5. Übungsbesprechung Programmkonstruktion

Karl Gmeiner

karl@complang.tuwien.ac.at

12.01.2012

Rückblick und Ausblick

Test 2

- Einsichtnahme 17.01., 10:00, Argentinierstraße 8/4.
- Invarianten Gültigkeit auch vor und nach der Schleife.

Test 4

- Gesamter Stoff

Aufgabe 1: Exceptions

```
class IntNumber {
  private int value;

  IntNumber(int initValue) {
    this.value = initValue;
  }

  boolean isZero() {
    return value == 0;
  }

  IntNumber div(IntNumber that) {
    return new IntNumber(
        this.value / that.value);
  }
}
```

Fehlerfall Division durch 0

- Java wirft eine Exception bei einer Division durch 0.
- Möglichkeiten, diesen Fehler zu vermeiden:
  1. Fehlerhaften Aufruf mit `if` vermeiden
  2. Checked Exception - Exception muss behandelt werden.
  3. Unchecked Exception - RuntimeException muss nicht behandelt werden.
  4. `return null;` im Fehlerfall

Aufgabe 1: Fehlerhaften Aufruf mit `if` vermeiden

```
class IntNumber {
  boolean isZero() {
    return value == 0;
  }

  IntNumber div(IntNumber that) {
    return new IntNumber(
      this.value / that.value);
  }
}
```

```
IntNumber n1, n2;

if ( ! n2.isZero() ) {
  n1.div( n2 );
} else {
  // Fehlerbehandlung
}
```

Eigenschaften

- Fehlerfall wird vorher ausgeschlossen (vgl. auch Bereichsgrenzen-Checks)
- Fehlerbehandlung lokal.
- Kein Rollback nötig.
- Oft einziger Weg, Fehler zu vermeiden.
- Ist eine Division durch 0 ein Ausnahmefall?
- Fehler kann an anderer Stelle auch auftreten.

Aufgabe 1: Exception

```
class IntNumber {
  IntNumber div(IntNumber that)
      throws DivByZeroException {
    if ( that.isZero() )
        throw new DivByZeroException();

    return new IntNumber(
      this.value / that.value);
  }
}
```

```
IntNumber n1, n2;
try {
  n1.div( n2 );
} catch (DivByZeroException e) {
  // Fehlerbehandlung
}
```

Eigenschaften

- Fehler wird behandelt, nachdem er aufgetreten ist.
- Fehlerbehandlung auch über mehrere Methoden hinweg.
- Aufrufer MUSS Fehler behandeln, auch wenn er z.B. durch `if` zuvor ausgeschlossen wurde.

Aufgabe 1: RuntimeException

```
class IntNumber {
  IntNumber div(IntNumber that) {
    if ( that.isZero() )
        throw new DBZRunTimeEx();

    return new IntNumber(
      this.value / that.value);
  }
}
```

```
IntNumber n1, n2;

if ( ! n2.isZero() ) {
  n1.div( n2 );
} else {
  // Fehlerbehandlung
}
```

Eigenschaften

- Aufrufer kann Exception fangen, oder mit `if` ausschließen bzw. ignorieren.
- "Unchecked Exceptions - The Controversy"

Aufgabe 1: return null

```
class IntNumber {
  IntNumber div(IntNumber that) {
    if ( that.isZero() ) {
        return null;
    } else {
      return new IntNumber(
        this.value / that.value);
  }
}
```

Eigenschaften

- Problem: Bedeutung von `null`?
- Alternative: Unterklasse `InvalidNumber` von `IntNumber`

```
IntNumber n1, n2;

IntNumber n = n1.div( n2 );

if ( n == null ) {
  // Fehlerbehandlung
}
```

Aufgabe 2: BufferedReader

Schreiben Sie ein Programm, das die Zeilen einer Datei zählt und am Ende ausgibt.

- `Reader`: Einlesen von Text
- `InputStream`: Einlesen von Binärdaten (kein `readLine`)

Datei + Stream öffnen

```
try {
  BufferedReader br = new BufferedReader(
       new FileReader(filename));
} catch (FileNotFoundException e) {
  System.err.println("Datei nicht gefunden");
}
```

Aufgabe 2: BufferedReader cont'd

Datei + Stream öffnen

```
try {
  BufferedReader in = new BufferedReader(
       new FileReader(filename));
} catch (FileNotFoundException e1) {
    System.err.println(e.getMessage());
}
```

Aufgabe 2: BufferedReader cont'd

Datei einlesen - 1. Versuch

```
try {
  BufferedReader in = new BufferedReader(
       new FileReader(filename));

  while(in.readLine() != null) { lineCount++; }

  in.close(); // Wird immer ausgefuehrt?
}
catch (IOException e2) {
  System.err.println(e2.getMessage());
}
catch (FileNotFoundException e1) { // Already been caught?
  System.err.println(e1.getMessage());
}
```

Aufgabe 2: BufferedReader cont'd

Datei einlesen - 2. Versuch

```
try {
  BufferedReader in = null;
  try {
    in = new BufferedReader(new FileReader(filename));
    while(in.readLine() != null) { lineCount++; }
  }
  finally {
    if(in != null) { in.close(); }
  }
}
catch (IOException e) {
  System.err.println(e.getMessage());
}
```

Aufgabe 3: Datei lesen und schreiben

```
in1 = new BufferedReader(new FileReader(inFilename1));
in2 = new BufferedReader(new FileReader(inFilename2));
out = new BufferedWriter(new FileWriter(outFilename));

while(true) {
  String line1 = in1.readLine();
  String line2 = in2.readLine();

  if(line1 == null || line2 == null) { break; }

  if( line1.equals(line2) ) {
    out.write(line1);
    out.newLine();
  }
}
```

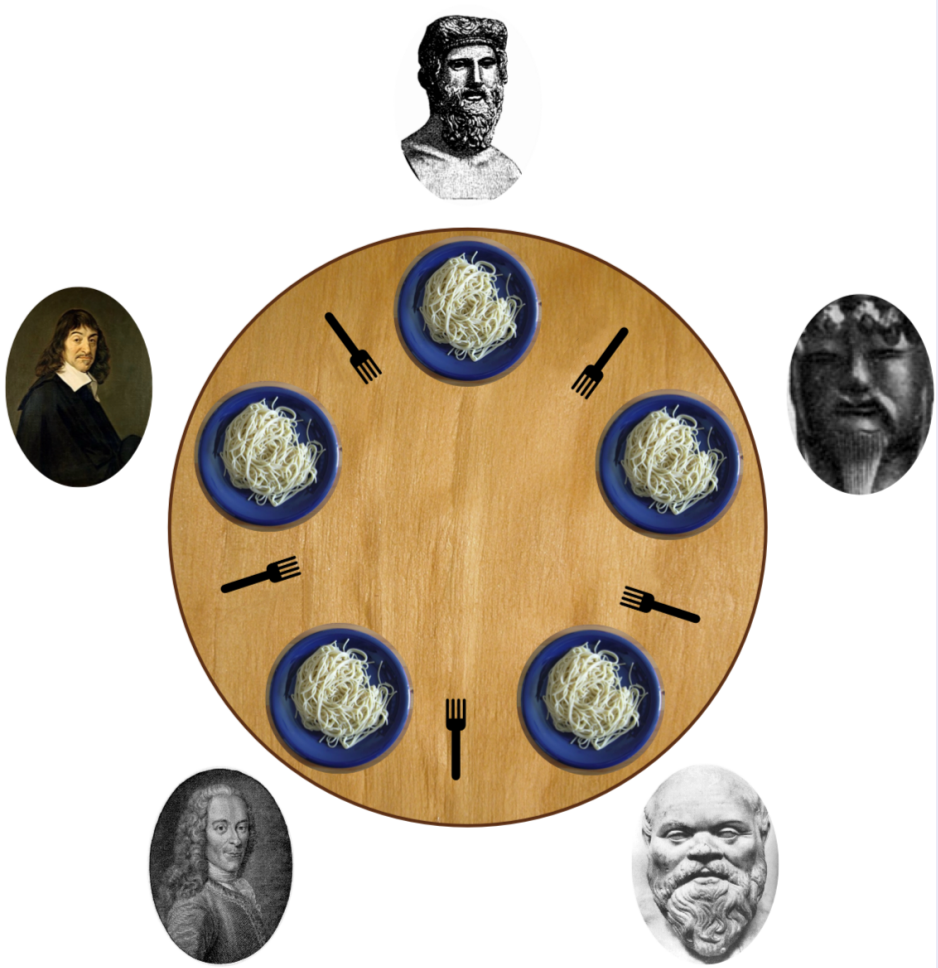
Aufgabe 3: Datei lesen und schreiben (ohne break)

```
in1 = new BufferedReader(new FileReader(inFilename1));
in2 = new BufferedReader(new FileReader(inFilename2));
out = new BufferedWriter(new FileWriter(outFilename));

String line1 = in1.readLine();
String line2 = in2.readLine();

while(line1 != null && line2 != null) {
  if( line1.equals(line2) ) {
    out.write(line1);
    out.newLine();
  }
  String line1 = in1.readLine();
  String line2 = in2.readLine();
}
```

Aufgabe 3: Fehlerbehandlung

```
BufferedReader in1 = null;
BufferedReader in2 = null;
BufferedWriter out = null;

try {
  try {
    /* ... siehe vorige Folie ... */
  }
  finally {
    if(in1 != null) in1.close();
    if(in2 != null) in2.close();
    if(out != null) out.close();
  }
}
catch (IOException e) { /*... */ }
```

Aufgabe 4: Philosopher's Problem



Benjamin D. Esham / Wikimedia Commons

- Tisch mit 5 Tellern, dazwischen je eine Gabel.
- 5 Philosophen
- Wenn Philosoph hungrig nimmt er linke, danach rechte Gabel und beginnt zu essen.

Aufgabe 4: Philosopher's Problem - Lösungen

- Mehr Gabeln.
- Kellner vergibt Gabeln
  - Strategien für Kellner?
- Reihenfolge, in der die Gabeln genommen werden, ändern
  - Nicht immer anwendbar
- Monitor: Philosophen essen, wenn Nachbarn nicht essen
  - Exclusive Lock auf Testen, ob Nachbarn essen, und Gabeln nehmen.
- Erlaube, dass Philosophen kommunizieren.