Fehler- und Ausnahmebehandlung

Fehlerhafte Programme

Programme können aus vielen Gründen unerwünschtes Verhalten zeigen:

- Fehler beim Entwurf, d.h. bei der Modellierung des Problems
- Fehler bei der Programmierung des Entwurfs
  - Programm bricht mit `NullPointerException` ab
  - Algorithmen falsch implementiert
  - ...
- Ungenügender Umgang mit außergewöhnlichen Situationen
  - fehlerhafte Benutzereingaben, z.B. Datum 31.11.2010
  - Abbruch der Netzwerkverbindung
  - Dateien können nicht gefunden werden
  - ...

Umgang mit außergewöhnlichen Situationen

Ausnahmesituationen unterscheiden sich von Programmierfehlern darin, dass man sie nicht (zumindest prinzipiell) von vornherein ausschließen kann.

Immer möglich sind zum Beispiel:

- unerwartete oder ungültige Eingaben
- Ein- und Ausgabe-Fehler beim Zugriff auf Dateien oder Netzwerk

Ausnahmen in Java

Erkennung und Behandlung eines Problems muss oft in ganz verschiedenen Teilen des Programms stattfinden.

Beispiel: Datei öffnen

- Erkennung: Konstruktor von `InputStream` erkennt, wenn eine Datei nicht gefunden werden kann.
- Behandlung: GUI soll ein Fenster öffnen, um den Benutzer nach Eingabe eines neuen Dateinamens zu fragen.

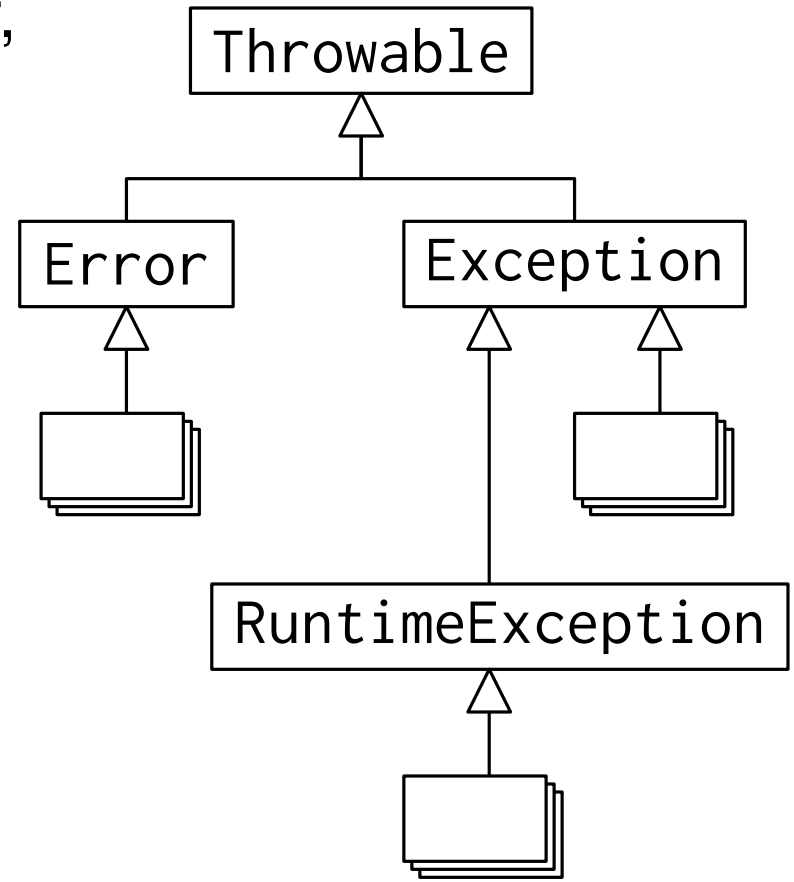
Exceptions (dt.: Ausnahmen) werden bei der Erkennung eines Problems ausgelöst und erlauben die Fehlerbehandlung an einer ganz andereren Stelle im Programm.

Ohne Exceptions muss man Informationen über Fehler „per Hand“ zurückgeben (z.B. Method gibt anstatt `void` eine Fehlernummer zurück, welche vom Aufrufer ausgewertet werden muss).

Laufzeitfehler und Ausnahmen in Java

In Java werden verschiedene Arten von Ausnahmen und Fehlern durch verschiedene Unterklassen von `Throwable` repräsentiert.

- Instanzen von `Error`:
  unbehandelbare unerwartete Fehler, z.B. Fehler der virtuellen Maschine
- Instanzen von `Exception`:
  vorhersehbare Ausnahmen, welche vom Programm behandelbar sind
- Instanzen von `RuntimeException`:
  Ausnahmen, die nicht behandelt werden müssen. Alle anderen Instanzen von `Exception` müssen behandelt werden.



Auslösen von Ausnahmen

Das Auslösen einer Ausnahme erfolgt mit der Anweisung

```
throw exp;
```

wobei `exp` ein Ausdruck vom Typ `Throwable` ist.

Möglichst existierende Ausnahmeklassen verwenden, z.B.:

- `NullPointerException`: Der Wert `null` ist aufgetreten, wo `null` nicht erwartet wurde.
- `IllegalArgumentException`: Eine Methode wurde mit unzulässigen Parametern aufgerufen.
- `IllegalStateException`: Eine Methode wurde auf einem Objekt aufgerufen, dass für diesen Methodenaufruf nicht im richtigen Zustand ist.

Eigene Ausnahmeklassen können durch Ableiten von `Exception` oder `RuntimeException` definiert werden.

Deklaration von Ausnahmen

Bei Methoden muss man möglicherweise auftretende Ausnahmen deklarieren *(checked)*:

Beispiele:

```
public void meth1() throws IOException {
  if (...) {
    throw new IOException();
  }
}

// Annotationen sagen nur, dass eine Ausnahme moeglicher-
// weise aufritt. Tatsaechlich kann sie auch nie auftreten.
public void meth2() throws IOException {
  System.out.println();
}
```

Für Unterklassen von `Error` und `RuntimeException` sind solche Deklarationen optional und auch unüblich *(unchecked)*.

Geprüfte Ausnahmen (checked exceptions)

Geprüfte Ausnahmen repräsentieren Ausnahmesituationen, mit denen gerechnet werden muss und auf die reagiert werden sollte.
Beispiel: `FileNotFoundException`, `IOException`

- Geprüfte Ausnahmen sind alle Unterklassen von `Exception`, die nicht auch Unterklassen von `RuntimeException` sind.
- Alle möglichweise auftretenden geprüfte Ausnahmen müssen entweder deklariert oder abgefangen werden.

Falls Methode `foo` eine Methode `bad` aufruft, welche möglichweise eine geprüft Ausnahme werfen kann, so muss `foo` ebenfalls die mögliche Ausnahme deklarieren oder abfangen.

Eine nicht-gefangene Ausnahme wird an den Aufrufer durchgereicht. Dadurch kann die Behandlung eines Problems an einer höheren Stelle erfolgen.

Ungeprüfte Ausnahmen (unchecked exceptions)

Ungeprüfte Ausnahmen (unchecked exceptions) repräsentieren Ausnahmesituationen, deren Ursache ein Programmierproblem ist.

Beispiele: `NullPointerException`, `IllegalArgumentException`, `IllegalStateException`, `ClassCastException`, `...`

- Alle Ausnahmeklassen, die von `RuntimeException` oder `Error` abgeleitet sind, sind ungeprüfte Ausnahmen.
- Ungeprüfte Ausnahmen müssen weder behandelt noch deklariert werden.

  Grund: Anstatt die Ausnahme zur Laufzeit zu behandeln, sollte das Programmierproblem behoben werden.

Abfangen von Ausnahmen

Behandlung von Ausnahmen:

```
try {
  // Block "normalen" Codes, in dem
  // eine Ausnahme auftreten koennte
} catch (ExceptionA ea) {
  // Ausnahmebehandlung fuer Fehler des Typs ExceptionA
} catch (ExceptionB eb) {
  // Behandlung fuer Fehler der Klasse ExceptionB
} finally {
  // Code, der in jedem Fall nach normalem
  // Ausnahmebehandlung ausgefuehrt werden
}
```

Jeder `catch`-Block kümmert sich um eine Java Ausnahme-Klasse.

Das mit `throw` geworfene Ausnahme-Objekt `ea` bzw. `eb` kann im `catch`-Block abgefragt werden, um weitere Information zu dem aufgetretenen Problem zu erhalten.

Fehlerbehandlung

Ein fertiges Programm sollte nicht mit einer Exception abbrechen.

- Geprüfte Exceptions sind an einer geeigneten Stelle mit `try` abzufangen und zu behandeln.
- Ausnahmesituationen müssen sinnvoll behandelt werden.
  Beispiele:
  - falsche Benutzereingabe
    $\Rightarrow$ neue Eingabeaufforderung
  - IO-Fehler, z.B. Netzwerkübertragung
    $\Rightarrow$ evtl. nochmal versuchen, aber nicht unendlich oft
  - nicht sinnvoll behandelbarer Fehler
    $\Rightarrow$ Benutzerdaten sichern, Programm beenden

Fehlerbehandlung

Ungeprüfte Ausnahmen, die Programmierfehler repräsentieren, werden nicht abgefangen.

Beispiele:

- `NullPointerException`: Der Wert `null` ist aufgetreten, wo `null` nicht erwartet wird.
- `IllegalArgumentException`: Eine Methode wurde mit unzulässigen Parametern aufgerufen.
- `ClassCastException`: Eine Typumwandlung `(C)e` ist fehlgeschlagen.

Die einzige sinnvolle Reaktion auf solche Exceptions ist das Programm zu korrigieren.
$\Rightarrow$ Kein Abfangen solcher Exceptions mit `try`.

Konvention

Öffentliche (`public`) Methoden überprüfen eventuelle Annahmen an ihre Parameter und lösen gegebenenfalls eine Exception aus.

(typisch: `IllegalArgumentException`, `IllegalStateException`, `NullPointerException`)

Beispiel:

```
/**
 * Spielzug waehlt Farbe einer Region
 *
 * @param region
 * Region, deren aktuelle Farbe gewaehlt wird.
 * Muss verschieden sein von aktuellen Spielerfarben.
 * @throws IllegalArgument Exception
 */
public void makeMoveRegion(Region region) {
  if (!farbeWaehlbar(region.getColor())) {
    throw new IllegalArgumentException("Farbe der Region ist nicht frei.");
  }
  // Spielzug durchfuehren...
}
```

Konvention

Öffentliche (`public`) Methoden überprüfen eventuelle Annahmen an ihre Parameter und lösen gegebenenfalls eine Exception aus.

Beispiel:

```
/**
 * Konstruiere Kartenknoten mit den gegebenen Koordinaten
 *
 * @param id eindeutiger Name fuer Node, nicht null
 * @param latitude Koordinate
 * @param longitude Koordinate
 * @throws IllegalArgument Exception
 */
public Node(String id, double latitude, double longitude) {
  if (id == null) {
    throw new NullPointerException();
  }
  this.id = id;
  this.longitude = longitude;
  this.latitude = latitude;
}
```

Allgemeine Hinweise zur Fehlerbehandlung

Bei der Behandlung von Ausnahmen möglichst die spezifischen Ausnahmen angeben.

Beispiel:

```
try {
  ...
} catch (IOException e) {
  ...
} catch (JSONException e) {
  ...
}
```

ist besser als ein „catch-all“

```
try {
  ...
} catch (Exception e) {
  ...
}
```

Allgemeine Hinweise zur Fehlerbehandlung

Ausnahmen sollten nicht ignoriert werden.

Beispiel:

```
try {
  ...
} catch (Exception e) { }
```

sollte nie ohne (dokumentierten!) guten Grund im Programm stehen.

Allgemeine Hinweise zur Fehlerbehandlung

Ausnahmen sollten nur in außergewöhnlichen Situationen ausgelöst werden.

Beispiel:

```
// So nicht!
// Exceptions nur in Ausnahmefaellen ausloesen!
try {
  Iterator<String> i = list.iterator();
  while (true) {
    String s = i.next();
    ...
  }
} catch (NoSuchElementException e) { }
```

Besser:

```
for (String s: list) {
  ...
}
```

Dokumentation

Ungeprüfte Exceptions werden üblicherweise nicht mit `throws` deklariert.

Die möglichen ungeprüften Exceptions sollten jedoch im Javadoc dokumentiert werden.

Beispiel:

```
/**
 * Returns the element at the specified position in this list.
 *
 * @param index index of the element to return
 * @return the element at the specified position in this list
 * @throws IndexOutOfBoundsException {@inheritDoc}
 */
public E get(int index) {
    rangeCheck(index);

    return elementData(index);
}
```

(Quelltext von `java.util.ArrayList`)

Definieren eigener Ausnahmen

Eigene Ausnahmen definiert man als Erbe von `Throwable`:

```
public class MeinFehler extends Exception {
    private final double wert1;
    private final boolean wert2;

    public MeinFehler(String msg, double w1, boolean w2) {
        super(msg);
        this.wert1 = w1;
        this.wert2 = w2;
    }
    public double getWert1() { return wert1; }
    public boolean isWert2() { return wert2; }
}
```

Im Normalfall von `Exception` erben; dessen `String` Attribut wird durch Aufruf des Ahnen-Konstruktors gesetzt.

Oft ist es sinnvoll, eine eigene Hierarchie von Ausnahmen zu definieren, um diese einheitlich abfangen zu können.

Verwenden eigener Ausnahmen

```
public static void main(String[] args) {
    try {
        trouble();
    } catch (MeinFehler fehl) {
        System.out.println("Fehler: " + fehl.getWert1()
                           + " und " + fehl.isWert2());
        System.out.println(fehl.getMessage());
    }
}
public static void trouble() throws MeinFehler {
    MeinFehler felr = new MeinFehler("Bah!", 6.9, true);
    System.out.println("Noch ist alles gut.");
    throw felr; // Jetzt wird der Fehler geworfen!
}
```

Ausgabe:

```
Noch ist alles gut.
Fehler ist: 6.9 und true
Bah!
```

Zusammenfassung

- Unterscheidung zwischen geprüften und ungeprüften Ausnahmen
- Geprüfte Ausnahmen abfangen oder mit `throws` deklarieren
- Ungeprüfte Ausnahmen, welche Programmierfehler repräsentieren, nicht abfangen
- Argumente in öffentlichen Methoden überprüfen und ggf. eine Ausnahme auslösen
- Mögliche Ausnahmen immer dokumentieren
- Ausnahmen möglichst spezifisch behandeln
- Ausnahmen nicht einfach nur ignorieren
- Ausnahmen nur in außergewöhnlichen Situationen verwenden