CSE 143
Lecture 25

I/O Streams; Exceptions; Inheritance

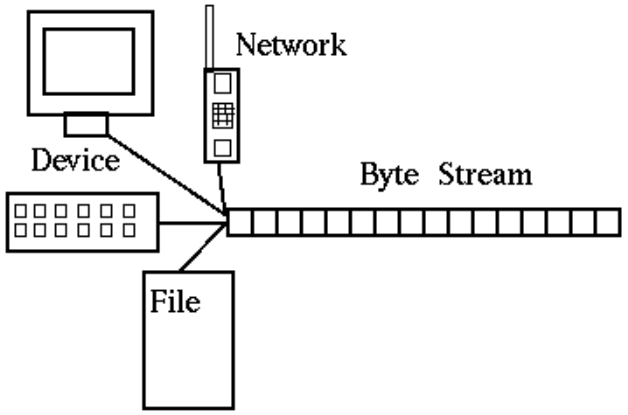
read 9.3, 6.4

slides adapted from Marty Stepp
http://www.cs.washington.edu/143/

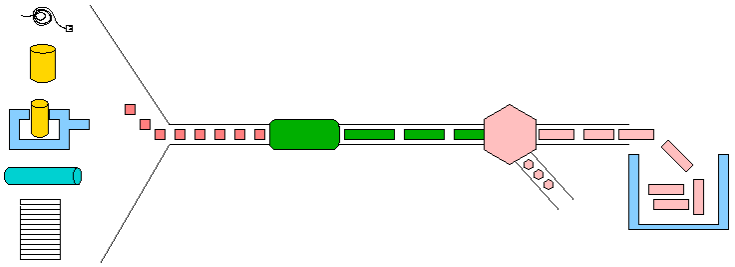
Input and output streams

- **stream**: an abstraction of a source or target of data
  - 8-bit bytes flow to (output) and from (input) streams

- can represent many data sources:
  - files on hard disk
  - another computer on network
  - web page
  - input device (keyboard, mouse, etc.)

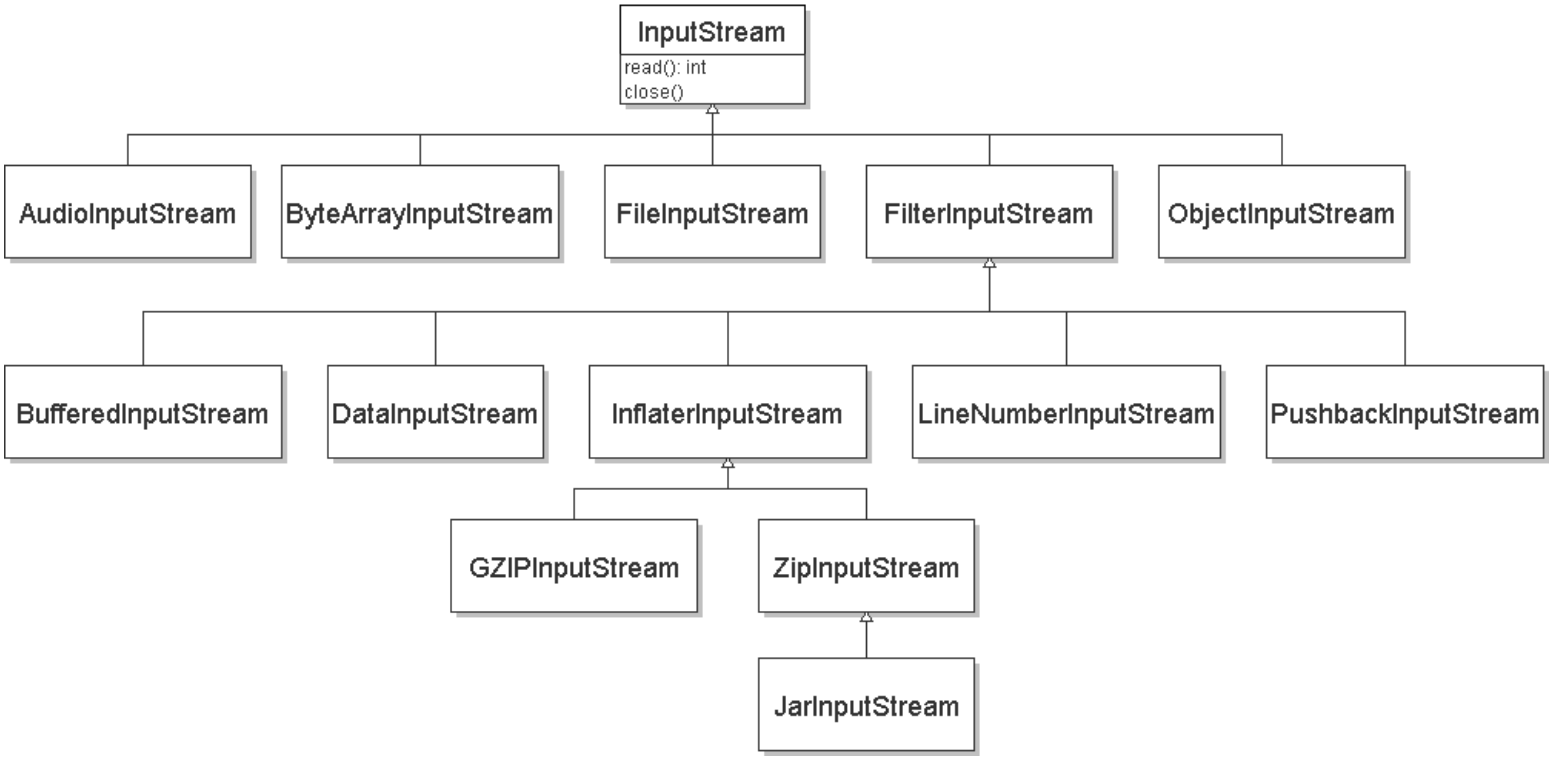


- represented by `java.io` classes
  - `InputStream`
  - `OutputStream`



Streams and inheritance

- all input streams extend common superclass `InputStream`;
  all output streams extend common superclass `OutputStream`
  - guarantees that all sources of data have the same methods
  - provides minimal ability to read/write one byte at a time



Input streams

- constructing an input stream:

Constructor
`public FileInputStream(String name) throws IOException`
`public ByteArrayInputStream(byte[] bytes)`
`public SequenceInputStream(InputStream a, InputStream b)`

(various objects also have methods to get streams to read them)

- methods common to all input streams:

Method	Description
`public int` **`read`**`() throws IOException`	reads/returns a byte (-1 if no bytes remain)
`public void` **`close`**`() throws IOException`	stops reading

Output streams

- constructing an output stream:

Constructor
`public FileOutputStream(String name) throws IOException`
`public ByteArrayOutputStream()`
`public PrintStream(File file)` `public PrintStream(String fileName)`

- methods common to all output streams:

Method	Description
`public void write(int b) throws IOException`	writes a byte
`public void close() throws IOException`	stops writing (also flushes)
`public void flush() throws IOException`	forces any writes in buffers to be written

I/O and exceptions

- **exception**: An object representing an error.
  - **checked exception**: One that must be handled for the program to compile.

- Many I/O tasks throw exceptions.
  - Why?

- When you perform I/O, you must either:
  - also **throw** that exception yourself
  - **catch** (handle) the exception

Throwing an exception

```
public type name(params) throws type {
```

- **`throws` clause**: Keywords on a method's header that state that it may generate an exception.
  - Example:

```
public void processFile(String filename)
                throws FileNotFoundException {
```

"I hereby announce that this method might throw an exception, and I accept the consequences if it happens."

Catching an exception

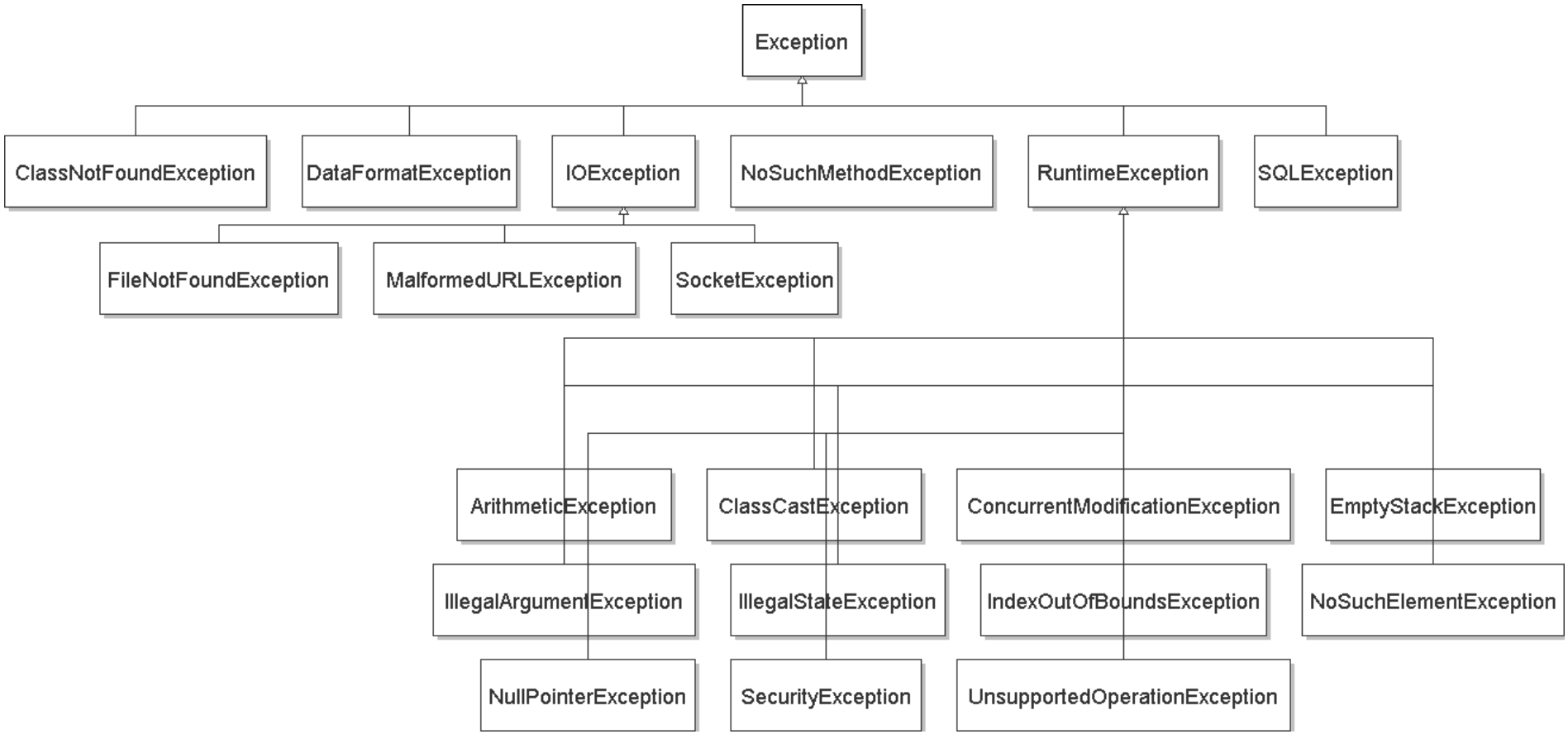
```
try {
    statement(s);
} catch (type name) {
    code to handle the exception
}
```

– The `try` code executes. If the given exception occurs, the `try` block stops running; it jumps to the `catch` block and runs that.

```
try {
    Scanner in = new Scanner(new File(filename));
    System.out.println(input.nextLine());
} catch (FileNotFoundException e) {
    System.out.println("File was not found.");
}
```

Exception inheritance

- All exceptions extend from a common superclass `Exception`



Dealing with an exception

- All exception objects have these methods:

Method	Description
`public String` **`getMessage`**`()`	text describing the error
`public String` **`toString`**`()`	a stack trace of the line numbers where error occurred
`getCause``(),` **`getStackTrace`**`(),` **`printStackTrace`**`()`	other methods

- Some reasonable ways to handle an exception:
  - try again; re-prompt user; print a nice error message; quit the program; do nothing (!)

Inheritance and exceptions

- You can catch a general exception to handle any subclass:

```
try {
    Scanner input = new Scanner(new File("foo"));
    System.out.println(input.nextLine());
} catch (Exception e) {
    System.out.println("File was not found.");
}
```

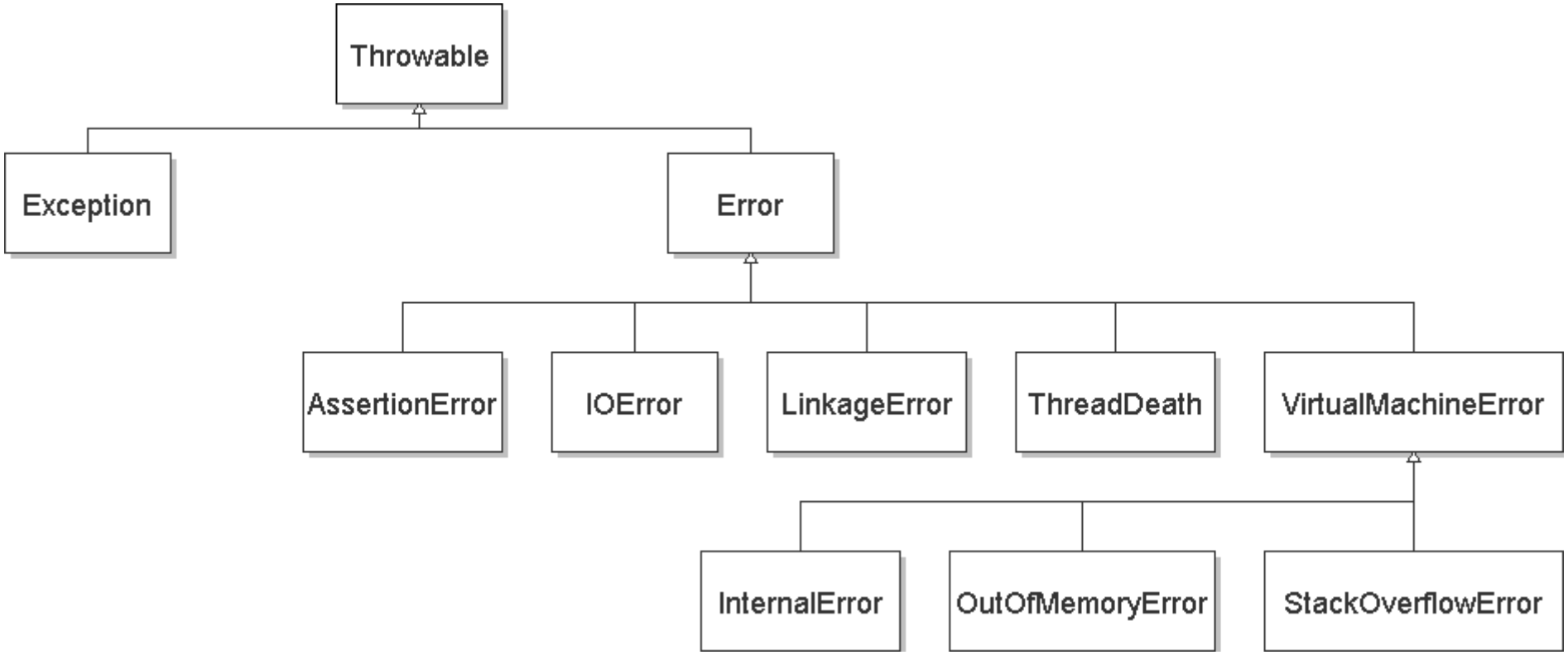
- Similarly, you can state that a method throws any exception:

```
public void foo() throws Exception { ...
```

  - Are there any disadvantages of doing so?

Exceptions and errors

- There are also `Error`s, which represent serious Java problems.
  - `Error` and `Exception` have common superclass `Throwable`.
  - You can catch an `Error` (but you probably shouldn't)



Exercise

- Write a class `Downloader` with the following behavior:
  - `public Downloader(String url)`
    - Initializes the downloader to examine the given URL.
  - `public void download(String targetFileName)`
    - Downloads the file from the URL to the given file name on disk.

- Write client program `DownloadMain` to use `Downloader`:

```
URL to download? foo bar
Bad URL! Try again: http://zombo.com/
Target file name: out.html
```

Contents of `out.html`:

```
<html>
<head>
<title>ZOMBO</title>
...
</body>
</html>
```

Reading from the web

- class `java.net.URL` represents a web page's URL
- we can connect to a URL and read data from that web page

Method/Constructor	Description
`public URL(String address) throws MalformedURLException`	creates a URL object representing the given address
`public String getFile(), getHost(), getPath(), getProtocol()` `public int getPort()`	returns various parts of the URL as strings/integers
`public InputStream openStream() throws IOException`	opens a stream for reading data from the document at this URL

```
http://www.foo.com:8080/dir1/dir2/readme.txt
protocol   host     port   path       file
```

Exercise solution

```
import java.io.*;
import java.net.*;

public class Downloader {
    private URL url;

    // Constructs downloader to read from the given URL.
    public Downloader(String urlString) throws MalformedURLException {
        url = new URL(urlString);
    }

    // Reads downloader's URL and writes contents to the given file.
    public void download(String targetFileName) throws IOException {
        InputStream in = url.openStream();
        FileOutputStream out = new FileOutputStream(targetFileName);
        while (true) {
            int n = in.read();
            if (n == -1) {      // -1 means end-of-file
                break;
            }
            out.write(n);
        }
        in.close();
        out.close();
    }
}
```

Exercise solution 2

```
import java.io.*;
import java.net.*;
import java.util.*;

public class DownloadMain {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("URL to download? ");
        String urlString = console.nextLine();

        Downloader down = null;    // create a downloader;
        while (down == null) {     // re-prompt the user if this fails
            try {
                down = new Downloader(urlString);
            } catch (MalformedURLException e) {
                System.out.print("Bad URL! Try again: ");
                urlString = console.nextLine();
            }
        }

        System.out.print("Target file name: ");
        String targetFileName = console.nextLine();

        try {    // download bytes to file (print error if it fails)
            down.download(targetFileName);
        } catch (IOException e) {
            System.out.println("I/O error: " + e.getMessage());
        }
    }
}
```

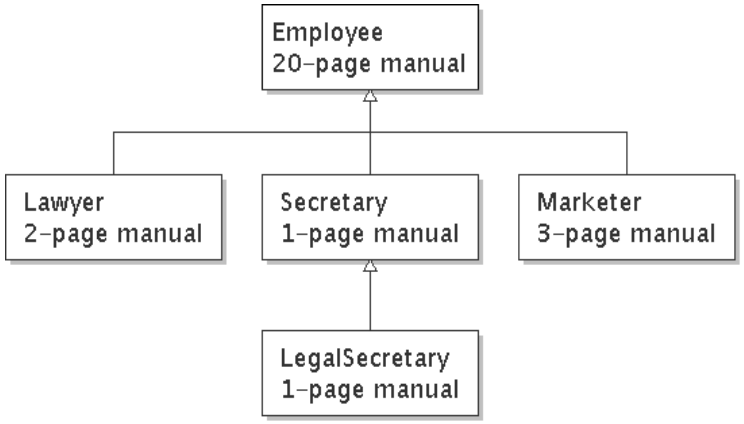
Exercise 2

- Write class `TallyDownloader` to add behavior to `Downloader`:
  - `public TallyDownloader(String url)`
  - `public void download(String targetFileName)`
    - Downloads the file, and also prints the file to the console, and prints the number of occurrences of each kind of character in the file.

```
URL to download? http://zombo.com/
<html>
<head>
<title>ZOMBO</title>
<!--Please Visit 15footstick.com our other website. ThankZ -->
...
</body>
</html>
{
=21,  =42, !=1, "=18, #=4, %=4, ,=3, -=14, .=10, /=18, 0=15, 1=9,
  2=2, 3=1, 4=5, 5=5, 6=4, 7=1, 8=3, 9=2, :=3, ;=1, <=17, ==24,
  >=17, ?=1, A=1, B=3, C=2, D=3, E=2, F=19, M=1, O=2, P=3, S=1, T=2,
  V=2, Z=2, _=2, a=42, b=13, c=27, d=18, e=47, f=7, g=10, h=28,
  i=32, j=2, k=5, l=24, m=21, n=17, o=36, p=12, q=3, r=17, s=24,
  t=37, u=8, v=10, w=15, x=5, y=6, z=2}
```

Inheritance

- **inheritance**: Forming new classes based on existing ones.
  - a way to share/**reuse code** between two or more classes
  - **superclass**: Parent class being extended.
  - **subclass**: Child class that inherits behavior from superclass.
    - gets a copy of every field and method from superclass
  - **is-a relationship**: Each object of the subclass also "is a(n)" object of the superclass and can be treated as one.



Inheritance syntax

```
public class name extends superclass {

public class Lawyer extends Employee {
    ...
}
```

- **override**: To replace a superclass's method by writing a new version of that method in a subclass.

```
public class Lawyer extends Employee {
    // overrides getSalary method in Employee class;
    // give Lawyers a $5K raise
    public double getSalary() {
        return 55000.00;
    }
}
```

`super` keyword

- Subclasses can call inherited methods/constructors with `super`

```
super.method(parameters)
super(parameters);
```

```
public class Lawyer extends Employee {
    public Lawyer(int years) {
        super(years);  // calls Employee constructor
    }

    // give Lawyers a $5K raise
    public double getSalary() {
        double baseSalary = super.getSalary();
        return baseSalary + 5000.00;
    }
}
```

  - Lawyers now always make $5K more than Employees.

Exercise solution

```
public class TallyDownloader extends Downloader {
    public TallyDownloader(String url) throws MalformedURLException {
        super(url);   // call Downloader constructor
    }

    // Reads from URL and prints file contents and tally of each char.
    public void download(String targetFileName) throws IOException {
        super.download(targetFileName);

        Map<Character, Integer> counts = new TreeMap<Character, Integer>();
        FileInputStream in = new FileInputStream(targetFileName);
        while (true) {
            int n = in.read();
            if (n == -1) {
                break;
            }
            char ch = (char) n;
            if (counts.containsKey(ch)) {
                counts.put(ch, counts.get(ch) + 1);
            } else {
                counts.put(ch, 1);
            }
            System.out.print(ch);
        }
        in.close();
        System.out.println(counts);   // print map of char -> int
    }
}
```

Exercise solution 2

```
import java.io.*;
import java.net.*;
import java.util.*;

public class DownloadMain {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("URL to download? ");
        String urlString = console.nextLine();

        Downloader down = null;    // create a tallying downloader;
        while (down == null) {     // re-prompt the user if this fails
            try {
                down = new TallyDownloader(urlString);
            } catch (MalformedURLException e) {
                System.out.print("Bad URL! Try again: ");
                urlString = console.nextLine();
            }
        }

        ...
    }
}
```