Exceptions

CoMa-Übung VI

TU Berlin

28.11.2012

Themen der Übung

1. Organisatorisches
2. Compilezeit- und Laufzeitfehler
3. Exceptions
4. Try-Catch-Finally

- Bewertung der OA 5 fehlerhaft, Madeleine arbeitet dran

Compilezeit- und Laufzeitfehler

Compilezeitfehler

Compilezeitfehler sind Fehler, die der Java-Kompiler (`javac`) beim Kompilieren eines Java-Programms findet.

- `int i = 1.0:`
- `int() array == null;`

Laufzeitfehler

Laufzeitfehler sind Fehler, die Java (`java`) beim Ausführen eines Java-Programms findet.

- `int i = 1/0;`

Andere Fehler

Es gibt Fehler, die weder der Java-Kompiler noch Java beim Ausführen findet. Solche Fehler sind weder Compilezeitzeit noch Laufzeitfehler.

- Endlosschleifen

Compilezeit- und Laufzeitfehler – Beispiele (1)

```
1  public class Fraction {
2
3    private long num;
4    private long denom;
5
6    public Fraction multiply(r) {
7      this.num = this.num * r.num
8      this.denom = this.denom * r.denom
9    }
10 }
```

Fehler

- Fehlende Kommas in Zeilen 7 & 8 → Compilefehler
- Fehlendes `return` / falscher Rückgabetyp → Compilefehler

Compilezeit- und Laufzeitfehler – Beispiele (2)

```
1  public class Fraction {
2
3    private long num;
4    private long denom;
5
6    public Void sum(Fraction r) {
7      num += r.num;
8      denom += r.denom;
9    }
10 }
```

Fehler

- `Void` ist kein Rückgabetyp → Compilefehler
- Falsche Implementierung → Weder noch

Compilezeit- und Laufzeitfehler – Beispiele (3)

```
1  public class HelloWorld {
2
3    public static void main(String[] args) {
4      double d = 0.0;
5      System.out.println((String) 1.0);
6      System.out.println((String) d);
7      System.out.println((int) Double.NaN);
8      System.out.println((long) (double) (long) Double.
           POSITIVE_INFINITY);
9    }
10 }
```

Fehler

- Zeilen 5 & 6 sind nicht konvertibel → Compilefehler
- Zeilen 7 & 8 funktionieren → Ergeben 0 und $2^{63} - 1$

Compilezeit- und Laufzeitfehler – Beispiele (4)

```
1  public class HelloWorld {
2
3      public static void main(String[] args) {
4          int[] lengths = new int[args.length];
5          int total;
6          for (int i=0; i<lengths.length; i++) {
7              lengths[i] = args.length;
8              total += lengths[i];
9          }
10         System.out.println("Durchschnitt: " + total / args
               .length);
11     }
12 }
```

Fehler

- Variable `total` nicht initialisiert → Compilefehler
- Potentielle `ArithmeticException` in Zeile 10 → Laufzeitfehler

Compilezeit- und Laufzeitfehler – Beispiele (5)

```
1  public class HelloWorld {
2
3      public static void main(String[] args) {
4          int[] lengths = new int[args.length];
5          double total = 0.0;
6          for (int i=0; i<=lengths.length; i++) {
7              total += args.length();
8          }
9          System.out.println("Durchschnitt: " + total / args
               .length);
10     }
11 }
```

Fehler

- Array hat keine `length()` Methode → Compilefehler
- `ArrayIndexOutOfBoundsException` in der Schleife → Laufzeitfehler

Compilezeit- und Laufzeitfehler – Beispiele (6)

```
1  public class HelloWorld {
2
3      int[] numbers2;
4
5      public static void main(String[] args) {
6          int[] numbers = new int[args.length];
7          for (int i=0; i<numbers.length; i++) {
8              numbers = Integer.parseInt(args);
9              numbers2 = numbers * numbers;
10         }
11     }
12 }
```

Fehler

- `Integer.parseInt` funktioniert nicht mit Arrays → Compilefehler
- `numbers2` nicht statisch → Compilefehler
- `*` funktioniert nicht mit Arrays → Compilefehler

Compilezeit- und Laufzeitfehler – Beispiele (7)

```
1  public class HelloWorld {
2
3      static int[] numbers2;
4
5      public static void main(String[] args) {
6          int[] numbers = new int[args.length];
7          for (int i=0; i<numbers.length; i++) {
8              numbers[i] = Integer.parseInt(args[i]);
9              numbers2[i] = numbers[i] * numbers[i];
10         }
11     }
12 }
```

Fehler

- `NumberFormatException` in Zeile 8 möglich → Laufzeitfehler
- `NullPointerException` in Zeile 9 möglich → Laufzeitfehler

Exceptions

Exceptions

Exceptions sind Javas Konzept, um mit Fehlern und außergewöhnlichen Situationen umzugehen.

```
1  try {
2    // Code, der eine Exception ausloesen kann
3  } catch ( ... ) {
4    // Code zum Behandeln der Exception
5  }
6  // Es geht normal weiter, die Exception wurde behandelt
```

try-catch-finally

- Exceptions werden in `try-catch-finally` behandelt
- Erlaubt das Behandeln von Ausnahmen (`catch`)
- Erlaubt garantierte Aufräumarbeiten (`finally`)

Exceptions (2)

```
1  String stringToConvert = "42%";
2  try {
3    Integer.parseInt(stringToConvert);
4  } catch (NumberFormatException e) {
5    System.out.println(stringToConvert + " kann nicht in
         eine Zahl konvertiert werden!");
6  }
```

Exceptions und Javadocs

Werden in einer Methode Exceptions ausgelöst, sollten sie im Javadoc dokumentiert sein. Das ist für die offiziellen Java-Klassen auch der Fall, wie etwa für `Integer.parseInt`.

Exceptions und Laufzeitfehler

Hinweis: Tritt eine *Exception* auf, liegt ein Laufzeitfehler vor.

Exceptions (3)

```
String stringToConvert = "42%";
try {
  Integer.parseInt(stringToConvert);
} catch (NumberFormatException e) {
  System.out.println(stringToConvert + " kann nicht in
      eine Zahl konvertiert werden!");
}
```

Exceptions – Ablauf

Tritt eine Exception auf,

- wird ein Exception-Objekt erzeugt,
- die Ausführung des Codes unterbrochen,
- die erste passende Catch-Klausel gesucht,
- und dort mit der Code-Ausführung fortgefahren.

Nach Beenden der Catch-Klausel werden alle anderen Catch-Klauseln übersprungen. Tritt keine Exception auf, werden Catch-Klauseln nicht ausgeführt.

Exceptions (4)

```
String stringToConvert = "42%";
try {
  Integer.parseInt(stringToConvert);
} catch (NumberFormatException e) {
  System.out.println(stringToConvert + " kann nicht in
      eine Zahl konvertiert werden!");
}
```

Exceptions – Eigenschaften

Ein Exception-Objekt e hat,

- eine Klasse mit einem Namen `e.getClass().getName()`,
- eine Nachricht `e.getMessage()`,
- eine Möglichkeit, den Stack-Trace auf der Konsole auszugeben (`e.printStackTrace();`).

Wird eine Exception nicht gefangen, gibt Java alles drei auf der Konsole aus.

Exceptions (5)

```
try {

} catch (...) {

} catch (...)  {

}
```

Exceptions – Code-Wiederholungen

Ist eine Exception aufgetreten und in einem Catch-Block behandelt worden, gibt es keine Möglichkeit, automatisch an die Stelle zurückzuspringen, wo die Exception aufgetreten ist.

Exceptions – Mehrere Catch-Blöcke

Zu einem `try` können mehrere Catch-Blöcke gehören, wenn unterschiedliche Exceptions behandelt werden müssen.

Exceptions (6)

Exceptions – `throws`

Kann eine Methode eine Exception werfen, so wird das üblicherweise per `throws` in der Methodendeklaration angegeben.

```
int divide(int a, int b) throws ArithmeticException {
    return a / b;
}
```

`try-catch-finally`

Müssen bestimmte Aufräumarbeiten in jedem Fall durchgeführt werden, kann ein Try-Catch-Block mittels eines Finally-Blocks ergänzt werden.

- `finally` ist optional.
- Maximal 1x `finally` pro `try`.
- `finally`-Code wird immer ausgeführt, egal, ob der `try`-Block durch `return`, eine Exception oder normal beendet wurde.

Exceptions (7)

```
try {

} catch (...) {

} finally  {

}
```

Hinweis

- `try-finally` kann auch ohne `catch` benutzt werden.
- Wird oft im Umgang mit Dateien verwendet.

```
try {

} finally  {

}
```


Exceptions (7)

```
try {
  return 1;
} finally  {
  return 0;
}
```

Warnung

- `try-finally` behandelt keine Exceptions, die würden an den Aufrufer weitergereicht.
- Behandelt niemand die Exception, wird das Programm beendet.
- In `catch` & `finally` Blöcken können neue Exceptions geworfen werden – dann sind eventuell verschachtelte `try`-Blöcke erforderlich.
- Ein `return` im `finally`-Block überschreibt ein `return` im `try`-Block.

Exceptions (8)

Eigene Exceptions

Eigene Exceptions werden mittels `throw` ausgelöst.

- Es ist sinnvoll, direkt am Anfang einer Methode die Eingaben zu testen.
- → je schneller Fehler gefunden werden, desto besser (*fail-fast*).

```
double sqrt(double d) {
  if (d < 0 ) {
    throw new IllegalArgumentException("Keine negativen
        Zahlen erlaubt!");
  }
  return Math.sqrt(d);
}
```