File IOと例外処理

オブジェクト指向プログラミング特論

2020年度

只木進一：工学系研究科

今日のサンプルプログラム

- https://github.com/oop-mc-saga/FileIOSamples

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
- 一般的な例外処理は後述

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

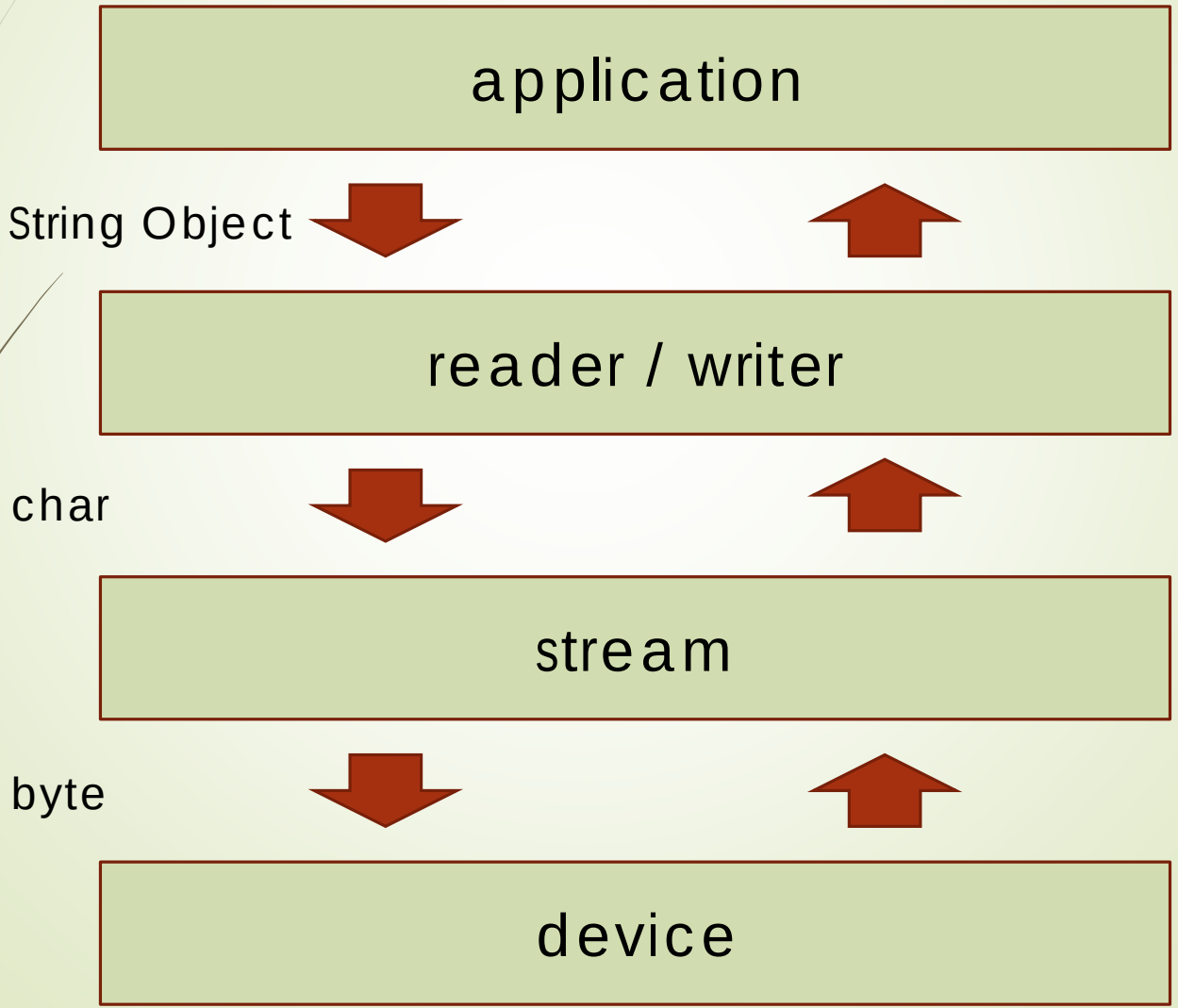
```
1.  StringBuilder b=new StringBuilder();
2.  int c;
3.  try {
4.      while ((c = System.in.read()) != -1) {
5.          b.append((char)c);
6.          //1バイトずつ読んでbに追加
7.      }
8.  } catch (IOException ex) {
9.      //エラー処理
10. }
```

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()メソッドを使用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

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
boolean exists()	ファイルが存在

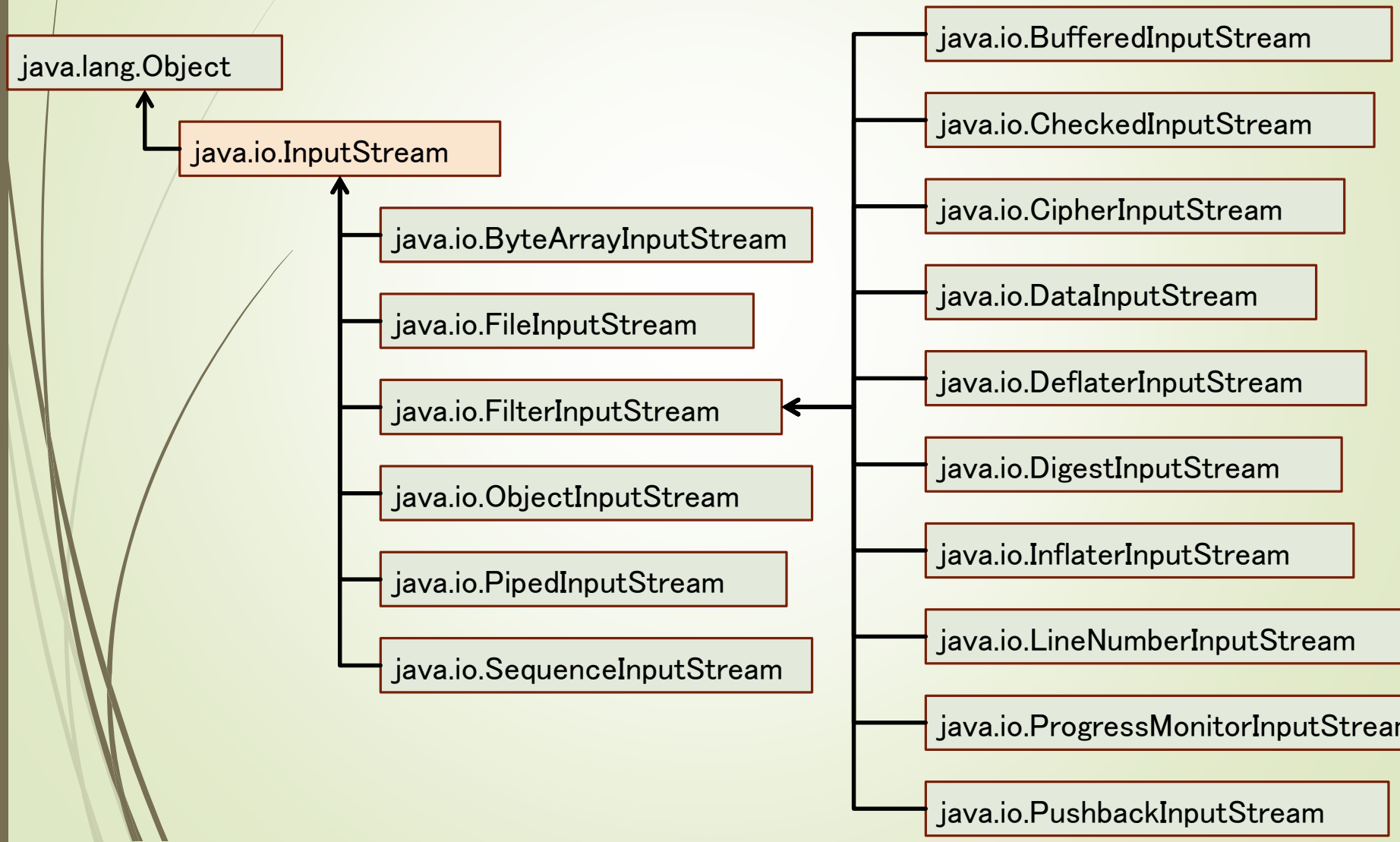
Fileに対応したInputStream

- 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

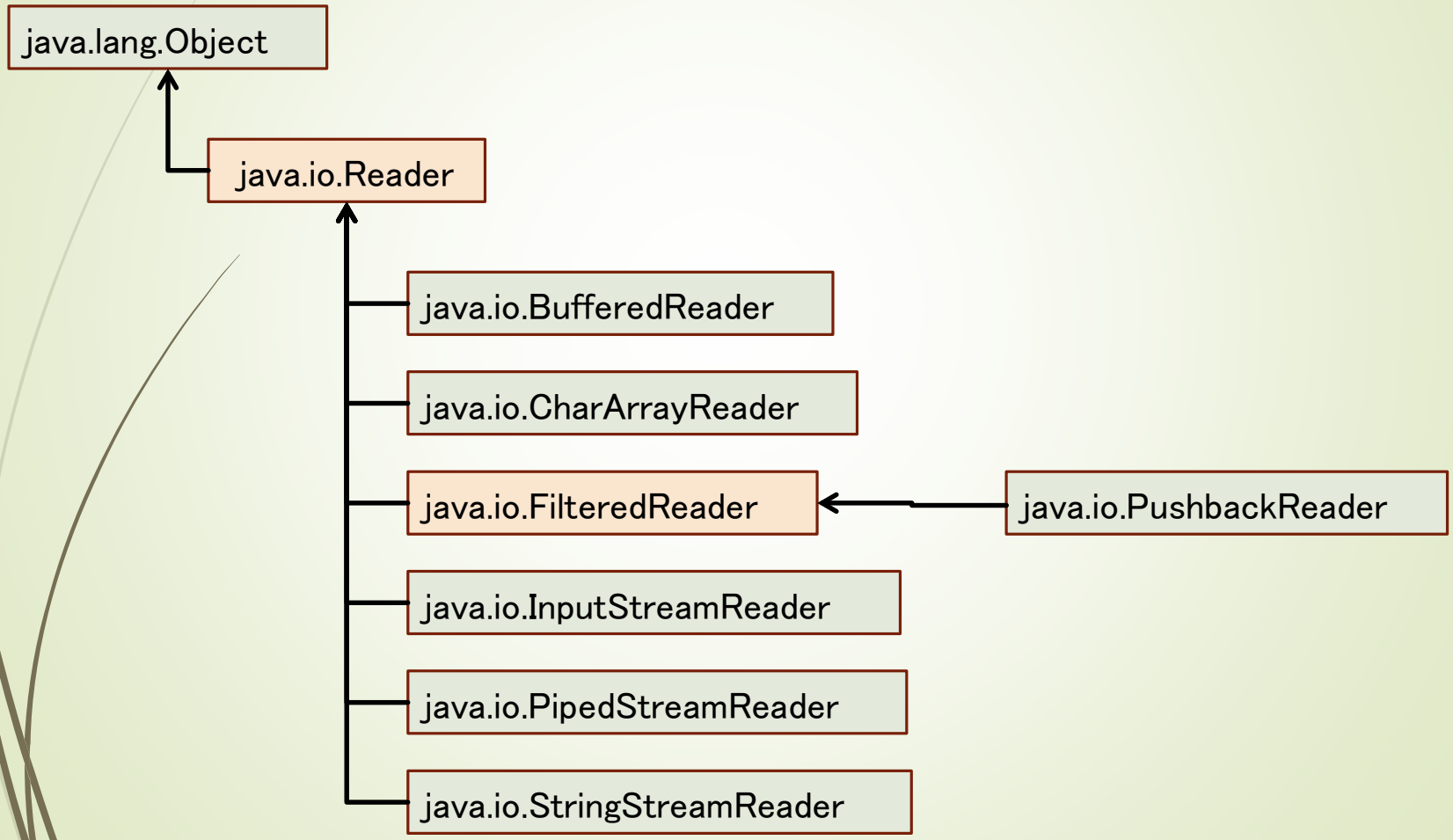
```
1.  //例外が発生すると呼び出し側に知らせる
2.      static public String openInputStream(String filename)
3.              throws IOException {
4.          File file = new File(filename);//ファイル指定
5.          StringBuilder sb = new StringBuilder();
6.          //入力バッファを開く
7.          try ( BufferedInputStream in
8.                  = new BufferedInputStream(
9.                          new FileInputStream(file))) {
10.             int n;
11.             while ((n = in.read()) != -1) {// 1バイト毎に読み込み
12.                 char c = (char) n;//コードを文字へ変換
13.                 sb.append(c);//ビルダへ追加
14.             }
15.         }
16.         return sb.toString();
17.     }
```

FileIOSamples/simplest/Input.java

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



```
1.      static List<String> openReader(String filename)
2.              throws IOException {
3.          File file = new File(filename);
4.          List<String> stringList
5.                  = Collections.synchronizedList(new ArrayList<>());
6.          try ( BufferedReader in = new BufferedReader(
7.                  new InputStreamReader(
8.                          new FileInputStream(file), ENC))) {
9.              String line;
10.             //一行毎に読み込み
11.             while ((line = in.readLine()) != null) {
12.                 stringList.add(line);
13.             }
14.         }
15.         return stringList;
16.     }
```

FileIOSamples/simplest/Input.java

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

標準入力をBufferedReaderに見せる
・出力先に依存しないコードを書く

```
1.  BufferedReader in
2.      = new BufferedReader(
3.          new InputStreamReader(System.in));
4.  try {
5.      String line;
6.      while ((line = in.readLine()) != null) {
7.          System.out.print(line);
8.      }
9.  } catch (IOException ex) {
10.     System.err.println(ex);
11.}
```

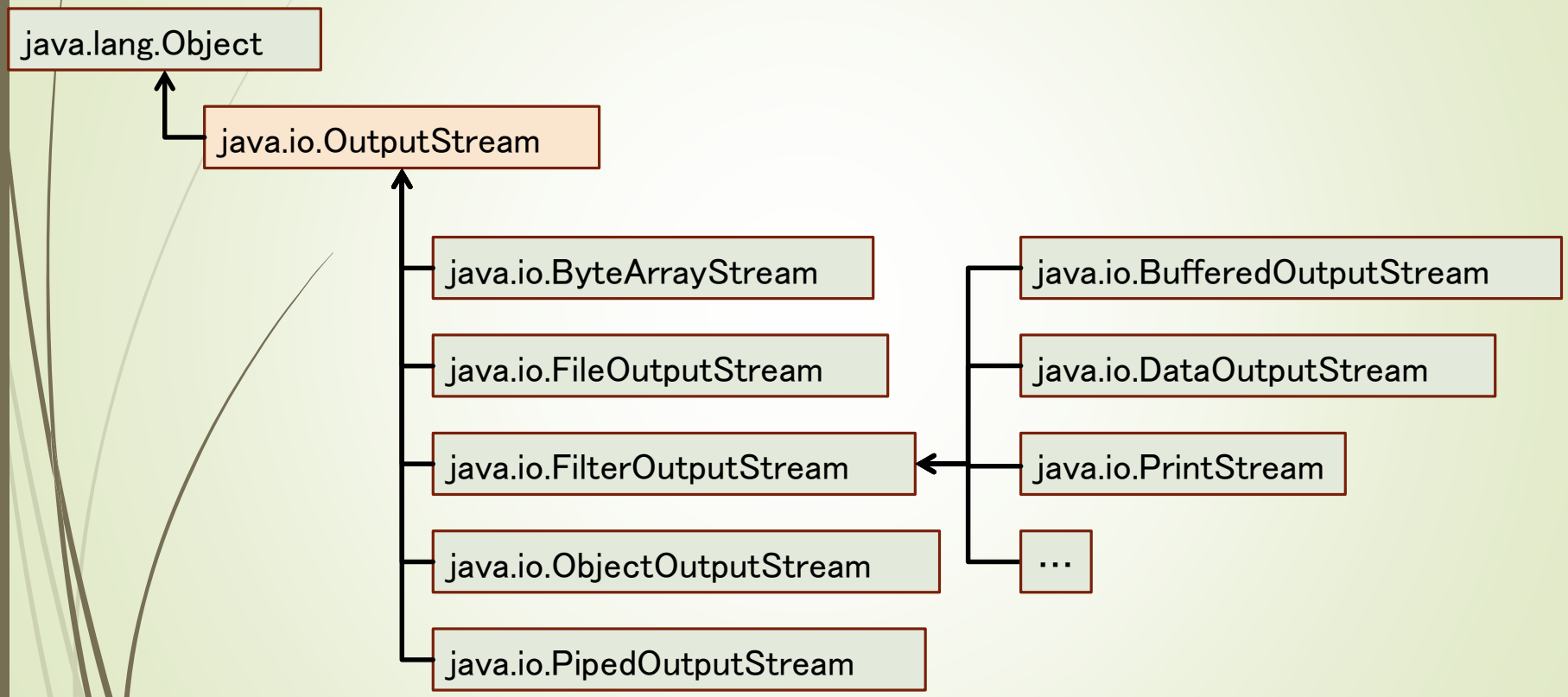
出力

- 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);

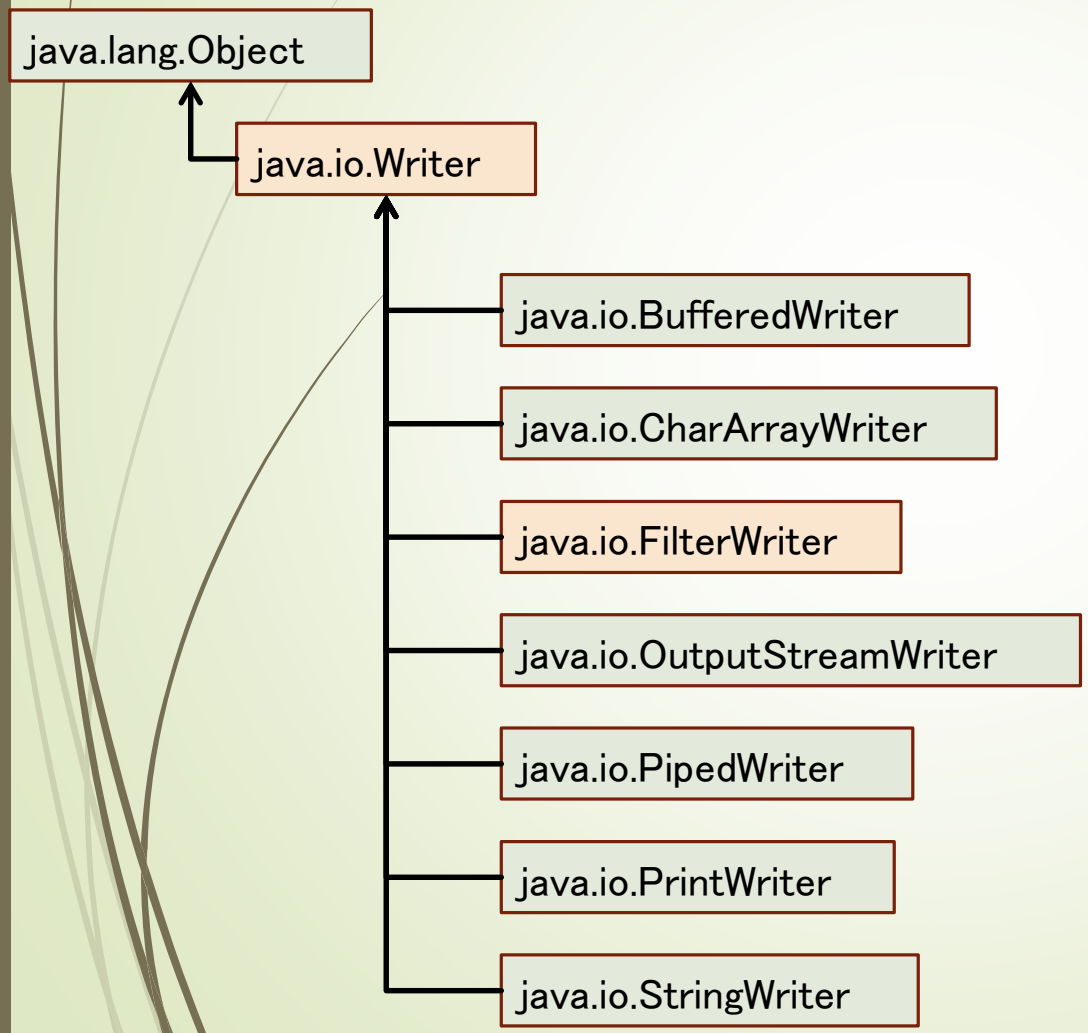
```
1.      public static void main(String[] args)
2.              throws FileNotFoundException {
3.          File file = new File("PrintStreamSampleOutput.txt");
4.          try ( PrintStream out = new PrintStream(file)) {
5.              for (int i = 0; i < 100; i++) {
6.                  int x = (int) (100 * Math.random());
7.                  out.println(x);
8.              }
9.          }
10.     }
```

FileIOSamples/simplest/PrintStreamSample.java

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

```
1.  public static void main(String[] args) throws IOException {
2.      File file = new File("WriterSampleOutput.txt");
3.      try (BufferedWriter out = new BufferedWriter(
4.              new OutputStreamWriter(
5.                      new FileOutputStream(file)))) {
6.          for (int i = 0; i < 100; i++) {
7.              int x = (int) (100 * Math.random());
8.              out.write(String.valueOf(x));
9.              out.newLine();
10.         }
11.     }
12. }
```

FileIOSamples/simplest/WriterSample.java

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){
}
```

具体例

- テキストファイルを、一行ずつ、別ファイルにコピーする
  - fileCopy/FileCopy.java
- バイナリファイルを、1バイトずつ、別ファイルにコピーする
  - fileCopy/BinaryFileCopy.java

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

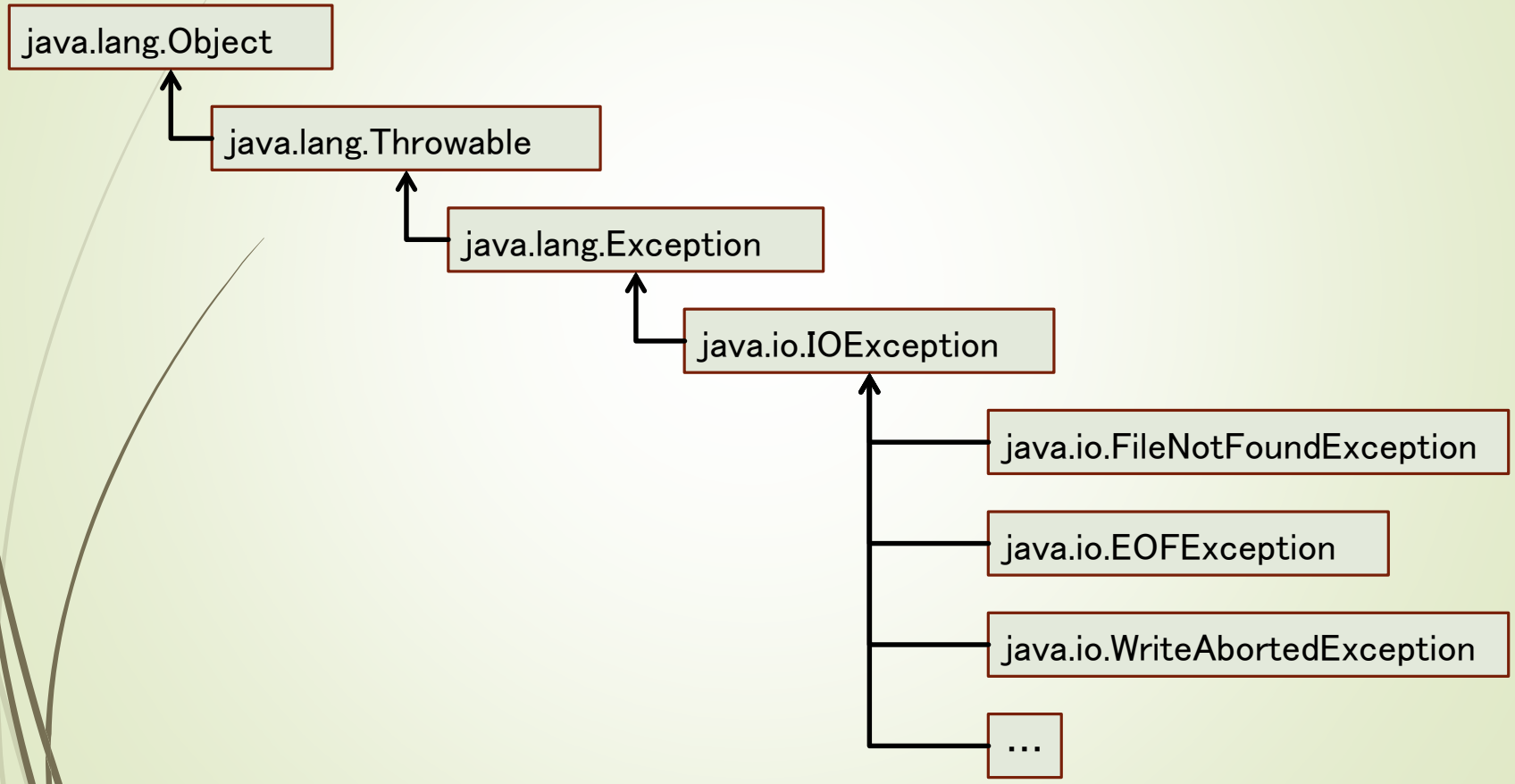
```
String nl = System.getProperty("line.separator");
```

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



例外発生に対する処理

- 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
}
```

例外処理を発生させる

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

例外の例

- ArithmeticException
  - 算術計算の例外。ゼロでの割算など
- ArrayIndexOutOfBoundsException
  - 不正なインデクスを用いた配列アクセス
- IllegalArgumentException
  - 不正な引数
- NumberFormatException
  - 文字列から数値への変換の例外

例外を発生させる例

- ファイルから読み込んだ文字列が数字に変化できない例
  - Exception/ExceptionSample.java
- メソッドの引数が不適切な例
  - Exception/NewtonMethod.java

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」