Manipulating Files and IO exceptions

Object Oriented Programming
2022 First Semester
Shin-chi Tadaki (Saga University)

Today's sample programs

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

File IO (Input and Output) in Java

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

IO exceptions

- IO exceptions are inevitable
  - Can not read a file. Can not write a file.
  - File not found
- General exceptions will be shown later.
- Handling exceptions enables us to prevent applications fail.

Standard input and output

```java
package java.lang;
import java.io.*;
public final class System{
    private System(){}
    public final static InputStream in;
    public final static OutputStream out;
    public final static PrintStream err;
}
```

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

Standard input: keyboard

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

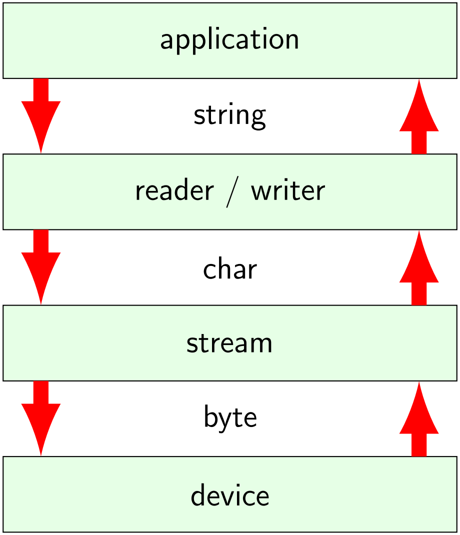
```java
StringBuilder b = new StringBuilder();
int c;
try {
    while((c = System.in.read()) != -1){
        b.append((char)c);
        //read 1byte data and append b
    }
} catch (IOException ex){
    //Error handling
}
```

Standard output

- `void print()`: print
- `void println()`: print then terminate the line
- Arguments of methods
  - primitive data types
  - objects: using `toString()` method

Improving IO functions

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


application
string
reader / writer
char
stream
byte
device

Buffering

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

Input

- 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 file.
- Need to test the existence and accessability of the file before using.

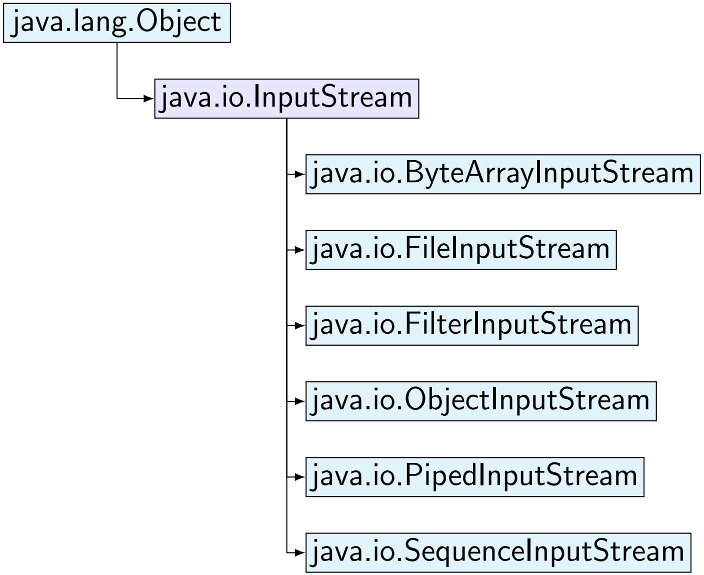
method	operation
`boolean canRead()`	test the file readable
`boolean canWrite()`	test the file writable
`boolean createNewFile()`	create a new file
`boolean exists()`	test the file existence

FileInputStream class

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

- `int read()`
  - Read data one byte
  - return -1 if end

Hierarchy of InputStream classes


java.lang.Object
java.io.InputStream
java.io.ByteArrayInputStream
java.io.FileInputStream
java.io.FilterInputStream
java.io.ObjectInputStream
java.io.PipedInputStream
java.io.SequenceInputStream

Example of InputStream

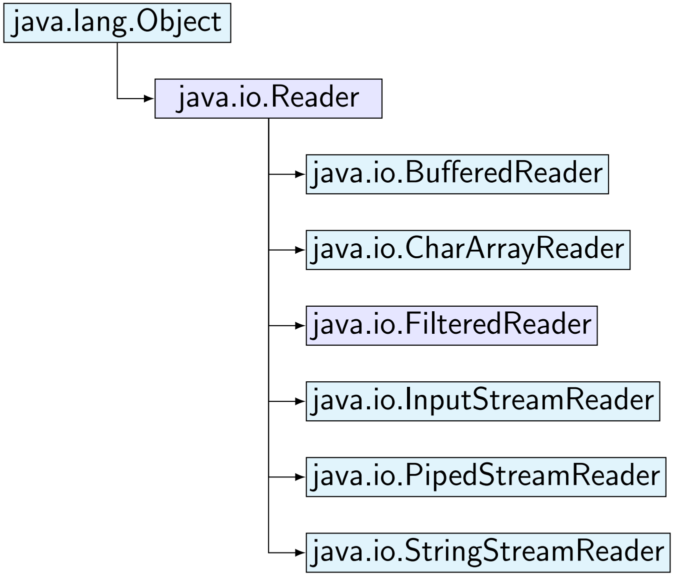
```java
static public String openInputStream(String filename)
        throws IOException {
    File file = new File(filename);//Specify file for reading
    StringBuilder sb = new StringBuilder();
    //Open input buffer
    try ( BufferedInputStream in
            = new BufferedInputStream(
                    new FileInputStream(file))) {
        int n;
        while ((n = in.read()) != -1) {//Read byte by byte
            char c = (char) n;//Convert byte to character
            sb.append(c);//append to string builder
        }
    }
    return sb.toString();
}
```

simplest/Input.java

`BufferedReader` class

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

Hierarchy of Reader classes



```java
static List<String> openReader(String filename)
        throws IOException {
    File file = new File(filename);
    List<String> stringList
            = Collections.synchronizedList(new ArrayList<>());
    try ( BufferedReader in = new BufferedReader(
            new InputStreamReader(
                    new FileInputStream(file), ENC))) {
        String line;
        //read line by line
        while ((line = in.readLine()) != null) {
            stringList.add(line);
        }
    }
    return stringList;
}
```

`simplest/Input.java`

Wrapping standard input

```java
public static void wrapping() {
    BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));
    try {
        String line;
        while ((line = in.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException ex) {
        System.err.println(ex);
    }
}
```

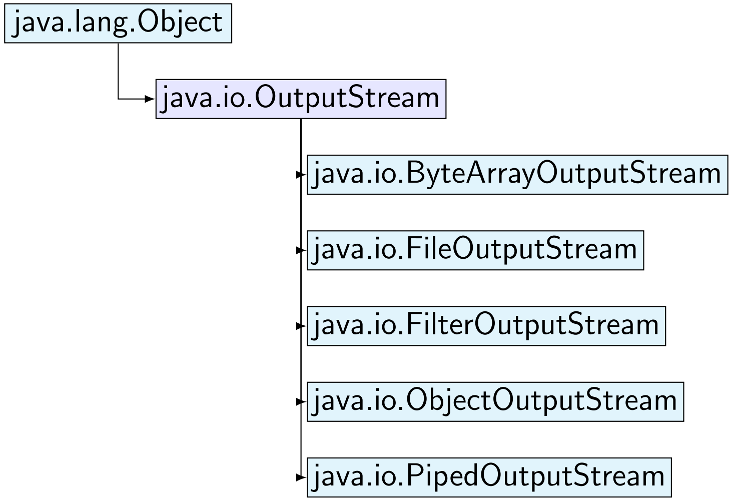
Output

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

`OutputStream` class

- Write by bytes
  - `void write(byte[])`
- Flush this output stream
  - `void flush()`
- Close this stream
  - `void close()`

Hierarchy of output streams


java.lang.Object
java.io.OutputStream
java.io.ByteArrayOutputStream
java.io.FileOutputStream
java.io.FilterOutputStream
java.io.ObjectOutputStream
java.io.PipedOutputStream

`PrintStream` classNode

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

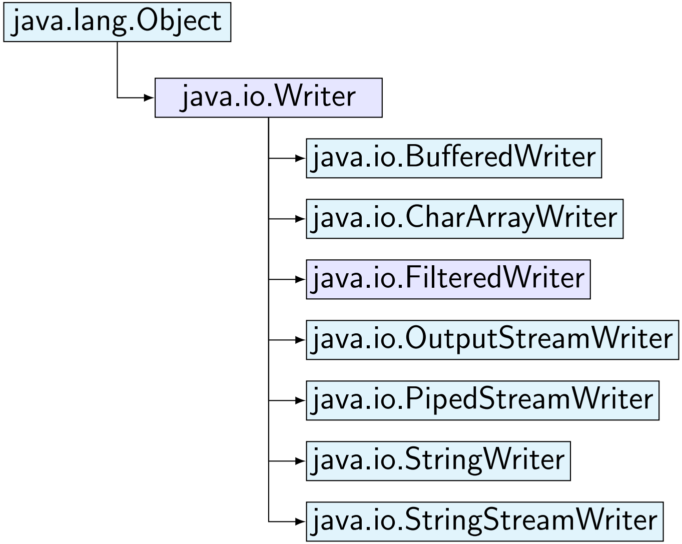
```
public static void main(String[] args)
        throws FileNotFoundException {
    File file = new File("PrintStreamSampleOutput.txt");
    try ( PrintStream out = new PrintStream(file)) {
        for (int i = 0; i < 100; i++) {
            int x = (int) (100 * Math.random());
            out.println(x);
        }
    }
}
```

simplest/PrintStreamSample.java

`BufferedWriter` class

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

Hierarchy of writers



```java
public static void main(String[] args) throws IOException {
    File file = new File("WriterSampleOutput.txt");
    try (BufferedWriter out = new BufferedWriter(
            new OutputStreamWriter(
                    new FileOutputStream(file)))) {
        for (int i = 0; i < 100; i++) {
            int x = (int) (100 * Math.random());
            out.write(String.valueOf(x));
            out.newLine();
        }
    }
}
```

`simplest/WriterSample.java`

Wrapping standard output

```java
public static void wrapping() {
    BufferedWriter out = new BufferedWriter(
            new OutputStreamWriter(System.out));
    try {
        out.write("Something");
        out.newLine();
    } catch (IOException ex) {
        System.err.println(ex);
    }
}
```

Examples

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

Note: line break codes

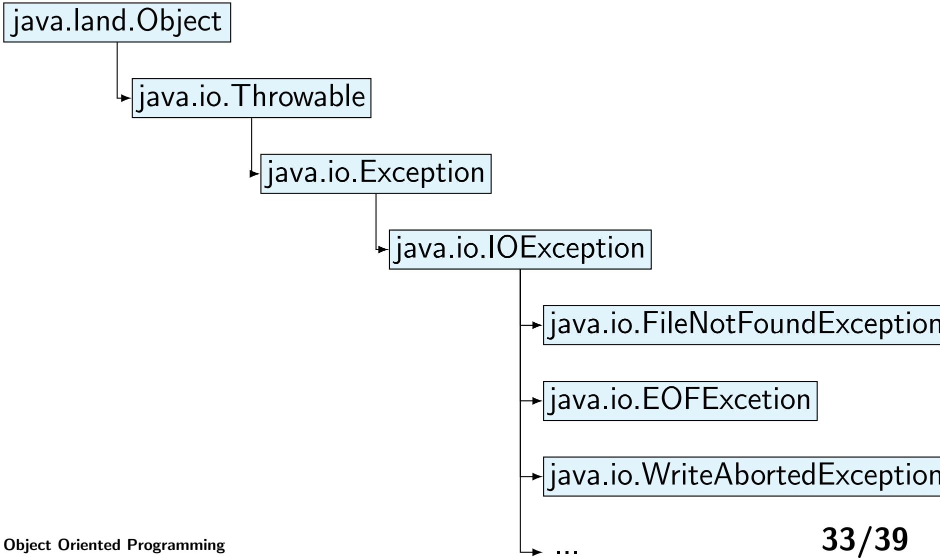
- Line break codes depend on OS.
  - UNIX, Linux, MacOS(> 9): LF (`0x0a`)
  - Windows: CR+LF (`0x0d0a`)
- Write OS independent code by Java

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

Exceptions

- Exceptions are inevitable in IO
- Applications should handle exceptions for preventing applications from being aborted
- Uniform method for handling exceptions
- Java defines exceptions as class

Hierarchy of exception classes



Handling exceptions

- Inside method

```
try{
    Something will throw exceptions
} catch (Exception ex){
    Error Handling
}
```

- Notify exception to caller

```
public void method() throws Exception{
    Something will throw exceptions
}
```

Generating exceptions

```
public void method() throws Exception{
    if(something){
        String message="error message";
        throw new Exception(message);
    }
}
```

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/ExceptionSample.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`.