File I/O

File I/O

- For file I/O, analogous to its standard I/O, are the classes `FileInputStream` and `FileOutputStream`.
- Whether reading or writing, a file should be *opened* first and then *closed* at the end.
- When a file is opened, it is either for *reading* or *writing*, but not both.

FileInputStream

- The class `FileInputStream` provides simple machinery to read files, analogous to the standard input stream `System.in`.
- With this class, we regard a file as a `FileInputStream` object. To read its content, we often create a `Scanner` object from it.

Scanner

The class `Scanner` can also facilitate extracting meaningful information from `FileInputStream`. It extracts information separated by *blank spaces*.

- `next()` — extracts the next word as a single `String` object.
- `nextInt()` — extracts the next integer as an `int` value.
- `nextDouble()` — extracts the next real number as a `double` value.

Scanner

Example.

```
FileInputStream f =
  new FileInputStream(”input.txt”);
Scanner s = new Scanner(f);
ArrayList<Integer> numbers =
  new ArrayList<Integer>();

int i = 0;
while (s.hasNextInt()) {
  numbers.add(i, s.nextInt());
  i++;
}
f.close();
```

Scanner

Example.

```
FileInputStream f =
  new FileInputStream(”input.txt”);
Scanner s = new Scanner(f);
ArrayList<String> words =
  new ArrayList<String>();

int i = 0;
while (s.hasNext()) {
  words.add(i, s.next());
  i++;
}
f.close();
```

FileOutputStream

- The class `FileOutputStream` provides simple machinery to output to files, analogous to the standard output stream `System.out`.
- With this class, we regard a file as a `FileOutputStream` object. To print, we often create a `PrintWriter` object from it.

PrintWriter

Example.

We assume from the last example that we have an `ArrayList<String>` object `words`.

```
FileOutputStream f =
  new FileOutputStream(”output.txt”);
PrintWriter p = new PrintWriter(f);

for (int i = 0; i < words.size(); i++)
  p.println(words.get(i));
f.close();
```

File

- The classes `FileInputStream` and `FileOutputStream` form streams for reading and writing, respectively, raw data.
- On the other hand, the class `File` represents file/directory paths and metadata (i.e., data about data).

`File`

While it cannot read or write data by itself, the class `File` can determine

- whether a file/directory exists,
- whether a name refers to a file or directory.
- the size of a file/directory, etc.

It can also create and delete files and directories.

Validation with `File`

```
public boolean isReadable(String name) {
  try {
    File f = new File(name);
    if (!f.exists())
      throw (new FileNotFoundException(name + " does not exist."));
    if (!f.canRead())
      throw (new IOException(name + " is not readable."));
    return true;
  } catch (FileNotFoundException e) {
    System.out.println("I/O error: " + e.getMessage());
    return false;
  } catch (IOException e) {
    System.out.println("I/O error: " + e.getMessage());
    return false;
  }
}
```

`File`

While it cannot read or write data by itself, the class `File` can determine

- whether a file/directory exists,
- whether a name refers to a file or directory.
- the size of a file/directory, etc.

It can also be used, with a `Scanner` object, to read and write data.

Scanner

Example.

```
File f = new File("input.txt");
Scanner s = new Scanner(f);
ArrayList<String> words =
  new ArrayList<String>();

int i = 0;
while (s.hasNext()) {
  words.add(i, s.next());
  i++;
}
f.close();
```

PrintWriter

Example.

We assume from the last example that we have an `ArrayList<String>` object `words`.

```
File f = new File("output.txt");
PrintWriter p = new PrintWriter(f);

for (int i = 0; i < words.size(); i++)
  p.println(words.get(i));
f.close();
```

Laboratory 8

Most web servers maintain log files of client accesses.

Webmasters often extract and analyze access patterns from these logs to determine, for example, the busiest periods over the course of a day, which pages are more popular than others, etc.

Laboratory 8

For Laboratory 8, you will design and implement classes to read and analyze such log files. You will build on the classes

- **`LogAnalyzer`**,
- `LogEntry`,
- `LogfileReader`, and
- `LoglineTokenizer`.