Day 3

Working with Path

- `Path` and `Paths` are newer classes for working with files and file system
  - `Path` and `Paths` are in `java.nio.file` package
- Get a path

```
Path readme = Paths.get("src/readme.md")
```

- Create a new file

```
Path p = Path.createFile(readme)
```

- Convert a Path to File

```
File f = p.toFile()
```

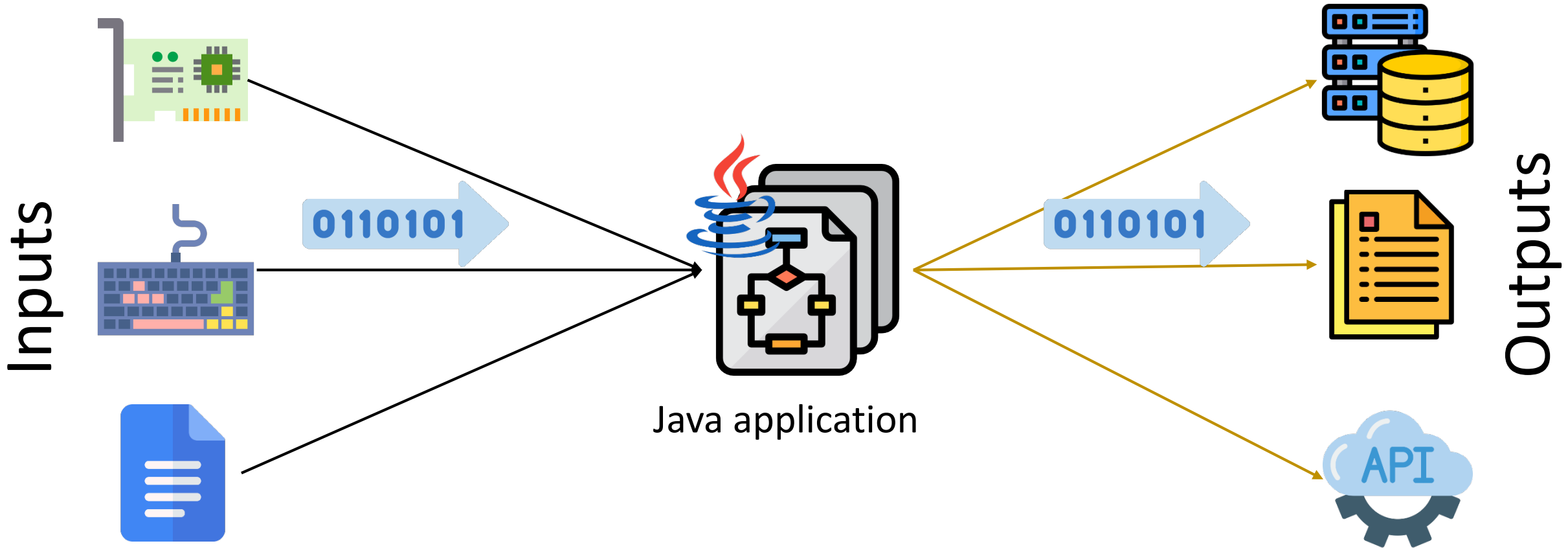
Working with Files

- Java `File` class is a representation of a file
  - See `Files` for advance file operations
- Get information about the file
  - Permissions - `canRead()`, canWrite `()`, `canExecute()`
  - Information - `isFile()`, `isDirectory()`, `length()`, `exists()`
  - File operations - `delete()`, `mkdir()`, `renameTo()`

```
File file = Paths.get("/home/fred/readme.txt");
if (!file.exists())
   System.out.printf("The file does not exists");
else if file.isFile()
   System.out.printf("The size of the file is %d\n", f.length());
else // it is a directory
   for (File f: file.listFiles())
      System.out.printf("filename: %s\n", f.getName());
```

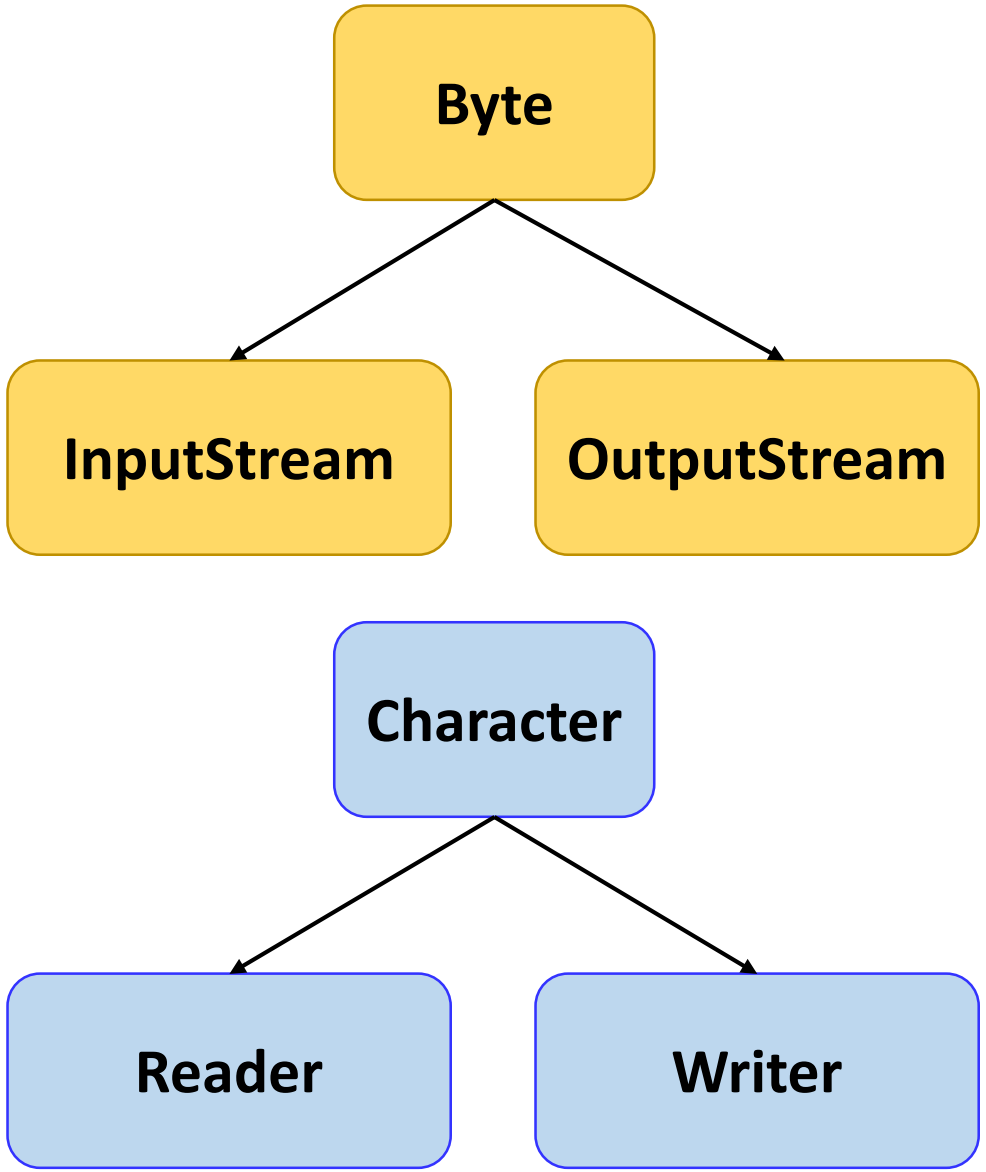
Note: ignoring exceptions

Input and Output



Java IO

- `java.io` package provides libraries for performing IO
  - `java.nio` for more advance IO usage
- Inputs and outputs are performed in bytes or character (2 bytes)
- `InputStream` and `OutputStream`
  - Bytes
- `Reader` and `Writer`
  - Character (2 bytes)



Byte Stream

- InputStream
- FileInputStream
- ByteArrayInputStream
- DataInputStream
- ObjectInputStream
- GZIPInputStream
  - java.util.zip
- CipherInputStream
  - javax.crypto

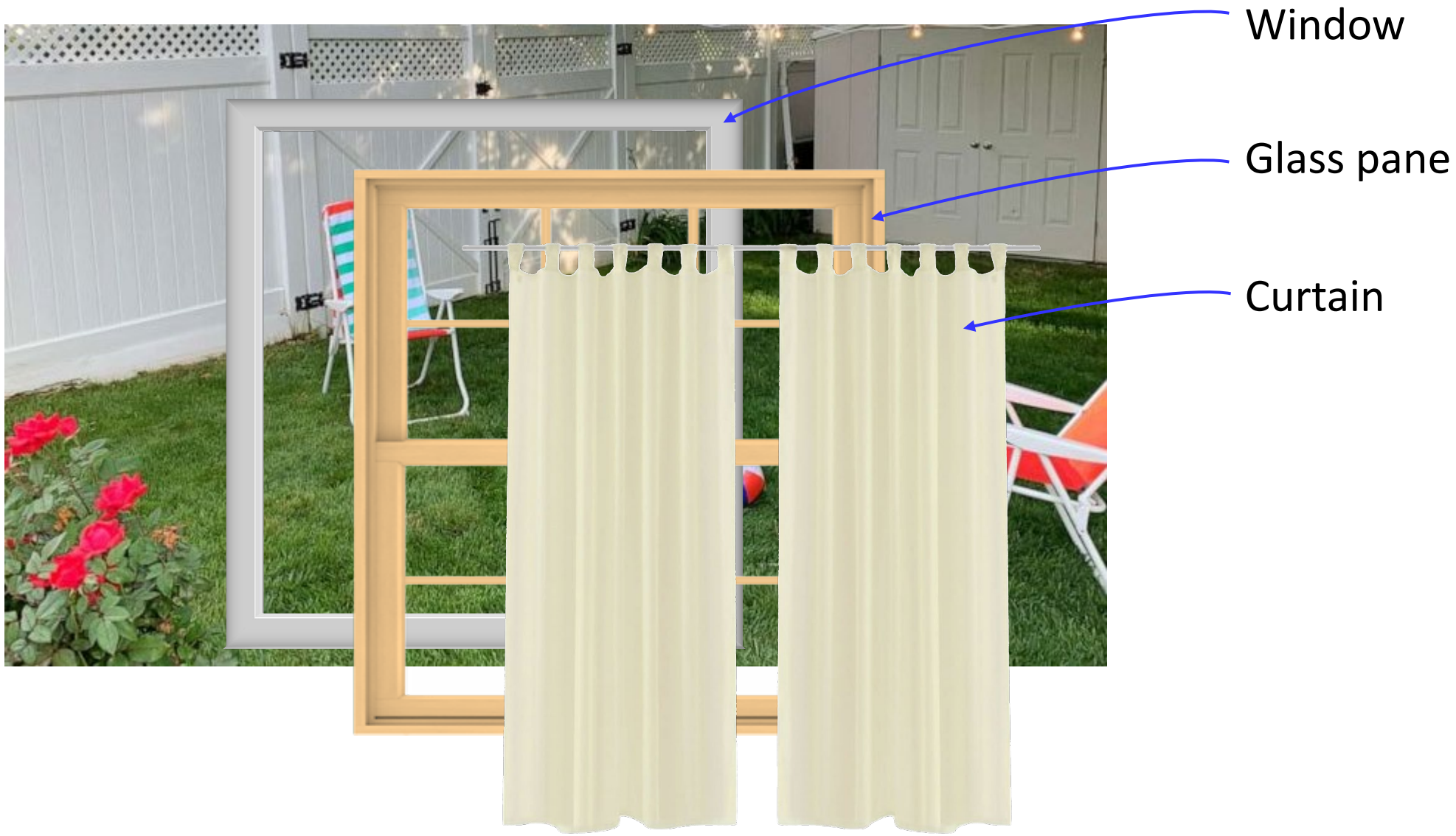
- OutputStream
- FileOutputStream
- ByteArrayOutputStream
- DataOutputStream
- ObjectOutputStream
- GZIPOutputStream
  - java.util.zip
- CipherOutputStream
  - javax.crypto

Example

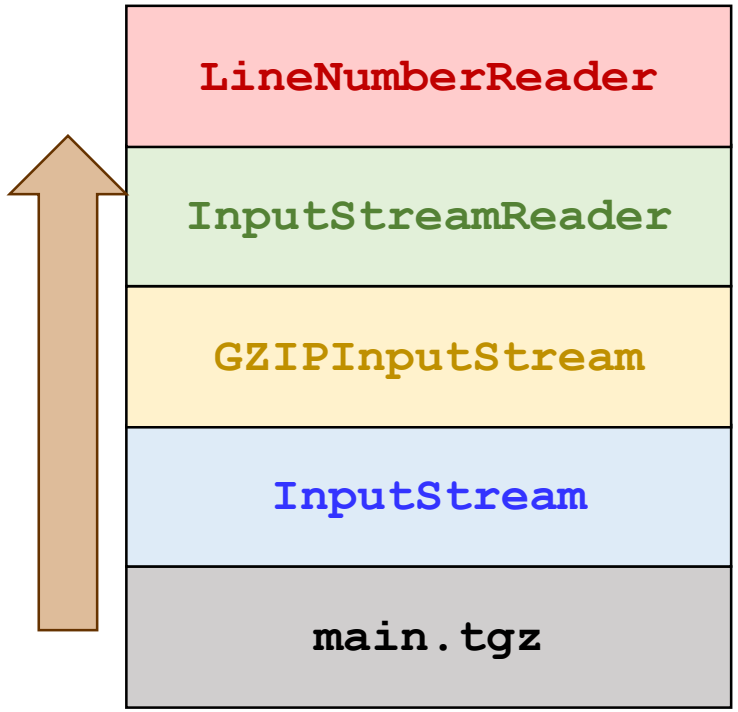
```
byte[] buffer = new byte[1024];
int size = 0;
InputStream is = new FileInputStream("myfile.txt");
OutputStream os = new FileOutputStream("copy of
myfile.txt");

while (-1 != (size = is.read(buffer)))
   os.write(buffer, 0, size);

os.flush();
os.close();
is.close();
```

Note: ignoring exceptions

Decorator Pattern



Example - Java IO Decorator




```
Path p = Paths.get("src/main.tgz");
InputStream is = new FileInputStream(p.toFile());
GZIPInputStream gis = new GZIPInputStream(is);
InputStreamReader isr = new InputStreamReader(gis);
LineNumberReader lnr = new LineNumberReader(isr);
String line;

while (null != (line = lnr.readLine()))
   System.out.printf"%d: %s\n", line, lnr.getLineNumber());

// Closes all the stream
is.close();
```

Note: ignoring exceptions

Example - Java IO Decorator

`try`-with-resource will automatically closes InputStream when program exits the `try` block

```
Path p = Paths.get("src/main.tgz");

try (InputStream is = new FileInputStream(p.toFile()) {

   GZIPInputStream gis = new GZIPInputStream(is);
   InputStreamReader isr = new InputStreamReader(gis);
   LineNumberReader lnr = new LineNumberReader(isr);
   String line;

   while (null != (line = lnr.readLine()))
      System.out.printf"%d: %s\n", line, lnr.getLineNumber());

   // Closes all the stream before leaving block
}
```

Note: ignoring exceptions

Exceptions

- An exception is an event that arises during the execution of a program
  - Eg. reading a non existent file
  - Eg. index of an array is greater than the array length
  - Eg. running out of memory
- Exceptions are thrown
  - By constructor and methods
  - Performing illegal/erroneous operations during runtime
  - When JVM detacts a fault

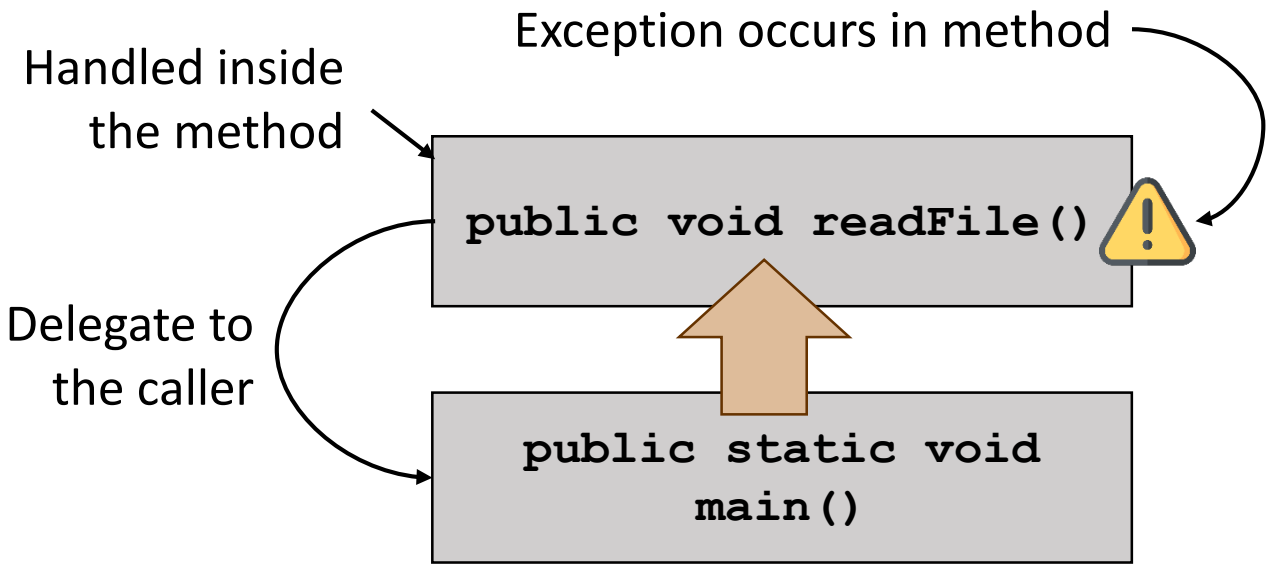
```
Path p = Paths.get("src/data.csv");
try {
    InputStream is = new FileInputStream(p.toFile());
    ...
} catch (FileNotFoundException e) {
    //Handle exception
}
```

Control transfer to exception handler by passing the rest of the block

Exception thrown by `FileInputStream` constructor

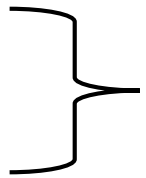
Handling Exceptions

- Exceptions can be handled
  - Where it occurs
  - Delegate to the caller
- Where it occurs
  - Use the try catch block
- Delegate to the caller
  - Declare the method will throw one or more exceptions



Example - Handling Exception

```
public static String readFile(String fn) {
    StringBuilder sb = new StringBuilder();
    String line;
    try (Reader reader = new FileReader(fn)) {
        BufferedReader br = new BufferedReader(reader);
        while (null != (line = reader.readLine()))
            sb.append(line).append("\n");
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    return sb.toString();
}
```



Exception is handle by the method within the `catch` block

Example - Delegating

```
public static String readFile(String fn) throws FileNotFoundException {
    StringBuilder sb = new StringBuilder();
    String line;
    try (Reader reader = new FileReader(fn)) {
        BufferedReader br = new BufferedReader(reader);
        while (null != (line = reader.readLine()))
            sb.append(line).append("\n");
    }
    return sb.toString();
}

public static void main(String... args) {
    try {
        String f = readFile(args[0]);
        ...
    } catch(FileNotFoundException e) {
        System.out.printf("An exception has occurred\n:s\n"
                , e.getMessage());
    }
}
```

Declares that this method can throw `FileNotFoundException`

Use try/catch block to handle exception

`finally` Clause

- Execute a block of code before leaving the try block
- Leave try block due to one of the following conditions
  - Leaving the `try` block
  - Exception
  - From `return` statement
- Flow
  - Exception: 1 -> 3 -> 4
  - Normal: 1 -> 4 -> 2

- `finally` can be used without the `catch` block

```
try {
    InputStream is = new FileInputStream(fn);
    ...
    return
} catch (FileNotFoundException e) {
    ...
} finally { }
```

1 2 3 4