Java – Fehlerbehandlung

Bislang werden bei Fehlern sogenannte „Exceptions“ ausgegeben, mit denen der Benutzer meist wenig anfangen kann. Java kann angewiesen werden zu versuchen, den gewünschten Programmteil auszuführen. Sollten bestimmte Fehler dabei aufkommen, werden diese abgefangen und eine entsprechende Java-Routine ausgeführt.

```
public class Dividieren
{
  public static void main(String args[])
  {
    int a=7;
    int b=0;

    System.out.println("a : b = "+ a / b);
  }
}
```

Der Aufruf dieses kleinen Programms erzeugt natürlich eine Fehlermeldung (Division durch 0 nicht möglich):

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Dividieren.main(Dividieren.java:9)
```

Ein Benutzer kann mit dieser Fehlermeldung meist wenig anfangen, eventuell erkennt er den Teil “by zero” und kann sich das Problem denken, eventuell liest er die Meldung auch gar nicht richtig, da sie ihm zu kryptisch erscheint.

Es wäre also wünschenswert, wenn wir als Entwickler diese Fehlermeldungen unterdrücken und durch eine eigene, verständliche Meldung ersetzen könnten. Hierzu dient das Befehlspärchen `try` und `catch`.

Der Aufbau besteht aus `try` und `catch`, unter `try` steht das gewünschte Programm, sollte es zu Problemen kommen, können diese in `catch` abgefangen und eine Reaktion des Programms implementiert werden. In `catch` wird die genaue Exception als Fehlertyp aufgeführt, diese kann ggf. aus der Java-Meldung entnommen und an eine lokale Variable, z.B. `e`, übergeben werden. Innerhalb von `catch` können aus `e` weitere Informationen ausgelesen und ausgegeben werden. Die eigentliche Fehlermeldung (/ by zero) bekommt man z.B. per `e.getMessage()`.

Um verschiedene Exceptions abzufangen, können nacheinander mehrere catch-Befehle definiert werden, im Gegensatz zu `if`, `else if` und `else` wird aber immer einfach nacheinander `catch` mit geschweiften Klammern geschrieben.

Unser Divisions-Programm wird daher wie folgt geändert:

```
public class Dividieren
{
  public static void main(String args[])
  {
    int a=7;
    int b=0;

    try
    {
      System.out.println("a : b = " + a / b);
    }
    catch(ArithmeticException e)
    {
      System.out.println("Bei der Berechnung gab es ein Problem");
      System.out.println("Fehlerbeschreibung: "+e.getMessage());
    }
  }
}
```

Aufgabe
Entwickle ein Programm, das zwei ganzzahlige Eingaben entgegennimmt und die erste Zahl durch die zweite Zahl dividiert. Es sollen zwei Exceptions abgefangen und durch eine aussagekräftige Fehlermeldung ersetzt werden. Einmal die Möglichkeit, eine falsche Eingabe (Dezimalzahl, Buchstaben,…) vorzunehmen, sowie Fehler, die bei der Berechnung auftreten können.

Eigene Fehler „werfen“
Nicht immer reagiert Java so auf Probleme und Fehler, wie wir das gerne hätten. Daher gibt es die Möglichkeit, selbst eine Exception auszugeben, in Javasprache zu „werfen“ (`throw`). Das könnte z.B. so aussehen:

```
public static int berechneDivision(int a, int b){
    if (b == 0){
      throw new IllegalArgumentException("Der Nenner darf nicht Null
sein");
    }

    int c = a/b;

    return c;
  }
```

Dieser Programmcode führt (bei korrektem Aufruf) zu folgender Fehlermeldung bei Division durch 0:

```
Exception in thread "main" java.lang.IllegalArgumentException: Der Nenner
darf nicht Null sein
        at Division.berechneDivision(Division.java:10)
        at Division.main(Division.java:5)
```

Auch diesen Fehler könnte man wieder abfangen:

```
try{
   System.out.println(berechneDivision(10,0));
 }
 catch(IllegalArgumentException e){
   System.out.println("Fehler: " + e.getMessage());
 }
```

Die Meldung sähe dann so aus:

```
Fehler: Der Nenner darf nicht Null sein
```

Durch geeignetes „Werfen“ eigener Fehlermeldungen und Abfangen an passender Stelle können also kryptische Exceptions von Java verhindert und dem Benutzer eine klare Fehlermeldung angezeigt werden.