# Enum – Aufzählungstypen in Java

Dr Steffen Jost

Institut für Informatik der LMU München

# Endliche Aufzählungen



Endliche Aufzählungen (engl. Enumeration) bieten sich immer dann an, wenn Menge der Optionen vor Kompilieren bekannt

## Beispiele

- Booleans: `true`, `false` kein enum in Java
- Wochentage: Montag, Dienstag, . . . , Sonntag
- Noten: “Sehr gut”,. . . ,“Ungenügend”
- Spielkarten: Kreuz, Pik, Herz, Karo
- Nachrichtentypen eines Protokolls: login, logout, chat, . . .
- Optionen, z.B. Kommandozeilenparameter

Aufzählungen dürfen sich mit Programmversion ändern

# Probleme ohne `enum`



Aufzählungen mit finalen `int`/`String` Konstanten modelliert

Nachteile:

- Keine Typsicherheit: `int`-Konstante `MONTAG` kann auch dort verwendet werden, wo eine Spielkartenfarbe erwartet wird.
- Keine Bereichsprüfung: Wert 42 kann übergeben werden, wo eine Wochentag `int` Konstante erwartet wird.
- Sprechende Namen nicht erzwungen:
  “Hacks” mit direkten Zahlen können auftauchen
- Geringe Effizienz: z.B. Vergleich von `String` Konstanten;
- Keine Modularität: Gesamt-Rekompilation bei Änderungen

Seit Java 1.5: `enum` für endl. Aufzählungen möglich!

# `enum` Deklarationen



## Beispiel

```
public enum Kartenfarbe { KREUZ, PIK, HERZ, KARO; };
```

Definiert Typ `Kartenfarbe` mit 4 Konstanten.
mit Komma getrennt, mit Semikolon abgeschlossen

Verwendung durch `Kartenfarbe.PIK`

`Kartenfarbe` ist reguläre Java-Klasse:

- Nur jeweils eine Instanz pro statischer Konstante, d.h. es kann gefahrlos == verwendet werden
- Verschiedene Enums können gleiche Konstanten haben: Verwechslung wird durch Typsystem ausgeschlossen
- Erbe von `java.lang.Enum`, Methoden automatisch erstellt

# java.lang.Enum



Folegende Methoden werden über `java.lang.Enum` automatisch für jedes `enum` bereitgestellt:

| | |
|---|---|
| `boolean equals(Object other)` | Direkt verwendbar |
| `int hashCode()` | Direkt verwendbar |
| `int compareTo(E o)` | Vergleich gemäß Definitionsreihenfolge |
| `String toString()` | Umwandlung zur Anzeige |
| `static <T extends Enum<T>> valueOf(String)` | |
| `String name()` | NICHT verwenden! |
| `int ordinal()` | NICHT verwenden! |

Erlaubt optimierte Versionen `EnumMap<K extends Enum<K>,V>` und `EnumSet<E extends Enum<E>>`

anstatt Bit-Felder immer `EnumSet` verwenden!

# values()



Weiterhin wird für jedes enum eine statische Methode `values()` definiert, welche ein Array aller Konstanten liefert:

```
for (Kartenfarbe f : Kartenfarbe.values()) {
  System.out.println(f);
}
```

Reihenfolge der Konstanten wie in der Deklaration des enums

# Attribute



Konstanten können mit anderen Werten assoziiert werden, welche wie Attribute der enum-Klasse behandelt werden.

- Dazu Konstruktor und getter-Methoden definieren
- Konstruktoren müssen immer `private` sein

# Attribute



Konstanten können mit anderen Werten assoziiert werden, welche wie Attribute der enum-Klasse behandelt werden.

- Dazu Konstruktor und getter-Methoden definieren
- Konstruktoren müssen immer `private` sein

```
public enum Feld {
  FOREST("Wald",2), MEADOW("Wiese",2), MOUNTAIN("Berg",1);
  private final String typ;
  private final int ertrag;

  private Feld(String typ, int ertrag) {
    this.typ = typ;
    this.ertrag = ertrag;
  }
  public ertrag() {
    return ertrag;
  }
}
```

# Methoden



enum-Klassen können auch Methoden definieren

- reguläre statische Methoden
- Methoden in Abhängigkeit der Konstante:
  Abstrakte Methode kann in Konstanten-abhängiger Deklaration spezifiziert werden

enum-Klassen können somit Interfaces implementieren, wodurch man enum-Typen zusammenfassen kann.

Man sollte `switch` in Methoden der enum-Klasse möglichst vermeiden, da Änderung der Konstanten dann kein Fehler beim Kompilieren liefert! Syntax: nach `case` nur Konstante angeben

# Zusammenfassung



- Alle Optionen vor Kompilation bekannt
- Optionen können sich mit Programmversion ändern
- `enum` Deklaration generiert Klasse mit statischen Instanzen (kein öffentlicher Konstruktor)
- Konstanten sind automatisch geordnet
- Nützliche Methoden automatisch generiert, z.B. `values()`
- `EnumSet` anstatt Bit-Felder verwenden
- Werte über `EnumMap` oder Attribute assoziieren

# Quellen




- “Effective Java” (2nd Edition) von Joshua Bloch, Addison-Wesley Verlag, 2008
- Vorlesungsfolien zum Softwareentwicklungspraktikum, Wintersemester 2008/09, LMU München
- Dokumentation der Java, Platform Standard Edition 7, von Oracle
  `http://docs.oracle.com/javase/7/docs/api/`