Implementierung von Exception Handlern in Java

Abnormale Ereignisse, die während des Programmablaufs auftreten, brechen die normale Ausführung eines Programms ab und bringen es zum Absturz. Diese abnormalen Ereignisse werden **Ausnahmen** bzw. **Exceptions** genannt.

In Java werden Ausnahmen, die vom Programmierer zwingend behandelt werden müssen (**Checked Exceptions**) von solchen unterschieden, die nicht zwingend behandelt werden müssen (**Unchecked Exceptions**)

Bei Unchecked Exceptions wird der Programmierer vom Compiler nicht darauf hingewiesen, dass er sich um die Exceptions kümmern muss. Bei Checked Exceptions wird zur Kompilierzeit überprüft, ob dieses vom Programmierer behandelt werden.

Führt die Ausführung eines Programmes zu einer Situation, in der eine reguläre Weiterarbeit nicht sinnvoll ist, z.B. bei einer Division durch Null, dann liegt eine Ausnahme Situation (exception) vor. In Java wird in einer solchen Situation eine Exception ausgelöst. Der Programmverlauf verzweigt von der Stelle, an der das Ausnahmeereignis aufgetreten ist, zu einer vom Programmierer anzugebenden Stelle.

Dazu werden kritische Befehle in einem try-Block (Exception Area) zusammengefasst. Führt einer der Befehle zu einem Fehler, wird der Block verlassen, um die Aufräum- bzw. Wiederbelebungsmassnahmen zu veranlassen.

Das Werfen (**throw**) von Exceptions ist jedoch nur innerhalb einer **Exception Area** möglich

Zu jedem **try-Block** gehört mindestens ein catch- oder ein **finally-Block**.

Tritt nun in diesem Programmstück ein Fehler auf, d.h. wird darin eine Exception geworfen, so verzweigt der Programmablauf unmittelbar zur ersten catch-Anweisung, die diese Exception behandelt.

Weiterleiten von Exceptions mit *throws*

Entdeckt der Compiler eine Funktion, die bei einem Fehler eine **Exception** auslösen könnte, ohne dass eine entsprechende **try/catch-Konstruktion** bereitgestellt wird, weist er den Benutzer durch folgende Fehlermeldung darauf hin:
unreported exception <exceptionname>; must be caught or decleared to be thrown

Ein Fehler kann aber zur Behandlung "nach oben" weitergereicht werden.
In der Deklaration einer Klassenmethode muss dazu angegeben werden, welche Fehler innerhalb der Funktion auftreten können. Dadurch kann die Bearbeitung der Fehler an den Aufrufer der Methode delegiert werden:

```
public static void main(String[] args)throws IOException
 {
  System.out.println("Text eingeben und mit Return abschliessen");
  BufferedReader in = new BufferedReader(new
   InputStreamReader(System.in));
  String s;
  while(( s = in.readLine()).length() != 0) {
   System.out.println(s);

  } /* end while */
  System.out.println("Programm Excep1 wird beendet");
 }
}
```

Bsp:1 IOException wird weitergeleitet

Behandlung einer Exception innerhalb des Programms

```
public static void main(String[] args)
{
 try {
  System.out.println("Text eingeben und mit Return abschliessen");
  BufferedReader in = new BufferedReader(new
   InputStreamReader(System.in));
  String s;
  while(( s = in.readLine()).length() != 0) {
   System.out.println(s);

  } /* end while */
  System.out.println("Programm Excep1 wird beendet");
 }
 catch ( IOException e ) {
    System.out.println("IOException wurde ausgelöst");
    e.printStackTrace
 }
 finally {
 System.out.println("Programm Excep1 ist beendet");
 }
}
```

Bsp:2 IOException wird mit try-catch-Block abgefangen

Die grundlegende Anweisung für das Exception-Handling ist die *try-catch* Anweisung

```
FileInputStream fis = null;
try {

 …
 fis = new FileInputStream ("Datei");
 …
}
```

try-Block
treten Innerhalb dieses Blockes Fehler auf, wird er verlassen und durch den entsprechenden catch-Block abgefangen

```
...
```

```
catch ( IOException e ) {

System.out.print( "Kann Datei nicht öffnen");
}
```

catch-**Block**

fängt die IOException ab

```
...
```

```
catch (<ExceptionKlassenname> <ExceptionObjekt>){

System.out.print( "Kann Datei nicht öffnen");
}
```

Beispiel für weiteren

catch-**Block**

```
finally {

   fis.close();
}
```

finally-**Block**

wird immer ausgeführt

Welche Exceptions von Klassen aus der Java-Bibliothek geworfen werden, gehört mit zu den Abklärungen, die beim Design des Programms gemacht werden müssen. Informationen dazu finden Sie in der Java Dokumentation. Z.B. java.lang.exceptions

Auslösen von Exception mit throw-Anweisung

Um eine Ausnahmebedingung zu signalisieren, muss eine throw-Anweisung verwendet werden, die mit einem neu erzeugten **ExceptionObjekt** die Art der Ausnahmebedingung kennzeichnet.

```
throw <ExceptionObjekt>;
```

```
Beispiel:
```

```
if ( x == 0.0f )
   throw new MathematikException( " Division durch Null in einsDurchX! ");
```

Im Folgenden wird ein Auszug aus der bekannten Problemstellung unseres Stacks gezeigt. Ein spezielles **ExceptionObjekt** wird ausgelöst, wenn ein Stack Overflow – Ereignis eintrifft.

```
public class StackException extends Exception
{

    public StackException() {
      super("StackException");
    }
    public StackException(String s) {
      super(s);
    }

}
```

```
public class StackOverflowException extends StackException
{

    public StackOverflowException()
    {
        super("StackOverflowException");
    }
    public StackOverflowException(String s)
    {
        super(s);
    }

}
```

```
// setzt ein Element auf den Stack
public void push ( int value ) throws StackOverflowException {
  if ( full() ){
   throw new StackOverflowException("Overflow");
  }else {
   stack[top] = value;
   top++;
  } // end if
 } // end push
```

Quelle: Java als erste Programmiersprache, Cornelia Heinisch, 7. Auflage S 539

Dominik Waldvogel, 7. Dezember 2014