OOP with Java
6. Console I/O

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

I/O

- Programs usually determine their actions and produce output based on their input data

I/O

- Programs usually determine their actions and produce output based on their input data
- We already know that we can create output using

```
System.out.println(...)
```

I/O

- Programs usually determine their actions and produce output based on their input data
- We already know that we can create output using

```
System.out.println(...)
```

- We now want to look a bit deeper into the concept of input and output

I/O

- Programs usually determine their actions and produce output based on their input data
- We already know that we can create output using

```
System.out.println(...)
```

- We now want to look a bit deeper into the concept of input and output
- For now, we only consider the simplest cases of console interaction

- `System.out` is a `PrintStream` which allows us to write data to the console

- `System.out` is a `PrintStream` which allows us to write data to the console
- It has the methods `System.out.print` and `System.out.println` for all basic Java types we had so far

- `System.out` is a `PrintStream` which allows us to write data to the console
- It has the methods `System.out.print` and `System.out.println` for all basic Java types we had so far:
  - `System.out.print(a)` prints the value of expression `a`

- `System.out` is a `PrintStream` which allows us to write data to the console
- It has the methods `System.out.print` and `System.out.println` for all basic Java types we had so far:
  - `System.out.print(a)` prints the value of expression `a`
  - `System.out.println(a)` prints the value of expression `a` and then starts a new line

- `System.out` is a `PrintStream` which allows us to write data to the console
- It has the methods `System.out.print` and `System.out.println` for all basic Java types we had so far:
  - `System.out.print(a)` prints the value of expression `a`
  - `System.out.println(a)` prints the value of expression `a` and then starts a new line
  - `System.out.println()` without argument just starts a new line

Examples for using `System.out`

Listing: A program demonstrating how to use `System.out`.

```
/** Examples for using System.out */
public class SystemOut {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    System.out.println("This text is printed and afterwards , a new line is started."); //$NON-NLS-1$
    System.out.println(34); // just write a number and start a new line

    System.out.print("Here , we write the integer value "); // no new line afterwards! //$NON-NLS-1$
    System.out.print((Integer.MAX_VALUE / Short.MAX_VALUE) / Byte.MAX_VALUE);
    System.out.print(" and then \""); //$NON-NLS-1$
    double a=0.66d/3d;
    System.out.print(a>0.25d ? "bla" : "blubb"); //$NON-NLS-1$ //$NON-NLS-2$
    System.out.print("\" , the result of the expression '"); //$NON-NLS-1$
    System.out.print(a);
    System.out.println(">0.25d ? \"bla\" : \"blubb\"'."); //$NON-NLS-1$
    System.out.println("Here we just print some empty lines:");// new line afterwards //$NON-NLS-1$
    System.out.println(); // another new (empty) line
    System.out.println(); // another new (empty) line
    System.out.println("...and that's all.");//$NON-NLS-1$
  }
}
```

Remark: These `//NON-NLS-1` things can safely be ignored, they are just there to tell Eclipse that a `String` literal is not internationalized/stored in a resource but to be used as it. Ignore them.

Piping Output to a File

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm

Piping Output to a File

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console

Piping Output to a File

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console, then this output can be written to a file instead.

Piping Output to a File

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console, then this output can be written to a file instead.
- Example: `java SystemOut > SystemOut.txt` creates a text file called `SystemOut.txt` in which the output of our program `SystemOut` (previous slide) is stored. This file looks as follows:

Piping Output to a File

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console, then this output can be written to a file instead.
- Example: `java SystemOut > SystemOut.txt` creates a text file called `SystemOut.txt` in which the output of our program `SystemOut` (previous slide) is stored. This file looks as follows:

Listing: File `SystemOut.txt` created via `java SystemOut > SystemOut.txt`.

```
This text is printed and afterwards, a new line is started.
34
Here, we write the integer value 516 and then "blubb", the result of the expression '0.22>0.25d ? "bla" : "blubb"'.
Here we just print some empty lines:


...and that's all.
```

Piping Output to another Program 1

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console, then this output can be written to a file instead

Piping Output to another Program 1

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console, then this output can be written to a file instead or to the input of another program!

Piping Output to another Program 1

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console, then this output can be written to a file instead or to the input of another program!
- Example 1: `java SystemOut | head -n 3 > SystemOutToHead.txt` writes the output of our `SystemOut` program to the input of the program `head` and tells it to keep print the first three lines of its input to its output, which is then piped to file `SystemOutToHead.txt`

Piping Output to another Program 1

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console, then this output can be written to a file instead or to the input of another program!
- Example 1: `java SystemOut | head -n 3 > SystemOutToHead.txt` writes the output of our `SystemOut` program to the input of the program `head` and tells it to keep print the first three lines of its input to its output, which is then piped to file `SystemOutToHead.txt`

Listing: Contents of file `SystemOutToHead.txt`

```
This text is printed and afterwards, a new line is started.
34
Here, we write the integer value 516 and then "blubb", the result of the expression '0.22>0.25d ? "bla" : "blubb"'.
```

Piping Output to another Program 2

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console, then this output can be written to a file instead or to the input of another program!
- Example 2: `java SystemOut | sed 's/blubb/COOL!/g'` writes the output of our `SystemOut` program to the input of the *stream editor* `sed` and tells sed to replace all occurrences of `blubb` with `COOL`. The output of `sed` is thus:

Piping Output to another Program 2

- In Unix/Linux (and also Windows), there exists the "Pipes & Filters" Paradigm
- If an program writes output to the console, then this output can be written to a file instead or to the input of another program!
- Example 2: `java SystemOut | sed 's/blubb/COOL!/g'` writes the output of our `SystemOut` program to the input of the *stream editor* `sed` and tells sed to replace all occurrences of `blubb` with `COOL`. The output of `sed` is thus:

Listing: Output of `java SystemOut | sed 's/blubb/COOL!/g'`

```
This text is printed and afterwards, a new line is started.
34
Here, we write the integer value 516 and then "COOL!", the result of the expression '0.22>0.25d ? "bla" : "COOL!"'.
Here we just print some empty lines:


...and that's all.
```

- `System.in` is an `InputStream`* which allows us to write read single characters** from the console

* We will discuss in Lesson 28: *I/O and Streams* in detail what `InputStream`s are and how to create and use them.
** actually, `byte`s, but let's ignore this for now

`System.in` and `Scanner`

- `System.in` is an `InputStream`* which allows us to write read single characters** from the console
- Usually, we want to read structured stuff, like words or numbers

* We will discuss in Lesson 28: *I/O and Streams* in detail what `InputStream`s are and how to create and use them.
** actually, `byte`s, but let's ignore this for now

`System.in` and `Scanner`

- `System.in` is an `InputStream`* which allows us to write read single characters** from the console
- Usually, we want to read structured stuff, like words or numbers
- We can get this by wrapping `System.in` into a `Scanner` object `scanner` by doing `Scanner scanner = new Scanner(System.in);`

* We will discuss in Lesson 28: *I/O and Streams* in detail what `InputStream`s are and how to create and use them.
** actually, `byte`s, but let's ignore this for now

`System.in` and `Scanner`

- `System.in` is an `InputStream`* which allows us to write read single characters** from the console
- Usually, we want to read structured stuff, like words or numbers
- We can get this by wrapping `System.in` into a `Scanner` object `scanner` by doing `Scanner scanner = new Scanner(System.in);`
- Well, we did not yet learn what an object is and what `new` does, so let us ignore this aspect for now and focus just on reading data

* We will discuss in Lesson 28: *I/O and Streams* in detail what `InputStream`s are and how to create and use them.
** actually, `byte`s, but let's ignore this for now

`System.in` and `Scanner`

- `System.in` is an `InputStream`* which allows us to write read single characters** from the console
- Usually, we want to read structured stuff, like words or numbers
- We can get this by wrapping `System.in` into a `Scanner` object `scanner` by doing `Scanner scanner = new Scanner(System.in);`
- Well, we did not yet learn what an object is and what `new` does, so let us ignore this aspect for now and focus just on reading data:
  - `scanner.nextLine()` reads a full line of text as a `String`

* We will discuss in Lesson 28: *I/O and Streams* in detail what `InputStream`s are and how to create and use them.

** actually, `byte`s, but let's ignore this for now

`System.in` and `Scanner`

- `System.in` is an `InputStream`* which allows us to write read single characters** from the console
- Usually, we want to read structured stuff, like words or numbers
- We can get this by wrapping `System.in` into a `Scanner` object `scanner` by doing `Scanner scanner = new Scanner(System.in);`
- Well, we did not yet learn what an object is and what `new` does, so let us ignore this aspect for now and focus just on reading data:
  - `scanner.nextLine()` reads a full line of text as a `String`
  - `scanner.nextInt()` reads an `int` number (from text)

* We will discuss in Lesson 28: *I/O and Streams* in detail what `InputStream`s are and how to create and use them.
** actually, `byte`s, but let's ignore this for now

- `System.in` is an `InputStream`* which allows us to write read single characters** from the console
- Usually, we want to read structured stuff, like words or numbers
- We can get this by wrapping `System.in` into a `Scanner` object `scanner` by doing `Scanner scanner = new Scanner(System.in);`
- Well, we did not yet learn what an object is and what `new` does, so let us ignore this aspect for now and focus just on reading data:
  - `scanner.nextLine()` reads a full line of text as a `String`
  - `scanner.nextInt()` reads an `int` number (from text)
  - `scanner.nextDouble()` reads a `double` number (from text)

* We will discuss in Lesson 28: *I/O and Streams* in detail what `InputStream`s are and how to create and use them.
** actually, `byte`s, but let's ignore this for now

`System.in` and `Scanner`

- `System.in` is an `InputStream`* which allows us to write read single characters** from the console
- Usually, we want to read structured stuff, like words or numbers
- We can get this by wrapping `System.in` into a `Scanner` object `scanner` by doing `Scanner scanner = new Scanner(System.in);`
- Well, we did not yet learn what an object is and what `new` does, so let us ignore this aspect for now and focus just on reading data:
  - `scanner.nextLine()` reads a full line of text as a `String`
  - `scanner.nextInt()` reads an `int` number (from text)
  - `scanner.nextDouble()` reads a `double` number (from text)
  - . . . and so on. . .

* We will discuss in Lesson 28: *I/O and Streams* in detail what `InputStream`s are and how to create and use them.

** actually, `byte`s, but let's ignore this for now

`System.in` and `Scanner`

- `System.in` is an `InputStream`* which allows us to write read single characters** from the console
- Usually, we want to read structured stuff, like words or numbers
- We can get this by wrapping `System.in` into a `Scanner` object `scanner` by doing `Scanner scanner = new Scanner(System.in);`
- Well, we did not yet learn what an object is and what `new` does, so let us ignore this aspect for now and focus just on reading data:
  - `scanner.nextLine()` reads a full line of text as a `String`
  - `scanner.nextInt()` reads an `int` number (from text)
  - `scanner.nextDouble()` reads a `double` number (from text)
  - ...and so on...
  - `scanner.hasNext()` check if there is something else to read

* We will discuss in Lesson 28: *I/O and Streams* in detail what `InputStream`s are and how to create and use them.

** actually, `byte`s, but let's ignore this for now

Examples for using `System.out` and `Scanner`

Listing: A program reading a line and a number.

```
import java.util.Scanner; // import the scanner class: ignore this for now

/** Examples for using System.in and Scanner */
public class SystemIn {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    Scanner scanner = new Scanner(System.in); // initiate reading from System.in, ignore for now

    String string;
    System.out.println("Please␣enter␣string:␣");//$NON-NLS-1$
    string = scanner.nextLine(); // read next line from input and store in variable "string"
    System.out.print("You␣wrote:␣'"); //$NON-NLS-1$
    System.out.print(string); // print the stuff we read
    System.out.println("'.");//$NON-NLS-1$

    System.out.println("Please␣enter␣int␣number:␣"); // tell user to enter number //$NON-NLS-1$
    int value = scanner.nextInt(); // read the next integer from the input
    System.out.print("You␣wrote␣"); //$NON-NLS-1$
    System.out.print(value); // print the value we read
    System.out.print("␣and␣5␣times␣"); //$NON-NLS-1$
    System.out.print(value); // print the value we read
    System.out.print("␣is␣"); //$NON-NLS-1$
    System.out.println(value * 5); // print the value we read times 5, start new line
  }
}
```

Remark: These `//NON-NLS-1` things can safely be ignored, they are just there to tell Eclipse that a `String` literal is not internationalized/stored in a resource but to be used as it. Ignore them.

Piping Input into Our Program

- We have already seen that we can "pipe" the output of (our) one program to the input of another program

Piping Input into Our Program

- We have already seen that we can "pipe" the output of (our) one program to the input of another program
- We created a program `SystemOut` which prints one line of text to its output, then prints one line with a number, then prints some more text.

Piping Input into Our Program

- We have already seen that we can "pipe" the output of (our) one program to the input of another program
- We created a program `SystemOut` which prints one line of text to its output, then prints one line with a number, then prints some more text.
- We created a program `SystemIn` which reads one line of text from its input, then reads one line with a number, then ignores everything else and is finished.

Piping Input into Our Program

- We have already seen that we can "pipe" the output of (our) one program to the input of another program
- We created a program `SystemOut` which prints one line of text to its output, then prints one line with a number, then prints some more text.
- We created a program `SystemIn` which reads one line of text from its input, then reads one line with a number, then ignores everything else and is finished.
- Let's plug them together and do `java SystemOut | java SystemIn` or even

```
java SystemOut | java SystemIn > SystemOutToSystemInViaPipe.txt
```

Piping Input into Our Program

- We have already seen that we can "pipe" the output of (our) one program to the input of another program
- We created a program `SystemOut` which prints one line of text to its output, then prints one line with a number, then prints some more text.
- We created a program `SystemIn` which reads one line of text from its input, then reads one line with a number, then ignores everything else and is finished.
- Let's plug them together and do `java SystemOut | java SystemIn` or even

```
java SystemOut | java SystemIn > SystemOutToSystemInViaPipe.txt
```

Listing: Output of `java SystemOut | java SystemIn`

```
Please enter string:
You wrote: 'This text is printed and afterwards, a new line is started.'.
Please enter int number:
You wrote 34 and 5 times 34 is 170
```

Piping Input into Our Program

- We have already seen that we can "pipe" the output of (our) one program to the input of another program
- We created a program `SystemOut` which prints one line of text to its output, then prints one line with a number, then prints some more text.
- We created a program `SystemIn` which reads one line of text from its input, then reads one line with a number, then ignores everything else and is finished.
- Or let's pipe file `SystemOut.txt` into `SystemIn` and do

  ```
  java SystemIn < SystemOut.txt > SystemOutToSystemInViaFile.txt
  ```

  (which writes its output to `SystemOutToSystemInViaFile.txt`)

Piping Input into Our Program

- We have already seen that we can "pipe" the output of (our) one program to the input of another program
- We created a program `SystemOut` which prints one line of text to its output, then prints one line with a number, then prints some more text.
- We created a program `SystemIn` which reads one line of text from its input, then reads one line with a number, then ignores everything else and is finished.
- Or let's pipe file `SystemOut.txt` into `SystemIn` and do

  `java SystemIn < SystemOut.txt > SystemOutToSystemInViaFile.txt`

  (which writes its output to `SystemOutToSystemInViaFile.txt`)

Listing: Contents of `SystemOutToSystemInViaFile.txt`

```
Please enter string:
You wrote: 'This text is printed and afterwards, a new line is started.'.
Please enter int number:
You wrote 34 and 5 times 34 is 170
```

- So far, we have written to the console and read data from the console, which can either come from a user or from a pipe

- So far, we have written to the console and read data from the console, which can either come from a user or from a pipe
- The stuff we have used is called "Streams" and we used the two streams `stdout` (to write) `stdin` (to read)

- So far, we have written to the console and read data from the console, which can either come from a user or from a pipe
- The stuff we have used is called "Streams" and we used the two streams `stdout` (to write) `stdin` (to read)
- There is one more output stream: `stderr`

`stderr` vs. `stdout`

- So far, we have written to the console and read data from the console, which can either come from a user or from a pipe
- The stuff we have used is called "Streams" and we used the two streams `stdout` (to write) `stdin` (to read)
- There is one more output stream: `stderr`
- While `stdout` is for the real output of our program, `stderr` is for status and error information

stderr vs. stdout

- So far, we have written to the console and read data from the console, which can either come from a user or from a pipe
- The stuff we have used is called "Streams" and we used the two streams `stdout` (to write) `stdin` (to read)
- There is one more output stream: `stderr`
- While `stdout` is for the real output of our program, `stderr` is for status and error information
- In the console, it looks similar, but the difference becomes clear when we modify our `SystemIn` program a bit.

Examples for using `System.err` for status

Listing: System.in, System.out, and System.err.

```
import java.util.Scanner; // import the scanner class: ignore this for now

/** Examples for using System.in and Scanner using System.err instead of System.out for user
    interaction */
public class SystemInSystemErr {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    Scanner scanner = new Scanner(System.in); // initiate reading from System.in, ignore for now

    String string;
    System.err.println("Please enter string: ");//$NON-NLS-1$ // <--- using System.err for status
    string = scanner.nextLine(); // read next line from input and store in variable "string"
    System.out.print("You wrote: '"); //$NON-NLS-1$
    System.out.print(string); // print the stuff we read
    System.out.println("'.");//$NON-NLS-1$

    System.err.println("Please enter int number: "); // tell user to enter number using StdErr
        //$NON-NLS-1$
    int value = scanner.nextInt(); // read the next integer from the input
    System.out.print("You wrote "); //$NON-NLS-1$
    System.out.print(value); // print the value we read
    System.out.print(" and 5 times "); //$NON-NLS-1$
    System.out.print(value); // print the value we read
    System.out.print(" is "); //$NON-NLS-1$
    System.out.println(value * 5); // print the value we read times 5, start new line
  }
}
```

Remark: These `//$NON-NLS-1$` things can safely be ignored, they are just there to tell Eclipse that a `String` literal is not internationalized/stored in a resource but to be used as it. Ignore them.

The Output

Listing: `java SystemOut | java SystemInSystemErr`

```
You wrote: 'This text is printed and afterwards, a new line is started.'.
You wrote 34 and 5 times 34 is 170
```

- And we can also pipe the contents of a file into the input of our program...

Listing: `java SystemInSystemErr < SystemOut.txt`

```
You wrote: 'This text is printed and afterwards, a new line is started.'.
You wrote 34 and 5 times 34 is 170
```

- As you see, the questions to the user are no longer in the output...

Vertical Ball Throw Revisited

- We can now improve our "vertical ball throw" program from the previous lesson to read all of the parameters x_0, v_0, t from `stdin`

Listing: The vertical ball throw revisited.

```
import java.util.Scanner;

/**
 * A ball is thrown vertically upwards into the air by a x0m tall person
 * with velocity v0m/s. Where is it after t seconds?<br/>
 * x(t) = x0 + v0 * t - 0.5 * g * t^2
 */
public class VerticalBallThrowRevisited {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    Scanner scanner = new Scanner(System.in); // initiate reading from System.in, ignore for now
    System.err.println("Enter size x0 of person in m:"); //$NON-NLS-1$
    double x0 = scanner.nextDouble(); // read initial vertical position x0
    System.err.println("Enter initial upward velocity v0 of ball in m/s:"); //$NON-NLS-1$
    double v0 = scanner.nextDouble(); // read initial velocity upwards v0
    double g = 9.80665d; // free fall acceleration downwards
    System.err.println("Enter time t in s:"); //$NON-NLS-1$
    double t = scanner.nextDouble();  // read the time $t$
    double xt = x0 + (v0*t) - 0.5d*g*t*t; // x(t) = x0 + v0 * t - 0.5 * g * t^2
    System.out.println((xt > 0d) ? xt : 0d); // prints result and makes sure the ball stops at ground
  }
}
```

Summary

1. We have learned how to write output to the console.
2. We have learned how to read input from the console.
3. We can now write programs which interact with a user.
4. We have learned how to write status output to the console.
5. We have learned how to write the output of a program to a file, the output of a program to the input of another program, and a file to the input of a program.
6. We can now write programs which can be plugged together with many other tools.
7. All of this has nothing to do with Java, it works for all console programs!

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog