

# 発展プログラミング演習II

## 10. 入出力

# 入出力

- ・ キーボードからの入力、ターミナルへの出力
- ・ ファイルの読み書き
- ・ java.io (およびjava.nio) パッケージ  
(クラスライブラリ) を使う
- ・ 教科書 pp.155-169

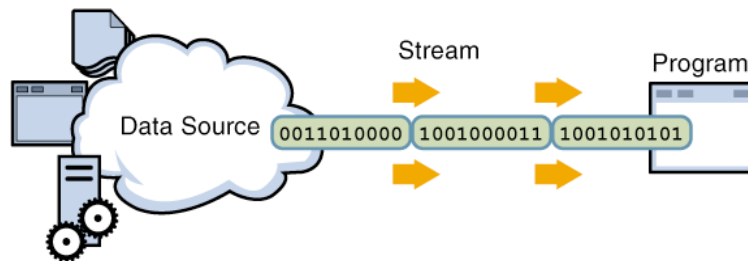
# 入出力クラスの大まかな分類

- ・ 入力 \*Input\*, \*Reader\*
  - ・ 出力 \*Output\*, \*Writer\*
- 
- ・ バイナリ \*Stream
  - ・ 文字 \*Reader, \*Writer

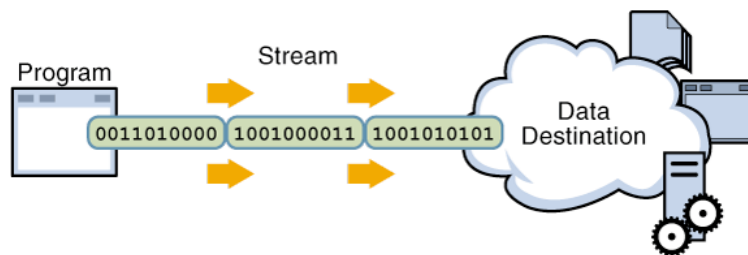
	入力	出力
バイナリ	InputStream	OutputStream
文字	Reader	Writer

# ストリーム

- ・ 入力



- ・ 出力



- ・ データの流れ
- ・ 書き手と受け手がセット、一方通行

# 標準入出力

- ・ 予め用意されたクラスオブジェクト
- ・ 標準出力=ターミナルへの表示（出力）
  - ・ System.out      **System = クラス名**
- ・ 標準入力=ターミナル(キーボード)からの入力
  - ・ System.in
- ・ 標準エラー出力
  - ・ System.err

# 例題10.1

- ・ 次のプログラムを実行して動作を確認せよ

```
import java.io.*;

class StandardIOExample {
    public static void main(String[] args) {
        String num1, num2;
        try {
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("input 1st number: ");
            num1 = in.readLine();
            System.out.print("input 2nd number: ");
            num2 = in.readLine();
            int sum = Integer.parseInt(num1) + Integer.parseInt(num2);
            System.out.println("sum = " + sum);
        } catch (IOException e) {
            System.err.println("input error");
        } catch (NumberFormatException nfe) {
            System.err.println("invalid number error");
        }
    }
}
```

発展プログラミング演習II

# try, catch

- ・ 例外処理を書く必要がある

```
import java.io.*;

class StandardIOExample {
    public static void main(String[] args) {
        String num1, num2;
        try {
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("input 1st number: ");
            num1 = in.readLine();
            System.out.print("input 2nd number: ");
            num2 = in.readLine();
            int sum = Integer.parseInt(num1) + Integer.parseInt(num2);
            System.out.println("sum = " + sum);
        } catch (IOException e) {
            System.err.println("input error");
        } catch (NumberFormatException nfe) {
            System.err.println("invalid number error");
        }
    }
}
```


発展プログラミング演習II

# 標準出力

- 利用可能なメソッドはPrintStreamを参照

```
import java.io.*;

class StandardIOExample {
    public static void main(String[] args) {
        String num1, num2;
        try {
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("input 1st number: ");
            num1 = in.readLine();
            System.out.print("input 2nd number: ");
            num2 = in.readLine();
            int sum = Integer.parseInt(num1) + Integer.parseInt(num2);
            System.out.println("sum = " + sum);
        } catch (IOException e) {
            System.err.println("input error");
        } catch (NumberFormatException nfe) {
            System.err.println("invalid number error");
        }
    }
}
```



何が起きているか？

発展プログラミング演習II



# 標準エラー出力

- 標準出力との違いはリダイレクトするとわかる

```
import java.io.*;

class StandardIOExample {
    public static void main(String[] args) {
        String num1, num2;
        try {
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("input 1st number: ");
            num1 = in.readLine();
            System.out.print("input 2nd number: ");
            num2 = in.readLine();
            int sum = Integer.parseInt(num1) + Integer.parseInt(num2);
            System.out.println("sum = " + sum);
        } catch (IOException e) {
            System.err.println("input error");
        } catch (NumberFormatException nfe) {
            System.err.println("invalid number error");
        }
    }
}
```

**Buffer = ためる**

# 標準入力

- なぜこんな複雑なことをする必要があるか？

```
import java.io.*;

class StandardIOExample {
    public static void main(String[] args) {
        String num1, num2;
        try {
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("input 1st number: ");
            num1 = in.readLine();
            System.out.print("input 2nd number: ");
            num2 = in.readLine();
            int sum = Integer.parseInt(num1) + Integer.parseInt(num2);
            System.out.println("sum = " + sum);
        } catch (IOException e) {
            System.err.println("input error");
        } catch (NumberFormatException nfe) {
            System.err.println("invalid number error");
        }
    }
}
```

発展プログラミング演習II

# 標準入力からの入力

- ・ Cのscanf()関数のようなものはJavaには無い
- ・ InputStream
  - ・ バイト入力ストリームのスーパークラス
  - ・ 1バイトずつ読み込む
  - ・ 1行をまとめて読み込むメソッドは無い
  - ・ ⇒リターンキーが入力されるまでをまとめて読み込む機能が必要

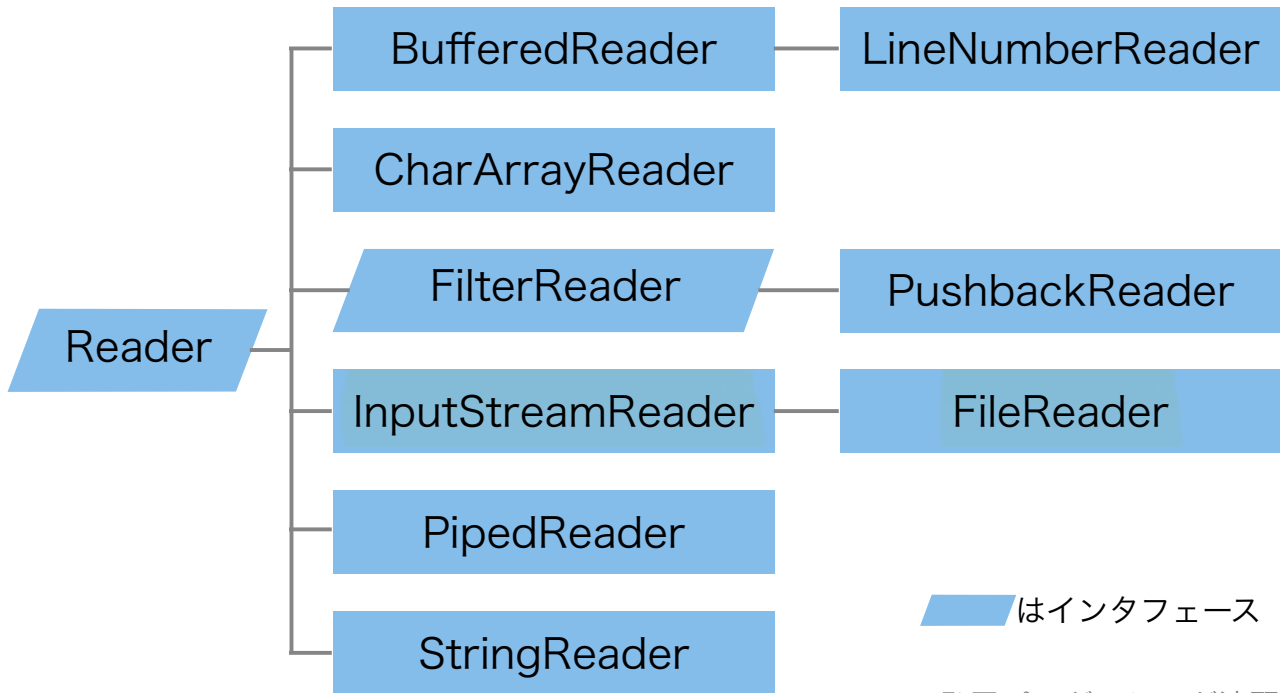
# ストリームラッパー stream wrapper

- ・ 他のストリームにかぶせて（ラップして）機能を追加する
- ・ InputStreamReader
  - ・ バイトデータを読み込んで文字列に変換する
- ・ BufferedReader
  - ・ バッファリングすることで効率を上げる
  - ・ `readLine()`メソッドで1行分を読み込む

# 文字列ストリーム

- ReaderとWriter
- Javaの基本エンコードはUnicode
  - 内部ではUnicodeで扱う
  - 入出力時にはその他のエンコード（JIS\_Encoding, Shift\_JIS, EUC-JPなど）も利用可能
- バイトデータと文字の変換を行う

# Reader



## 例題10.2

- ・ 次のプログラムをserohikino\_goshu\_utf8.txtを引数にして実行せよ
- ・ 行番号を付けて表示するように修正せよ

```
import java.io.*;

class FileReaderExample {
    public static void main(String[] args) throws IOException {
        BufferedReader in = null;
        try {
            in = new BufferedReader(new FileReader(args[0]));
            String str;
            while ((str = in.readLine()) != null) {
                System.out.println(str);
            }
        } catch (IOException ioe) {
            // do nothing
        } finally {
            if (in != null) in.close();
        }
    }
}
```

発展プログラミング演習II

# 例題10.2 解説

```
import java.io.*;
```

```
class FileReaderExample {
```

```
    public static void main(String[] args) throws IOException {
```

```
        BufferedReader in = null;
```

```
        try {
```

```
            in = new BufferedReader(new FileReader(args[0]));
```

```
            String str;
```

```
            while ((str = in.readLine()) != null) {
```

```
                System.out.println(str);
```

```
            }
```

```
        } catch (IOException ioe) {
```

```
            // do nothing
```

```
        } finally {
```

```
            if (in != null) in.close();
```

```
        }
```

```
    }
```

```
}
```

1行ずつ読み込むために  
BufferedReaderでラップする

1行読み込む  
ストリームの終わりに達して  
いる場合はnullが返る

使い終わったら閉じるのを忘れずに

※行番号を付けるにはLineNumberReaderを使ってもよい

発展プログラミング演習II



# 文字コードの変換

- ・ 文字コードを指定してファイルを読み込むには  
InputStreamReaderに文字コードを指定する
- ・ インスタンス生成時に指定する必要がある

## コンストラクタ

```
InputStreamReader(InputStream in, String charsetName)
```

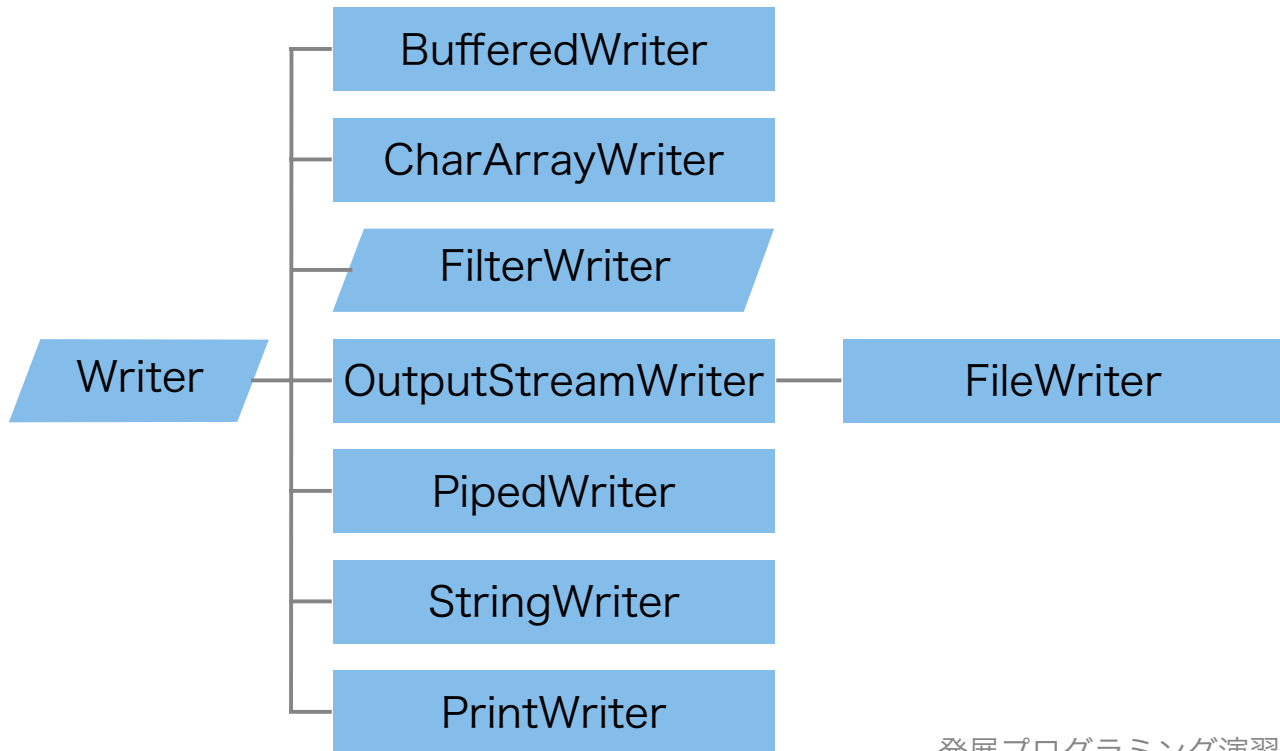
## 例

```
BufferedReader in = new BufferedReader(new InputStreamReader(  
    new FileInputStream(filename), "SJIS"));
```

## 例題10.3

- ・ 例題10.2のプログラムを  
ChartsetConverter.javaにコピーして、  
Shift\_JISの文字コードで読み込むように  
修正せよ
- ・ 作成したプログラムに対して、  
serohikino\_goshu\_sjis.txtと  
serohikino\_goshu\_utf8.txtで実行して  
動作を確認せよ

# Writer



# 例題10.4

- ・ 次のプログラムを実行して動作を確認せよ

```
import java.io.*;

class FileWriterExample {
    public static void main(String[] args) throws IOException {
        BufferedWriter out = null;
        try {
            out = new BufferedWriter(new FileWriter(args[0]));
            for (int i = 0; i < 100; i++) {
                out.write(i + "は");
                if (i % 2 == 0) out.write("2で割り切れる ");
                if (i % 3 == 0) out.write("3で割り切れる ");
                if (i % 5 == 0) out.write("5で割り切れる ");
                out.newLine();
            }
            out.close();
        } catch (IOException ioe) {
            // do nothing
        } finally {
            if (out != null) out.close();
        }
    }
}
```

発展プログラミング演習II

# 入出力とバッファ

- ・ ファイルの入出力は遅い
  - ・ 一般的にブロック単位で読み込む／書き込む
  - ・ 少しずつ読み込む／書き出すのは効率が悪い
- ・ ⇒効率のためにバッファを使うべき
- ・ バッファをフラッシュしないと書き込まない
  - ・ 例：バッファを使ってターミナルに表示させても表示のタイミングがずれることがある

# PrintWriter ・ PrintStream

- ・ System.out、System.in
- ・ 結構便利
  - ・ 様々なデータ型を文字列で出力できる
  - ・ 書式指定も可能 `printf()` メソッド
  - ・ ファイル名、文字コードを指定できる  
コンストラクタもある

# 出力時の文字コードの指定

- ・ 文字コードを指定してファイルを書き出すには  
OutputStreamWriterに文字コードを指定する
- ・ InputStreamReaderと似たようなやりかた

## コンストラクタ

```
OutputStreamWriter(OutputStream out, String charsetName)
```

## 例

```
BufferedWriter out = new BufferedWriter(new OutputStreamWriter(  
    new FileOutputStream(filename), "EUC_JP"));
```

# 課題10

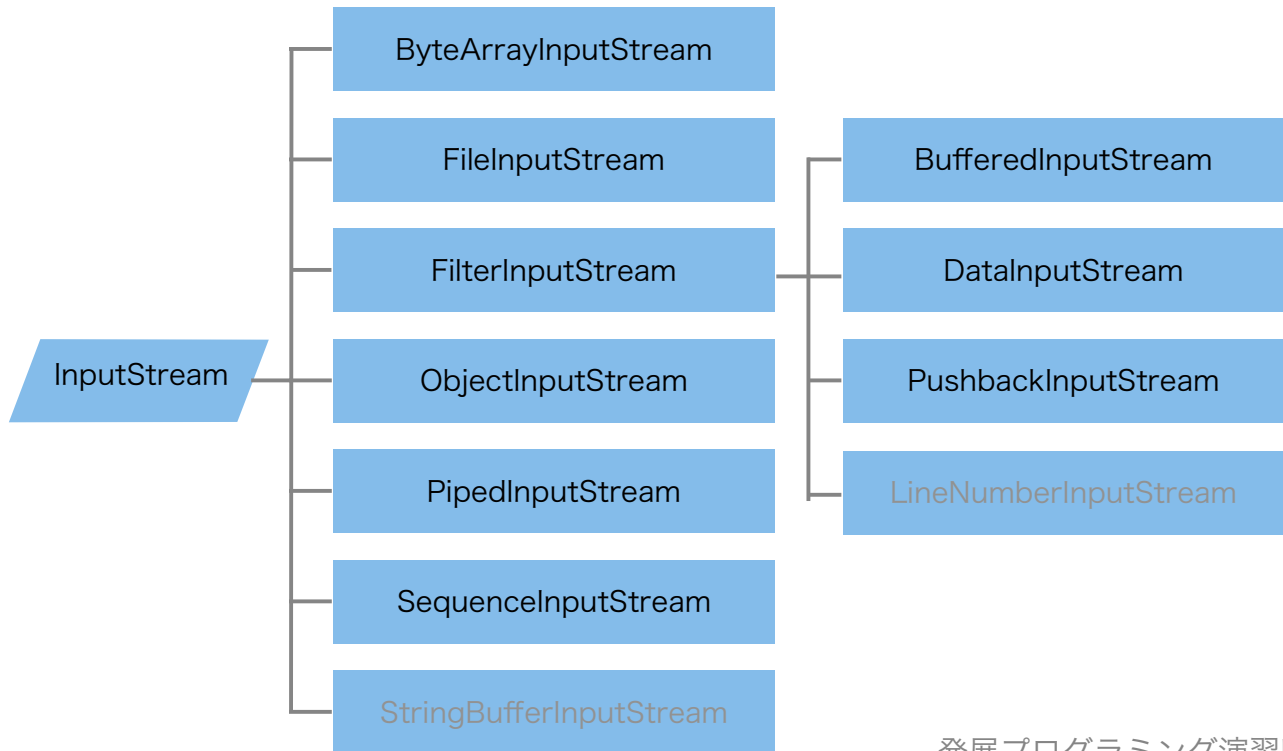
- ・ 次の仕様のプログラムを作成せよ
  - ・ 第1引数で指定されたファイル名のUnicodeのテキストファイルを読み込み、
  - ・ 奇数行は第2引数で指定されたファイル名にEUC\_JPで書き出し、
  - ・ 偶数行は第3引数で指定されたファイル名にSJISで書き出す



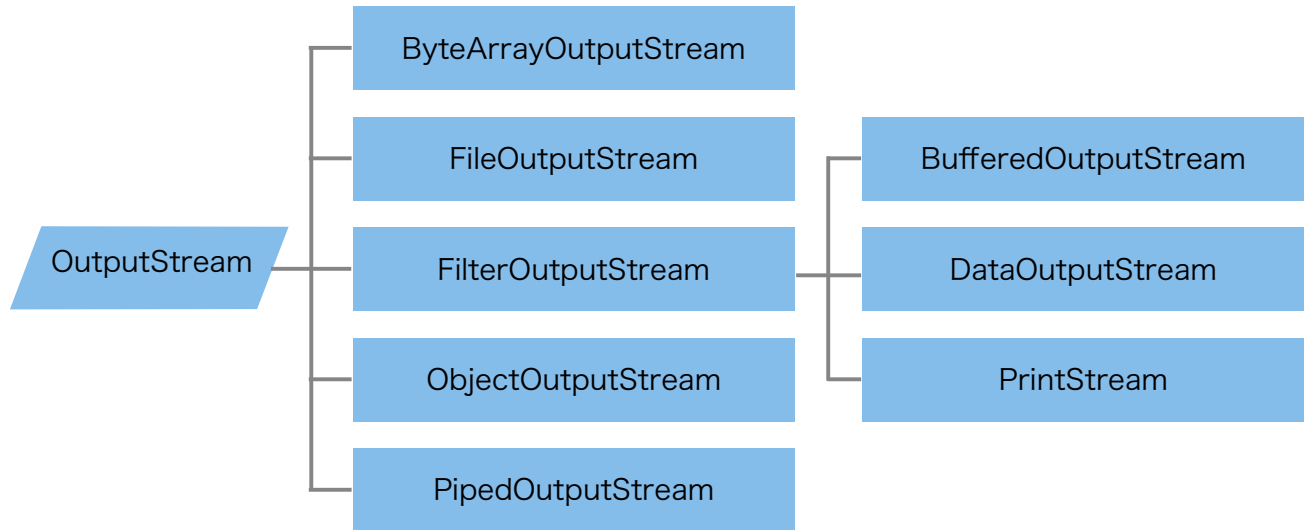
# データストリーム

- ・ InputStreamとOutputStream
- ・ データを生のまま読み込む／書き出す
  - ・ バイトデータ
  - ・ プリミティブ型のデータ
  - ・ オブジェクト

# InputStream



# OutputStream



# バイトデータの読み書き

- ・ ファイルからの入出力：  
FileInputStreamとFileOutputStreamを使う
- ・ readメソッドで読み込み
  - ・ readメソッドの返値がint型であることに注意
- ・ writeメソッドで書き出し

# プリミティブ型の読み書き

- ・ DataInputStreamとDataOutputStream
  - ・ InputStreamとOutputStreamをラップする
- ・ 書き出す時は引数の型に応じたメソッドを使う
- ・ 読み込むときは読み込む型に応じたメソッドで読み込む
- ・ 書き出した時と型が違っていたらどうなる??

# オブジェクトの読み書き\*

- ・ `ObjectInputStream`と`ObjectOutputStream`
- ・ `java.io.Serializable`インタフェースをサポートするオブジェクトのみ（「直列化可能」）
- ・ 具体的な使い方は他のデータストリームと同様なので省略

## 例題10.5

1. 次の仕様のプログラムを作成せよ
  - a. 0から100までのint型の値を引数で指定されたファイルにバイナリデータで書き出す
  - b. a.でできたファイルをint型の配列に読み込んで表示する
2. 1-aで作成したファイルの中身をod -xコマンドで確認せよ
3. 1.のプログラムにfloat型の配列に読み込む処理を追加し、1-aで作成したファイルを読み込んで表示させてみよ

# Stringクラスと正規表現

- 正規表現を用いて一致判定、分割、置換できる
  - `boolean matches(String regex)`
  - `String[] split(String regex)`
  - `String replaceAll(String regex, String replacement)`



## 例題10.6

- serohikino\_goshu\_utf8.txt中には、ルビが含まれている。（《》で囲まれる範囲がルビ、|は区切りを示す記号）  
これらを削除してターミナルに出力するプログラムを作成せよ
- ヒント
  - 削除するにはマッチした部分を  
"""（空文字列）で置換する
  - ルビの部分を探す正規表現は《.\*?》