上級プログラミング１（第９回）

工学部　情報工学科
杉本　徹

今日のテーマ

- 入出力に関わるプログラミング
- 例外処理
- マルチスレッドプログラミング

入出力に関わるプログラミング

TXTファイルからデータを読み出そう

```
import java.io.*;
class Smpl0501 {
	public static void main(String[] args){
		BufferedReader bf=null;   String strData="";
		try{	FileInputStream fI= new FileInputStream("Sample.txt");
			InputStreamReader iS=new InputStreamReader(fI,"SJIS");
			bf=new BufferedReader(iS);
			while((strData=bf.readLine())!=null){
				System.out.println(strData);
			}
		}catch(IOException e){
			e.printStackTrace();
		}finally{
			try {
				bf.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}

	}
}
```

先週の復習(import文)

```
import java.io.*;

class Smpl0501 {
    public static void main(String[] args){

            .....
          ファイル入出処理の記述
            .....
    }
}
```

ファイル入出力用のクラスを
使うのでそのパッケージを
インポートしておく

参考：Java標準ライブラリの主要パッケージ

- java.lang（Javaの基本的なクラス群）
  - Objectクラス，Stringクラス，Systemクラスなど
  - java.langパッケージのクラスはimport無しで使える
- java.io（入出力関連のクラス群）
  - BufferedReader, InputStream, File など
- java.util（様々な有用なクラス群）
  - ArrayList, HashMap, Date, Random など
- java.net（ネットワーク関連のクラス群）
  - Socket, URL, HttpURLConnection など
- java.awt（UI，グラフィックス関連のクラス群）
  - Component, Image, Graphics, Font など

テキストファイルの読み出し

```
String line="";

FileInputStream fi
            = new FileInputStream("S
InputStreamReader is
            =new InputStreamReader(fi,"SJIS");
BufferedReader br
            =new BufferedReader(is);
while((line=br.readLine())!=null){
            System.out.println(line);
}
```

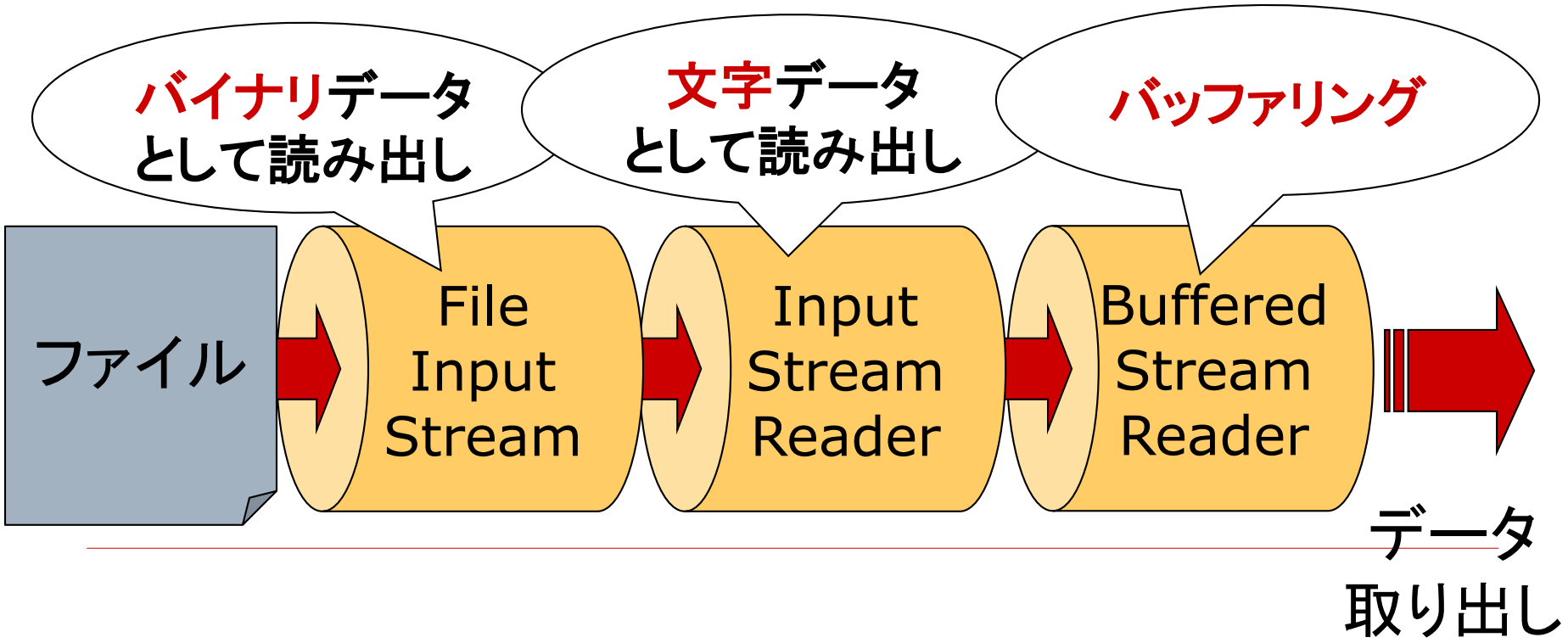
バイナリデータ
として読み出し

文字データ
として読み出し

バッファリング

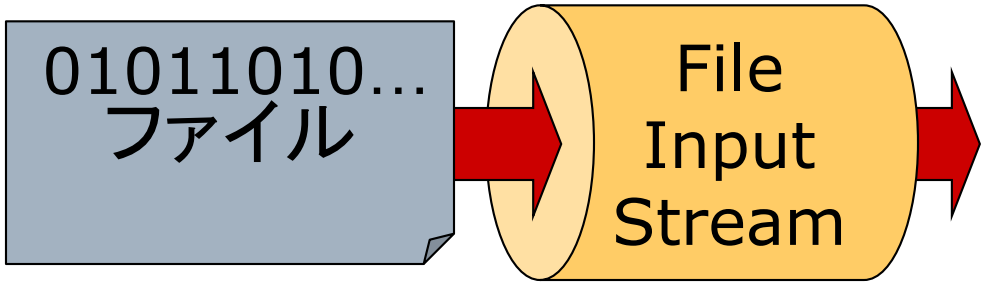
テキストファイルの読み込み関係のクラス

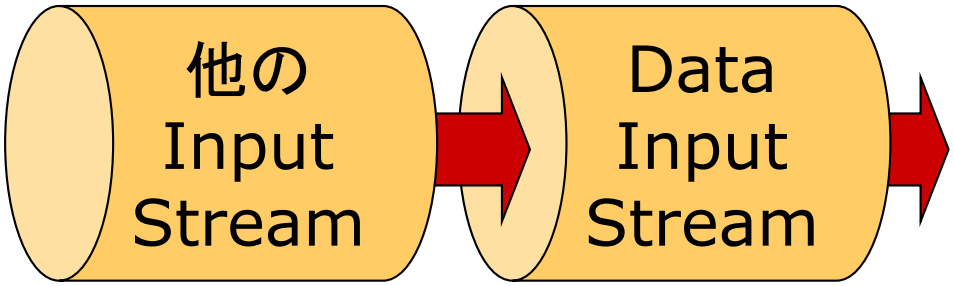
テキストファイルからデータを読み込むときには、
通常、三段構えで行う



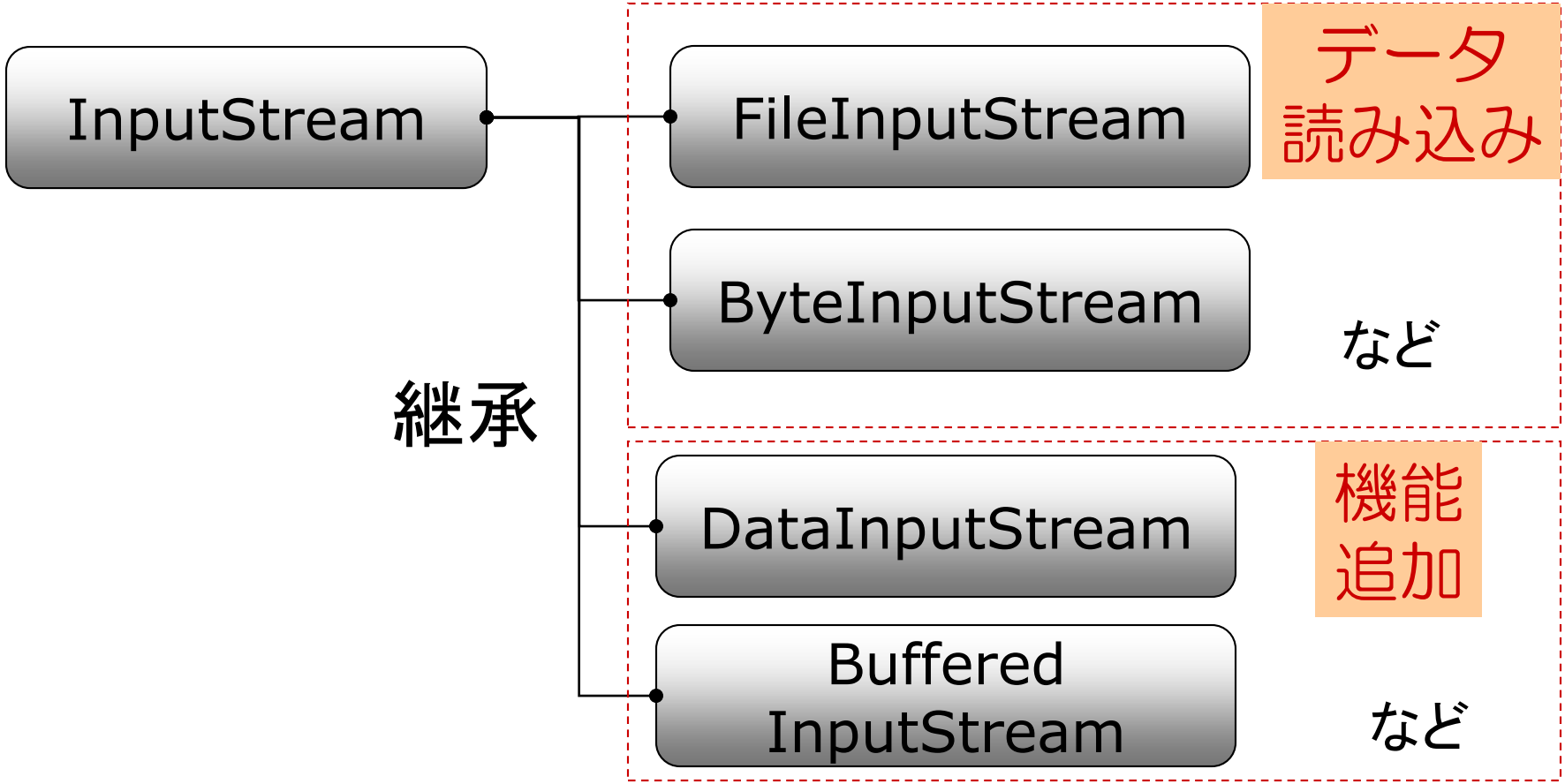
バイト入力ストリーム

byte単位でプログラムに渡すデータの流れ



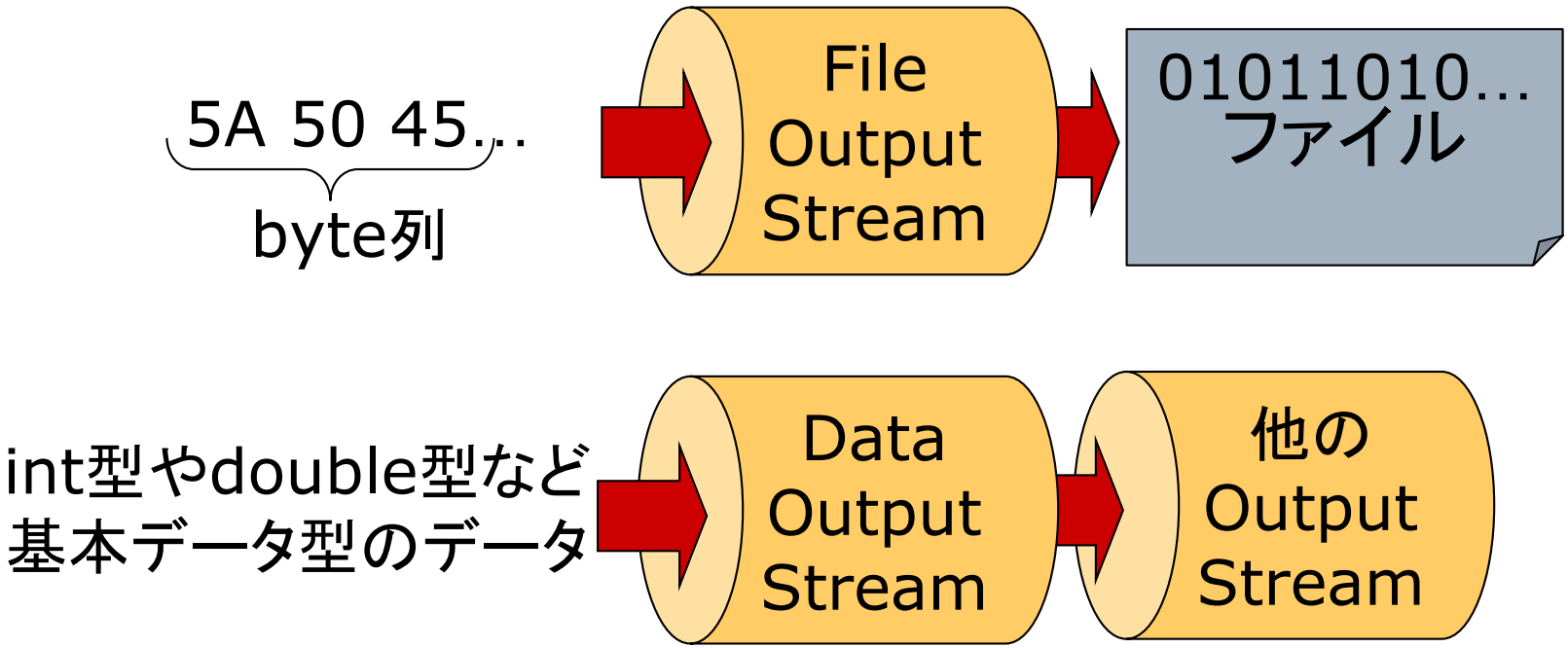


バイト入力ストリーム

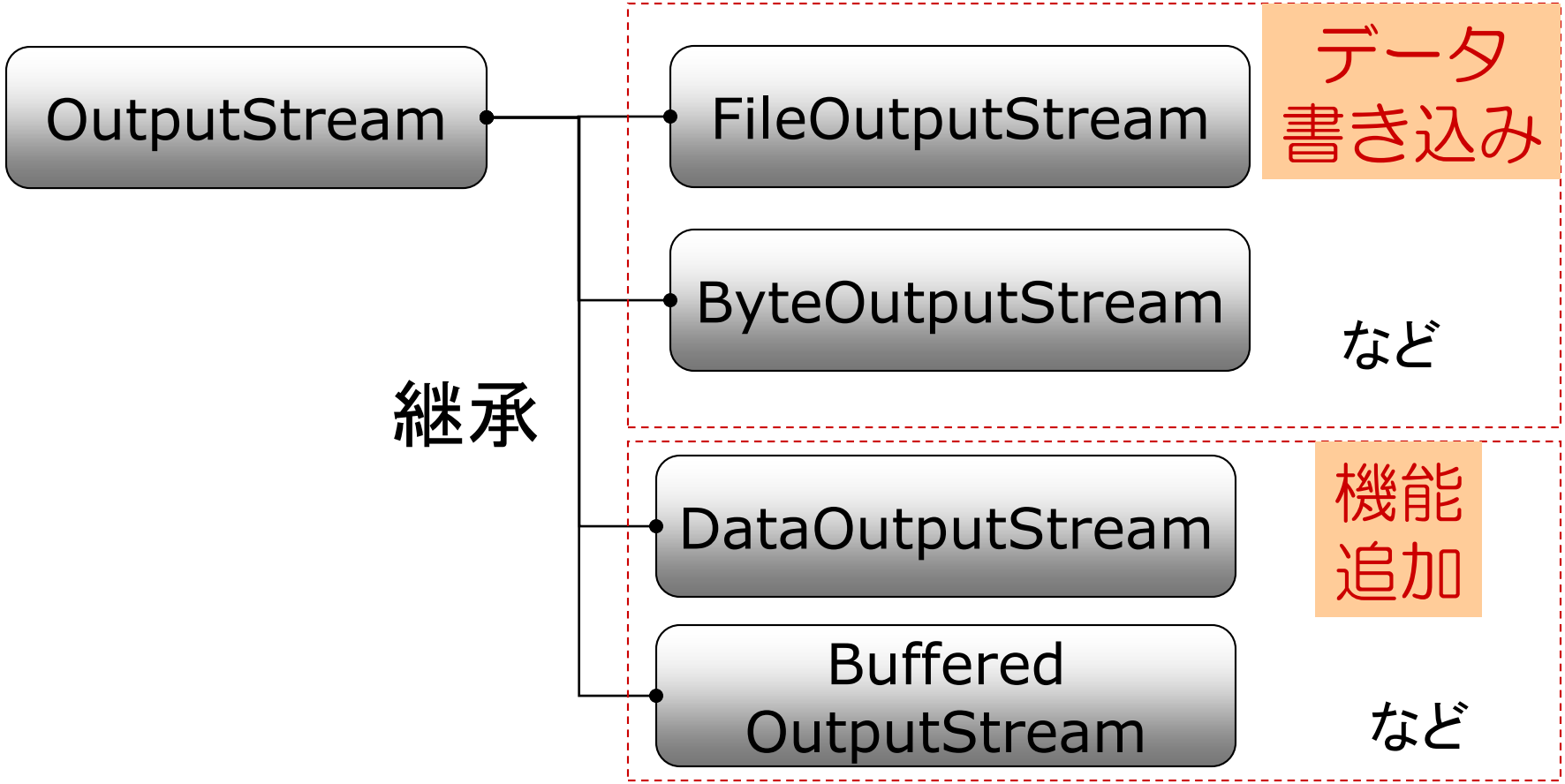

InputStream
継承
FileInputStream
ByteInputStream
など
データ
読み込み
DataInputStream
Buffered
InputStream
など
機能
追加

バイト出力ストリーム

byte単位で出力するデータの流れ

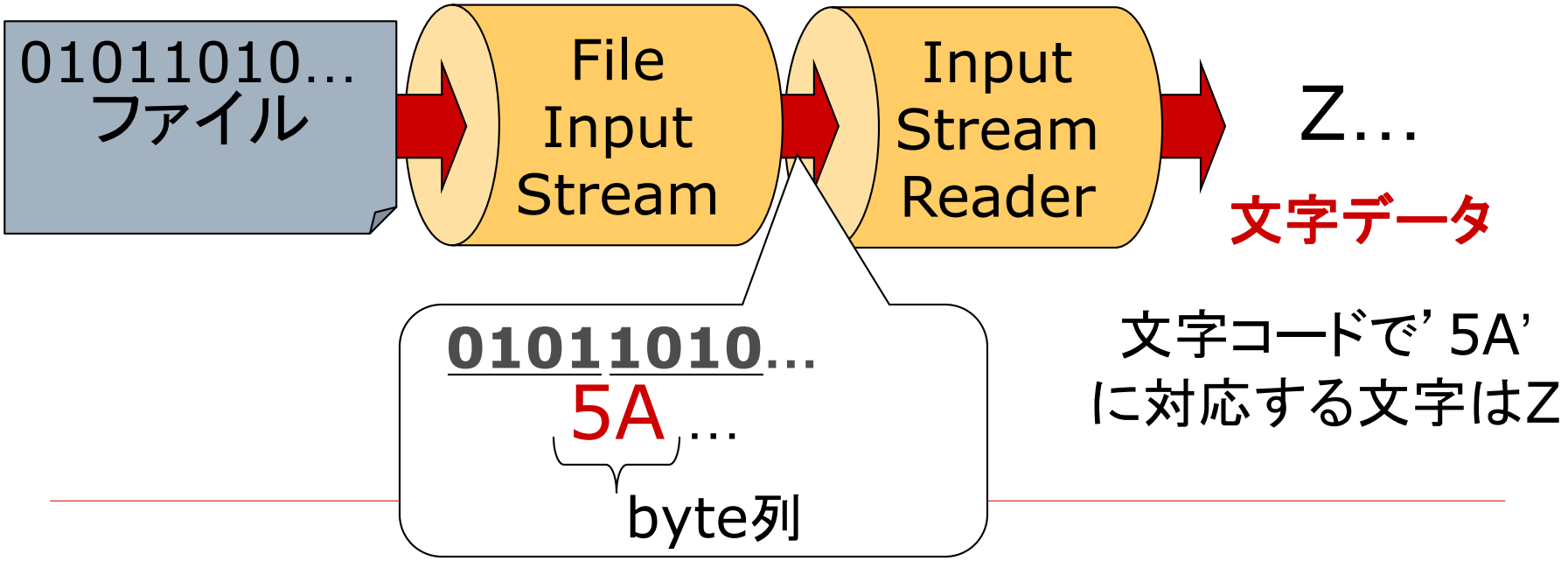


バイト出力ストリーム

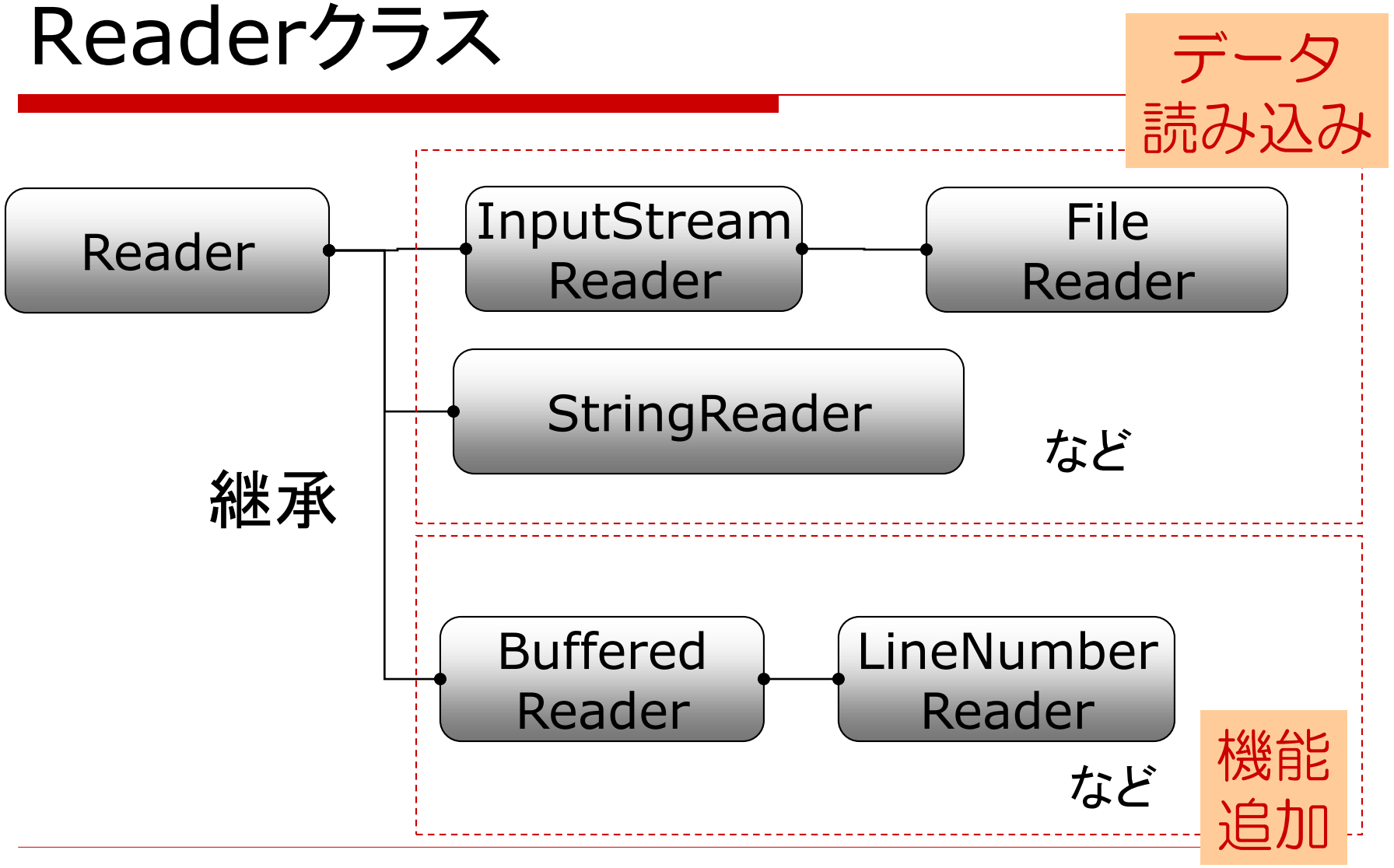

OutputStream
継承
FileOutputStream
ByteOutputStream
など
データ
書き込み
DataOutputStream
Buffered
OutputStream
など
機能
追加

文字ストリーム

文字単位でプログラムに渡すデータの流れ

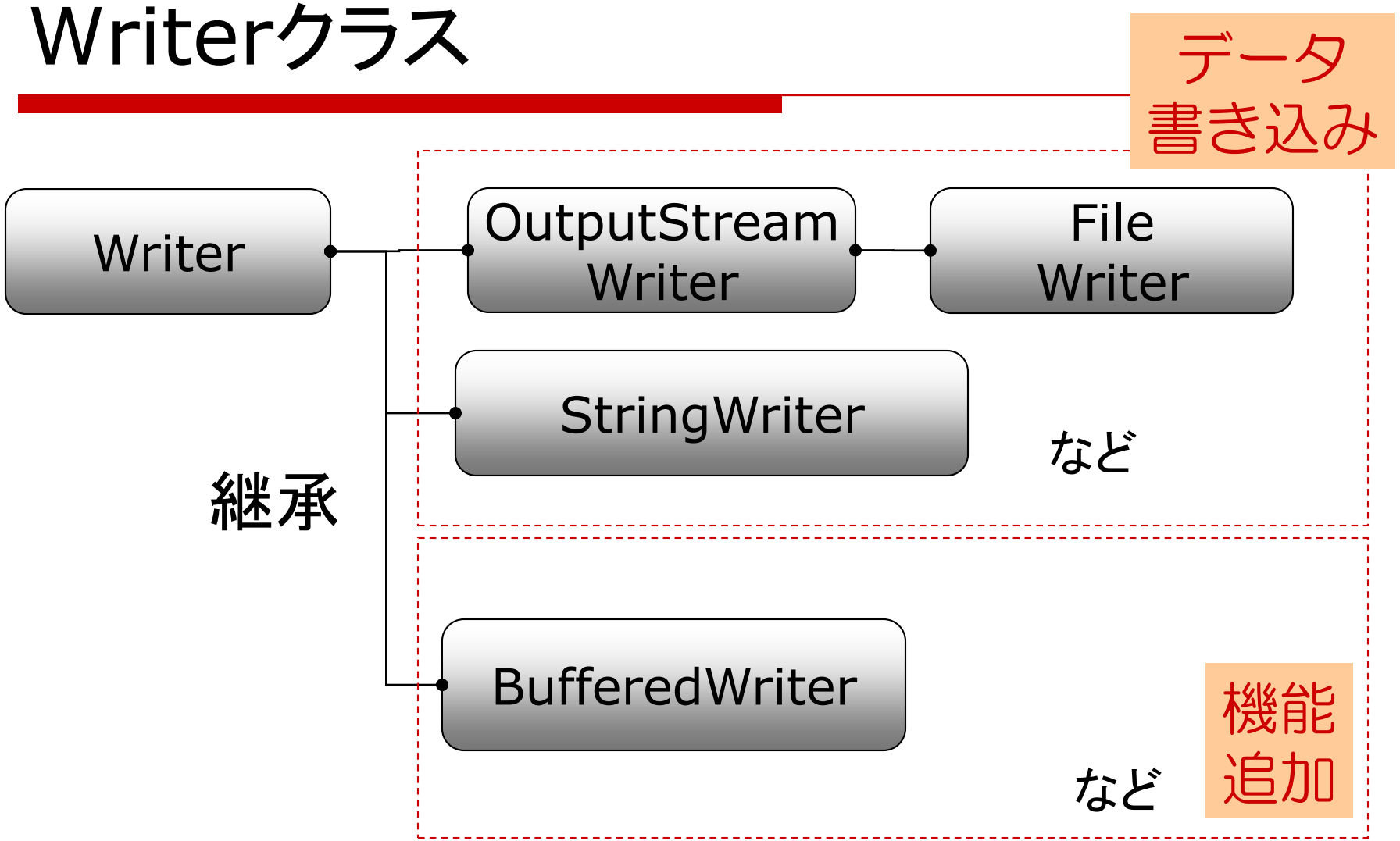


Readerクラス



※クラス名はスペースの都合上改行している

Writerクラス



※クラス名はスペースの都合上改行している

注意

ファイルを使ったら閉じるべし

BufferedReaderをcloseするとiSやfIも内部的にcloseされる

```
FileInputStream fI= new  FileInputStream(“…”);
InputStreamReader iS=new InputStreamReader(fI,"SJIS");
BufferedReader bf=new BufferedReader(iS);
            while((strData=bf.readLine())!=null){
                  System.out.println(strData);
            }
bf.close();
```

まとめ

- バイト列での入出力部分とそれを文字列として扱う部分を、プログラムに必要な機能に応じて組み合わせる
  - バイト列入出力部分：InputStream, OutputStreamとその子クラス
  - 文字列取り扱い部分:Reader, Writerの子クラス

補足1

- 1文にまとめて書いてもよい

```
BufferedReader br =
  new BufferedReader(
    new InputStreamReader(
      new FileInputStream("Sample.txt"), "SJIS"));
```

ファイルの文字コード
シフトJIS → "SJIS"
UTF-8 → "UTF-8"

- FileReaderを使うとさらに短くなる

```
BufferedReader br =
  new BufferedReader(
    new FileReader("Sample.txt"));
```

ただし文字コードの指定ができない（デフォルトの文字コードで処理される）

補足2

- ファイルに書き出す場合の例

```
PrintWriter pw =
  new PrintWriter(
    new BufferedWriter(
      new OutputStreamWriter (
        new FileOutputStream(ファイル名), “SJIS”));
```

- PrintWriterクラスを使うことでprintln, printfなどのメソッドが使える（例： pw.println(…);）
- FileWriterを使うと少し短くなる

例外処理

TXTファイルからデータを読み出そう(再掲)

```
import java.io.*;
class Smpl0501 {
	public static void main(String[] args){
		BufferedReader bf=null;   String strData="";
		try{	FileInputStream fI= new FileInputStream("Sample.txt");
			InputStreamReader iS=new InputStreamReader(fI,"SJIS");
			bf=new BufferedReader(iS);
			while((strData=bf.readLine())!=null){
				System.out.println(strData);
			}
		}catch(IOException e){
			e.printStackTrace();
		}finally{
			try {
				bf.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}

	}
}
```

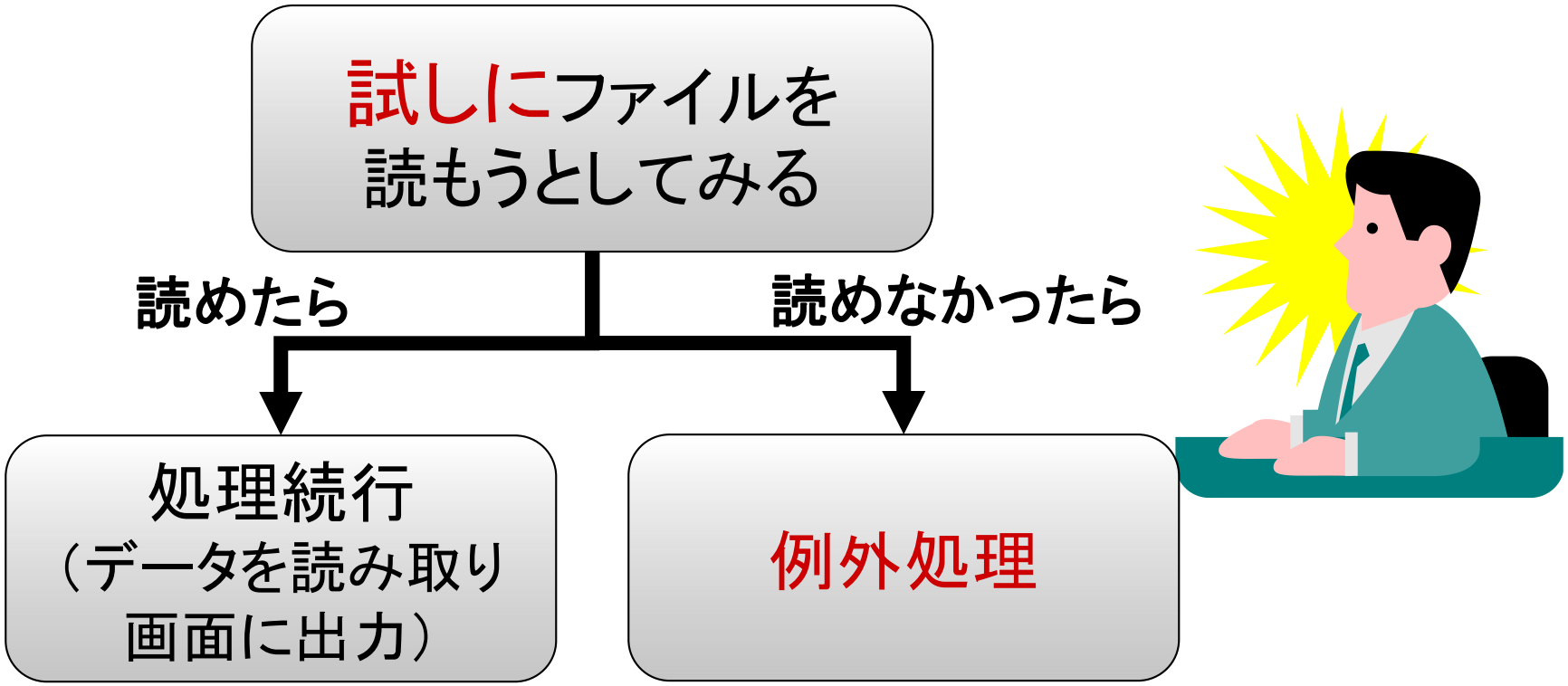
・・・こんなときどうしよう？

ファイルが
ないよ！

ファイルはあるけど
読み取れない！

などなど

こんなことができたら便利



うまくいかなかったこと（ファイルが開けなかった等）を例外という

TXTファイルからデータを読み出そう(再掲)

```
public static void main(String[] args){
   String strData="";   BufferedReader  bf=null;

   try{
        FileInputStream fI
                = new FileInputStream("Sample.txt");
        InputStreamReader iS
                =new InputStreamReader(fI,"SJIS");
        bf=new BufferedReader(iS);
        while((strData=bf.readLine())!=null){
                System.out.println(strData);
        }
   }
   catch(IOException e){
      e.printStackTrace();
   }
      ・・・・・(略)・・・・・
}
```

試しに

もし**例外が発生**したなら、その例外を**捕えて**{}内の**処理**を行う

複数の例外に対して別の処理を行わせることも可能

```
try{
        ・・・・ファイル読み出し処理・・・・
}
catch(FileNotFoundException e){
    System.out.println(“そんなファイルはありません。”);
    e.printStackTrace();
}
catch(IOException e){
    System.out.println(“入出力エラーが発生しました。”);
    e.printStackTrace();
}

    ・・・・・(略)・・・・・
}
```

Finally文

- どんな例外処理をしても、正常処理であっても必ず実施する処理を記述する

```
try{   …
       bf=new BufferedReader(iS);
       …
} catch(FileNotFoundException e){
    System.out.println(“そんなファイルはない。”);
} catch(IOException e){
    System.out.println(“入出力エラーが発生しました。”);
}finally{   …
                bf.close();
                …
}
```

必ず
入力ストリーム
を閉じる

例外とエラーの違い

エラー

ハードウェア障害
OSのバグによるエラー

システムエラー

例外

ゼロで除算している
文法が間違っている
ファイルがない
入出力がうまくいかない
など

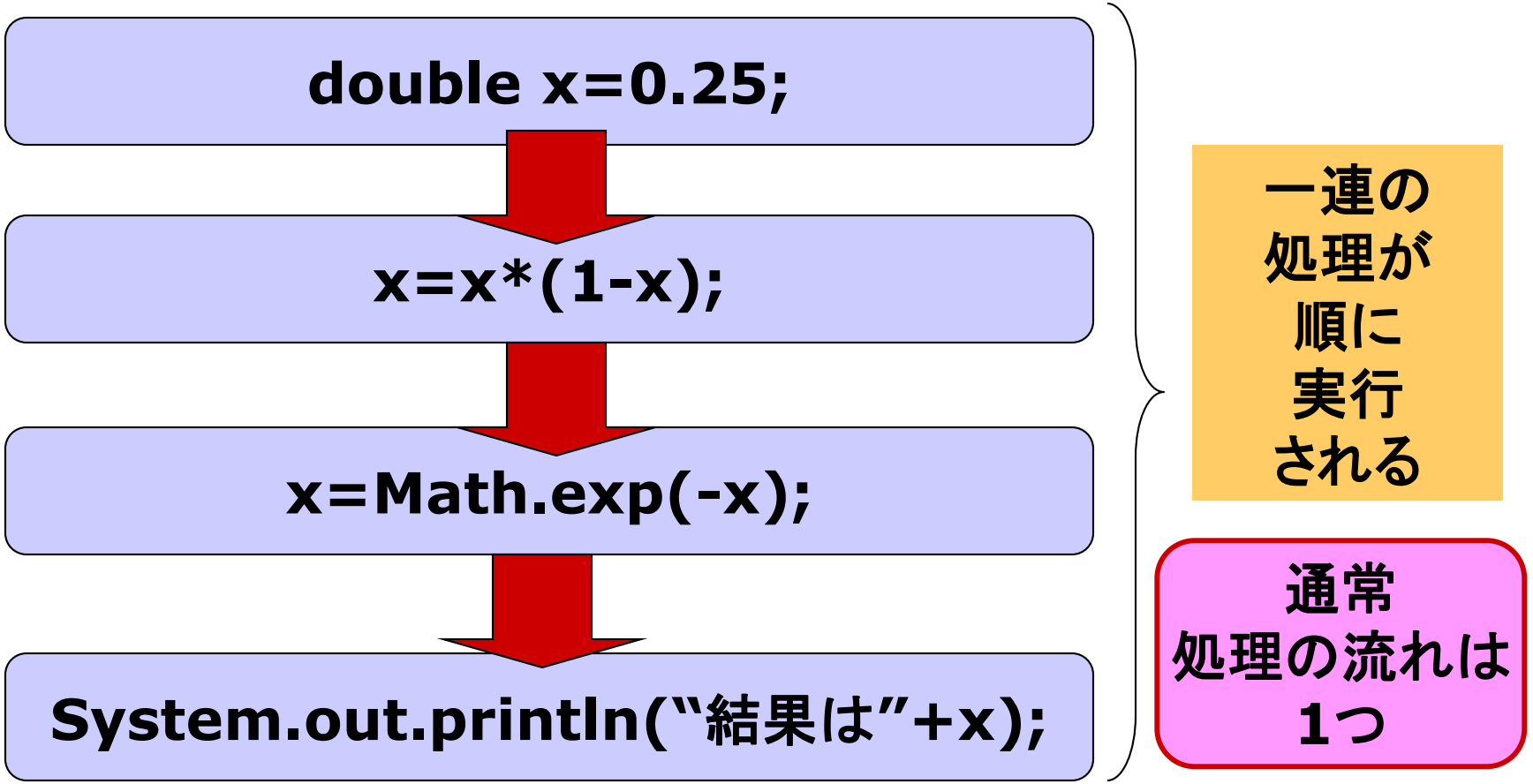
プログラムエラー

補足：よく起こる例外の種類

- NullPointerException
  - 未定義のオブジェクト（null）のフィールドを参照したりメソッドを呼び出したりした際に発生
  - 例： String str=null;
    System.out.println(str.length());
- ArrayIndexOutOfBoundsException
  - 配列の添え字の範囲を超えてアクセスした際発生
  - 例： int[] a = new int[3];
    System.out.println(a[3]);

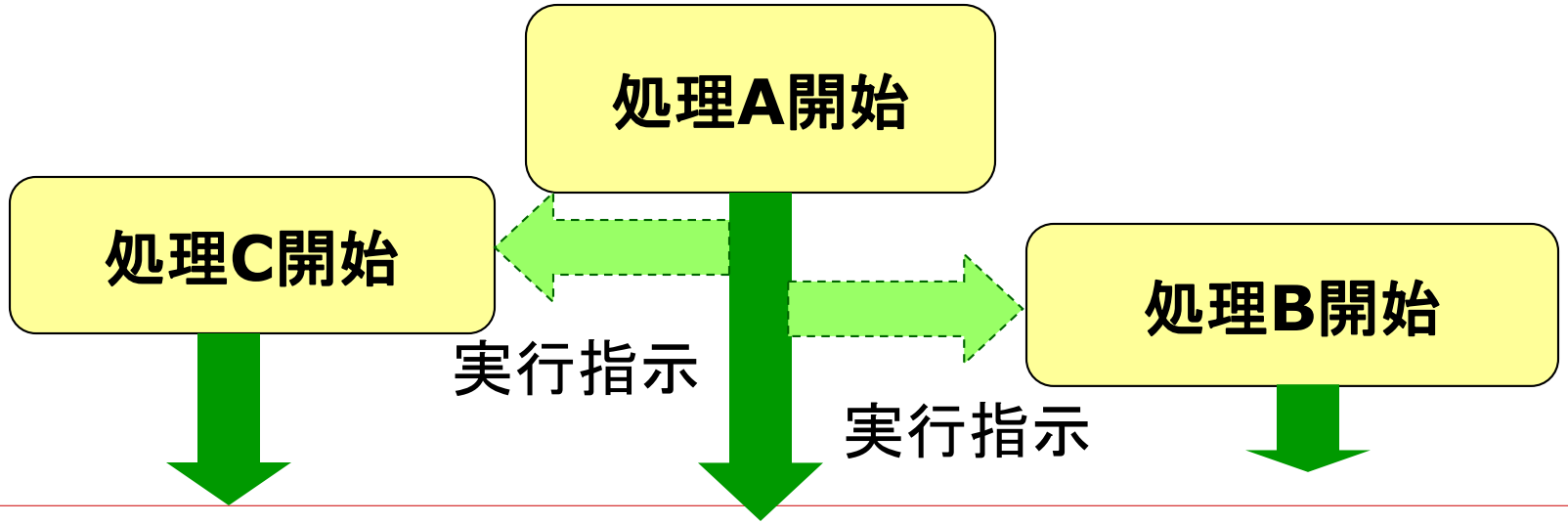
スレッド

普通のプログラム

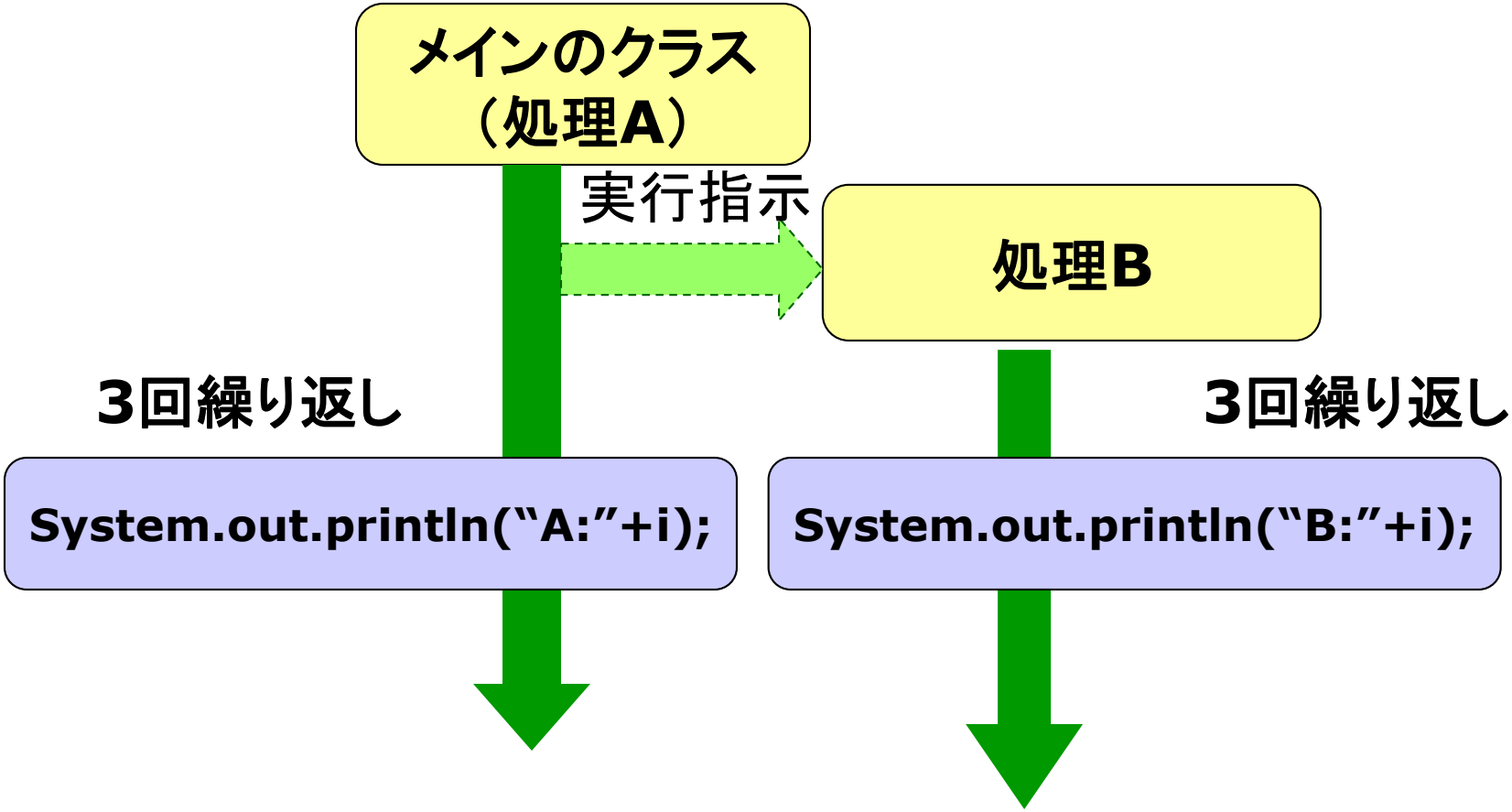

double x=0.25;
x=x*(1-x);
x=Math.exp(-x);
System.out.println(“結果は”+x);
一連の
処理が
順に
実行
される
通常
処理の流れは
1つ

マルチスレッド

- 1つのプログラム（正確にはプロセス）の中で、処理の流れ（スレッド）を同時に複数実行するしくみ
- 複数のスレッドが同時に複数実行されるプログラムをマルチスレッドプログラムと呼ぶ（普通のプログラムはシングルスレッドプログラム）



マルチスレッドプログラムの簡単な例



ソースプログラム(1)　処理B

```
class ThreadSmpl extends Thread {
        public void run(){
                for(int i=0; i<3; i++){
                        System.out.println("B:"+i);
                }
        }
}
```

① Treadクラスを継承する
② run()メソッドをオーバーライドし、同時実行させる処理を記述する。
run()メソッドはstart()メソッドから呼び出される

ソースプログラム(2) 処理A

```
class ThreadMain {
        public static void main(String[] args) {
            ThreadSmpl shoriB = new ThreadSmpl();
            shoriB.start();
            for(int i=0; i<3; i++){
                        System.out.println("A:"+i);
            }
            System.out.println("Done(main).");
        }
}
```

処理Bを新規スレッドとしてstart()メソッドで実行後、A:・・・を表示

startメソッド：新たにスレッドを生成し，そのrunメソッドを呼び出す

結果

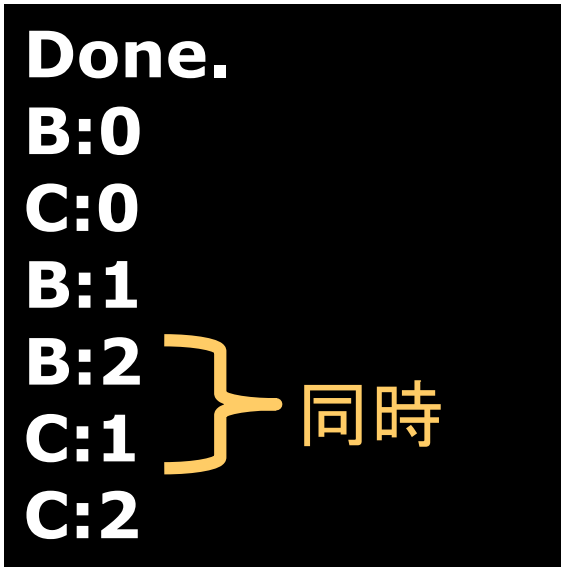
```
A:0
A:1
A:2
Done(main).
B:0
B:1
B:2
```

処理Aと処理Bの実行順は
環境に依存するが、この結果では
Bの処理の前に
Aの処理が終わっている
ことに注意

シングルスレッドでは、
処理Bの実行前に
処理Aが終わることはありえない

処理A(ThreadMain)のmainメソッドも
Threadクラス(ThreadSmpl)のオブジェクトのrunメソッドも
同等のスレッドとして扱われる

結果



BとCのスレッドは同時に
始まり、Bは1秒おき、
Cは1.5秒おきに
画面に表示する

それぞれのスレッドが独立に動いている
ことがわかる