# Programmieren II

## Input/Output

Alexander Fraser

fraser@cl.uni-heidelberg.de

(Based on material from T. Bögel)

May 21, 2014

# Outline

# Outline

# Outline

# Object methods

- `boolean equals( Object obj )`
- `String toString()`
- `String getClass()`
- `int hashCode()`
- `Object clone()`

- and some others...

# Object methods

```
boolean equals( Object obj )
```

- implements an equivalence relation for objects
- *intended* meaning: "structural" equality, not necessarily same object identity
- But: `equals` method in `Object` tests for object identity:

  `o1.equals(o2)` $\Leftrightarrow$ `o1==o2` for `Object` instances `o1`, `o2`
- to define useful equivalence relations, `equals` is almost always overwritten (alongside the hashCode method ...)
- is called by the Java API. For instance, when storing a new object in a `Set` (discussed later), `equals` is automatically used to check whether the object is already in the set.

# Implementing equals(Object obj)

```
public class Book {
    ...
        public boolean equals(Object obj) {
            if (obj instanceof Book)
                return ISBN.equals((Book)obj.getISBN());
            else
                return false;
        }
}
```

Book.java

http://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html

- Note the use of the instanceof operator
- And the cast to Book
- Start from this code when you implement equals(Object obj)
- The hashCode() for equal objects must be the same (discussed later)

# Outline

# The class String

- Objects of the class `java.lang.String` (standard library)
- can be written as literals

```
String s1 = "I'm a String object";
String s3 = new String( s1 );
boolean eqTest = s3 == s1; // false!!
eqtest = s1 == "I'm a String object"; // true!
```

- Strings – like all objects – are compared with `equals` (not with ==)
- == inconsistently works for Strings (because of reuse in the String pool), danger: can lead to programming errors

# The class String

- Objects of the class `java.lang.String` (standard library)
- can be written as literals

```
String s1 = "I'm a String object";
String s3 = new String( s1 );
boolean eqTest = s3 == s1; // false!!
eqtest = s1 == "I'm a String object"; // true!
```

- Strings – like all objects – are compared with `equals` (not with ==)
- == inconsistently works for Strings (because of reuse in the String pool), danger: can lead to programming errors

# The class String

- Objects of the class `java.lang.String` (standard library)
- can be written as literals

```
String s1 = "I'm a String object";
String s3 = new String( s1 );
boolean eqTest = s3 == s1; // false!!
eqtest = s1 == "I'm a String object"; // true!
```

- Strings – like all objects – are compared with `equals` (not with ==)
- == inconsistently works for Strings (because of reuse in the String pool), danger: can lead to programming errors

# The class String

- Objects of the class `java.lang.String` (standard library)
- can be written as literals

```
String s1 = "I'm a String object";
String s3 = new String( s1 );
boolean eqTest = s3 == s1; // false!!
eqtest = s1 == "I'm a String object"; // true!
```

- Strings – like all objects – are compared with `equals` (not with ==)
- == inconsistently works for Strings (because of reuse in the String pool), danger: can lead to programming errors

# The class String

- Objects of the class `java.lang.String` (standard library)
- can be written as literals

```
String s1 = "I'm a String object";
String s3 = new String( s1 );
boolean eqTest = s3 == s1; // false!!
eqtest = s1 == "I'm a String object"; // true!
```

- Strings – like all objects – are compared with `equals` (not with ==)
- == inconsistently works for Strings (because of reuse in the String pool), danger: can lead to programming errors

# The class String

```
charAt        indexOf       contains
startsWith    endsWith      length
toLowerCase   toUpperCase   trim
substring     replace       split
...
```

http://download.oracle.com/javase/7/docs/api/java/lang/String.html

# Outline

# Formatting strings

## Formatting streams

- Stream classes implementing formatting options: `PrintWriter` (character) and `PrintStream` (bytes)

## `print & println`

- Outputs a string after calling appropriate `toString` method
- We already know this: `System.out.print(r);`
- Formatting for standard/error output: `System.out.format(formatString, arguments)`

## `String.format`

- `format` method: formats multiple arguments based on *format string*
- *format string*: static text embedded with format specifiers
- Documentation for format strings: `http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html#syntax`

# Formatting strings – example

```
public class Root2 {
    public static void main(String[] args) {
        int i = 2;
        double r = Math.sqrt(i);

        System.out.format("
         The square root of %d is %f.%n", i, r);
    }
}
```

# Format strings I

## Format strings

- Format specifiers begin with a %
- End with a 1- or 2-character *conversion*

## Examples for conversions

- d formats integer value as a decimal value
- f formats floating point values
- n formats platform-specific new line
- s formats any value as a string

# Format strings II

## Additional elements

- Precision (e.g. for floats)
- Width (minimum width)
- Flags (special formatting options)
- Argument index

## Example

```
double amount = 34002005.2450;
System.out.format("Money gained/lost since last
                   statement: %,.2f", amount);
```

**Output:**

```
Money gained/lost since last statement: 34,002,005.25
```

# Example for common scenario

## Printing information about an object – toString()

```
public class Patient {

    private String name;
    private int healthInsuranceNumber;
    ...
    /**
     * Overrides toString() method in Object.
     * Returns string representation of a Patient.
     */
    public String toString() {
        return String.format("Name: %s %n Health insurance
            number: %s", this.name, this.healthInsuranceNumber)
            ;
    }
    ...
}
```

# Literature


The Java tutorials – Numbers and Strings
`http://docs.oracle.com/javase/tutorial/java/data/numberformat.html`

Ullenboom, Ch.
*Java ist auch eine Insel*. (Chapter 4.7)
Galileo Computing, 2012.

# Outline

# java.util.regex

- Package `java.util.regex` (standard library)
- to use classes from the package:

```
import java.util.regex.*;
```

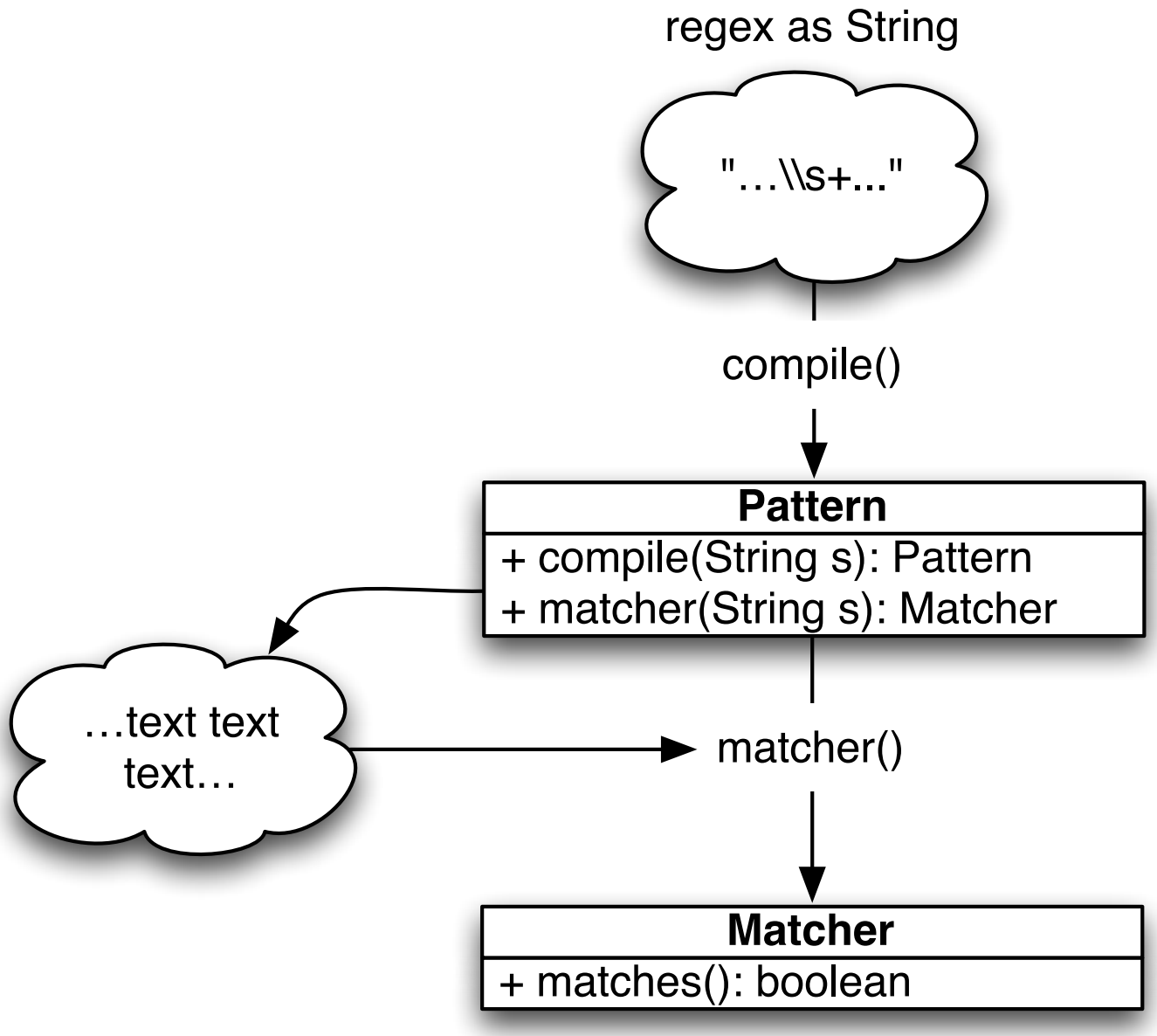
- Package contains classes `Pattern` and `Matcher`

# java.util.regex

- Regular expressions are specified as Strings
- ...and then compiled to `Pattern` objects
- `Matcher` objects are the result of the application of a pattern to a certain String

```
Pattern p = Pattern.compile( "a*b" );
Matcher m = p.matcher( "aaaaab" );
boolean b = m.matches();
```

# Java Regex Workflow



## Methods of the class Pattern

```
static Pattern   compile( String regex )
static Pattern   compile( String regex, int flags )
       Matcher   matcher( CharSequence input )
static boolean   matches( String regex, CharSequence input )
      String[]   split( CharSequence input )
```

## Methods of the class Pattern

```
static Pattern  compile( String regex )
static Pattern  compile( String regex, int flags )
       Matcher  matcher( CharSequence input )
static boolean  matches( String regex, CharSequence input )
      String[]  split( CharSequence input )
```

## Methods of the class Pattern

```
static Pattern   compile( String regex )
static Pattern   compile( String regex, int flags )
       Matcher   matcher( CharSequence input )
static boolean   matches( String regex, CharSequence input )
      String[]   split( CharSequence input )
```

## Methods of the class Pattern

```
static Pattern   compile( String regex )
static Pattern   compile( String regex, int flags )
       Matcher   matcher( CharSequence input )
static boolean   matches( String regex, CharSequence input )
      String[]   split( CharSequence input )
```

## Methods of the class Pattern

```
static Pattern   compile( String regex )
static Pattern   compile( String regex, int flags )
       Matcher   matcher( CharSequence input )
static boolean   matches( String regex, CharSequence input )
      String[]   split( CharSequence input )
```

## Methods of the class Matcher

- 3 Methods:

| | | |
|---|---|---|
| `boolean` | `matches( )` | complete match |
| `boolean` | `lookingAt( )` | match with prefix |
| `boolean` | `find( )` | match with sub-sequence (iterative) |

```
Pattern p = Pattern.compile( "ab" );
Matcher m = p.matcher( "abcdabcd" );
```

## Methods of the class Matcher

- 3 Methods:

| | | |
|---|---|---|
| `boolean` | `matches( )` | complete match |
| `boolean` | `lookingAt( )` | match with prefix |
| `boolean` | `find( )` | match with sub-sequence (iterative) |

```
Pattern p = Pattern.compile( "ab" );
Matcher m = p.matcher( "abcdabcd" );
```

## Methods of the class Matcher

- 3 Methods:

| | | |
|---|---|---|
| `boolean` | `matches( )` | complete match |
| `boolean` | `lookingAt( )` | match with prefix |
| `boolean` | `find( )` | match with sub-sequence (iterative) |

```
Pattern p = Pattern.compile( "ab" );
Matcher m = p.matcher( "abcdabcd" );
```

## Methods of the class Matcher

- 3 Methods:

| | | |
|---|---|---|
| `boolean` | `matches( )` | complete match |
| `boolean` | `lookingAt( )` | match with prefix |
| `boolean` | `find( )` | match with sub-sequence (iterative) |

```
Pattern p = Pattern.compile( "ab" );
Matcher m = p.matcher( "abcdabcd" );
```

## Methods of the class Matcher

- 3 Methods:

| | | |
|---|---|---|
| `boolean` | `matches( )` | complete match |
| `boolean` | `lookingAt( )` | match with prefix |
| `boolean` | `find( )` | match with sub-sequence (iterative) |

```
Pattern p = Pattern.compile( "ab" );
Matcher m = p.matcher( "abcdabcd" );

boolean result = m.matches(); // result = ?
```

## Methods of the class Matcher

- 3 Methods:

| | | |
|---|---|---|
| `boolean` | `matches( )` | complete match |
| `boolean` | `lookingAt( )` | match with prefix |
| `boolean` | `find( )` | match with sub-sequence (iterative) |

```
Pattern p = Pattern.compile( "ab" );
Matcher m = p.matcher( "abcdabcd" );

result = m.lookingAt(); // result = ?
```

## Methods of the class Matcher

- 3 Methods:

| | | |
|---|---|---|
| boolean | matches( ) | complete match |
| boolean | lookingAt( ) | match with prefix |
| boolean | find( ) | match with sub-sequence (iterative) |

```
Pattern p = Pattern.compile( "ab" );
Matcher m = p.matcher( "abcdabcd" );

result = m.find(); // result = ?
```

## Methods of the class Matcher

- 3 Methods:

| | | |
|---|---|---|
| boolean | matches( ) | complete match |
| boolean | lookingAt( ) | match with prefix |
| boolean | find( ) | match with sub-sequence (iterative) |

```
Pattern p = Pattern.compile( "ab" );
Matcher m = p.matcher( "abcdabcd" );

result = m.find(); // result = ?
result = m.find(); // result = ?
```

## Additional methods of the class Matcher

| | |
|---|---|
| `String group( )` | Previously matched substring |
| `String group(int group)` | Last match of the group group |
| `int groupCount( )` | Number of groups in a pattern |
| `Matcher reset( )` | Sets current position to 0 |
| `Matcher reset(CharSeq. input )` | Resets the input |
| `String replaceAll(String repl )` | Replaces each match by repl (impl. Reset) |
| `String replaceFirst(String repl)` | Replaces the first match (ipml. Reset) |

## Additional methods of the class Matcher

| | |
|---|---|
| `String group( )` | Previously matched substring |
| `String group(int group)` | Last match of the group group |
| `int groupCount( )` | Number of groups in a pattern |
| `Matcher reset( )` | Sets current position to 0 |
| `Matcher reset(CharSeq. input )` | Resets the input |
| `String replaceAll(String repl )` | Replaces each match by repl (impl. Reset) |
| `String replaceFirst(String repl)` | Replaces the first match (ipml. Reset) |

## Additional methods of the class Matcher

| | |
|---|---|
| `String group( )` | Previously matched substring |
| `String group(int group)` | Last match of the group group |
| `int groupCount( )` | Number of groups in a pattern |
| `Matcher reset( )` | Sets current position to 0 |
| `Matcher reset(CharSeq. input )` | Resets the input |
| `String replaceAll(String repl )` | Replaces each match by repl (impl. Reset) |
| `String replaceFirst(String repl)` | Replaces the first match (ipml. Reset) |

## Additional methods of the class Matcher

| | |
|---|---|
| `String group( )` | Previously matched substring |
| `String group(int group)` | Last match of the group group |
| `int groupCount( )` | Number of groups in a pattern |
| `Matcher reset( )` | Sets current position to 0 |
| `Matcher reset(CharSeq. input )` | Resets the input |
| `String replaceAll(String repl )` | Replaces each match by repl (impl. Reset) |
| `String replaceFirst(String repl)` | Replaces the first match (ipml. Reset) |

## Additional methods of the class Matcher

| | |
|---|---|
| `String group( )` | Previously matched substring |
| `String group(int group)` | Last match of the group group |
| `int groupCount( )` | Number of groups in a pattern |
| `Matcher reset( )` | Sets current position to 0 |
| `Matcher reset(CharSeq. input )` | Resets the input |
| `String replaceAll(String repl )` | Replaces each match by repl (impl. Reset) |
| `String replaceFirst(String repl)` | Replaces the first match (ipml. Reset) |

# Regular expressions

- Regex syntax:
  `http://download.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#sum`
- Very useful online tool to live-check regular expressions:
  `http://gskinner.com/RegExr/`
- Java Regex Tutorial:
  `http://download.oracle.com/javase/tutorial/essential/regex/`
- J. Friedl (1997). Mastering Regular Expressions. O'Reilly.

# Literature


The Java tutorials
`http://docs.oracle.com/javase/tutorial/java/data/strings.html`

Ullenboom, Ch.
*Java ist auch eine Insel*. (Chapter 4.4)
Galileo Computing, 2012.

# Outline

# Outline

# I/O in Java

## What is I/O?

- I/O = Input & Output
- Communication of a program with the "world"
- In Java: package `java.io` (and `java.nio`)

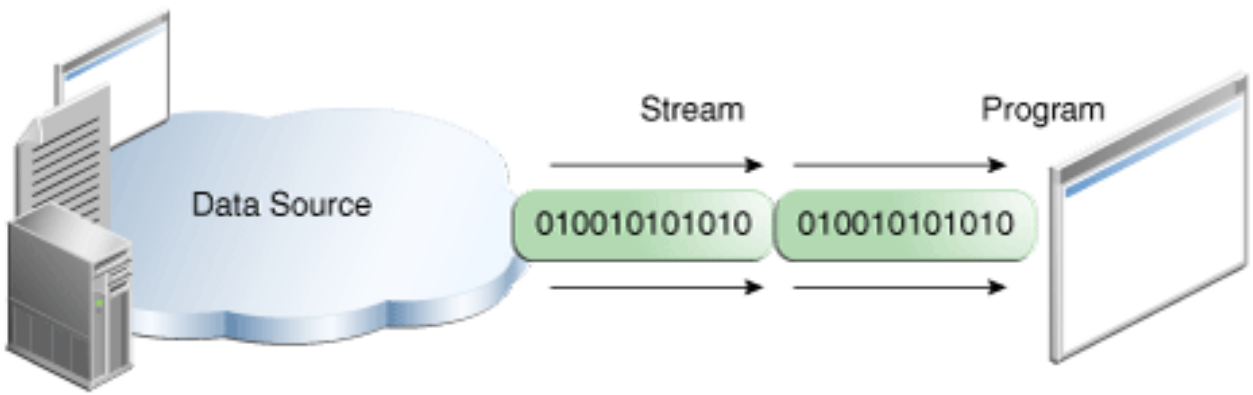
## Types of Communication

- Output to the console (standard out)
- Writing into a file
- Reading files
- Reading interactively from the user (standard input)

# The concept of Streams

## Streams

- I/O Stream: input source or output destination
- Various sources/destinations: files, devices, memory array etc.
- Streams support various data: bytes, primitive types, Strings, objects etc.
- Streams add abstraction and simplification
- Stream is always a sequence of data




source: `http://docs.oracle.com/javase/tutorial/essential/io/streams.html`

# Byte Streams

## Byte Streams

- Input and Output of 8-bit bytes
- For example: File Byte Streams
- Classes: `FileInputStream` and `FileOutputStream`
- Low-level stream, rarely used

# Character streams

## Character streams in Java

- Input and output with streams automatically translates to and from local character set
- Helpful for internationalization
- Character streams are sub-classes of `Reader` and `Writer`
- Character streams for files: `FileReader` and `FileWriter`

# Buffered Streams I

## Buffered Streams

- Unbuffered I/O: read/write requests handled directly by the OS
- This is inefficient: I/O is expensive
- Solution: buffers to reduce the overhead
- Buffered input: read from memory area (buffer)
- Buffered output: write to buffer
- Native I/O method is called if the buffer is full/empty

# Buffered Streams II

## Unbuffered to buffered: wrapping

- Unbuffered stream object passed to constructor of buffered stream class
- `inputStream = new BufferedReader(new FileReader("xanadu.txt"));`
- `outputStream = new BufferedWriter(new FileWriter("characteroutput.txt"));`

## Buffered character streams

- `BufferedReader`
- `BufferedWriter`

# Reading a file line-by-line (HFJ, pp. 452 – 454)

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class PlaintextReader {
  public static void main(String[] args) {
    try {
      BufferedReader br = new BufferedReader(new FileReader("
          test.txt"));
      String line = br.readLine();
      while (line != null) {
          line = br.readLine();
          System.out.println(line);
      }
      // always ensure that a stream is closed!
      br.close();
    } catch (IOException e) {e.printStackTrace();}
  }
}
```

code/PlaintextReader.java

# Writing to a file

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class PlaintextWriter {
  public static void main(String[] args) {
    try {
      BufferedWriter bw = new BufferedWriter(new FileWriter("
          test.txt"));
      bw.write("bla, bla, bla\n");
      bw.close();
    } catch (IOException e) {
      e.printStackTrace();
    }

  }
}
```

code/PlaintextWriter.java

# Outline

## Standard Input access

- Standard input and output often used to interact with a user
- Usually: output to standard output (terminal screen), followed by user input (from keyboard)
- Two possibilities in Java:
  - Standard streams
  - The Console

# Standard streams

## Standard streams

- By default: reading input from keyboard
- Writing output to the display
- (Of course, streams can be piped, bypassed etc.)

## Usage of Standard Streams in Java

| OS Stream | Java Stream |
|---|---|
| Standard Input | `System.in` |
| Standard Output | `System.out` |
| Standard Error | `System.err` |

# Using Standard Streams in Java

## `System.out` & `System.err`

- Can be used just like character streams
- Example: `System.err.println("How could that happen?");`

## `System.in`

- Needs to be used as a **byte stream**
- E.g. `InputStreamReader cin = new InputStreamReader(System.in);`
- Complete example for interacting with user: next slide

# Read lines from standard input

```
import java.io.*;
/** Read a string from console using a BufferedReader.*/
public class BRReadLines {
    public static void main(String args[]) throws IOException
    {
        // Create a BufferedReader using System.in
        BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));
        String str;
        System.out.println("Enter lines of text.");
        System.out.println("Enter 'end' to quit.");
        do {
            str = br.readLine();
            System.out.println(str);
        } while(!str.equals("end"));
    }
}
```

code/BRReadLines.java

# The Console

## `Console` class

- More advanced methods for user input
- Useful for secure password entry
- Directly provides character stream methods (e.g. `readline()`)
- To obtain a `Console` object: `Console c = System.console();`

- Example: Password.java (Moodle)

# What you should know...

- How to read contents from files
- How to write strings to a file

## Serialization

- You can also write objects
- This is called **serialization**
- Analogy: deep-freezing
- Serialized objects can be stored, sent via network etc. and de-serialized again
- More details: `http://docs.oracle.com/javase/tutorial/essential/io/objectstreams.html`

# Next time

- Next time: java.nio
- Operating system independent file paths
- Manipulate files and directories

# Literature


The Java tutorials
`http://docs.oracle.com/javase/tutorial/essential/io/`

Ullenboom, Ch.
*Java ist auch eine Insel*. (Chapter 17 - Streams)
Galileo Computing, 2012.

Sierra, K. & Bates, B.
*Head First Java*. (Chapter 14 - Covers Serialization and Text I/O)
O'Reilly Media, 2005.