3.6 Ausnahmebehandlung

Unter einer **Ausnahme** verstehen wir eine Situation, bei der der normale Programmablauf nicht fortgeführt werden kann und daher unterbrochen wird. Man spricht auch von einem **Laufzeitfehler** oder einer **Exception**. Beispiele für typische Ausnahmen sind

- Division durch 0
- Zugriff auf Feldelemente mit undefiniertem Index
- Dereferenzierung eines Zeigers mit Wert `null`
- Öffnen einer nicht vorhandenen Datei

Zuverlässige Programme sollten auch in solchen Situationen nicht abstürzen! Dies gilt insbesondere für Programme zur Kraftwerkssteuerung, Flugzeugsteuerung etc., ist aber natürlich auch für andere Programme wünschenswert. Die Konsequenz ist, dass eine gute Programmiersprache Konstrukte zur Behandlung von Ausnahmen anbieten muss; solche wurden zuerst in PL/I eingeführt.

Das übliche Konzept zum Arbeiten mit Ausnahmen gliedert sich in zwei Teile:

Auslösung von Ausnahmen

- durch das Laufzeitsystem (Illegaler Feldindex, Division durch 0, ...),
- durch das Betriebssystem (Dateien, ...),
- durch den Programmierer (falsche Eingaben, ...),

mittels einer speziellen Anweisung „`raise <Fehlertyp>`“, z. B. könnte der Fehlertyp dann eine Division durch 0 beschreiben.

Behandlung von Ausnahmen durch

- einen speziellen Anweisungsteil in Blöcken oder Prozeduren (z. B. in Ada), der nur bei Ausnahmen in diesem Block oder dieser Prozedur ausgeführt wird, oder
- eine spezielle Anweisung mit Ausnahmebehandler (z. B. in Java). Das Prinzip ähnelt dem Prinzip bei einem Block, die Behandlung ist aber als Anweisung umgesetzt.

Der **Kontrollfluss** beim Auftreten einer Ausnahme ist wie folgt:

- Falls für die Anweisung/Block/Prozedur, in der die Ausnahme auftritt, ein Ausnahmebehandler vorhanden ist, werden die Anweisungen in diesem Behandler ausgeführt.
- Ansonsten wird dieser Block/Prozedur verlassen und die Ausnahme im nächsten umgebenden Block bzw. Prozedur ausgelöst. (evtl. bis zum Hauptprogramm, dies führt dann zu einem Programmabbruch).

Damit entspricht die operationelle Semantik in etwa einer Folge aus wiederholten `break`- und `raise`-Anweisungen, bis schließlich eine Behandlung erfolgt.

Eine weitere wichtige Frage ist die Stelle, an der nach der Ausnahmebehandlung weitergemacht werden soll. Hier bieten sich zwei Alternativen an:

Resumption Model: Fortführung an der Stelle, an der der Fehler auftrat (z. B. in PL/I). Hierbei ergibt sich allerdings ein Problem: Der Ausnahmebehandler kennt in der Regel nicht die exakte Stelle, wo der Fehler auftrat. Damit stellt sich die Frage, wie man *eine* Behandlung für unterschiedliche Ursachen erreicht. Außerdem ist es manchmal schwierig, den Fehler wirklich zu „reparieren“, um danach sinnvoll weiter zu machen. Daher ist heute das „termination model“ üblich.

Termination Model: Beende den aktuellen Block/Prozedur nach der Ausnahmebehandlung und mache im umgebenden Block bzw. in der aufrufenden Prozedur normal weiter.

Beispiel 3.15 (Ausnahmebehandlung in Ada)**.**

- Programmierer kann Ausnahmen definieren und auslösen
- Ausnahmebehandlung ist Teil jedes Blocks
- Behandlung spezieller Ausnahmetypen
- Folgt dem Termination Model

Beispielprogramm:

```
⋮
Invalid: exception; -- Deklaration einer Ausnahme
begin
  ...
   if Data<0 then raise Invalid; end if;
  ...
exception -- Beginn der Ausnahmebehandlung in diesem Block
  when Constraint_Error => Put ("Error - data out of range");
  when Invalid          => Put ("Error - negative value used");
  when others           => Put ("Some other error occured");
    -- others ist ein Schluesselwort (default-Behandlung)
end;
```

Beispiel 3.16 (Ausnahmebehandlung in Java)**.**

- Programmierer kann Ausnahmen (spezielle Objekte) deklarieren und auslösen

- Ausnahmebehandlung in spezieller Anweisung (`try`/`catch`/`finally`)
- spezielle Behandlung verschiedener Ausnahmetypen
- jede Prozedur (Methode) muss mögliche Ausnahmen behandeln oder explizit deklarieren, z. B.

  ```
  ... p(...) throws exception1, exception2, ...
  ```

  sonst: Compilerfehler (bis auf Standardausnahmen wie `RuntimeException`, d. h. arithmetische Fehler, Feldfehler etc.)
- Folgt dem Termination Model

Syntax der **Ausnahmebehandlung**:

```
try { <normale Berechnung> }
catch (ExceptionTypeA e1) { <Behandlungsblock dieses Ausnahmetyps,
                             wobei e1 Objekt der Ausnahme ist> }
catch (ExceptionTypeB e2) { <Behandlungsblock> }
⋮
finally { <Anweisungen, die immer (mit oder ohne Ausnahme)
           am Ende ausgefuehrt werden: Aufraeumanweisungen
           wie Schliessen von Dateien etc.> }
```

Entweder der `catch`- oder der `finally`-Teil können auch fehlen.

Auslösen von Ausnahmen mittels `throw`-Anweisung:

```
throw new IllegalArgumentException ("negative value");
```

Beispielprogramm

```
⋮
public static void main(String[] args) {
    int i;
    // Pruefe, ob Programm mit Integer-Argumenten aufgerufen wurde:
    try {
        i = Integer.parseInt(args[0]);
    } catch (ArrayIndexOutOfBoundsException _) {
        // Argument fehlt
        System.err.println ("Bitte Integer-Argument angeben!");
        System.exit(1);
    } catch (NumberFormatException _) {
        // Argument ist keine Zahl
```

```
            System.err.println ("Argument muss eine ganze Zahl sein!");
            System.exit(1);
        }
        p(i); // mache mit der normalen Verarbeitung weiter
    }
```