

Tutorium Programmieren

Tut Nr.11: Exceptions, java.util

Michael Friedrich | 21. / 23.11.2013

INSTITUT FÜR THEORETISCHE INFORMATIK



- 1 Exceptions
 - generelle Fehlerbehandlung
 - try-catch-finally
 - Exceptions vs Errors

- 2 java.util
 - Theorie: dynamische Programmierung

Wichtig bei:

- Öffnen einer Datei, welche nicht existiert
- Zugriff auf Array-Element mit ungültigem Index
- Division durch Null
- ...

Wichtig bei:

- Öffnen einer Datei, welche nicht existiert
- Zugriff auf Array-Element mit ungültigem Index
- Division durch Null
- ...

Probleme:

- Fehler abfangen, weiterleiten, behandeln?
- Wie Programmfluss dadurch nicht stören?

Eine Möglichkeit: Prüfen mittels if-Abfragen

```
1 public int positivMultiply(int x, int y) {  
2     if ((x>0 && y<0) || (x<0 && y>0)) {  
3         System.out.println("Fehler: ein Argument ist negativ");  
4         return -1;  
5     } else {  
6         return x*y;  
7     }  
8 }  
9
```

Eine Möglichkeit: Prüfen mittels if-Abfragen

```
1 public int positivMultiply(int x, int y) {  
2     if ((x>0 && y<0) || (x<0 && y>0)) {  
3         System.out.println("Fehler: ein Argument ist negativ");  
4         return -1;  
5     } else {  
6         return x*y;  
7     }  
8 }  
9
```

Funktioniert, aber:

- Stört Kontrollfluss
 - evtl. Ausgaben, wo man sie nicht haben will bzw. sollte!
- ⇒ Vermischung von Logik und Fehlerbehandlung

Exceptions

- Werden in Fehlersituationen zur Laufzeit ausgelöst („geworfen“)
- Können in einem übergeordneten Block abgefangen und behandelt werden
- Unterbrechen den Kontrollfluss

Vorteile:

- + Fehler muss nicht lokal behandelt werden, sondern kann weitergereicht werden
- + saubere Trennung von Logik und Fehlerbehandlung

- Sind Objekte (*Exception* und Unterklassen)
⇒ werden mit `new` erzeugt

- Sind Objekte (*Exception* und Unterklassen)
⇒ werden mit `new` erzeugt
- werden mit `throw` geworfen

- Sind Objekte (*Exception* und Unterklassen)
⇒ werden mit `new` erzeugt
- werden mit `throw` geworfen
- Methode, welche den Fehler erzeugt (wirft), wird mit `throws` gekennzeichnet

```
public void methode(..) throws IllegalArgumentException
```

- Sind Objekte (*Exception* und Unterklassen)
⇒ werden mit `new` erzeugt
- werden mit `throw` geworfen
- Methode, welche den Fehler erzeugt (wirft), wird mit `throws` gekennzeichnet

```
public void methode(..) throws IllegalArgumentException
```

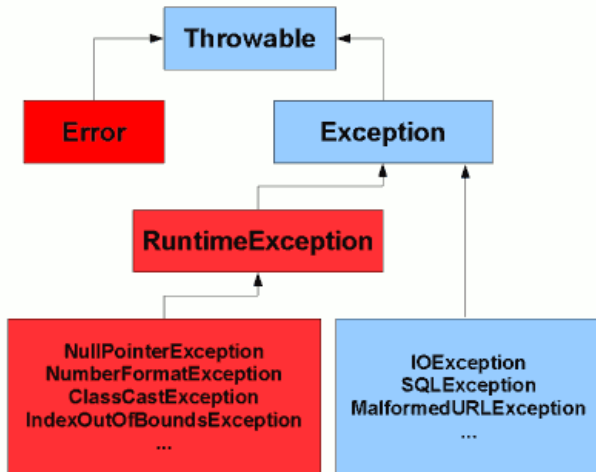
- Wird die Exception nicht in der Methode selbst gecatched, so wird sie an die aufrufende Methode weitergeleitet

- Sind Objekte (*Exception* und Unterklassen)
⇒ werden mit `new` erzeugt
- werden mit `throw` geworfen
- Methode, welche den Fehler erzeugt (wirft), wird mit `throws` gekennzeichnet

```
public void methode(..) throws IllegalArgumentException
```

- Wird die Exception nicht in der Methode selbst gecatched, so wird sie an die aufrufende Methode weitergeleitet
- Viele Exceptions in Java-API vorhanden, eigene sind möglich
- Java-Doc dokumentation mittels `@throws`

Exceptions: Vererbungshierarchie



Exceptions: Verwendung

```
1 public int positivMultiply(int x, int y) throws  
    IllegalArgumentException {  
2     if ((x>0 && y<0) || (x<0 && y>0)) {  
3         throw new IllegalArgumentException();  
4     }  
5     return x * y;  
6 }  
7
```

```
1 public int positivMultiply(int x, int y) throws  
    IllegalArgumentException {  
2     if ((x>0 && y<0) || (x<0 && y>0)) {  
3         throw new IllegalArgumentException();  
4     }  
5     return x * y;  
6 }  
7
```

An **anderer (aufrufender) Stelle**, wo der Fehler behandelt werden soll:

```
1 try {  
2     int z = positivMultiply(x,y);  
3 } catch (IllegalArgumentException e) {  
4     System.out.println("Fehler: ein Argument ist negativ");  
5 }  
6
```

For when you just Gotta Catch 'Em All



```
1  try {  
2      ...  
3  } catch (Exception e) {  
4      // Gotcha!  
5  }  
6
```


RuntimeException

- Programmierfehler, der zur Laufzeit festgestellt wird, teilweise behandelbar
- *NullPointerException, ArrayIndexOutOfBoundsException*

Errors

- kritische Fehler bei Programmausführung
- Nicht behandelbar!
- *VirtualMachineError, OutOfMemoryError*

sonstige Exceptions

- behandelbare Programmfehler, sollte man auf jeden Fall abfangen

optionaler Block, welcher auf jeden Fall ausgeführt wird.

```
1 try {  
2     openFile();  
3 } catch (IOException e) {  
4     printError("File couldn't be read!");  
5 } finally {  
6     closeFile();  
7 }  
8
```

```
1 public static void main(String[] args) {
2     String input = "";
3     BufferedReader buf = new BufferedReader( new
4         InputStreamReader(System.in));
5     while(!input.equals("quit")) {
6         try {
7             input = buf.readLine();
8         } catch (IOException e1) {
9             System.out.println("unable to read - shutting down...");
10            input = "quit";
11        }
12        try {
13            int i = Integer.parseInt(input);
14        } catch (NumberFormatException e) {
15            System.out.println("Eingabe war keine Zahl!");
16        }
17    }
18    System.out.println("shutting down...");
19 }
```

Was war bei dem Beispiel wichtig?!

Was war bei dem Beispiel wichtig?!

- Programm stürzt nie unkontrolliert ab

Was war bei dem Beispiel wichtig?!

- Programm stürzt nie unkontrolliert ab
- User wird über seine Fehler informiert und kann diese verbessern

Was war bei dem Beispiel wichtig?!

- Programm stürzt nie unkontrolliert ab
- User wird über seine Fehler informiert und kann diese verbessern

Was war bei dem Beispiel wichtig?!

- Programm stürzt nie unkontrolliert ab
- User wird über seine Fehler informiert und kann diese verbessern

Wir als Programmierer müssen unfähigen User (und Kollegen...) entgegen arbeiten.

Beispiel?

Was war bei dem Beispiel wichtig?!

- Programm stürzt nie unkontrolliert ab
- User wird über seine Fehler informiert und kann diese verbessern

Wir als Programmierer müssen unfähigen User (und Kollegen...) entgegen arbeiten.

Beispiel? NullPointerException abfangen, falsche Werte geliefert, falsche Formattierung...

Java bietet von sich aus schon sehr viel Funktionalität, z.Bsp LinkedList.

Java bietet von sich aus schon sehr viel Funktionalität, z.Bsp LinkedList.

Example

weitere eingebaute Datenstrukturen

Java bietet von sich aus schon sehr viel Funktionalität, z.Bsp LinkedList.

Example

weitere eingebaute Datenstrukturen

- Collections: ungeordneter Pool an Objekten

- `Collection<Product> products;`

Java bietet von sich aus schon sehr viel Funktionalität, z.Bsp LinkedList.

Example

weitere eingebaute Datenstrukturen

- Collections: ungeordneter Pool an Objekten

- `Collection<Product> products;`

- SortedSet: Menge mit totaler Ordnung

- `SortedSet<Product> products;`

Java bietet von sich aus schon sehr viel Funktionalität, z.Bsp LinkedList.

Example

weitere eingebaute Datenstrukturen

- Collections: ungeordneter Pool an Objekten
 - `Collection<Product> products;`
- SortedSet: Menge mit totaler Ordnung
 - `SortedSet<Product> products;`
 - Product MUSS hier Comparable<Product> implementieren

Java bietet von sich aus schon sehr viel Funktionalität, z.Bsp LinkedList.

Example

weitere eingebaute Datenstrukturen

- Collections: ungeordneter Pool an Objekten
 - `Collection<Product> products;`
- SortedSet: Menge mit totaler Ordnung
 - `SortedSet<Product> products;`
 - Product MUSS hier Comparable<Product> implementieren
- ArrayList: ähnlich LinkedList, aber mit Index
 - `ArrayList<Product> products;`

Java bietet von sich aus schon sehr viel Funktionalität, z.Bsp LinkedList.

Example

weitere eingebaute Datenstrukturen

- Collections: ungeordneter Pool an Objekten
 - `Collection<Product> products;`
- SortedSet: Menge mit totaler Ordnung
 - `SortedSet<Product> products;`
 - Product MUSS hier Comparable<Product> implementieren
- ArrayList: ähnlich LinkedList, aber mit Index
 - `ArrayList<Product> products;`
- Maps: key-value Paare
 - `TreeMap<Product, Customer> orders;`
 - `HashMap<Product, Customer> orders;`

Nehmt die Hinweise auf dem Übungsblatt als Einstiegspunkt.

Was ist das?

- **kein** Programmierparadigma
- sondern:
 - Methode zum Lösen von Optimierungsproblemen
 - benutzt "Divide and Conquer"
 - wichtig: Gesamtproblem muss ich aus kleineren Teilproblemen zusammensetzen lassen!
 - speichere Werte kleinerer Probleme in Tabelle ab

⇒ Begegnet euch wieder in Algorithmen 1 (2. Semester)

Levenshtein-Distanz

Die **Levenshtein-Distanz** (auch **Editierdistanz**) zwischen zwei Zeichenketten ist die minimale Anzahl von **Einfüge**-, **Lösch**- und **Ersetz**-Operationen, um die erste Zeichenkette in die zweite umzuwandeln

Levenshtein-Distanz

Die **Levenshtein-Distanz** (auch **Editierdistanz**) zwischen zwei Zeichenketten ist die minimale Anzahl von **Einfüge**-, **Lösch**- und **Ersetz**-Operationen, um die erste Zeichenkette in die zweite umzuwandeln

Beispiel: Levenshtein-Distanz von „Tier“ und „Tor“

- 1 „Tier“
- 2 „Toer“ (Ersetze i durch o)
- 3 „Tor“ (Lösche e)

Levenshtein-Distanz

Die **Levenshtein-Distanz** (auch **Editierdistanz**) zwischen zwei Zeichenketten ist die minimale Anzahl von **Einfüge**-, **Lösch**- und **Ersetz**-Operationen, um die erste Zeichenkette in die zweite umzuwandeln

Beispiel: Levenshtein-Distanz von „Tier“ und „Tor“

- 1 „Tier“
- 2 „Toer“ (Ersetze i durch o)
- 3 „Tor“ (Lösche e)

elegant zu lösen mittels dynamischer Programmierung!

DP: Beispielaufgabe

Eingabe: Zeichenketten u und v

Matrizeinträge $D_{i,j}$ speichern die Distanz der Präfixe $u_{0,i}$ und $v_{0,j}$.

Die Distanz-Werte berechnen sich wie folgt:

$$m = |u|$$

$$n = |v|$$

$$D_{0,0} = 0$$

$$D_{i,0} = i, 1 \leq i \leq m$$

$$D_{0,j} = j, 1 \leq j \leq n$$

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} & +0 \text{ falls } u_i = v_j \\ D_{i-1,j-1} & +1 \text{ (Ersetzung)} \\ D_{i,j-1} & +1 \text{ (Einfügung)} \\ D_{i-1,j} & +1 \text{ (Löschung)} \end{cases}$$
$$1 \leq i \leq m, 1 \leq j \leq n$$