File IO

オブジェクト指向プログラミング特論

2016年度

只木進一：工学系研究科

JavaでのFile IO

- JavaでのFile IOの仕組み
  - 言語(java.langパッケージ)にはFile IOが含まれない
  - 標準入出力のみ
  - java.ioパッケージが別に用意されている

例外処理の必要性

- IOでは、エラーが発生しやすい
  - 読めない、書けない
  - ファイルが存在しない

標準入出力

- 標準入力と出力
- 標準エラー出力

```
package java.lang;
import java.io.*;
public final class System {
  private System() {}//インスタンスは作成不能
  public final static InputStream in;
  public final static PrintStream out;
  public final static PrintStream err;
...
}
```

標準入力：キーボード

- 一文字ずつの入力
  - メソッドread()を使用
  - 一行をまとめて読めない
- 戻り値
  - 正整数：文字
  - -1：終了
- 例外発生可能性
  - IOException

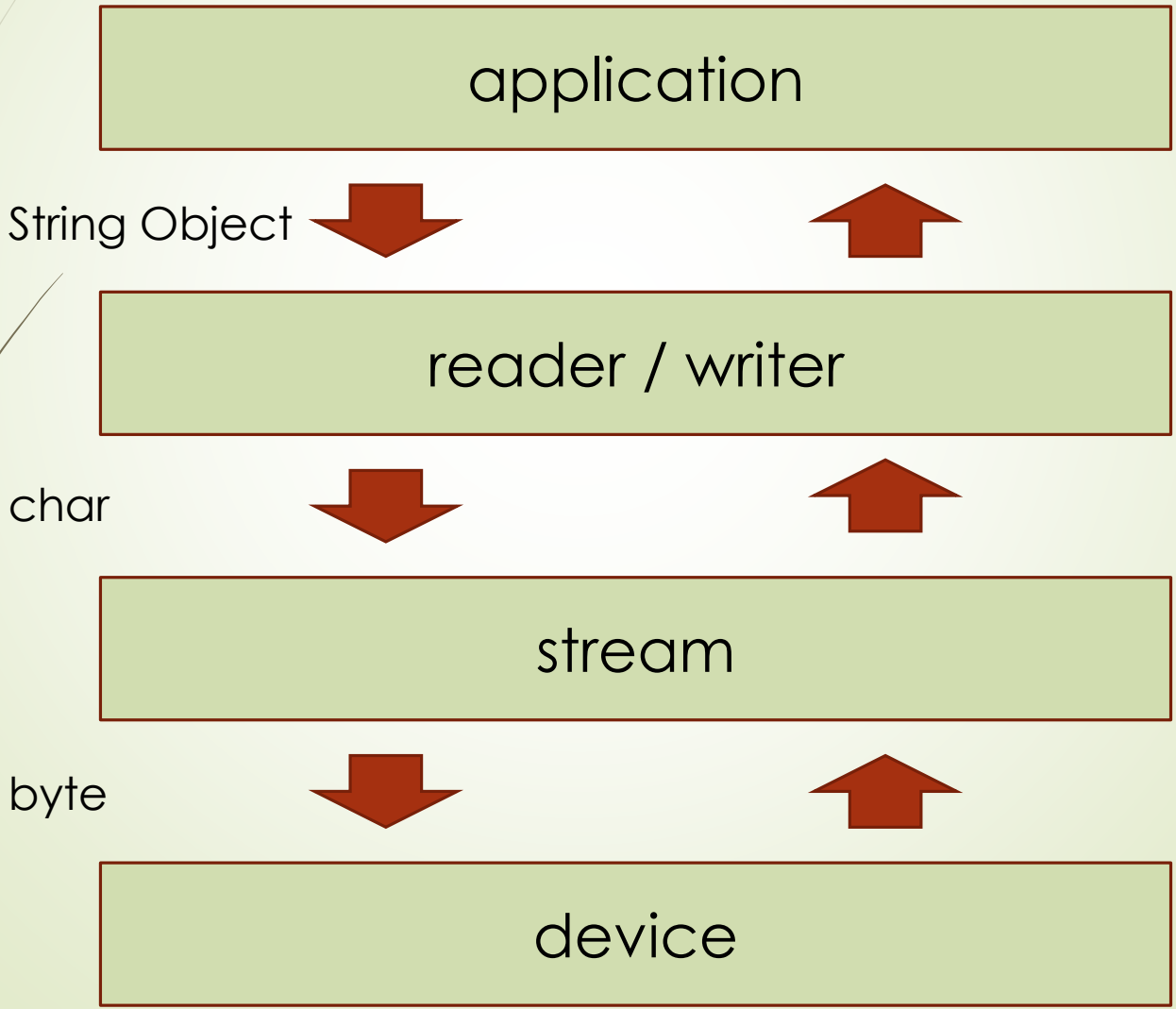
```
StringBuilder b=new StringBuilder();
int c;
try {
    while ((c = System.in.read()) != -1) {
        b.append((char)c);
        //1バイトずつ読んでbに追加
   }
} catch (IOException ex) {
  //エラー処理
}
```

標準出力：端末へ

- メソッドprint()：改行なし
- メソッドprintln()：改行あり
- 引数
  - 原始型
  - オブジェクト
    - toString()メソッドを使うなどして、文字列に変換して出力
    - Object.toString()

再検討

- 入出力先デバイスは多様
  - 標準入出力、ファイル、ネットワーク
- アプリケーションと最終のデバイスの間を階層化・モデル化
- アプリケーションから操作しやすいように


application
String Object
reader / writer
char
stream
byte
device

buffering

- コンピュータとデバイスでは、データ処理速度が大きく異なる
- 一定以上の量のデータの送受信では緩衝装置（buffer）が必要
  - streamで行うか、reader/writerで行うか

入力

- Fileクラスによるファイルの指定
- FileInputStream：ストリーム
- InputStreamReader：Reader
- BufferedReader：buffering

Fileを指定する

- クラスFileで指定する。

```
File file = new File(String filename)
```

- インスタンス作成だけでは、ファイルの存在や読み書きの可否は不明
  - 本当に読み書きする前に、その可能性をチェックすること

メソッド	処理
boolean canRead()	読み込み可能
boolean canWrite()	書き込み可能
boolean createNewFile()	空のファイルを生成

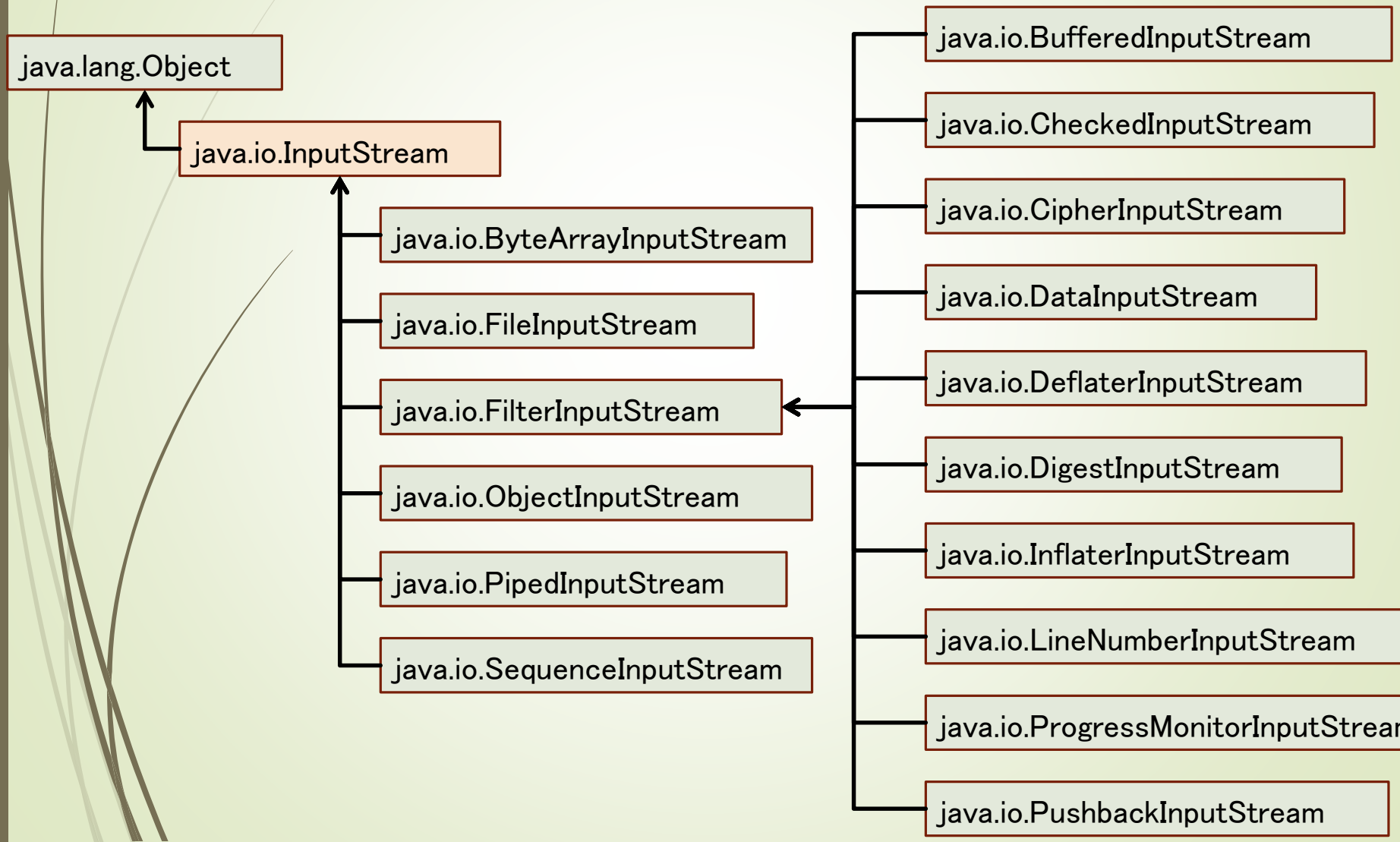
Fileに対応したInputStream

- FileInputStreamクラス

```
File file;
FileInputStream fStream=
        new FileInputStream(file);
```

- 読み込みはbyte
  - Int read()：1 byte 読む。戻り値が-1ならば、ファイル終端

入力ストリームのクラス階層



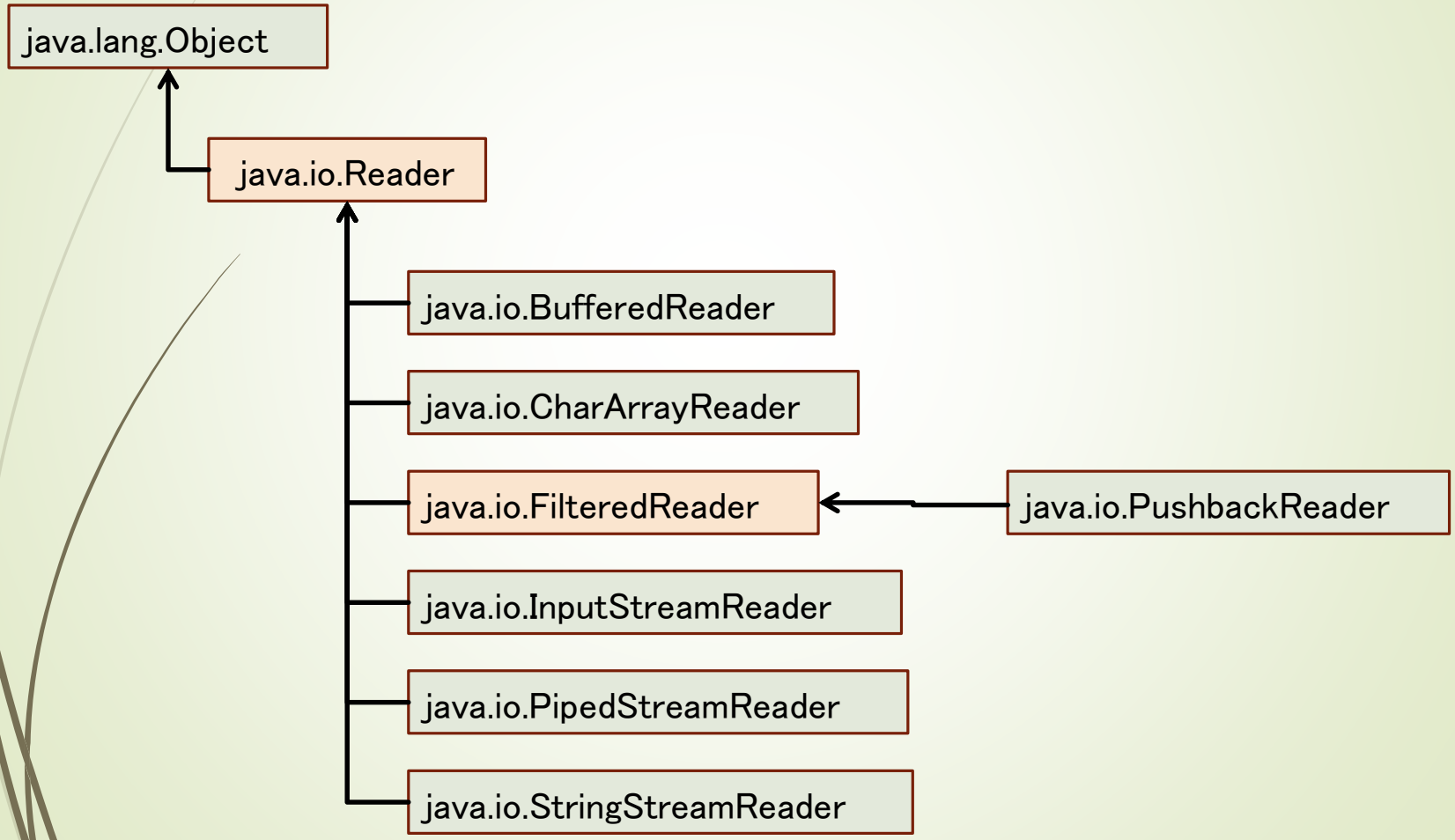
InputStreamの例

```
//例外が発生すると呼び出し側に知らせる
static public String openInputStream() throws IOException {
    File file = new File("input.txt");//ファイル指定
    //入力バッファを開く
    BufferedInputStream in =
        new BufferedInputStream(new FileInputStream(file));
    StringBuilder sb = new StringBuilder();
    int n;
    while ((n = in.read()) != -1) {// １バイト毎に読み込み
        sb.append((char) n);
    }
    return sb.toString();
}
```

Readerを使う

- バイト単位の読出しでは不便
- 文字、文字列単位での読み込み
  - int read(); //一文字読み込み
  - int read(char[]);//文字配列へ読み込み
  - String readLine();//一行を文字列へ読み込み
- 文字コードを指定できる

Readerのクラス階層



```
static List<String> openReader() throws IOException {
    File file = new File("input.txt");
    String enc = "UTF-8";
    List<String> stringList
            = Collections.synchronizedList(new ArrayList<>());
    try (BufferedReader in = new BufferedReader(
            new InputStreamReader(new FileInputStream(file), enc))) {
        String line;
        while ((line = in.readLine()) != null) {
            stringList.add(line);
        }
    }
    return stringList;
}
```

一行読み込んだ後で、スペース区切りで分割

- String[] String.split(String delimiter)
- 文字列を区切り文字列delimiterで分けて、結果を文字列配列で返す
- delimiterには、正規表現が使える
  - 例：空白文字(様々な種類、数)
    - “¥¥s+”

標準入力のwrapping

```
BufferedReader in
    = new BufferedReader(new InputStreamReader(System.in));
try {
    String line;
    while ((line = in.readLine()) != null) {
        System.out.print(line);
    }
    in.close();
} catch (IOException ex) {
    System.err.println(ex);
}
```

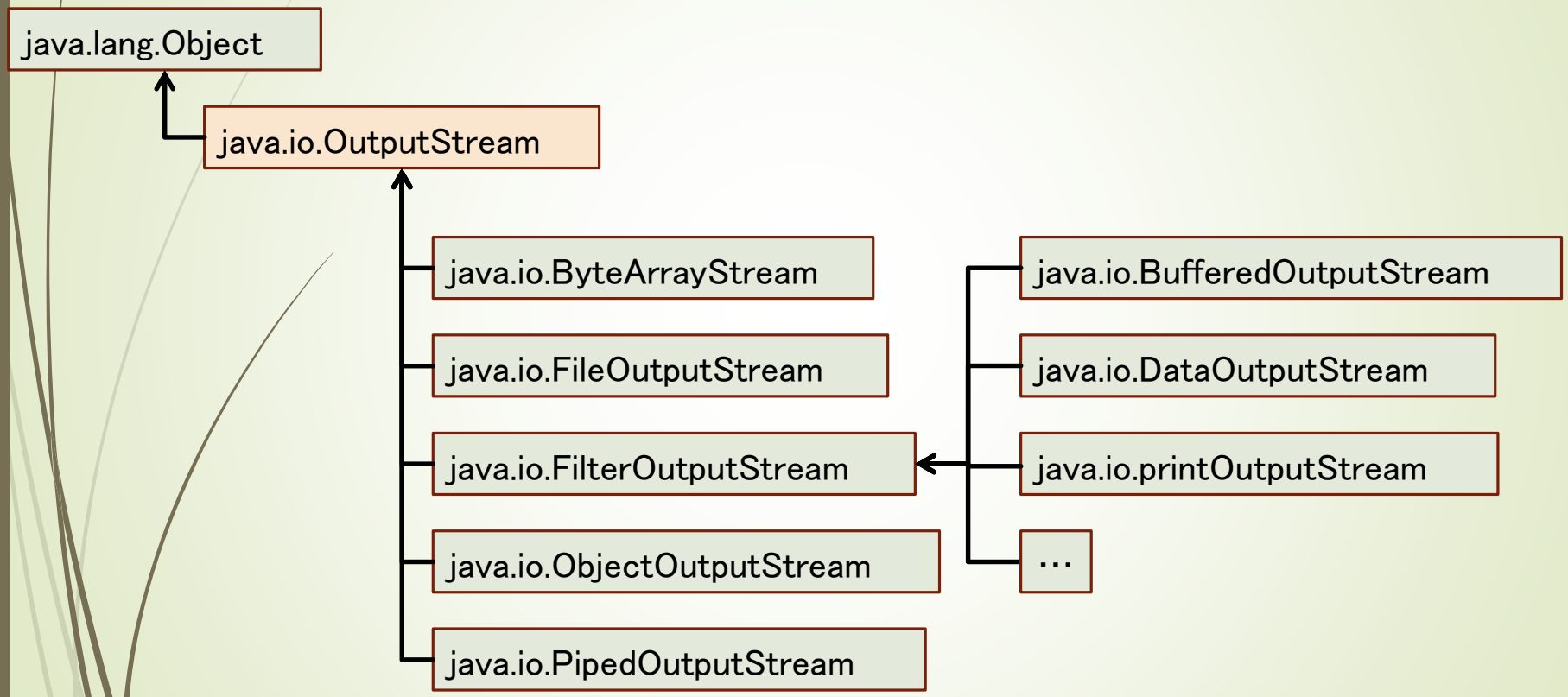
出力

- Fileクラスによるファイルの指定
- FileOutputStream：ストリーム
- OutputStreamWriter：Writer
- BufferedWriter：buffering

OutputStreamの基本

- バイト単位の書き出し
  - void write(byte[]);
- flush：強制排出
  - void flush();
- 閉鎖
  - void close();

出力ストリームのクラス階層



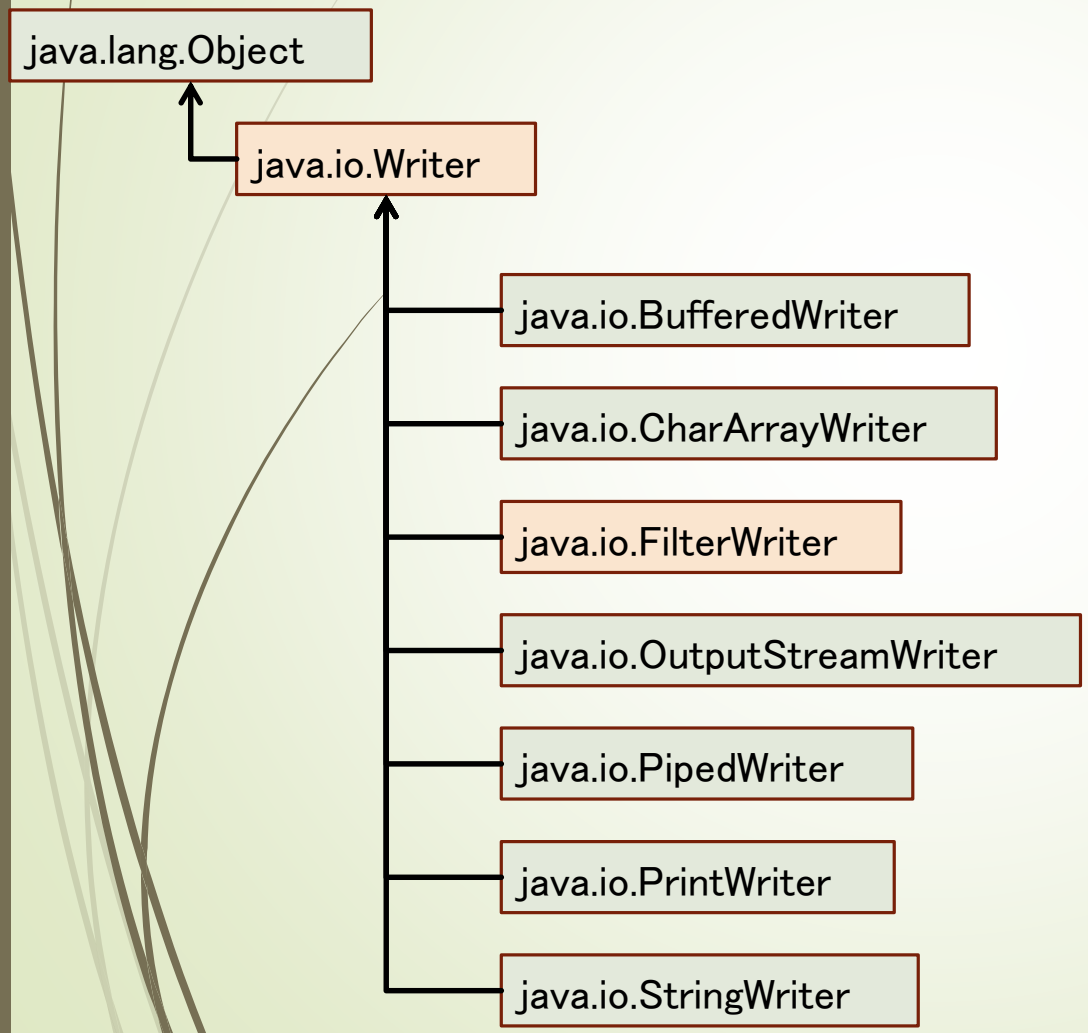
PrintStream

- OutputStreamに機能を追加
  - 文字列書き出し
  - print(String);
  - print(Object)
    - //Object.toStream()が使用される
  - println(String)
  - println(Object)
- 一文字追加
  - append(char);

Writer

- 文字、文字列をストリームに書く
  - void write(char);
  - void write(String);

Writerのクラス階層



Writerの例

```
BufferedReader in;
BufferedWriter out;
try {
    in = new BufferedReader(
        new InputStreamReader(new FileInputStream(inFile)));
    out = new BufferedWriter(
        new OutputStreamWriter(new FileOutputStream(outFile)));
} catch (FileNotFoundException ex) {System.err.println(ex);}
try {
    String line;
    while ((line = in.readLine()) != null) {
        out.write(line);
        out.newline()l//改行
}
in.close();
out.close();
} catch (IOException ex) {System.err.println(ex);}
```

標準出力のwrapping

```
BufferedWriter out
        = new BufferedWriter(new OutputStreamWriter(System.out));
String nl=System.getProperty("line.separator");
try{
    out.write(“Something”);
    out.write(nl);
}catch(IOException ex){
}
```

改行

- 改行コードは、OS依存
  - LF：UNIX、Mac OS X
  - CR+LF：Windows
  - CR：Mac OS 9以前
- Javaは、OS非依存にすべき
  - 実行時に、OSの改行コードを取得

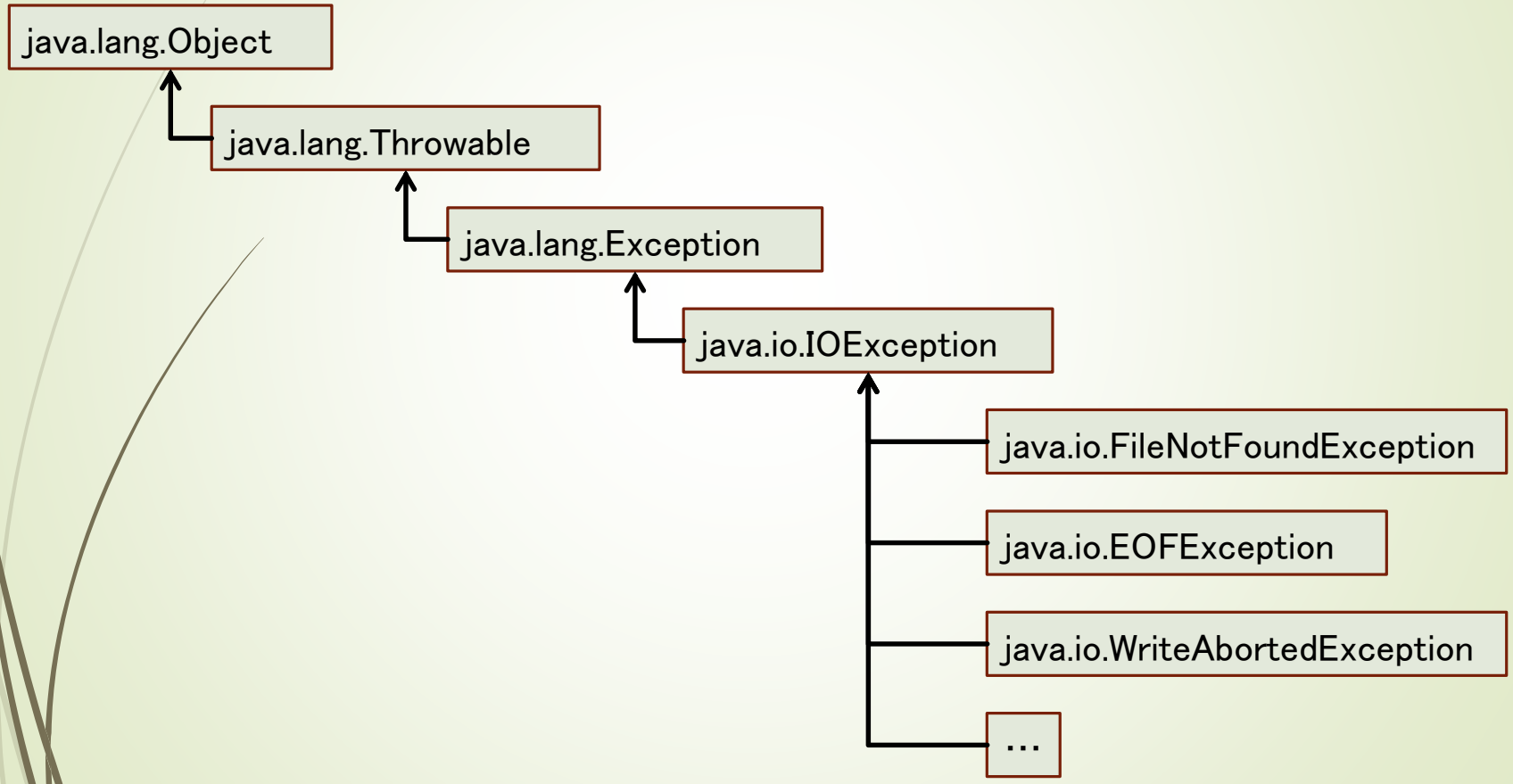
```
String nl = System.getProperty("line.separator") ;
```

例外処理

- ファイルの入出力では、実行時エラーが発生
- ファイルが読めない、ファイルに書けない
- 例外が発生した後、単に停止するのではなく、適切に処理が継続できるように

- メソッド間での例外処理の方法の統一が必要
  - ライブラリとしての挙動の統一
  - ユーザプログラムでの例外処理の簡素化
- 例外もクラスとして定義する

IOExceptionのクラス階層



例外を捕まえる

- 例外が発生する可能性のあるメソッドの実行
  - tryブロックで囲む
- 例外時の処理
  - 例外をcatchする
- 例：input streamを開く
  - FileNotFoundException
- 例：input streamから読む
  - IOException

例外が発生する処理

- メソッド内での処理

```
try{
      例外が発生する処理
   } catch(Exception e){
      エラー処理
   }
```

- 呼び出し側への通知

```
public void method() throws Exception{
....
    例外が発生する処理
}
```

例外処理が発生しない場合でも

```
public void method() throws Exception{
....
   if(条件){
       String message =”メッセージ”;
       throw new Exception(message);
   }
}
```

jdk中のソースファイルの参照

- Netbeans使用中にjdkのソースを見ることができる
- 見たいクラス名の文字列をマウスでダブルクリックして選択
- マウス右ボタン：「ナビゲート」→「ソースへ移動」

課題

- java.net.URLクラスは、URLを開くクラスである。URLクラスは、openStream()メソッドでInputStreamを開くことが可能である。
- 適当なURLからHTMLファイルを取り出すプログラムを作成しなさい。