DIT948 Programming H16
Lecture 11

Instructor: Musard Balliu, `musard@chalmers.se`

October 15, 2016

QUESTIONS?

Plan

- Last time
  1. Robot APIs
  2. Graphical User Interfaces
- Today's Plan:
  1. Input/Output Streams
  2. Network Programming

Streams

In Lecture 10 we have done quite a bit of input-output, in a graphical way. The users were presented with a small number of messages they could answer in a small number of ways (by clicking one of a couple of buttons).

In many applications, this model is too limiting. For example, sending e-mail is not done by choosing the words from drop-down menus.

Additionally, not all input-output involves a human partner. In fact, the fraction of all input-output with a human on at least one end is, in terms of quantity, insignificant.

In these situations, we use *streams*

The example

To illustrate input-output, we shall use the following example: an instance of `RandomRobot` which moves randomly and communicates its moves to another *program* with a robot which will execute those moves (a kind of *imitation game*).

Output

To that end, we'll create an `IORobot`: a version of `RandomRobot` which does input and output via streams.

We start with the simplest version: a robot which prints its actions via *standard output* (usually the console).

Code

`System.out.println`

We have printed to the console using `System.out.println`.

Documentation of class `System`

`out` is a static member of class `System`. Its type is `PrintStream`.

Documentation of class `PrintStream`

An important exercise!

We have overloaded the `randomMove` method of `RandomRobot`. Why not overload instead the `turn` methods and the `move` method instead?

In other words, why doesn't the following work?

Alternative `SysOutRobot`

Input

The counterpart to `System.out` for reading from the standard input (usually the keyboard) `System.in`.

`in is of type InputStream.`
`Documentation of class InputStream`

`Scanner`

Fortunately, there exists a class which saves us from manually converting bytes to `int`, `double`, etc.
Documentation for class `Scanner`

Now we can have a `SysInRobot` which can read messages and execute them, and we can run two programs at the same time, communicating via `System.out` and `System.in`.

Code

Using `SysInRobot`

If we start up `SysInRobot` with

```
java SysInRobot
```

at the command line, then we can enter commands from the keyboard and the robot will act accordingly.

However, we can *pipe* the output of `SysOutRobot` to the output of `SysInRobot` and have the outputting robot "in the driver's seat":

```
java SysOutRobot | java SysInRobot
```

Communicating via a file

Another possibility is to direct the output of `SysOutRobot` to a file, and to have `SysInRobot` read that file:

```
java SysOutRobot > x
java SysInRobot  < x
```

This reveals a problem with our treatement of input.

(Mis-)Using `Scanner`

The robots are out of sync. That is because we are using `Scanner` in the wrong way. `Scanner` has several limitations:

- it doesn't do buffering
- it isn't thread-safe
- it gets easily confused, etc.

Using `Scanner`

The rules for using `Scanner` are:

1. **Do not use** `Scanner` to retrieve data!
2. **Only use** `Scanner` to parse (make sense of) data which has already been retrieved, and which is not in danger of being changed by other processes.

In other words, **always separate data retrieval from parsing**!

Using `BufferedReader`

The solution is to use `BufferedReader` to read the data, line by line, from the input.

Documentation for `BufferedReader`

Once a line has been retrieved, we can use a `Scanner` to interpret it (which in our case is very easy, since there is only a digit in the whole line).

The code

Using `Files`

We have used the operating system to write to files and read from files by diverting the standard output and input of programs.

This is a very limited way of working with files. For instance, we couldn't work with more than one file at a time (every process has exactly one of standard output and input).

Java provides a type for working with files directly.

Documentation for `File`

The most important method here is the constructor

```
File(String pathname)
```

Using `Files`

Once we have a file, we can use it to obtain a `PrinterStream` for writing to it using `print` or `println`.

We can also use it to obtain a `FileInputStream`, which “is an” `InputStream`.
Documentation for `FileInputStream`

Using `Files`

We abstract away from the specific `PrintStream` that `OutRobot` is using; and similarly, from the specific `InputStream` from which `BufInRobot` creates its instance of `BufferedReader`.

This will make it easier to reuse the robots, as we shall see in the sequel.

`OutRobot`
`BufInRobot`

Network programming

Another source of streams is the network. While an advanced treatement of network programming requires a course for itself, the basics are surprisingly simple.

Communication over the network is done via *sockets*.
Documentation for `Socket`

Using `Sockets`

There are two main ways for obtaining sockets:

1. from a `ServerSocket`
2. by connecting to an existing socket

You can see that for any socket-based communication, there must be at least one instance of `ServerSocket`.
Documentation for `ServerSocket`

Input via sockets

Once we construct a `ServerSocket`, we can obtain a socket by *accepting* connections.

The connection is bi-directional, so the sockets can be used for both input and output. We shall use the server-side socket to control the `BufInRobot`.

Note that we do not need to create another robot, only to provide the `BufInRobot` with an input stream obtained from the socket.

Code for socket input.

Output via sockets

Similarly, once we have connected to a socket, we can obtain from it an `OutputStream`, from which we construct a `PrintStream` object for our `OutRobot`.

Again, the robot class remains unchanged: we can use it with any kind of `PrintStream`.

Code for socket output.

Homework

- As always, make sure you understand every line of code in this lecture.