Manipulating Files and IO exceptions

Object Oriented Programming
2024 First Semester
Shin-chi Tadaki (Saga University)

Today's sample programs

- `https://github.com/oop-mc-saga/FileIOExamples`

File IO (Input and Output) in Java

- File IO functions are not included in `java.lang`
  - `java.lang` contains only standard IO functions
- A separate package `java.io` provides File IO functions.

IO exceptions

- IO exceptions are inevitable. They will happen when the specified file is not
  - readable, or writable by access controls,
  - found,
  - etc.
- General exceptions will be discussed later.
- Handling exceptions enables us to prevent applications failures.
  - If not, applications will be aborted by exceptions

Standard input and output

```
1  package java.lang;
2  import java.io.*;
3  public final class System{
4      private System(){}
5      public final static InputStream in;
6      public final static PrintStream out;
7      public final static PrintStream err;
8  }
```

- Standard input and output are aliases for `java.io.InputStream` and `java.io.PrintStream`.

Standard input: from keyboard

- Read character by character.
  - `int read()`: reads the next one byte and returns character code.
  - `int read(byte[] b)`: reads some number of bytes and returns the number of bytes.
  - Both methods will throws `IOException`

```
1  StringBuilder b = new StringBuilder();
2  int c;
3  try {
4      while((c = System.in.read()) != -1){
5          b.append((char)c);
6          //read 1byte data and append to b
7      }
8  } catch (IOException ex){
9      //Error handling
10 }
```

See `simplest/StandardInput.java`

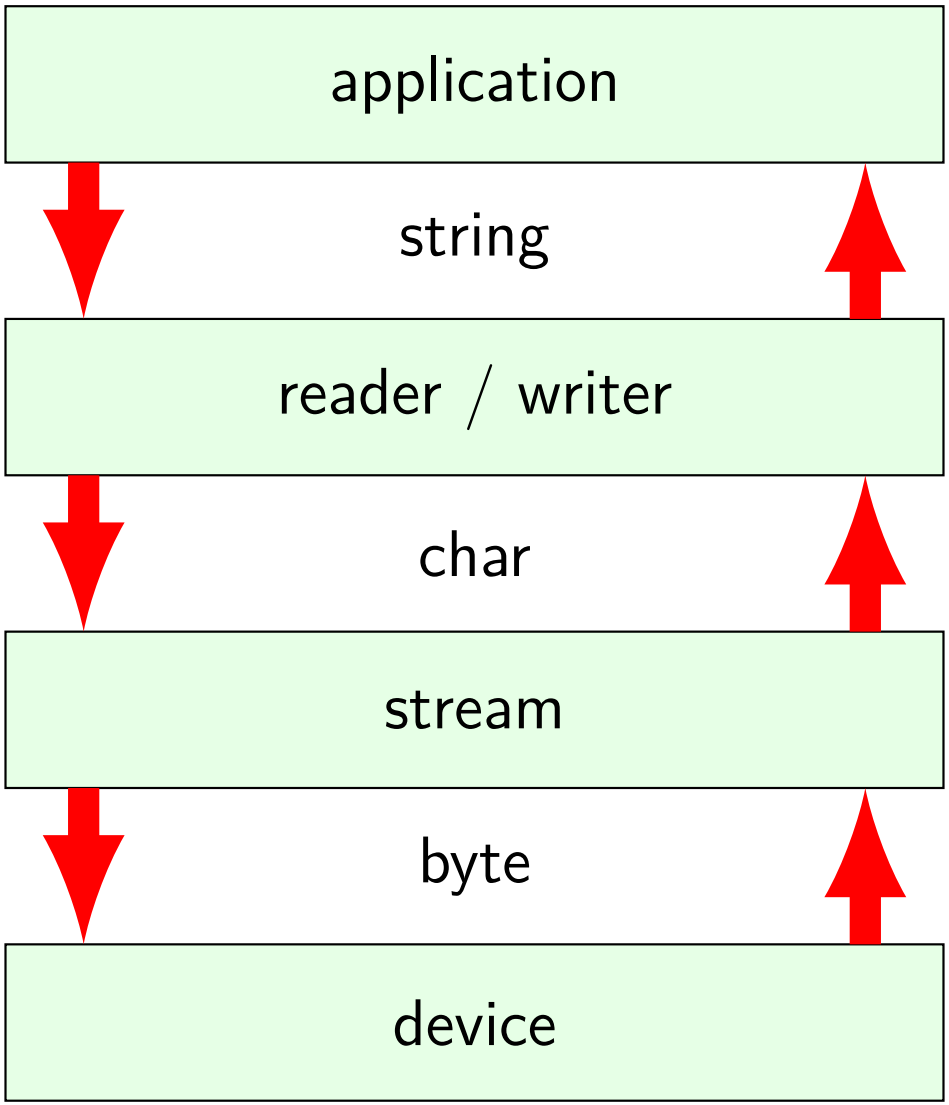
Standard output

- `void print()`: prints a string
- `void println()`: prints a string then terminates the line
- Arguments of the methods
  - primitive data types
  - objects: converting to a string using `toString()` method

Improving IO functions

- Various sources and destinations of IOs
  - standard IO, files, network resources
- Hierarchical structure between applications and IO resources

Hierarchical structure of IOs



Buffering

- Peripherals are slower than CPU
- Buffering is necessary for sending and receiving data
- Use `stream` or `reader/writer`

Input classes

- Specify a file by `File` class
- `FileInputStream`
- `InputStreamReader`
- `BufferedReader`

Specify a file

- `File` class
  - `File file = new File(String filename)`
- Note: the constructor of `File` class does not check the existence and accessability of the specified file.
- The File class provides functions for testing the existence and accessability of the file.

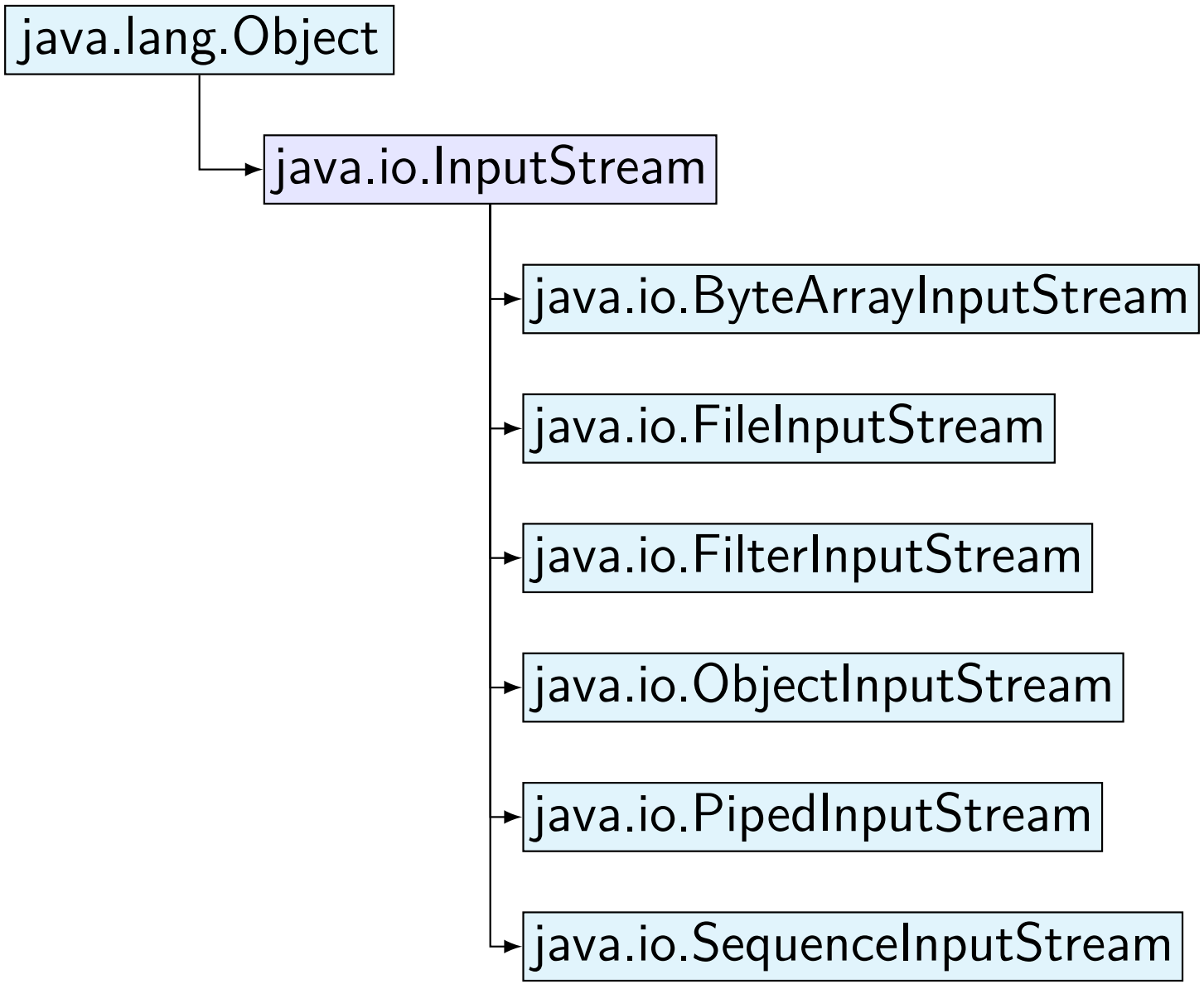
methods	operations
`boolean canRead()`	test the file readable
`boolean canWrite()`	test the file writable
`boolean createNewFile()`	create a new file
`boolean exists()`	test the file existence

FileInputStream class

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

- `int read()`
  - Reads data one byte
  - returns -1 if end

Hierarchy of InputStream classes



Example 4.1: InputStream

```
1  static public String readFromInputStream(String filename)
2          throws IOException {
3      File file = new File(filename);//Specify file for reading
4      StringBuilder sb = new StringBuilder();
5      //Open input buffer
6      try ( BufferedInputStream in
7              = new BufferedInputStream(
8                      new FileInputStream(file))) {
9          int n;
10         while ((n = in.read()) != -1) {//Read byte by byte
11             char c = (char) n;//Convert byte to character
12             sb.append(c);//append to string builder
13         }
14     }
15     return sb.toString();
16 }
```

simplest/Input.java

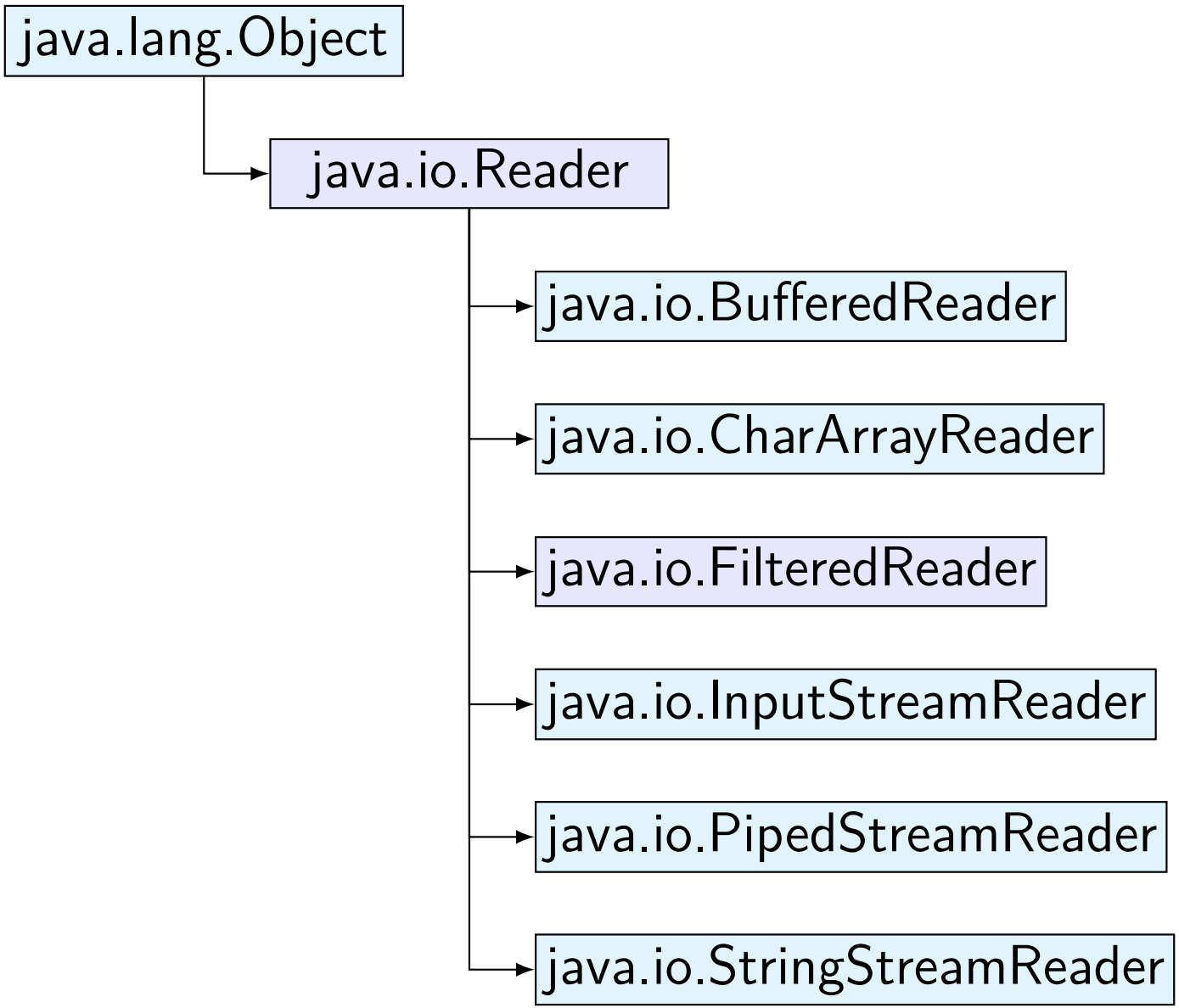
try-with-resources

- The previous example use `try` clause without `catch`.
- It catches exceptions relating to resources.
- It also closes the used resources.

`BufferedReader` class

- Reading by byte is inconvenient for handling text
- `Reader` class provide reading string lines from stream
  - `int read()`: reads one character
  - `int read(char[])`: reads characters into the array.
  - `String readLine()`: reads one string line

Hierarchy of Reader classes



Example 4.2: BufferedReader

```
1  static List<String> readFromReader(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         //read line by line
11         while ((line = in.readLine()) != null) {
12             stringList.add(line);
13         }
14     }
15     return stringList;
16 }
```

simplest/Input.java

Example 4.3: Wrapping standard input

- Standard input can be wrapped into `inputStreamReader`

```
1  public static List<String> wrapping() {
2      List<String> stringList
3              = Collections.synchronizedList(new ArrayList<>());
4      BufferedReader in = new BufferedReader(
5              new InputStreamReader(System.in));
6      try {
7
8          String line;
9          while ((line = in.readLine()) != null) {
10             stringList.add(line);
11         }
12     } catch (IOException ex) {
13         System.err.println(ex);
14     }
15     return stringList;
16 }
```

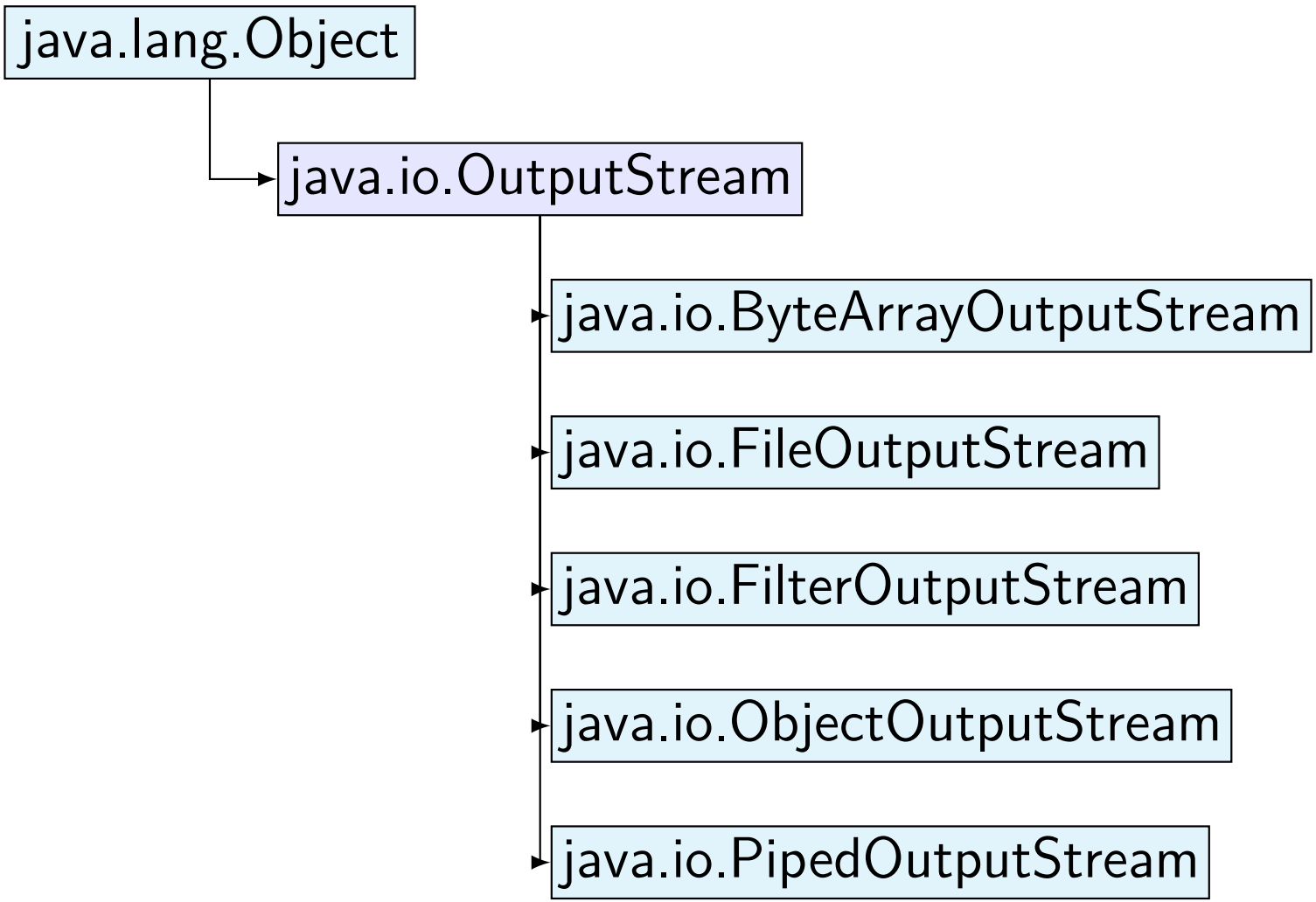
Output classes

- Specify file by `File` class
- `FileOutputStream`
- `PrintStream`
- `BufferedWriter`

`OutputStream` class

- Write by bytes
  - `void write(byte[])`
- Flush this output stream
  - Output processes delay because of buffering.
  - Sometime we need to flush buffered data to destinations.
  - `void flush()`
- Close this stream
  - `void close()`

Hierarchy of output streams



`PrintStream` classNode

- Extends `FilterOutputStream`
- Add some methods to `OutputStream`
- Output strings
  - `print(Object)`
  - `println(Object)`
- Add one character
  - `append(char)`

Example 5.1: PrintStream

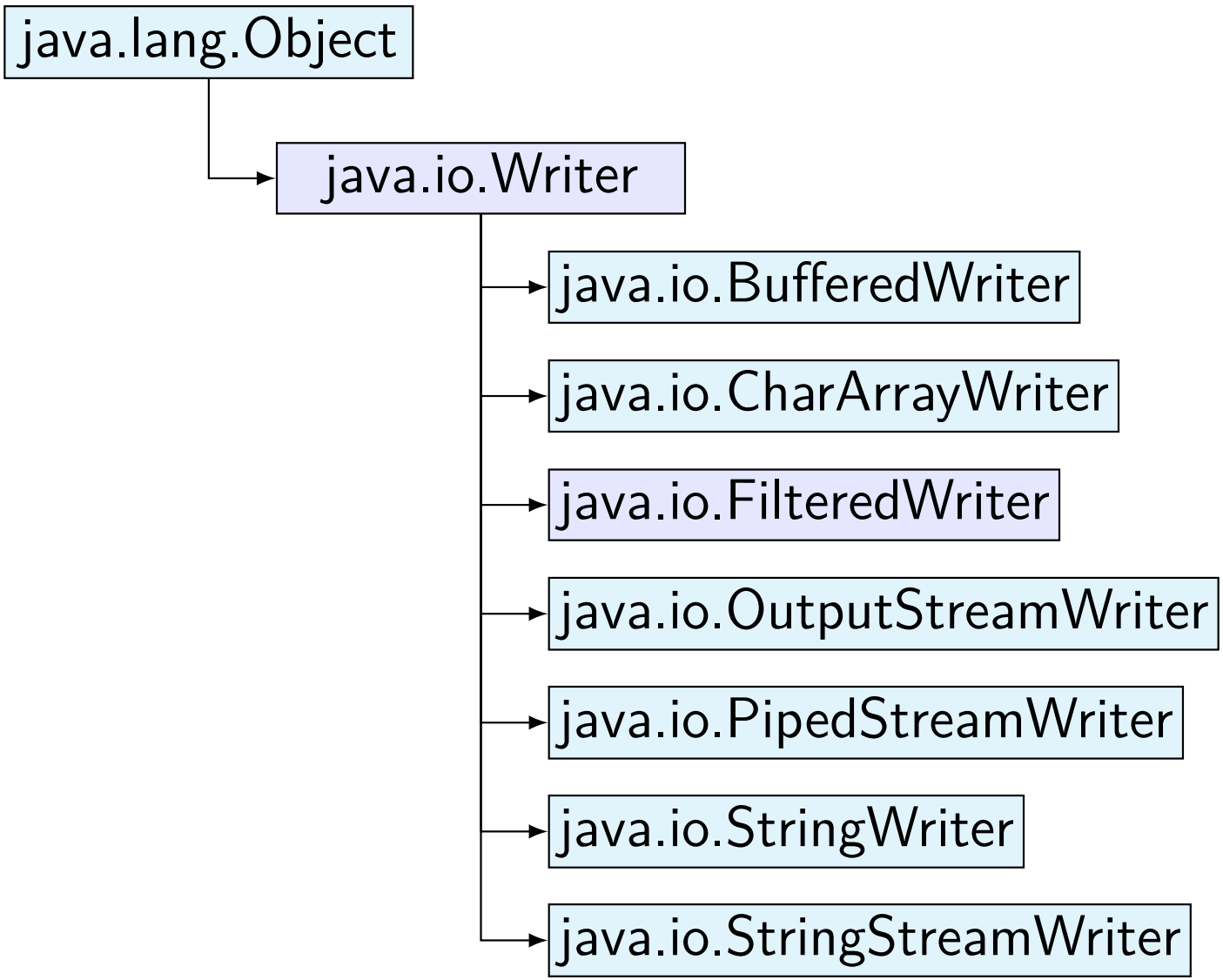
```
1  public static void main(String[] args)
2          throws FileNotFoundException {
3      File file = new File("PrintStreamSampleOutput.txt");
4      try ( PrintStream out = new PrintStream(file)) {
5          for (int i = 0; i < 100; i++) {
6              int x = (int) (100 * Math.random());
7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

simplest/PrintStreamExample.java

`BufferedWriter` class

- Put characters and strings into the stream
  - `void write(char)`
  - `void write(String)`
  - `void newLine()`

Hierarchy of writers



Example 5.2: BufferedWriter

```
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 }
```

simplest/WriterExample.java

Example 5.3: Wrapping standard output

```
1  public static void wrapping() {
2      BufferedWriter out = new BufferedWriter(
3              new OutputStreamWriter(System.out));
4      try {
5          out.write("Something");
6          out.newLine();
7      } catch (IOException ex) {
8          System.err.println(ex);
9      }
10 }
```

Other IO examples

- Copy text file by line
  - `fileCopy/FileCopy.java`
- Copy binary file by byte
  - `fileCopy/BinaryFileCopy.java`

Note: line break codes

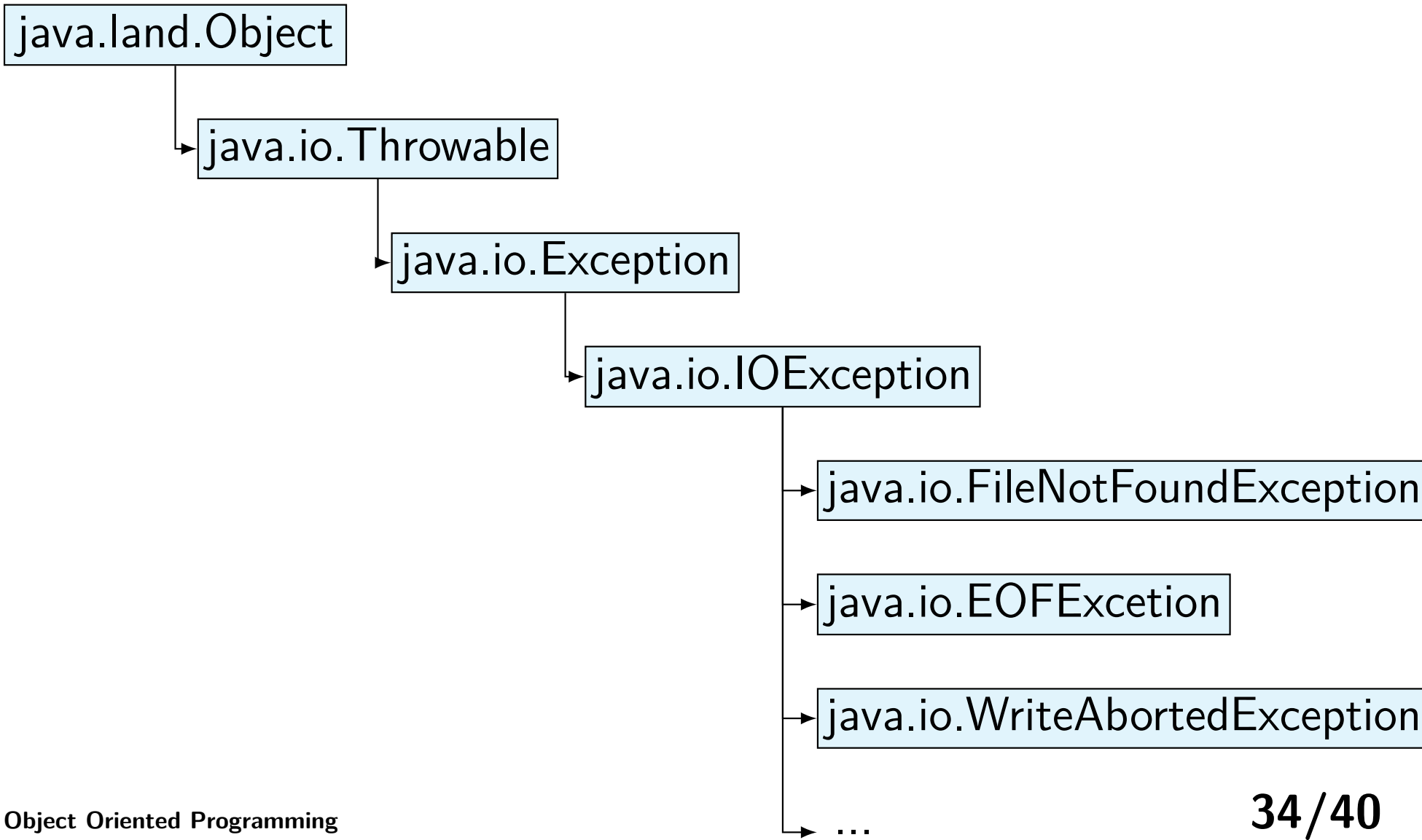
- Line break codes depend on OS.
  - UNIX, Linux, MacOS(> 9): LF (`0x0a`)
  - Windows: CR+LF (`0x0d0a`)
- Write OS independent code by Java

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

Exceptions

- Exceptions are inevitable in IO operations
- Applications should handle exceptions for preventing applications from being aborted
- Java defines exceptions as class
- Exception classes provide consistent methods for handling exceptions

Hierarchy of exception classes



General ways for handling exceptions

- Inside method

```
1  try{
2      //Something will throw exceptions
3  } catch (Exception ex){
4      //Error Handling
5  }
```

- Notify exception to caller

```
1  public void method() throws Exception{
2      //Something will throw exceptions
3  }
```

Example 6.1: Generating exceptions

```
1  public void method() throws Exception{
2      if(something){
3          String message="error message";
4          throw new Exception(message);
5      }
6  }
```

Other exceptions

- `ArithmeticException`: exceptional arithmetic conditions
- `ArrayIndexOutOfBoundException`: an array has been accessed with an illegal index
- `IllegalArgumentException`: a method has been passed an illegal or inappropriate argument
- `NumberFormatException`: the string does not have the appropriate format for expressing numbers.

Examples

- The application tries to read numerics from a file, which contains non-numeric strings
  - `Exception/ExceptionExample.java`
- The method receives inappropriate Arguments
  - `Exception/NewtonMethod.java`

How to see source files of jdk libraries

- in Netbeans
  - select class name by double-click
  - mouse right button: navigate $\rightarrow$ go to source

Exercise

Implement `copyData()` method in `BinaryFileCopy.java`.