Programmieren I

Dr. Klaus Höppner

Hochschule Darmstadt – Wintersemester 2009/2010

Exceptions

Motivation

Beim Ablauf von Programmen können verschiedene Probleme auftreten:

- Fehler beim Öffnen von Dateien,
- Einlesen ungültiger Daten,
- Zugriff auf nicht vorhandene Elemente eines Feldes,
- unzulässige Casts
- Division durch Null

In einigen Programmiersprachen führen diese Probleme entweder zu unsinnigem Verhalten oder zum Absturz des Programms.

In Java ist es möglich, solche Probleme abzufangen und „geeignet“ zu behandeln.

Beispiel

Zur Illustration betrachten wir folgende einfache Klasse zur Berechnung des Mittelwerts einer Reihe:

```
public class Mittelwert {
    private int size = 0;
    private int[] werte = new int[10];
    public int getSize() { return size; }
    public void add(int wert) {
        werte[size++] = wert;
    }
    public double getAverage() {
        double summe = 0;
        for (int index=0; index<size; ++index) {
            summe += werte[index];
        }
        return summe/size;
    }
}
```


Beispiel (Forts.)

Diese Klasse kann man nun anwenden, um für mehrere Werte den Mittelwert zu berechnen:

```
public class Anwendung {
    public static void main(String[] args) {
        Mittelwert m = new Mittelwert();

        m.add(7);
        m.add(3);
        m.add(-17);
        m.add(10);

        System.out.println(m.getAverage());
    }
}
```

In diesem Beispiel erhält man als Lösung den Wert 0.75.

Mögliche Probleme

Im vorigen Beispiel funktioniert die Anwendung fehlerfrei.

Trotzdem können in anderen Anwendungen Probleme auftreten, die mit Ausnahmen zu tun haben, die beim Programmablauf auftreten können.

Dies sind hier folgende Probleme:

- Es wird versucht, eine Reihe von mehr als 10 Zahlen zu bilden.
- Es wird versucht, von einer leeren Reihe den Mittelwert zu berechnen.

Mögliche Lösung

Wie kann man diese Ausnahmesituationen abfangen?

Im Fall der Methode `add` könnte man z. B. einen Rückgabewert vom Typ **`boolean`** verwenden:

```
public boolean add(int wert) {
    if (size<10) {
        werte[size++] = wert;
        return true;
    } else {
        return false;
    }
}
```

Dieser Rückgabewert müsste jeweils geprüft werden, ob die Reihe schon „voll“ war.

Mögliche Lösung (Forts.)

Die Möglichkeit zur Rückgabe eine boolean-Wertes existiert für `getAverage` nicht. Hier könnte man folgende Lösung wählen:

```
public double getAverage() {
    if (size>0) {
        double summe = 0;
        for (int index=0; index<size; ++index) {
            summe += werte[index];
        }
        return summe/size;
    } else {
        return -9999;
    }
}
```


Analyse

In der Methode `getAverage` findet eine Ausnahmebehandlung statt:

Normalfall Der Mittelwert der Reihenelemente wird als Rückgabewert geliefert.

Ausnahmefall Die Reihe ist leer.

Behandlung Es wird der Wert -9999 zurück gegeben.

Problem Wie unterscheidet man bei Rückgabe von -9999, ob

- der Mittelwert zufällig den Wert -9999 hatte, oder
- die Reihe leer war

Exceptions in Java

Neben dem Rückgabewert von Methode existiert in Java ein weiterer Mechanismus, mit dem Methoden den aufrufenden Programmteil Nachrichten zukommen lassen können, in diesem Fall über auftretende Problemfälle:

- Methoden können so genannte *Exceptions* werden,
- die im aufrufenden Programmteil *gefangen* und geeignet *behandelt* werden können.

Werfen von Exceptions

Betrachten wir nun folgende neue Version von `getAverage`:

```
public double getAverage() throws Exception {
    if (size>0) {
        double summe = 0;
        for (int index=0; index<size; ++index) {
            summe += werte[index];
        }
        return summe/size;
    } else {
        throw new Exception("No values");
    }
}
```


Analyse

In der neuen Version wird nun eindeutig unterschieden zwischen dem

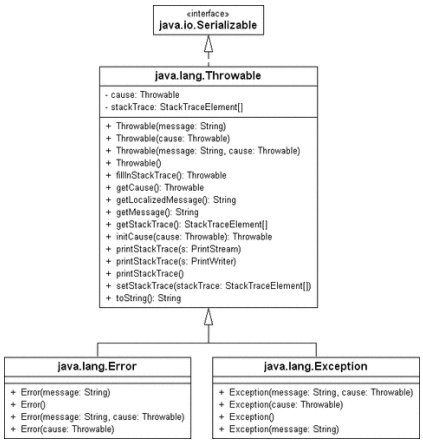
Normalfall in dem die Methode fehlerfrei abläuft und ein (sinnvoller) Wert zurück gegeben wird, und dem

Ausnahmefall einer leeren Reihe, in dem beim Versuch der Mittelwertbildung eine Exception geworfen wird.

Hierbei wird die Exception mit dem Schlüsselwort **`throw`** geworfen, wobei im Kopf der Methode deklariert ist, dass diese geworfen werden kann.

Die Klasse Exception

Die Klasse `Exception` erbt von der Klasse (kein Interface) `Throwable`:



Details

- Die Basisklasse der Hierarchie von Fehlern und Ausnahmen heißt `Throwable`, in der insbesondere die Methoden `getMessage()` und `printStackTrace()` implementiert sind (daher ist `Throwable` trotz des Namens kein Interface).
- Hiervon erben die Klassen `Error` und `Exception`.
- Das Auftreten von `Error` führt grundsätzlich zum Beenden des Programms.
- Objekte vom Typ `Exception` werden bei Bedarf gefangen und behandelt.

Fangen von Exceptions

Nachdem im Beispiel die Methode `getAverage` nun für eine leere Reihe eine Exception wirft, muss dies im aufrufenden Programmteil berücksichtigt werden:

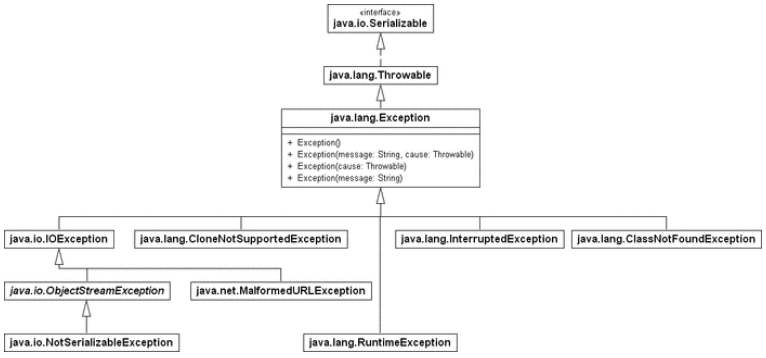
```
public class Anwendung {
    public static void main(String[] args) {
        Mittelwert m = new Mittelwert();
        m.add(7);
        m.add(3);
        m.add(-17);
        m.add(10);

        try {
            System.out.println(m.getAverage());
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

Fangen von Exceptions nach Typ

Die **`catch`**-Anweisung in einem **`try-catch`**-Block fängt eine Exception, wenn diese vom Typ her passt.
Dies bedeutet, dass Ausnahmesituationen unterschieden werden können, wenn diese Ausnahmen unterschiedlichen Typs (verschiedene Kindklassen von `Exception`) werfen.
Hierfür dürfen zu einem **`try`** *mehrere* **`catch`**-Anweisungen existieren, die dann unterschiedliche Ausnahmen verschieden behandeln.

Bei mehreren **`catch`**-Anweisungen gilt: Das erste passende **`catch`** gewinnt! Daher prüft man zuerst auf spezielle Exceptions und dann auf allgemeine (d. h. man beginnt mit den Kindklassen und endet bei der Basisklasse `Exception`).

Hierarchie von `Exception`



Exceptions müssen behandelt werden

In Java müssen Exceptions im Allgemeinen behandelt werden.
Dies bedeutet:

- Aufrufe von Methoden, die per Deklaration eine Exception werfen, müssen von einem try-catch-Block umschlossen sein, wobei ein catch-Statement existiert, das auf die Exception passt, *oder*
- die Methode, in der die fragliche Methode benutzt wird, muss ebenfalls per Deklaration diese Exception werfen können (oder eine Exception von einer Elternklasse).

Von dieser Regel gibt es eine wichtige Ausnahme, nämlich Exceptions vom Typ `RuntimeException` (oder einer Kindklasse davon). Diese müssen weder im Kopf einer Methode deklariert werden, noch müssen solche Ausnahmen zwingend behandelt werden.

Kindklassen von RuntimeException

ArithmeticException Ganzzahlige Division durch 0.

ArrayIndexOutOfBoundsExceptions Indexgrenzen bei einem Feld missachtet

ClassCastException Typumwandlung ist zur Laufzeit nicht möglich.

IllegalArgumentException Eine häufig verwendete Ausnahme, mit der Methoden falsche Argumente melden. Beispiel: `Integer.parseInt("Hallo");`

NullPointerException Meldet einen der häufigsten Programmierfehler, wenn versucht wird, für einen Nullpointer Methoden aufzurufen oder auf Attribute zuzugreifen.

Definieren eigener Exceptions

Da das Fangen von Exceptions nach Typ erfolgt, ist es häufig sinnvoll, eigene Exception-Klassen zu definieren.

Dies erlaubt dann, in einer Anwendung eine bestimmte Art auftretender Ausnahmen speziell abzufangen und zu behandeln.

Im Allgemeinen erben eigene Exception-Klassen entweder von `Exception` selber oder von `RuntimeError`. Das Ergebnis unterscheidet sich dann darin, dass im ersten Fall im aufrufenden Programmteil die entsprechenden Ausnahmen behandelt werden *müssen*, während sie im zweiten Fall auch ignoriert werden dürfen.

Beispiel

Im folgenden Beispiel wird eine neue Exceptionklasse definiert, die dann als spezieller Ausnahmetyp für die Methode `getAverage` verwendet werden kann.

```
public class NoValuesException extends Exception {

    public NoValuesException() {
        super();
    }

    public NoValuesException(String msg) {
        super(msg);
    }
}
```

Beispiel (Forts.)

Hierdurch ergibt sich in der Klasse `Mittelwert` folgende neue Version von `getAverage`:

```
public double getAverage() throws NoValuesException {
    if (size>0) {
        double summe = 0;
        for (int index=0; index<size; ++index) {
            summe += werte[index];
        }
        return summe/size;
    } else {
        throw new NoValuesException("No values");
    }
}
```


Beispiel (Forts.)

Dieser neue Ausnahmetyp kann nun speziell gefangen werden. Da der neue Typ von `Exception` erbt, würde das Fangen von `Exception` aber auch weiterhin funktionieren.

```
public class Anwendung {
    public static void main(String[] args) {
        Mittelwert m = new Mittelwert();
        m.add(7);
        m.add(3);
        m.add(-17);
        m.add(10);

        try {
            System.out.println(m.getAverage());
        } catch (NoValuesException e) {
            System.err.println(e.getMessage());
        }
    }
}
```