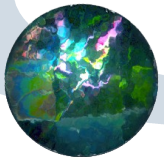
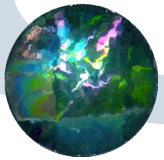
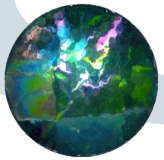


File IO



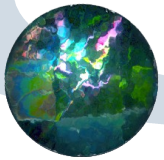
JavaでのFile IO

- 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
- 例外処理の実際
  - 読めない、書けない
  - ファイルが存在しない



標準入出力

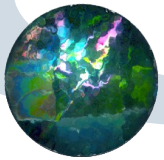
```
package java.lang;
import java.io.*;
public final class System {
  private System() {}//インスタンスは作成不能
  public final static InputStream in;
  public final static PrintStream out;
  public final static PrintStream err;
...
}
```



標準入力：キーボード

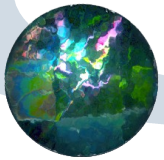
- 一文字ずつの入力
  - メソッドread()を使用
- 戻り値
  - 正整数：文字
  - -1：終了
- 例外発生可能性
  - IOException

```
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
}
```



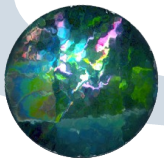
標準出力：端末へ

- 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
    - Object.toString()

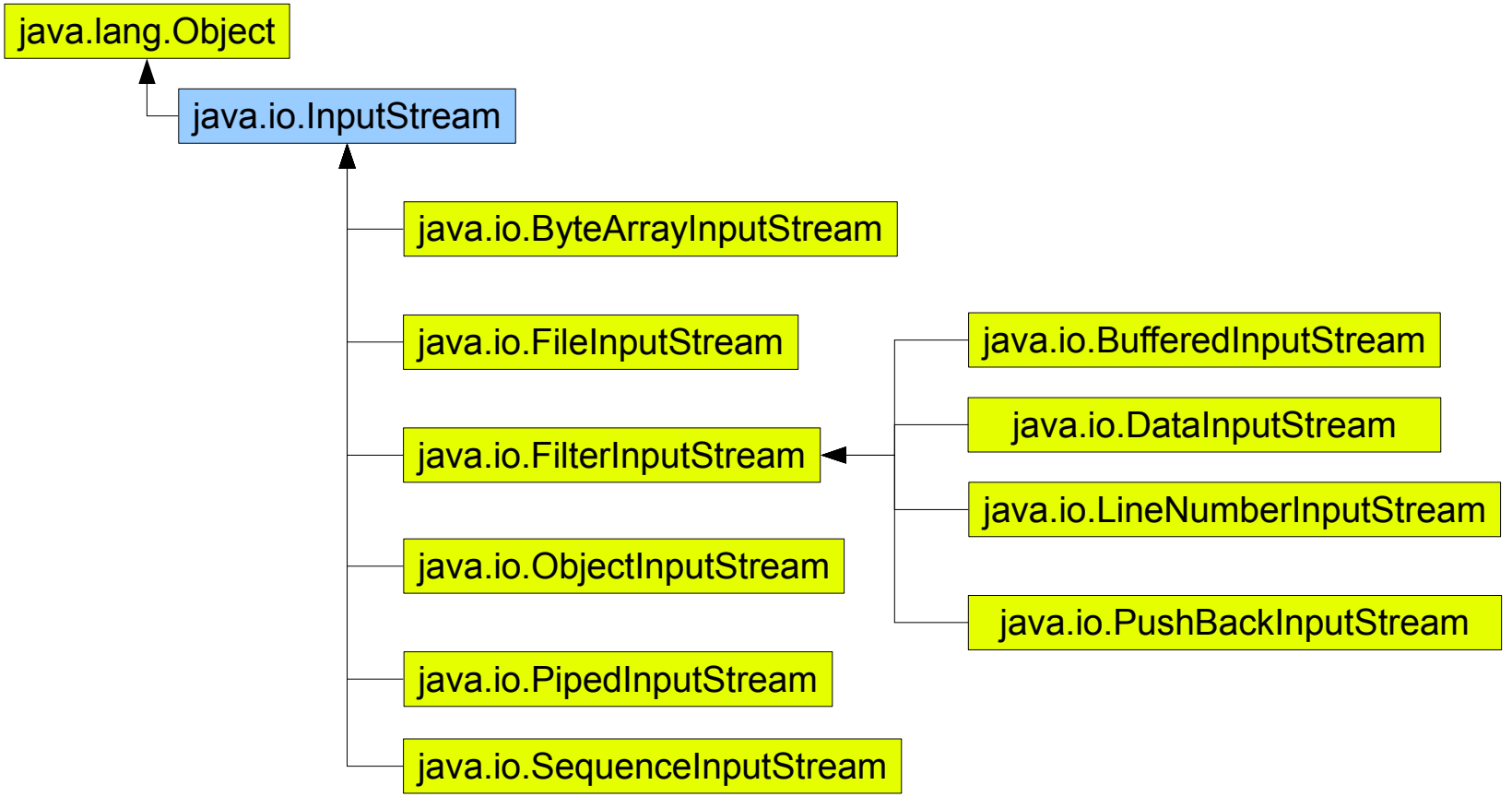


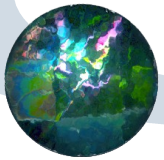
java.io.File

- ファイルそのもののクラス
- ファイルの属性を調べることができる
  - 存在するか否か
  - read/writeパーミッション
  - ファイルかディレクトリ
- ファイルの生成、消去が可能
- インスタンス生成時には、ファイルにアクセスしていないことに注意



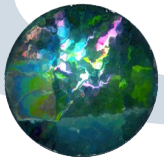
入力ストリーム


java.lang.Object
java.io.InputStream
java.io.ByteArrayInputStream
java.io.FileInputStream
java.io.FilterInputStream
java.io.ObjectInputStream
java.io.PipedInputStream
java.io.SequenceInputStream
java.io.BufferedInputStream
java.io.DataInputStream
java.io.LineNumberInputStream
java.io.PushBackInputStream



InputStreamの基本：1

- 読み込みはbyte単位
  - int read();//1 byte読み込み
  - int read(byte[]); //byte[]で一括読み込み
- 閉鎖
  - void close();



InputStreamの基本：2

- ファイルからの読み込み

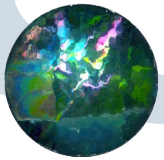
```
File file = new File("input.txt");
BufferedInputStream in=null;
try {
    in = new BufferedInputStream(new FileInputStream(file));
} catch (FileNotFoundException ex) {
    System.err.println(ex);
}

int n;
while((n=in.read())!=-1){
    System.out.print((char)n);
}
```

ファイルを指定

入力バッファを開く

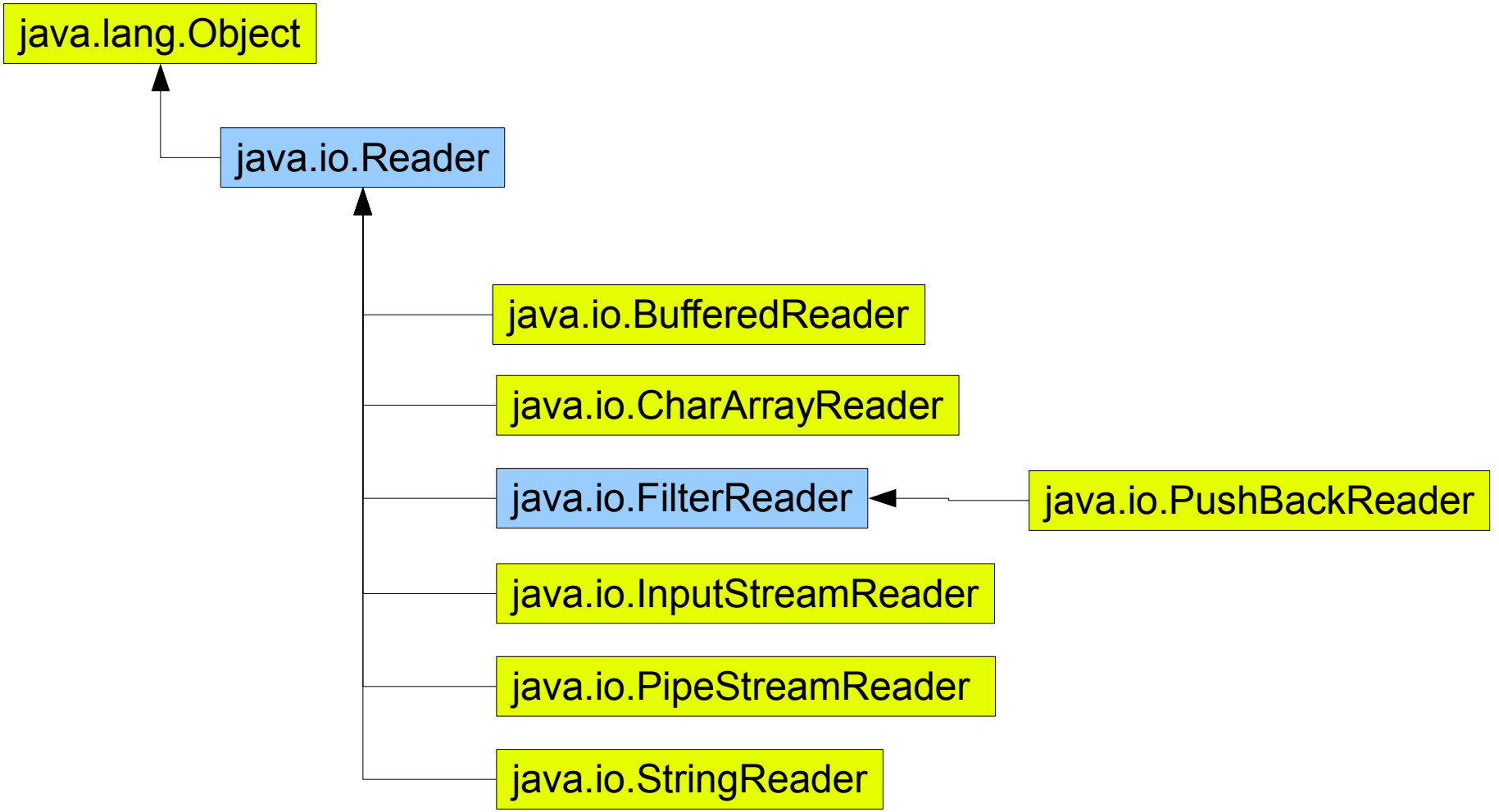
ファイルが見つけられない場合の処理

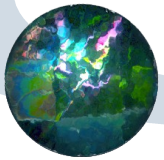
1バイトずつ読み込み



Readerを使う

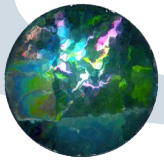
- 文字、文字列単位での読み込み
- int read(); //一文字読み込み
- int read(char[]);//文字配列へ読み込み
- String readLine();//一行を文字列へ読み込み


java.lang.Object
java.io.Reader
java.io.BufferedReader
java.io.CharArrayReader
java.io.FilterReader
java.io.PushBackReader
java.io.InputStreamReader
java.io.PipeStreamReader
java.io.StringReader



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

一行ずつ読み込む



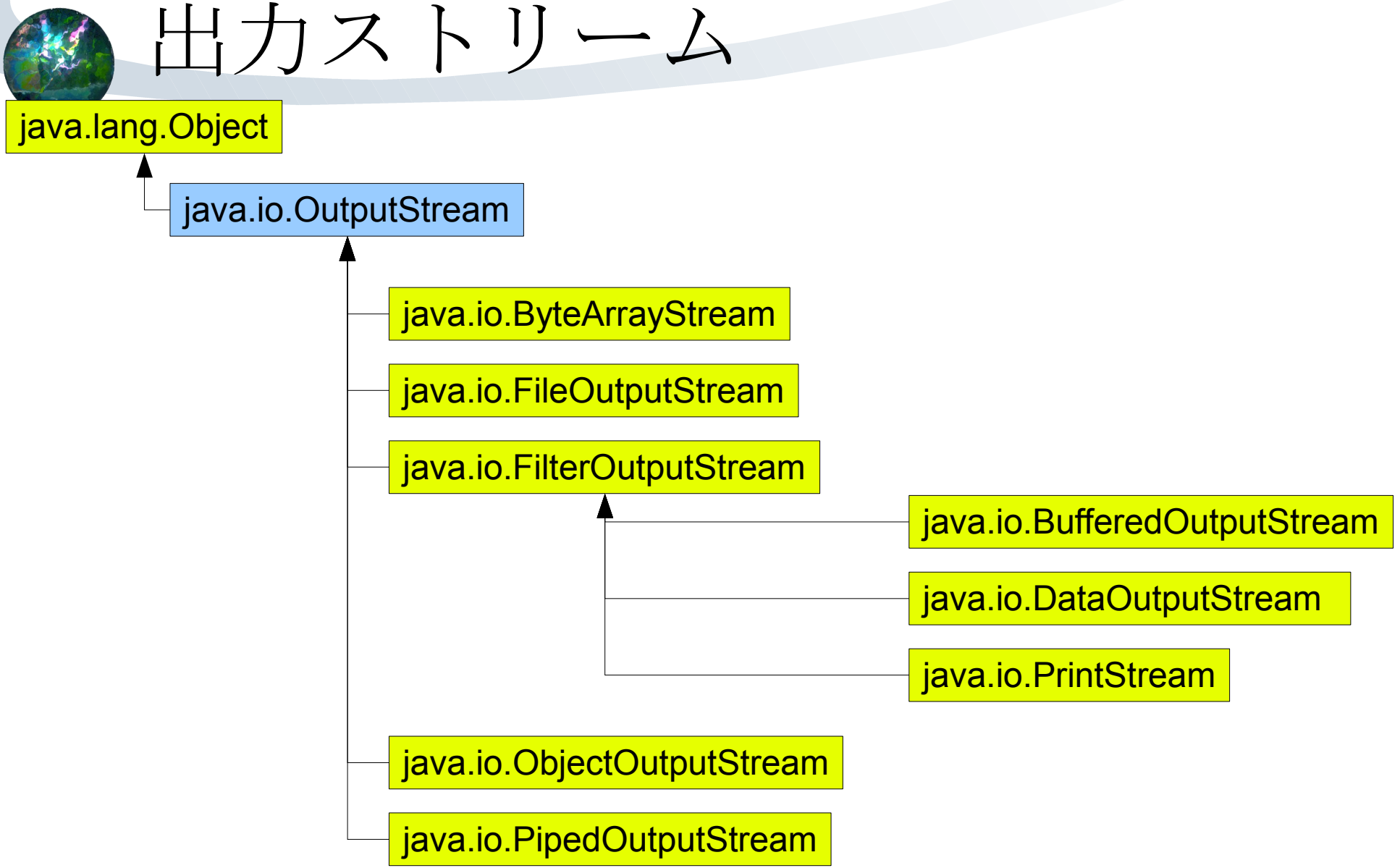
標準入力のwrapping

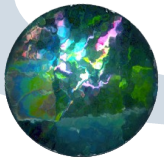
- 標準入力をBufferedReaderの一種に見せかける
  - ファイル書き込みと標準出力を切り替えることができる

標準入力からBufferedReaderを生成

```
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);
}
```

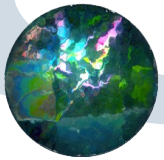
出力ストリーム


java.lang.Object
java.io.OutputStream
java.io.ByteArrayStream
java.io.FileOutputStream
java.io.FilterOutputStream
java.io.BufferedOutputStream
java.io.DataOutputStream
java.io.PrintStream
java.io.ObjectOutputStream
java.io.PipedOutputStream



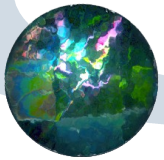
OutputStreamの基本

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



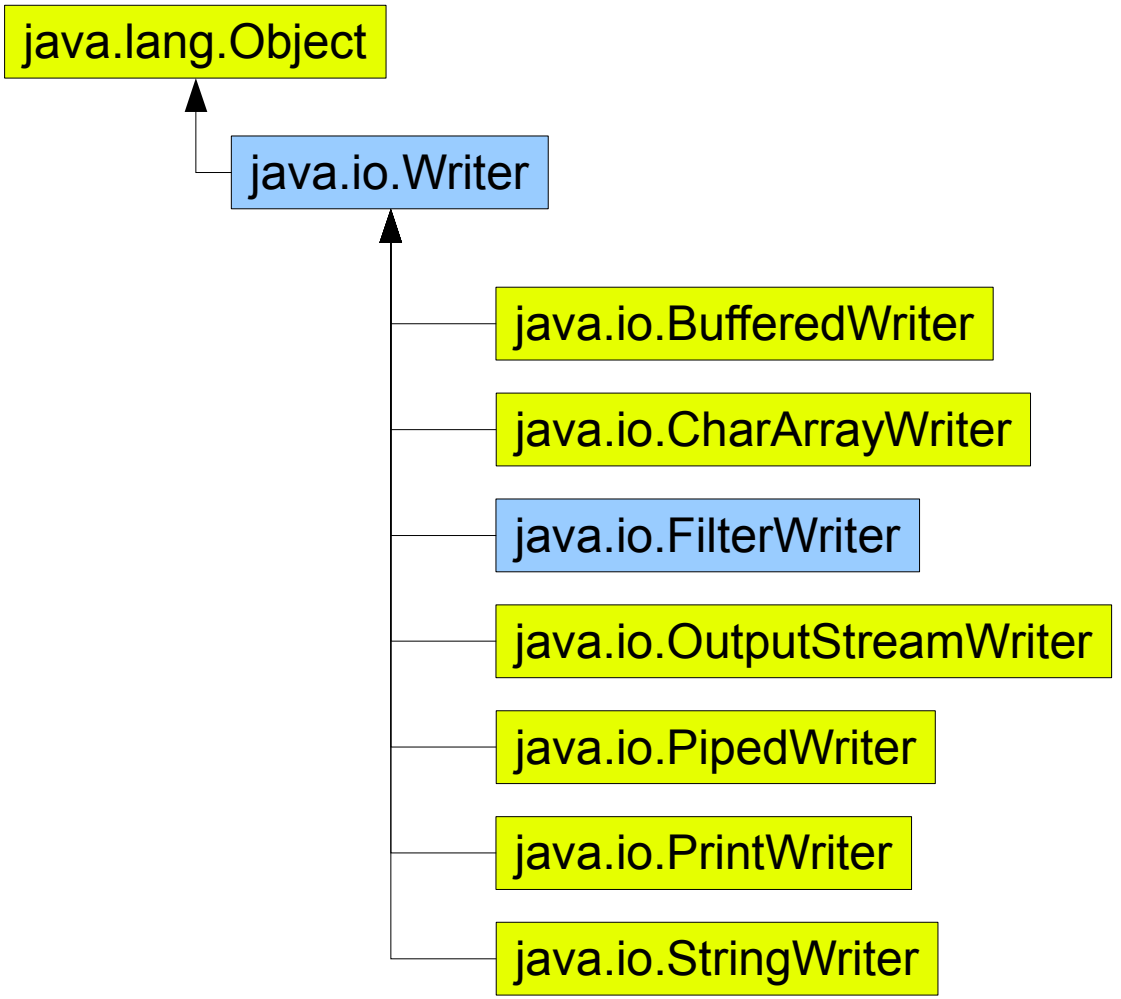
PrintStream

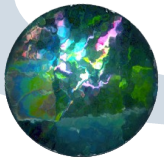
- OutputStreamに機能を追加
- 文字列書き出し
  - print(String);
  - print(Object)
    - //Object.toStream()が使用される
  - println(String)
  - println(Object)
- 一文字追加
  - append(char);



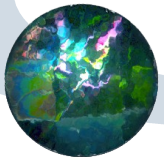
Writer

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


java.lang.Object
java.io.Writer
java.io.BufferedWriter
java.io.CharArrayWriter
java.io.FilterWriter
java.io.OutputStreamWriter
java.io.PipedWriter
java.io.PrintWriter
java.io.StringWriter

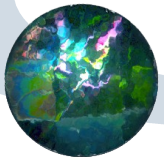


```
BufferedReader in = null;
BufferedWriter out = null;
try {
    in = new BufferedReader(new InputStreamReader(new FileInputStream(inFile)));
    out = new BufferedWriter(
            new OutputStreamWriter(new FileOutputStream(outFile)));
} catch (FileNotFoundException ex) {
    System.err.println(ex);
}
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);
}
```



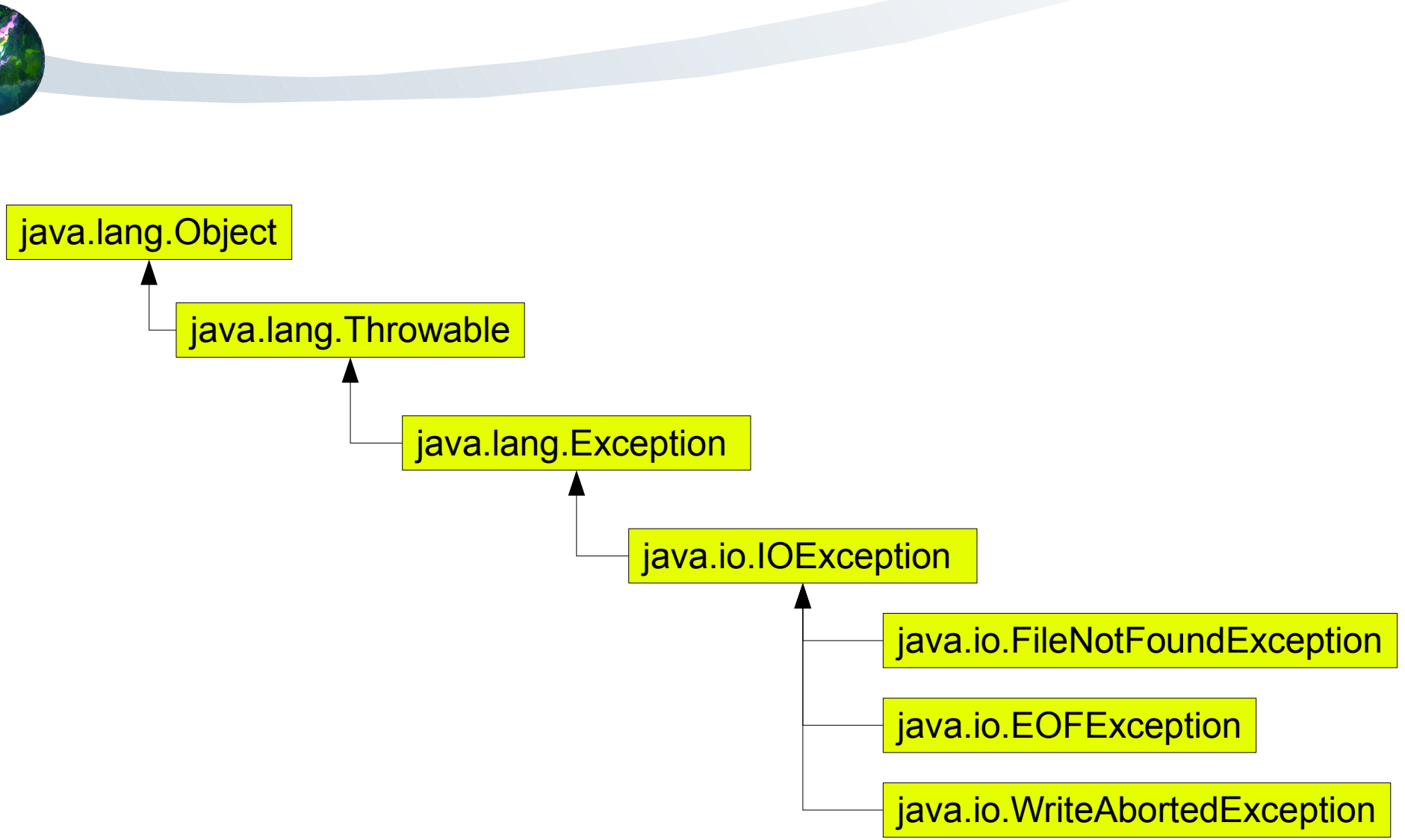
標準出力のwrapping

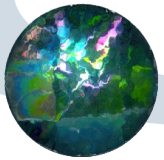
```
BufferedWriter out
        = new BufferedWriter(new OutputStreamWriter(System.out));
String nl=System.getProperty("line.separator");
try{
    out.write(“Something”);
    out.write(nl);
}catch(IOException ex){
}
```



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
- 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

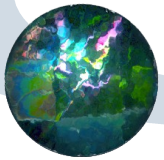

java.lang.Object
java.lang.Throwable
java.lang.Exception
java.io.IOException
java.io.FileNotFoundException
java.io.EOFException
java.io.WriteAbortedException



例外を捕まえる

- 例外を発生させるメソッドの実行
  - 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

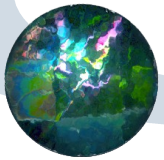
```
try{
  ここで例外発生
}catch(例外 e){
  例外処理
}
```



例外を呼出側に投げる

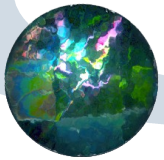
- メソッドの呼び出し時に発生した例外を呼び出し側に伝える

```
public void method() throws Exception{
....
   try{
      ....
   } catch(Exception e){
      throw e;
   }
}
```



例外を呼出側に投げる

```
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」

IOMain.java

```java
package standardIO;

import java.io.IOException;

/**
 *
 * @author tadaki
 */
public class IOMain {
    String src=null;
    public void getData(){
        StringBuilder b=new StringBuilder();
        int c;
        try {
            while ((c = System.in.read()) != -1) {
                b.append((char)c);
            }
        } catch (IOException ex) {
            System.err.println(ex);
        }
        src = b.toString();
    }

    public void printData(){
        System.out.println(src);
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        IOMain ioMain=new IOMain();
        ioMain.getData();
        ioMain.printData();
    }

}
```

FileIOMain.java

```
package FileIO;

import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author tadaki
 */
public class FileIOMain {

    private File inFile;
    private File outFile;
    private final String nl = System.getProperty("line.separator");

    public FileIOMain() throws FileNotFoundException {
        inFile = new File("input.txt");
        if (!inFile.isFile() || !inFile.canRead()) {
            String message = "入力ファイル"
                    + inFile.getAbsolutePath() + "がありません。";
            throw new java.io.FileNotFoundException(message);
        }
        outFile = new File("output.txt");
    }

    public int getData() {
        int n = 0;
        if (!outFile.exists()) {
            try {
                outFile.createNewFile();
            } catch (IOException ex) {
                Logger.getLogger(
                        FileIOMain.class.getName()).log(
                        Level.SEVERE, null, ex);
                return 0;
            }
        }
        if (outFile.canWrite()) {
            BufferedReader in = null;
            BufferedWriter out = null;
            try {
                in = new BufferedReader(new InputStreamReader(
                        new FileInputStream(inFile)));
```

```
                out = new BufferedWriter(
                        new OutputStreamWriter(
                        new FileOutputStream(outFile)));
            } catch (FileNotFoundException ex) {
                Logger.getLogger(
                        FileIOMain.class.getName()).log(
                        Level.SEVERE, null, ex);
                return 0;
            }
            try {
                String line;
                while ((line = in.readLine()) != null) {
                    n++;
                    out.write(line);
                    out.write(nl);
                }
                in.close();
                out.close();
            } catch (IOException ex) {
                Logger.getLogger(
                        FileIOMain.class.getName()).log(
                        Level.SEVERE, null, ex);
                n = 0;
            }
        }
        return n;
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        FileIOMain main = null;
        try {
            main = new FileIOMain();
        } catch (FileNotFoundException ex) {
            Logger.getLogger(
                    FileIOMain.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
        if (main != null) {
            int n = main.getData();
            System.out.println(n);
        }
```

```
        }
    }
```