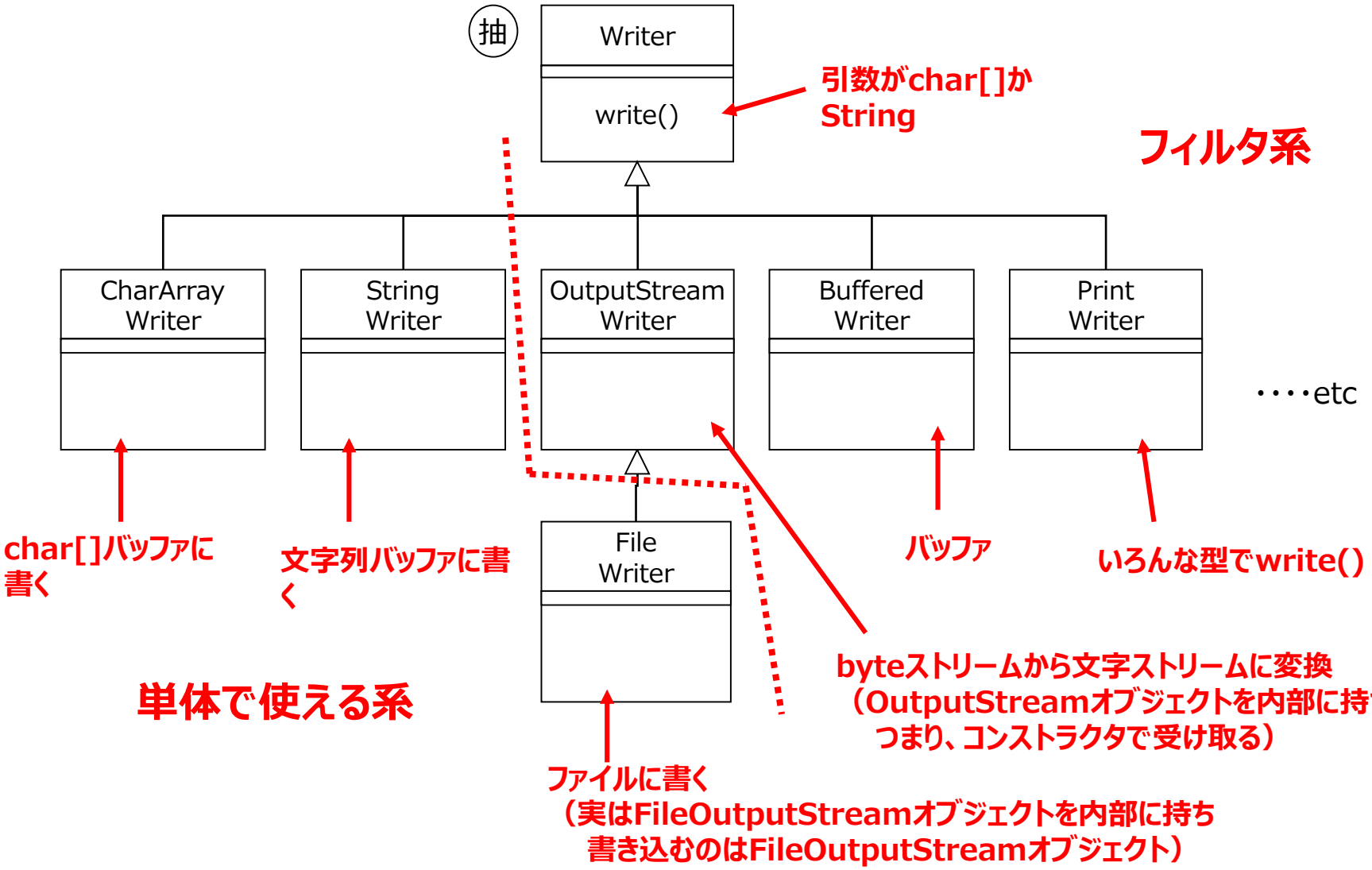
- 出力先を知らないのでコンストラクタで出力先を知っているストリームを渡している
- ObjectOutputStreamを使う
- 直列化する対象にSerializableインタフェースが必要

  →Serializableインタフェース　　…・マーカブルインタフェース

※ここまでStreamについて引数はbyte[]かObject

文字の読み書き(文字ストリーム)クラス

Writer



文字の読み書き(文字ストリーム)クラス

Writer

特徴

- char[]をbyte[]に変換して、OutputStreamのwriter()で書く
- Stringやchar[]に変換して取り出す（読む）
- 出力先はCharArrayWriter
- OutputStreamWriterは Unicode → 指定した文字コードへの変換

文字ストリームクラス（Writer/Reader）

- Writer(文字ストリーム) と OutputStream(byteストリーム) の違い。
  Writerは内部でOutputStreamを使用して書き込みを行う。
  Writerはchar[] をbyte[] に変換した後で、OutputStream のwrite() に渡す。

文字の読み書き(文字ストリーム)クラス

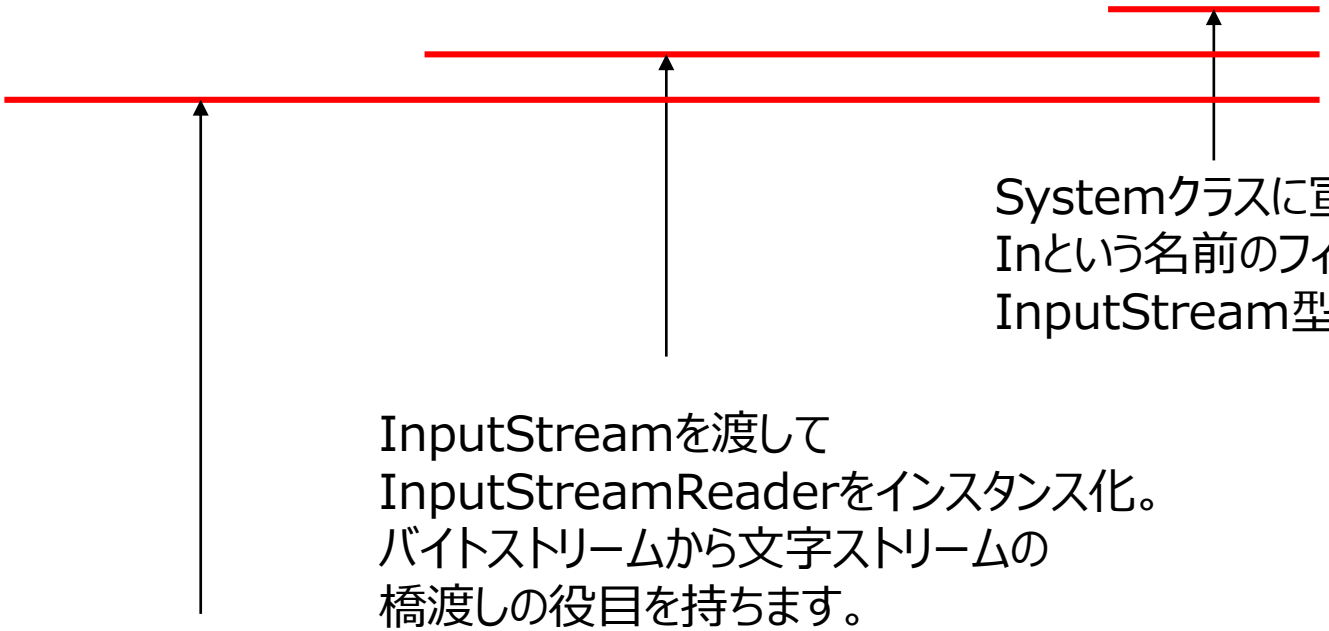
OutputStream/InputStream/Writer/Reader

	byte単位	文字単位
出力	OutputStream ・単独書き込み系 ・フィルタ系	Writer ・単独書き込み系 ・フィルタ系
入力	InputStream ・単独書き込み系 ・フィルタ系	Reader ・単独書き込み系 ・フィルタ系

文字の読み書き(文字ストリーム)クラス

例：

```
BufferedReader reader =
    new BufferedReader(new InputStreamReader(System.in));
```



Systemクラスに宣言された
Inという名前のフィールド。
InputStream型です。

InputStreamを渡して
InputStreamReaderをインスタンス化。
バイトストリームから文字ストリームの
橋渡しの役目を持ちます。

InputStreamReaderを渡して
BufferedReaderをインスタンス化。
バッファを利用して読み込みます。

java.nioとは

New I/Oのこと。

java.ioパッケージの入出力機能を補足する機能が用意されている。

主に、ファイルパス、ディレクトリパスを管理するjava.nio.file.Pathクラス。

ファイル、ディレクトリ操作を行うjava.nio.file.Filesクラスを使う。

java.io.fileの欠点

- 多くのメソッドで失敗時の適切な例外がスローされないため、適切なエラー・メッセージが取得できない。
  例).ファイルの削除が失敗した場合に、”削除が失敗した” 例外が発生するが、その理由が、“ファイルが存在しない” のか “ユーザーに権限がない” のか、他の問題があるのか が判別できない。
- 異なるプラットフォーム間で一貫していないメソッドがある。(renameメソッドの動作が異なるなど)
- シンボリック・リンクが十分にサポートされていない。
- ファイル権限、ファイル所有者、その他のセキュリティ属性に対して、サポートが十分では無い。
- 多くのファイル操作関係のメソッドにスケーラビリティがない。

→ クロスプラットフォーム、クラウドなどに対応したI/Oストリームクラスが求められる。

Pathオブジェクト

ファイル・システム内のファイルを特定するために使用可能なオブジェクト。
その場所にファイルが本当に存在するかどうかという情報は含まない。
ファイルにバイト列を読み書きする場合は、Pathの情報が必要になる。
※java.io.fileでは、Fileオブジェクト（ファイルパスの文字列のままで管理）を利用していた。

Pathsクラス

パス文字列またはURIを変換することによってPathオブジェクトを返すクラス。メソッドはstaticのみ。

```
Path path = Paths.get("C:¥¥dir¥¥hoge.txt");
Path path = Paths.get("C:", "dir", "hoge.txt");
Path path = FileSystems.getDefault().getPath("C:¥¥dir¥¥hoge.txt");
```

※すべて同じ結果となる。

※現在ではファイルパスを表すのにはFileではなくPathを使うことが推奨されている。
古いAPIでFileしか引数に取らないものがあるなど、Fileを使わざるを得ない場合があるが、Pathに用意されているtoFile()を用いて変換することも可能。

Filesクラス

ファイルやディレクトリなどの操作を行うstaticメソッドを持つクラス。

Filesクラスの各メソッドは、基本的にPathを引数に取る。

```
// Pathオブジェクトの取得
Path path = Paths.get(“foo.txt”);
// Pathオブジェクトを対象にファイル操作(ファイル作成)
Files.createFile(path);
```

主なメソッド

write	…	バイトをファイルに書き込む
createFile	…	ファイルの作成
createDirectory	…	ディレクトリの作成
delete	…	ファイル、ディレクトリの削除
move	…	移動・リネーム
exists	…	存在チェック

他にも

newInputStream、newBufferedReader、newOutputStreamメソッドなどを用いて
InputStream、BufferedReader、OutputStream オブジェクトを取得することも可能

→ ioストリームクラスを用いてのファイル操作も行える

Files.writeメソッド

引数で

- **Pathオブジェクト**
- **書込む内容 … byte[] または Iterable<? extends CharSequence>**
- **ファイルオプション … OpenOption**

を受け取り、Pathで指定されたファイルに対して byte[] または Iterable <? extends CharSequence> を OpenOptionで指定したオプション で書き込む。

writeメソッドは、

- すべてのバイトが書き込まれたとき、
- または 入出力エラーか 他の実行時例外がスローされたとき、

必ずそのファイルをクローズする。

※ Iterable <? extends CharSequence> の説明

・Iterable …… 拡張forに渡せるデータのかたまりのオブジェクト

・ <? extends CharSequence> …… ?クラスで限定したジェネリクス

? クラスはCharSequenceの子クラス

つまり、List<String>などを指す

StandardOpenOptionクラス

ファイルを開く、または作成する方法を指定するオブジェクト。（ファイルオプションを指定する）

APPEND：ファイルが WRITE 用に開かれた場合、バイトはファイルの先頭ではなく最後に書き込まれる。
CREATE　：ファイルが存在しない場合は新しいファイルを作成する。
READ　　：読み込みアクセス用に開く。
WRITE　　：書き込みアクセス用に開く。
DELETE_ON_CLOSE：閉じるときに削除する。
TRUNCATE_EXISTING：ファイルがすでに存在し、WRITEアクセス用に開かれた場合、その長さが0に切り詰められる。

など…