I/O

Adrian Schüpbach
adrian_laurent.schuepbach@alumni.ethz.ch

Worum geht es?

- I/O (Eingabe und Ausgabe)
  - Tastatur einlesen
  - Kommandozeilenparameter einlesen
  - Dateien lesen und schreiben
- Exceptions
  - Fehlerbehandlung (Bsp.: Datei existier ev. nicht)
  - Programmfehler

Exceptions

Exceptions

- Fehler können immer passieren
- Was sind Fehler?

Exceptions

- Fehler können immer passieren
- Was sind Fehler?

- Verschiedene Beispiele von Fehlern
  - Programmierfehler
  - Benutzerfehler (falsche/unerwartete Eingaben)
  - Berechnungsfehler (Division durch 0)
  - Resource nicht vorhanden (Datei, Netzwerk, ...)
  - Kein Speicher mehr
  - ...

Exceptions

- Fehler können in Klassen eingeteilt werden
  - Erwartete und unerwartete Fehler
  - Behandelbare und unbehandelbare Fehler

Exceptions

- Fehler muss man ...
  1. ... erkennen
  2. ... mitteilen
  3. ... behandeln

- $\rightarrow$ Dafür gibt es in Java den *Exception*-Mechanismus

Exceptions

- Fehler muss man ...
  1. ... erkennen
  2. ... mitteilen
  3. ... behandeln

- $\rightarrow$ Dafür gibt es in Java den *Exception*-Mechanismus

- Man nimmt an, dass Fehler die Ausnahme sind
  - $\rightarrow$ Exceptions, Exception handling
  - Ausnahmebehandlung

Exceptions

- Eine *Exception* ist ein “Fehlerobjekt”
  - Instanz einer “Fehlerklasse”
  - Ist wie normale Klasse in Java
  - Beinhaltet Details über den Fehler
- Wenn ein Fehler eintritt, wird Fehlerobjekt erzeugt

Exceptions

- Eine *Exception* ist ein “Fehlerobjekt”
  - Instanz einer “Fehlerklasse”
  - Ist wie normale Klasse in Java
  - Beinhaltet Details über den Fehler
- Wenn ein Fehler eintritt, wird Fehlerobjekt erzeugt

- Es gibt zwei Möglichkeiten, wie Fehlerobjekt erzeugt wird
  - System erzeugt es, wenn Fehler auftritt
    - Division durch 0, ungültiger Arrayindex, ...
  - Programmierer erzeugt Fehlerobjekt
    - Z.Bsp. ungültige Benutzereingabe

Exceptions

- Sobald Exception auftritt, muss sie abgefangen werden
- Normaler Programmfluss wird unterbrochen

Exceptions

- Sobald Exception auftritt, muss sie abgefangen werden
- Normaler Programmfluss wird unterbrochen

- Zwei Möglichkeiten zur Behandlung
  - Fehler “erwartet”: Sofort behandeln
  - Fehler nicht erwartet: Exception an Methode weiterleiten, die “mich” aufgerufen hat

Exceptions

Exception behandeln

- Code mit potentiellem Fehler in `try`-Block einbetten
- Exception in `catch`-Block behandeln
  - `try{ } catch(Exception e) { }`

Exceptions

Exception behandeln

```
public class Programm37 {
  public static void dateioeffnen(String dateiname){
    try {
      // Code, der ev. Fehler haben koennte
      // Z.Bsp. Datei oeffnen
      // "Fehler" = Kann Datei nicht oeffnen
    } catch (Exception e) {
      // Fehler behandeln
      // Z.Bsp. dem Benutzer sagen, dass
      // Datei nicht geoeffnet werden kann
    }
  }
  public static void main(String[] args) {
    dateioeffnen("bla");
  }
}
```

Exceptions

Exception weiterleiten

- Methodensignatur zeigt an, dass Fehler weitergeleitet werden
  - `public void m() throws Exception { }`

Exceptions

Exception nach oben weiterleiten

```
public class Programm38 {
  public static void dateioeffnen(String dateiname)
    throws Exception {

      // Code, der ev. Fehler haben k"onnte
      // Z.Bsp. Datei oeffnen
  }
  public static void main(String[] args) {
    try {
      dateioeffnen("bla");
    } catch (Exception e) {
      System.out.println("Fehler");
    }
  }
}
```

Exceptions

Exception erzeugen

- Exception-Objekt erzeugen (wie sonstiges Objekt auch)
- Exception-Objekt *werfen*
  - `throw new Exception();`

Exceptions

Exception werfen/erzeugen

```
public class Programm39 {
  public static void dateioeffnen(String dateiname)
    throws Exception {
      if(dateiname.length() == 0) {
          throw new Exception();
      }
  }
  public static void main(String[] args) {
    try {
      dateioeffnen("bla");
    } catch (Exception e) {
      System.out.println("Fehler");
    }
  }
}
```

Exceptions

Beispiele von typischen Fehlern

- Zugriff auf nicht erzeugtes Array oder Objekt
- Ungültiger Array-Index
- Division durch 0
- Datei nicht gefunden
- ...

Exceptions

NULL-Pointer-Exception

```
public class Programm40 {
  public static void main(String[] args) {
    String s = null;
    int laenge = s.length();

    int[] zahlenArray = null;
    zahlenArray[3] = 17;
  }
}
```

Exceptions

Besser: NULL-Pointer-Exception

```
public class Programm41 {
  public static void main(String[] args) {
    String s = null;
    int laenge;
    try {
      laenge = s.length();
    } catch (Exception e) {
      System.out.println("String nicht erzeugt");
    }

    int[] zahlenArray = null;
    try {
      zahlenArray[3] = 17;
    } catch (Exception e) {
      System.out.println("Array nicht erzeugt");
    }
```

Exceptions

Spezifische Exception fangen

- Bis jetzt gesehen: `catch (Exception e)`
- Was passiert im Folgenden Beispiel?

Welche Exception?

```
public class Programm42 {
  public static void main(String[] args) {
    String s = null;
    int[] zahlenArray = null;

    try {
      zahlenArray[10293] =  s.length();
    } catch (Exception e) {
      System.out.println("Ups, String oder Array"
        +" nicht erzeugt oder falsche Array-Index");
    }
  }
```

Exceptions

Spezifische Exception fangen

- Es gibt spezialisierte Exceptions
- Kann diese spezifisch fangen
  - `NullPointerException`
  - `ArrayIndexOutOfBoundsException`
  - `ArithmeticException`
  - `IOException`
  - ...

Exceptions

Spezifische Exception fangen

Spezifische Exception fangen

```
public class Programm43 {
  public static void main(String[] args) {
    String s = null;
    int[] zahlenArray = null;

    try {
      zahlenArray[10293] =  s.length();
    } catch (NullPointerException ne) {
      System.out.println("String oder Array"
                         +" nicht erzeugt.");
    } catch (ArrayIndexOutOfBoundsException obe) {
      System.out.println("Falscher Array-Index");
    }
  }
}
```

I/O

I/O

- Jedes Programm liest Daten ein oder gibt Daten aus
- Es gibt verschiedene Eingabe- und Ausgabemöglichkeiten
  - Bildschirm
  - Tastatur
  - Drucker
  - Netzwerk
  - Dateien
  - Kommandozeilenparameter
  - ...

Tastatureingaben

Tastatureingaben

- Tastatur ist wichtige Eingabequelle
- Gehört zum System
- Gibt eine Anzahl Methoden, wie Tastatur eingelesen werden kann

Tastatureingaben

- Eingaben über `String System.in.read(byte[] byteArray)`
  - System liest, bis ENTER gedrückt wird
  - Zeichen landen in Byte-Array
  - Byte-Array in String konvertieren:
    ```
    String eingabe = new String(byteArray);
    ```

- Achtung: Wirft IOException
  - `try{} catch(IOException ioe) { }`
  - `IOException` muss importiert werden
    - Mehr dazu später

Tastatureingaben

Tastatureingabe

```
import java.io.IOException;

public class Programm45 {
    public static void main(String[] args) {
      byte[] eingabe = new byte[256];
      // SEHR WICHTIG:
      System.out.println("Gib etwas ein: ");
      try {
        System.in.read(eingabe);
      } catch (IOException ieo) {
        System.out.println("Eingabefehler");
      }
      String textEingabe = new String(eingabe);
    }
}
```

Kommandozeilenparameter

Kommandozeilenparameter

- Zweite, sehr angenehme Art der Dateneingabe

Kommandozeilenparameter

- Gesehen: `public static void main(String[] args)`
  - `main` nimmt also ein `String`-Array entgegen
- Array beinhaltet einzelne Kommandozeilenparameter
- Alle Parameter sind Strings, auch die Zahlen!

Kommandozeilenparameter

- Parameter benutzen: Programm ...
  1. ... macht Schleife über ganze Parameterliste
  2. ... schaut sich jeden Parameter an
  3. ... prüft, ob er bekannt und gültig ist
  4. ... transformiert Text-Zahlen in echte Zahlenwerte
  5. ... speichert Parameterwert in Variable
  6. ... sollte testen, dass nötige Parameter angegeben wurden

Kommandozeilenparameter

- Typische Kommandozeilenaufrufe
  - java MeinProgramm -h
  - java MeinProgramm –help
  - java MeinProgramm -f dateiname.txt –anzahl 7 -s -l 1
  - java MeinProgramm -f=dateiname.txt –anzahl=7 -s -l=3

Kommandozeilenparameter

Kommandozeilenparameter einlesen (alle optional hier)

```
public class Programm44 {
  static int level=0, anzahl=0;
  static boolean s=false; static String file="";
    public static void main(String[] args) {
      for (int i=0; i< args.length; i++) {
        if (args[i].equals("--help")) {
          printUsage();
        } else if (args[i].equals("-f")) {
          inputFile = args[i+1];
          i++;
        } else if (args[i].equals("--anzahl")) {
          anzahl = Integer.parseInt(args[i+1]);
          i++;
        }
      }
    }
```

Kommandozeilenparameter

- Vorteile der Kommandozeilenparameter

Kommandozeilenparameter

- Vorteile der Kommandozeilenparameter
  - Schnell für Benutzer

Kommandozeilenparameter

- Vorteile der Kommandozeilenparameter
  - Schnell für Benutzer
  - Kann aus Skript/Programm mit Parametern aufgerufen werden

Kommandozeilenparameter

- Vorteile der Kommandozeilenparameter
  - Schnell für Benutzer
  - Kann aus Skript/Programm mit Parametern aufgerufen werden
  - Einfach, gewisse Eingaben optional zu machen

Dateien

Dateien

- Dateien: Datenspeicher

Dateien

- Dateien: Datenspeicher
- Eigenschaften
  - Grosse Datenmenge (Ein- und Ausgabe)
  - Aneinanderkettung von Bytes
  - Daten bleiben bestehen → Persistenz

Dateien

- Solange Inhalt der Datei nicht intepretiert wird, sind es nur Bytes

Dateien

- Solange Inhalt der Datei nicht intepretiert wird, sind es nur Bytes
- Dateiformate
  - Dateiformate erklären, wie Inhalt interpretiert werden soll
  - Bsp.: CSV, Bild, HTML, Zip, PDF, ...

Dateien

- Solange Inhalt der Datei nicht intepretiert wird, sind es nur Bytes
- Dateiformate
  - Dateiformate erklären, wie Inhalt interpretiert werden soll
  - Bsp.: CSV, Bild, HTML, Zip, PDF, ...
- Bsp.: CSV: Tabelle, Spalten mit Komma getrennt
  - Datei soll zeilenweise eingelesen werden
  - Komma trennt Spalten
  - Text zwischen Kommas ist Zelleninhalt

Dateien

- Wichtige Klassen im Umgang mit Dateien:
  - `File`
  - `FileReader`
  - `FileWriter`
  - `BufferedReader`
  - `BufferedWriter`
  - `RandomAccessFile`

Dateien

`File`

- `File` ist allgemeines Datei-Objekt
  - Informationen über Dateien
  - Datei erzeugen
  - Datei löschen
- File-Objekt erzeugen
  - `File datei = new File("dateiname.txt");`
  - Erzeugt File-Objekt, nicht Datei!

Dateien

File

Nützlichen File-Funktionen

Anweisung	Operation
`File f=new File("datei.txt");`	Erzeugt neues File-Objekt
`boolean b = f.exists()`	Testet, ob Datei existiert
`long l = f.length();`	Gibt Dateigrösse zurück
`f.createNewFile();`	Erzeugt Datei mit Namen in `f`
`f.delete();`	Löscht die Datei
`f.isFile()`	Abfrage, ob Pfad Datei ist

Dateien

File

Pfade

Anweisung	Operation
`File f=new File("./../datei.txt");`	Erzeugt neues File-Objekt
`String s=f.getName();`	dateiname.txt
`String s=f.getPath();`	./../dateiname.txt
`String s=f.getAbsolutePath();`	/home/adrian/programm/java/./../dateiname.txt
`String s=f.getCanonicalPath();`	/home/adrian/programm/java/dateiname.txt

Dateien

File

- Komplette Liste der Methoden der Klasse java.io.File in der API erklärt.

Dateien

File

File

```
public class Programm46 {
  public static void main(String[] args)
    throws IOException {
    File f = new File("testdatei01.txt");
    if (!f.exists()) {
      f.createNewFile();
    }
    String canonicalName = f.getCanonicalPath();
    long dateiGroesse = f.length();
    System.out.println("Datei " + canonicalName
        + " ist " + dateiGroesse + " Bytes gross.");
    f.delete();
  }
}
```

Dateien

Dateien

- Dateien einlesen

Dateien

- Dateien einlesen
- Vorgehensweise/“Lesealgorithmus”
  1. Datei öffnen
  2. Datei einlesen
  3. Datei **schliessen**

Dateien

Lesezeiger

- Datei hat Lesezeiger
  - Zeigt auf nächstes Byte, das eingelesen wird
  - Zeigt direkt nach dem Öffnen auf Byte 0 (erstes Byte)
  - Wird automatisch um 1 erhöht, nachdem ein Byte gelesen wurde

Dateien

`FileReader`

- `FileReader` verantwortlich ...
  - ... eine Datei zu öffnen
  - ... eine Datei einzulesen
  - ... eine Datei zu schliessen
  - ... Lesezeigermanipulationen

Dateien

FileReader

Nützlichen FileReader-Funktionen

Anweisung	Operation
`FileReader fr = new FileReader(File f)`	Öffnet Datei
`FileReader fr = new FileReader(String dateiName)`	Öffnet Datei
`fr.close()`	Schliesst Datei
`fr.read(char[] buffer)`	Liest Bytes in Buffer ein
`fr.skip(long n)`	Lässt n Bytes aus

Dateien

`FileReader`

- Kleine Dateien komplett einlesen
  1. Dateigrösse bestimmen
  2. Puffer dieser Grösse allozieren
  3. Datei einlesen
  4. Datei schliessen

Dateien

`FileReader`

- Kleine Dateien komplett einlesen
  1. Dateigrösse bestimmen
  2. Puffer dieser Grösse allozieren
  3. Datei einlesen
  4. Datei schliessen

- Grosse Dateien können kaum komplett eingelesen werden
- Müssen Blockweise/Zeilenweise eingelesen werden
  1. Puffer einer fixen Grösse allozieren
  2. Schleife implementieren
  3. In jedem Schleifendurchgang Block lesen
  4. Datei schliessen

Dateien

FileReader

Datei blockweise lesen

```
public class Programm47 {
  public static void main(String[] args)
    throws Exception {
    FileReader fr = new FileReader("datei.txt");
    char[] puffer = new char[1024];
    while (fr.read(puffer) != -1) {
      String s = new String(puffer);
      System.out.println("Naechtser Teil: " + s);
    }
    fr.close();
  }
}
```

Dateien

Textdatei einlesen

Dateien

Textdatei einlesen

- Oft speichern Dateien Text
  - Auch Zahlen in Textform
- Oft sind Tabellen in Textform gespeichert
  - Eine Zeile in der Datei = Eine Zeile der Tabelle
  - Komma trennt Spalten

Dateien

Textdatei einlesen

- Oft speichern Dateien Text
  - Auch Zahlen in Textform
- Oft sind Tabellen in Textform gespeichert
  - Eine Zeile in der Datei = Eine Zeile der Tabelle
  - Komma trennt Spalten

- Idealerweise zeilenweise einlesen

Dateien

Textdatei einlesen

- `BufferedReader` hat Methode `String readLine()`
  - Liest eine ganze Zeile
  - Gibt den String zurück
- `String` hat Methode `split(char trennung)`
  - String wird in mehrere Strings eingeteilt
  - Trennungszeichen wird in `char trennung` angegeben

- $\rightarrow$ Kombination liefert
  - Zeile
  - Auftrennung in Spalten

Dateien

Textdatei einlesen

Datei zeilenweise einlesen

```
public class Programm48 {
  public static void main(String[] args)
    throws Exception {
    FileReader fr = new FileReader("datei.txt");
    BufferedReader reader = new BufferedReader(fr);
    String zeile;

    while ((zeile = reader.readLine()) != null) {
      String[] spalten = zeile.split(",");
      int zahl = Integer.parseInt(spalten[3]);
    }
  }
}
```

Dateien

Textdatei schreiben

Dateien

Textdatei schreiben

- Dateien schreiben

Dateien

Textdatei schreiben

- Dateien schreiben
- “Speichern” == in eine Datei schreiben
  - Daten
  - Einstellungen
  - ...

Dateien

Textdatei schreiben

- Algorithmus
  1. Datei erstellen oder existierende Datei öffnen
  2. Daten schreiben
  3. Datei schliessen

Dateien

Textdatei schreiben

- Algorithmus
  1. Datei erstellen oder existierende Datei öffnen
  2. Daten schreiben
  3. Datei schliessen

- $\rightarrow$ ähnlich, wie Datei lesen

Dateien

Textdatei schreiben

- Kombination aus `FileWriter` und `BufferedWriter`
  - So wie `FileReader` und `BufferedReader`

Dateien

Textdatei schreiben

Nützlichen BufferedWriter-Funktionen

Anweisung	Operation
`BufferedWriter bw = new BufferedWriter(fileWriter)`	Öffnet Datei
`bw.close()`	Schliesst Datei
`bw.newLine()`	Fügt Zeichen für eine neue Zeile ein
`bw.write(String s, int off, int len)`	Schreibt Teil eines Strings

Dateien

Textdatei schreiben

Textdatei schreiben

```
public class Programm49 {
  public static void main(String[] args)
  throws Exception {
    File file = new File("datei.txt");
    file.createNewFile();
    BufferedWriter bw =
        new BufferedWriter(new FileWriter(file));
    String text1 = "Welt"; String text2 = "All";
    bw.write(text1, 0, text1.length());
    bw.newLine();
    bw.write(text2, 0, text2.length());
    bw.close();
  }
}
```

Dateien

Dateien

- Jetzt können wir ...
  - ... Dateien erstellen
  - ... Dateien löschen
  - ... Dateien vollständig einlesen
  - ... Dateien vollständig schreiben
- → Super für Bearbeitung wissenschaftlicher Daten

Dateien

- Jetzt können wir ...
  - ... Dateien erstellen
  - ... Dateien löschen
  - ... Dateien vollständig einlesen
  - ... Dateien vollständig schreiben
- → Super für Bearbeitung wissenschaftlicher Daten

- Manchmal Daten aus Datei von bestimmter Stelle lesen
- Manchmal in Datei an bestimmte Stelle schreiben

Dateien

- Jetzt können wir ...
  - ... Dateien erstellen
  - ... Dateien löschen
  - ... Dateien vollständig einlesen
  - ... Dateien vollständig schreiben
- → Super für Bearbeitung wissenschaftlicher Daten

- Manchmal Daten aus Datei von bestimmter Stelle lesen
- Manchmal in Datei an bestimmte Stelle schreiben
- → RandomAccessFile

Dateien

`RandomAccessFile`

- Methode `seek(long pos)`
  - Setzt Dateizeiger auf Position `pos`
  - Nächste Lese- oder Schreiboperation beginnt an Stelle `pos`

- Ausserdem grosse Methodensammlung für Lese- und Schreiboperationen

Dateien

RandomAccessFile

Nützlichen RandomAccessFile-Funktionen

Anweisung	Operation
`RandomAccessFile raf = new RandomAccessFile(String name, String mode)`	Öffnet Datei
`RandomAccessFile raf = new RandomAccessFile(File file, String mode)`	Öffnet Datei
`raf.close()`	Schliesst Datei

Dateien

RandomAccessFile

Nützlichen RandomAccessFile-Funktionen

Anweisung	Operation
`raf.seek(long pos)`	Setzt Dateizeiger auf `pos`
`raf.readInt()`	Liest Integer
`raf.readLine()`	Liest Textzeile
`raf.writeDouble(double d)`	Schreibt `double`
...	...

Um eine so geschrieben Datei richtig einlesen zu können, muss man das *Dateiformat* genaustens kennen!