Einführung in die Informatik

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Sommersemester 2014

Teil VII

Erweiterte Programmkonstrukte

Code-Konventionen

Konventionen

Konventionen dienen dazu, den Code lesbarer zu machen. Dazu gehört

- richtige Einrückung,
- Zeilenumbrüche,
- konsequente Bennenung von Variablen/Klassen/Methoden,
- Kommentare, insbesondere JavaDoc-Kommentare.

Einrückung

Die generellen Regeln der Einrückung sind:

- Blöcke (durch { und } gekennzeichnet) werden eingerückt.
  Alle Blöcke werden immer gleichweit eingerückt.
- Bei mehrzeiligen Befehlen, werden die folgenden Zeilen eingerückt.

Bei größeren Projekten sollte abgesprochen werden,

- wie weit eingerückt wird (4 Spaces, 2 Spaces, 8 Spaces?),
- ob Tabs benutzt werden sollen und wie breit ein Tab ist,
- ob geschweifte Klammern in extra Zeilen kommen; werden sie eingerückt?

Viele Entwicklungsumgebungen können automatisch einrücken. Man muss allerdings dem Tool die Konventionen mitteilen.

Zeilenumbrüche

- Generell gilt, eine Anweisung pro Zeile.
- Auch bei kurzen Anweisungen, z.B. `i++;`
- Ein einzeiliger Schleifenrumpf oder Then/Else-Teil kommen immer in eine eigene Zeile.
- Lange Zeilen sollten zusätzlich umgebrochen werden.

Namenskonventionen

Die Groß-/Kleinschreibung von Bezeichnern soll helfen Typen von Variablen zu unterscheiden.

- Bezeichner in Java benutzen üblicherweise `DieCamelCaseKonvention`.
- Klassennamen haben großen Anfangsbuchstaben.
- Methoden- und Komponentennamen und Variablen haben kleinen Anfangsbuchstaben.
- Konstanten werden komplett großgeschrieben.
- Paketnamen werden komplett kleingeschrieben.
- Methodennamen fangen üblicherweise mit einem Verb an.
- Der Bezeichner sollte kurz, knapp und präzise sein.
- Je globaler ein Bezeichner ist, desto besser sollte er gewählt sein.

Aufzählungstypen

Aufzählungstypen (Enumerations)

Ein Aufzählungstyp kann nur endlich viele Werte annehmen.

Beispiel

```
public enum CardSuit = {
    HEARTS, DIAMONDS, CLUBS, SPADES;
}
```

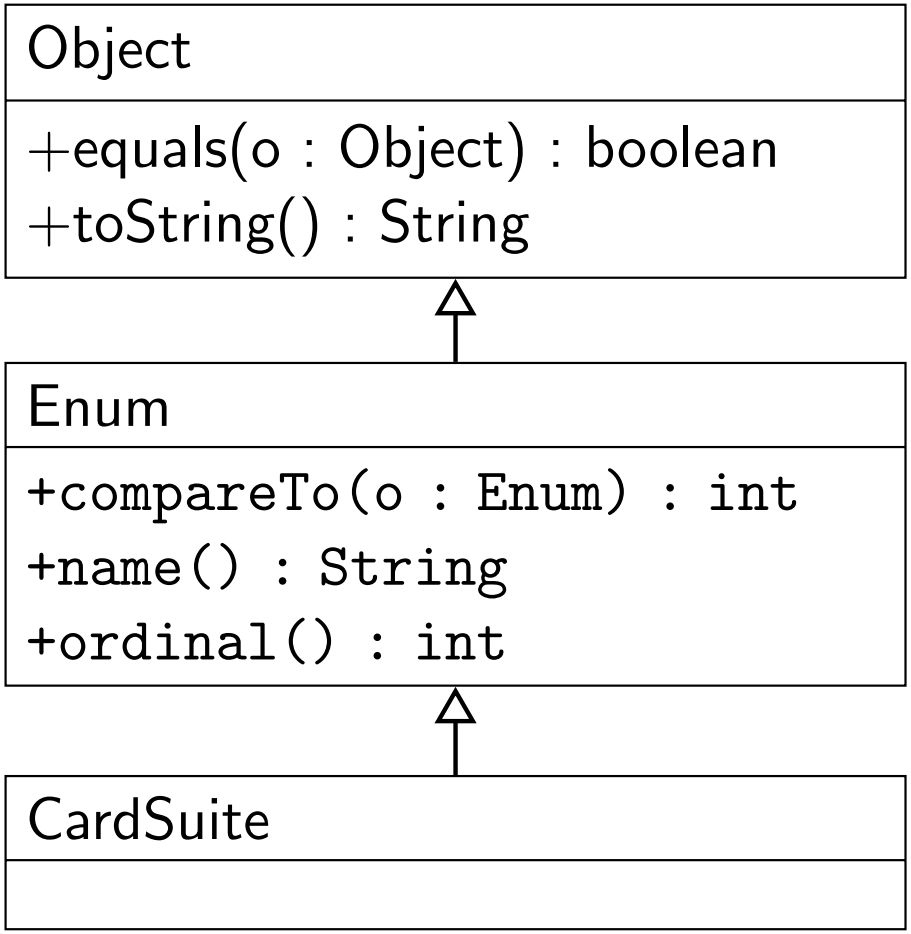
Ein Aufzählungstyp ist eine Klasse. Es gibt nur vier Objekte von dieser Klasse. Auf diese kann mit `CardSuit.HEARTS` ... zugegriffen werden.

Klassenhierarchie

Alle Aufzählungstypen erben von der Klasse `java.lang.Enum` die Methoden

- `ordinal()` gibt die Ordinalzahl zurück, 0 für die erste, usw.
- `name()` gibt den Namen zurück
- `toString()` gibt ebenfalls den Namen aus.

Aufzählungstypen haben keinen (öffentlichen) Konstruktor; man kann keine weiteren Werte erzeugen.



Aufzählungstypen und switch

Aufzählungstypen können in switch verwendet werden:

```
CardSuit suit = card.getSuit();
switch(suit) {
case HEARTS:
case DIAMONDS:
    System.out.println("rot");
    break;
case CLUBS:
case SPADES:
    System.out.println("schwarz");
    break;
}
```

Der Compiler warnt sogar, wenn man einen Fall vergessen hat.

Methoden in Aufzählungstypen

Man kann auch eigene Funktionen in Aufzählungstypen hinzufügen:

```
public enum CardSuit = {
    HEARTS, DIAMONDS, CLUBS, SPADES;

    public String getColor() {
        switch(this) {
        case HEARTS: case DIAMONDS:
            return "red";
        default:
            return "black";
        }
    }
}
```

Exceptions (Ausnahmen)

Fehler in Java

In Java werden Fehler durch spezielle Objekte, Exceptions, gekennzeichnet.
Wir haben schon einige kennengelernt:

- NullPointerException
- ArrayIndexOutOfBoundsException
- ArithmeticException

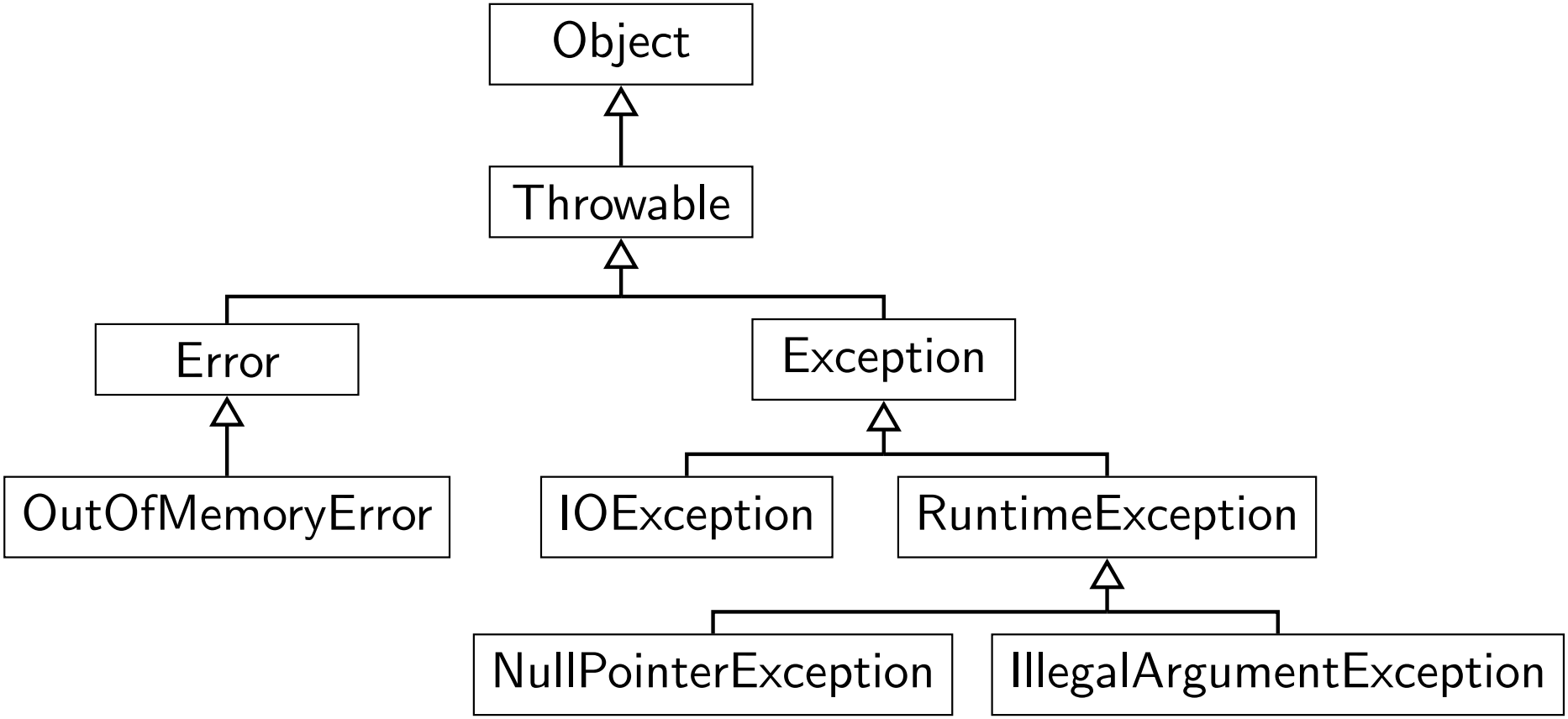
Fehlerausgabe

Wenn ein Fehler auftritt, wird normalerweise das Programm beendet und ein „Stacktrace“ ausgegeben, der besagt, wo der Fehler passiert ist:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java
    heap space
at de.uni_freiburg.informatik.ultimate.smtinterpol.convert.
    Clausifier.run(Clausifier.java:1955)
at de.uni_freiburg.informatik.ultimate.smtinterpol.convert.
    Clausifier.addFormula(Clausifier.java:2024)
at de.uni_freiburg.informatik.ultimate.smtinterpol.smtlib2.
    SMTInterpol.assertTerm(SMTInterpol.java:907)
at expriments.TestSMTInterpol.first_example(TestSMTInterpol.
    java:56)
at expriments.TestSMTInterpol.main(TestSMTInterpol.java:25)
```

- Erste Zeile gibt an, welcher Fehler aufgetreten ist.
- Zweite Zeile benennt die Klasse, Methode, Datei und Zeilennummer.
- Die Zeilen darunter geben an, wer die Methode aufgerufen hat.

Klassenhierarchie der Exceptions



Eigene Exceptions

Man kann in Java seine eigenen Exceptions bauen

```
public class MyException extends Exception
    public MyException(String message,
                       Throwable cause) {
        super(message, cause);
    }
    public MyException(String message) {
        super(message);
    }
}
```

Der Konstruktor nimmt bis zu zwei Parameter

- `String message` ist die Nachricht die am Ende ausgegeben werden soll.
- `Throwable cause` kann eine andere Exception benennen, die diesen Fehler verursacht hat. Es wird dann auch die andere Exception ausgegeben.

Exceptions werfen

In Java kann man das Kommando `throw` benutzen, um einen Fehler zu werfen:

```
public long factorial(int n) {
    if (n < 0)
        throw new IllegalArgumentException
            ("argument must not be negative");
    ...
}
```

Normalerweise wird `throw` mit *newException* benutzt. Man kann aber auch eine gefangene Exception erneut werfen.

Fehler fangen

Man möchte oft nicht, dass das Programm abstürzt, sondern stattdessen eine sinnvolle Fehlermeldung ausgeben und weitermachen.
Fehler können in Java mit folgender Syntax gefangen werden:

```
try {
    ... oeffne Datei ...
    ... lese Datei ...
} catch (FileNotFoundException ex) {
    System.out.println("Die Datei wurde nicht
       gefunden!");
}
... mache normal weiter ...
```

Wie werden Exceptions gefangen

Wenn eine Exception geworfen wird,

- wird zunächst der innerste `try`-Block der die Exception fängt gesucht.
- Gibt es in der Methode keinen, wird in der aufrufende Methode gesucht.
- Wenn der Block gefunden wurde und es einen passenden `catch`-Block gibt, wird mit dem Catch-Block weitergemacht.
- Gibt es keinen `try`-Block im gesamten Programm, wird das Programm beendet und die Exception ausgegeben.
- Wenn keine Exception geworfen wird, wird nach dem `try`-Block der `catch`-Block übersprungen.

Exception-Hierarchie

Ein `catch`-Block fängt alle Exception-Objekte von dem entsprechenden Typ *und alle Subklassen*.

Man kann also mit

```
catch (Exception ex) {
    ...
}
```

Alle `Exceptions`, auch `RuntimeException`, `NullPointerException`, `IOException` fangen.

Error, Exception, und RuntimeException

- Alle Subklassen von `Error` bezeichnen Fehler, die man nicht sinnvoll abfangen kann.
  Zum Beispiel: `OutOfMemoryError` (nicht genügend Speicher), `NoClassDefFoundError` (.class-Datei liegt nicht auf dem Rechner).
- Alle Subklassen von `RuntimeException` können jederzeit auftreten.
- Alle anderen Subklassen von `Exception` müssen explizit deklariert werden. Insbesondere `IOException`.

```
void readFile(String filename)
    throws IOException {...
```

Wenn eine Methode, die die Exception nicht deklariert eine andere aufruft, die sie deklariert, muss sie die Exception fangen und verarbeiten.