

# Programmieren II

## Aufzählungstypen (enum)



Heusch --  
Ratz 11.1

Institut für Automation und angewandte Informatik

```
final List<String> allResults = new ArrayList<String>();  
final Map<String, Integer> typeWordResultCount = new HashMap<String, Integer>();  
final Map<String, Integer> typePoints = new HashMap<String, Integer>();  
evaluation.put(type, typePoints);  
  
for (final Sheet sheet : this.sheets) {  
    final String sheetResult = sheet.getPlayerInput(type);  
    if (sheetResult.startsWith(start) && this.isValidWord(sheetResult, type)) {  
        validWordCountForType++;  
        allResults.add(sheetResult);  
    }  
}
```

# Aufzählungstypen (1)

- Unter einem Aufzählungstyp versteht man einen selbstdefinierten Datentyp, der nur eine bestimmte (endliche) Menge von Werten umfasst, die
  - explizit aufgezählt werden müssen
  - jeweils (einzeln) als Konstante betrachtet werden können
  - In ihrer Reihenfolge festgelegt sind (Ordinalzahl = Indexwert)
  - deren Anzahl und „Werte“ nach der Erzeugung unveränderlich sind
- Für die *Literale der Konstanten* (d.h. die Namen der Konstanten im Quelltext) gelten die gleichen Regeln wie für Bezeichner (Namen).

## Aufzählungstypen (2)

- Ab Version 1.5 führt Java mit dem Schlüsselwort **enum** einen eigenständigen und expliziten Mechanismus zur Definition von Aufzählungstypen ein
- Es können zwei Arten von Aufzählungstypen unterschieden werden (wobei die erste Art ein Spezialfall der zweiten Art ist):
  - einfache Aufzählungstypen
  - komplexe Aufzählungstypen



**Ab Java 1.5**

# Einfache Aufzählungstypen (1)

- Konzeptionell sind Aufzählungstypen Variablen und Attributen einer Klasse gleichgestellt
  - seit Java 16 lassen sich enums auch **lokal** (in Methoden) als **implizit statische** Variablen definieren

```
public class EnumTest {  
  
    enum Season {  
        WINTER, SPRING, SUMMER, FALL  
    };  
  
    public static void main(String[] args) {  
        Season s1 = Season.WINTER;    // Variable vom Typ Season  
        if (s1 == Season.WINTER) {    // Vergleich mit ==  
            System.out.println("It's " + s1);  
        }  
        Season s2 = Season.WINTER;  
    }  
}
```

## Einfache Aufzählungstypen (2)

- Verwendung von Aufzählungstypen entspricht der von primitiven Datentypen
  - Keine Anforderung von Speicherplatz durch `new` notwendig
  - Compiler verhindert Instanziierung eines Aufzählungstyps
  - Aufzählungsinstanzen haben keinen Vorgabewert bei der Initialisierung
  - Verwendung einer nicht initialisierten Ausprägung führt zu einem Übersetzungsfehler („variable ... might not have been initialized“)
- Aufzählungsinstanzen besitzen keine Identität (Bei einem Vergleich ist Belegung maßgeblich)

# Einfache Aufzählungstypen (3)

## ■ Klassenmethode von `enum`

- `<T>[] values()` liefert ein Array mit allen Elementen
- `<T> valueOf(String value)` liefert eine `enum`-Konstante

## ■ Instanzmethoden von `enum`-Konstanten

- `int ordinal()` liefert die Ordinalzahl der `enum`-Konstante
- `String toString()` enthält die Zeichenkette (Bezeichner) mit dem die Aufzählungsinstanz belegt wurde
- `boolean equals(Object o)` liefert `true`, wenn `o` und die `enum`-Konstante übereinstimmen, sonst `false`

```
for (Season s : Season.values()) {  
    System.out.println(s + " has the value " + s.ordinal());  
}
```

## Einfache Aufzählungstypen (4)

- Nicht identisch sind hingegen Ausprägungen verschiedener Aufzählungstypen, die vermeintlich denselben Wert enthalten, d.h. in deren zur Definition verwendeten Werteliste sich lexikalisch dieselben Einträge finden.

Nachfolgende Zuweisung wird bereits durch den Übersetzer (`incompatible types`) abgelehnt

```
enum SeasonE { WINTER, SPRING, SUMMER, FALL; };  
enum SeasonG { WINTER, FRÜHLING, SOMMER, HERBST; };  
  
// ...
```

```
✗ SeasonE s = SeasonG.WINTER; // Compiler-Fehler
```

## Einfache Aufzählungstypen (5)

- Dasselbe gilt auch für den Versuch des Vergleichs der Inhalte zweier Aufzählungsausprägungen, wie sie durch das nachstehende Codefragment versucht wird.

```
enum SeasonE { WINTER, SPRING, SUMMER, FALL; };  
enum SeasonG { WINTER, FRÜHLING, SOMMER, HERBST; };  
  
// ...  
  
SeasonE s1 = SeasonE.WINTER;  
SeasonG s2 = SeasonG.WINTER;  
✗ if (s1==s2) { // Compiler-Fehler  
    // ...  
}
```

- Auch hier wird bereits zum Übersetzungszeitpunkt durch die Fehlermeldung auf den Fehler hingewiesen:

 Ausgabe

Incompatible operand types SeasonE and SeasonG



# Komplexe Aufzählungstypen (1)

- Aufzählungstypen lassen sich wie Klassen ausbauen. Diese Erweiterung erlaubt es, die Elemente des Aufzählungstypen wahlfrei an **selbst definierte Eigenschaften** zu binden.
- Die Eigenschaften werden dabei als **Attribute des Aufzählungstypen** aufgefasst, die durch einen vom Programmierer bereitzustellenden **Konstruktor** zugewiesen werden.
- Der Konstruktoraufruf erfolgt automatisch durch das Laufzeitsystem zum Definitionszeitpunkt des Aufzählungstypen für alle Inhaltselemente.

# Komplexe Aufzählungstypen (2)

```
public enum Planet { // Beachte: enum statt class

    MERCURY(3.303e+23, 2.4397e6), // Konstantendeklarationen mit Parametern
    VENUS(4.869e+24, 6.0518e6),   // für den Konstruktor (Masse, Radius)
    EARTH(5.976e+24, 6.37814e6),
    MARS(6.421e+23, 3.3972e6),
    JUPITER(1.9e+27, 7.1492e7),
    SATURN(5.688e+26, 6.0268e7),
    URANUS(8.686e+25, 2.5559e7),
    NEPTUNE(1.024e+26, 2.4746e7); // Nach der letzten Konstante

    public double mass; // Planetenmasse in Kilogramm
    private double radius; // Radius in Meter
    public static final double G = 6.67300E-11; // Gravitationskonstante

    Planet(double mass, double radius) { // Darf NICHT public sein!
        this.mass = mass;
        this.radius = radius;
    }

    double surfaceGravity() { // Gravitation an der Oberfläche eines Planeten
        return G * this.mass / (this.radius * this.radius);
    }

    double surfaceWeight(double otherMass) { // Gewicht an der Oberfläche
        return otherMass * this.surfaceGravity(); // eines anderen Planeten (N)
    }
}
```

# Komplexe Aufzählungstypen (3)

```
public class PlanetWeight {
    public static void main(String[] args) {

        if (args.length != 1) {
            System.err.println("Usage: java PlanetWeight <weight>");
            System.exit(-1);
        }

        // Gewicht an Erdoberfläche in Newton
        double earthWeight = Double.parseDouble(args[0]);
        double mass = earthWeight / Planet.EARTH.surfaceGravity();

        for (Planet p : Planet.values()) {
            System.out.printf("Your weight on %s is %f%n", p, p.surfaceWeight(mass));
        }
    }
}
```

## Ausgabe

```
> java PlanetWeight 800
Your weight on MERCURY is 302,206092
Your weight on VENUS is 723,999280
Your weight on EARTH is 800,000000
Your weight on MARS is 302,989747
Your weight on JUPITER is 2024,446020
Your weight on SATURN is 852,812431
Your weight on URANUS is 724,101760
Your weight on NEPTUNE is 910,662458
```

# enum-Beispiele (1)

```
/**
 * Farben als "Klasse"
 */
public enum SimpleColor {
    WHITE, BLACK, RED, YELLOW, BLUE; // ; ist optional
}
```

```
/**
 * Eine äußere Klasse
 */
public class Outer {
    /**
     * Farben als "innere Klasse"
     */
    public enum SimpleColor {
        WHITE, BLACK, RED, YELLOW, BLUE
    }
}
```

```
for ( SimpleColor s : SimpleColor.values() ){
    System.out.println(s);
}
```

 **Ausgabe**  
WHITE  
BLACK  
RED  
YELLOW  
BLUE

## enum-Beispiele (2)

```
/**
 * Farben mit überschriebener toString()-Methode
 */
public enum SimpleColorExt {
    WHITE, BLACK, RED, YELLOW, BLUE; // ; ist hier notwendig

    @Override
    public String toString() {
        // nur erster Buchstabe groß
        String s = super.toString();
        return s.charAt(0) + s.substring(1).toLowerCase();
    }
}
```

```
for ( SimpleColorExt s : SimpleColorExt.values() ){
    System.out.println(s);
}
```



Ausgabe

White  
Black  
Red  
Yellow  
Blue

# enum-Beispiele (3)

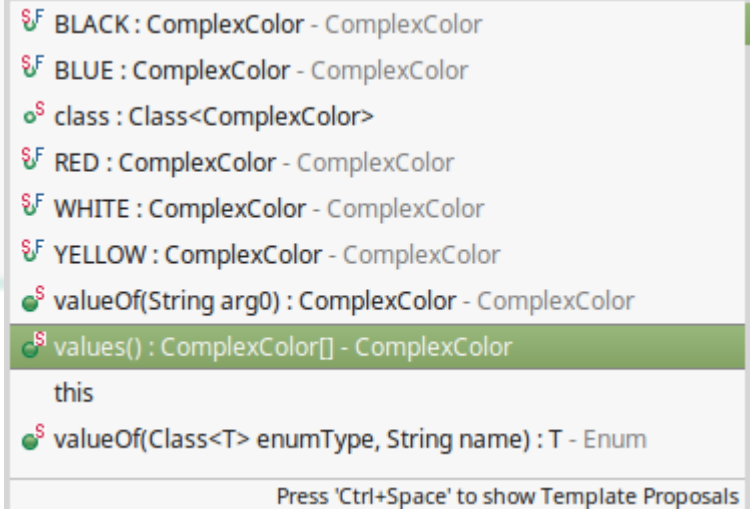
```
/**
 * Farben mit eigenen Codes. So könnte man z.B. zu jeder Farbe auch RAL-
 * oder RGB-Farbwerte verwalten.
 */
public enum ComplexColor {
    WHITE(21), BLACK(22), RED(23), YELLOW(24), BLUE(25);

    private int code;

    private ComplexColor(int c) {
        this.code = c;
    }

    public int getCode() {
        return this.code;
    }
}
```

## Nutzung von .values()



S F BLACK : ComplexColor - ComplexColor  
 S F BLUE : ComplexColor - ComplexColor  
 S class : Class<ComplexColor>  
 S F RED : ComplexColor - ComplexColor  
 S F WHITE : ComplexColor - ComplexColor  
 S F YELLOW : ComplexColor - ComplexColor  
 S valueOf(String arg0) : ComplexColor - ComplexColor  
**S values() : ComplexColor[] - ComplexColor**  
 this  
 S valueOf(Class<T> enumType, String name) : T - Enum

Press 'Ctrl+Space' to show Template Proposals

## enum-Beispiele (4)

- Aufzählungstypen lassen sich in `switch`-Anweisungen verwenden:

```
SimpleColor color = SimpleColor.values()[2];
switch (color) {
case WHITE:
    System.out.println("weiß"); break;
case BLACK:
    System.out.println("schwarz"); break;
case RED:
    System.out.println("rot"); break;
case YELLOW:
    System.out.println("gelb"); break;
case BLUE:
    System.out.println("blau"); break;
default:
    System.out.println("andere Farbe");
}
```

*Wie im 1. Semester / Kontrollstrukturen „versprochen“!*

# enum-Beispiele (5)

```
public class ColorExample {

    public static void main(String[] args) {
        // Farbe aufgrund ihres Wertes bestimmen
        SimpleColor col = SimpleColor.valueOf("BLACK");
        System.out.println("Color: " + col);

        // Nummern (Reihenfolge, Zählung beginnt bei 0) bestimmen
        System.out.println("#" + col + ": " + col.ordinal());
        System.out.println("#" + SimpleColor.RED + ": " +
                               SimpleColor.RED.ordinal());
        System.out.println("#" + SimpleColor.BLUE + ": " +
                               SimpleColor.BLUE.ordinal());
    }
}
```

```
> Ausgabe
Color: BLACK
#BLACK: 1
#RED: 2
#BLUE: 4
```

- `int ordinal()` liefert, das wievielte Element des Aufzählungstyps eine bestimmte Instanz ist. (Die Zählung beginnt wie bei Arrays bei 0)
- `<T> valueOf(String s)` liefert (sofern vorhanden) die zum String `s` passende Instanz eines Aufzählungstyps, ansonsten eine `IllegalArgumentException`



# Literatur

- Java-Tutorial für `enum`

<http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>