# Objektorientierte Programmierung

Statische Klassenelemente

Prof. Dr. Ulrike Hammerschall
Fakultät für Informatik und Mathematik

## Themen

- Klassenvariablen und Konstanten
- Statische Methoden
- Enums als Datentypen

## Objektvariablen versus Klassenvariablen

HM

- Objektvariablen:
  - werden in der Klasse für alle Objekte gleich definiert.
  - ihre Werte gelten jedoch immer nur für genau ein Objekt.
  - Änderungen an den Werten beziehen sich nur auf das jeweilige Objekt.

```
private int counter;
```

- Klassenvariablen
  - werden ebenfalls in der Klasse definiert.
  - ihre Werte gelten für alle Objekte zur Klasse.
  - Änderungen an Klassenvariablen gelten für alle Objekte.
  - sind mit dem Schlüsselwort *static* gekennzeichnet.

```
private static int counter;
```

*Siehe Beispiel Rational.java*



## Konstanten

HM

- Konstanten sind Klassenvariablen, die vor der Objektinitialisierung (vor Aufruf des Konstruktors) gesetzt sind und nicht verändert werden dürfen:
  - **static**: Klassenvariable
  - **final**: unveränderlich
- Sichtbarkeit
  - public-Konstanten sind öffentliche Konstanten, die von allen Klassen verwendet werden können.
  - private-Konstanten sind private Konstanten, die nur aus Methoden innerhalb der Klasse verwendet werden können.
- Namen von Konstanten: ganz in Großbuchstaben, Wortteile mit Unterstrichen ("_") getrennt.

```
public static final double EPSILON = 1E-16;
private static final int CAPACITY = 21;
```

## Statische Methoden

- Gehören wie Klassenvariablen allen Objekten einer Klasse.
- Werden ebenfalls mit dem Schlüsselwort **static** markiert.
- Der Aufruf erfolgt:
  - von Methoden innerhalb der gleichen Klasse über den Methodennamen.
  - von Methoden aus anderen Klassen über den Klassennamen und den Methodennamen (nicht über das Objekt).

*Siehe Beispiel Rational.java*



## Zugriffsregeln für statische Methoden

- Statische Methoden dürfen ohne Einschränkung auf Klassenvariablen und Konstanten zugreifen.
- Statische Methoden dürfen **nicht** auf Objektvariablen zugreifen.
- Statische Methoden dürfen **nicht** auf Objektmethoden zugreifen.
- Hintergrund: Es kann nicht eindeutig entschieden werden, welches Objekt gemeint ist. Gegebenenfalls existiert noch nicht einmal ein Objekt.

## Zugriffsregeln für Objektmethoden

- Objektmethoden dürfen ohne Einschränkung auf Klassenvariablen und Konstanten zugreifen.
- Objektmethoden dürfen ohne Einschränkung auf Objektvariablen zugreifen.
- Objektmethoden dürfen ohne Einschränkung auf statische Methoden und Objektmethoden zugreifen.
- Achtung: Methoden aus anderen Klassen werden immer über den Klassennamen, nie über ein Objekt der Klasse aufgerufen.



## Die main-Methode

- Startmethode für jedes Java-Programm. Die Signatur ist fest vorgegeben.
- Muss **static** deklariert sein, da zu Beginn noch kein Objekt existiert.
- Aus der main-Methode heraus wird sukzessive das Objektgeflecht der Anwendung aufgebaut.

```
public class AnyClass {

    // Objekt- und Klassenvariablen von AnyClass
    // Konstruktoren und Methoden von AnyClass

    public static void main(String[] args) {

        // Einlesen der Argumente von der Kommandozeile
        // Initialisieren eines Objekts der Klasse
        AnyClass anyClass = new AnyClass(...);

        // Aufruf von Methoden auf dem Objekt
        anyClass.methodeVonAnyClass(...);
        ...
    }
}
```

## Themen

- Klassenvariablen und Konstanten
- Statische Methoden
- Enums als Datentypen



## Eingeschränkte Wertebereiche

- Datentypen wie int, double und boolean speichern Zahlen und Wahrheitswerte.
- Oft werden nur eingeschränkte Wertebereiche gebraucht. Beispiele:
  - Bestimmte Farben aus dem Farbspektrum
  - die sieben Wochentage
  - die zwölf Monate im Jahr
  - weiblich/männlich
  - Schachfiguren
- Die Codierung solcher Wertemengen als Zahlen oder Wahrheitswerte ist technisch möglich, aber logisch willkürlich.

## Enumerations

- "Aufzählungstypen" ("Enumerations", "Enums", engl. "enumerations") erlauben Typdefinition für begrenzte Wertemengen.
- Definitionsschema:

```
enum <Type> {<Wert1>,<Wert2>,...}
```

- **Type** definiert den Namen des Typs. Es gelten die gleichen Namenskonventionen wie bei Klassennamen.
- **Wert**: ein Element der Enum-Liste. Die gültigen Werte werden in den geschweiften Klammern aufgelistet. Es gelten die Konventionen bei Konstanten.



## Beispiele für Enums

```
// Enumeration für Ampelfarben
enum Color {
       RED, YELLOW, GREEN
}
// Enumeration für englische Wochentage
enum Day {
       MON, TUE, WED, THU, FRI, SAT, SUN
}
// Enumeration für das Geschlecht
enum Gender {
       FEMALE, MALE, OTHER
}
// Enumeration für Schachfiguren
enum Chess {
       PAWN, ROOK, KNIGHT, BISHOP, QUEEN, KING
}
```

## Beispiel: Arbeiten mit Enums

```
enum Color {
	RED, GREEN, BLUE, YELLOW
}
class SavingBox {
	// Betrag im Sparschwein
	private double content = 0;
	// Farbe des Sparschweins
	private Color color;

	// Custom Konstruktor
	SavingBox(Color color){
		this.color = color;
	}
}
```



## Beispiel: Arbeiten mit Enums

```
// Verwendung des Enums in einer anderen Klasse
class SavingBoxMain {

	public static void main(String[] args) {
		// Initialisierung der Farbe
		Color color = Color.RED;
		SavingBox box = new SavingBox(color);

	}

}
```

## Definitionsorte für Enums

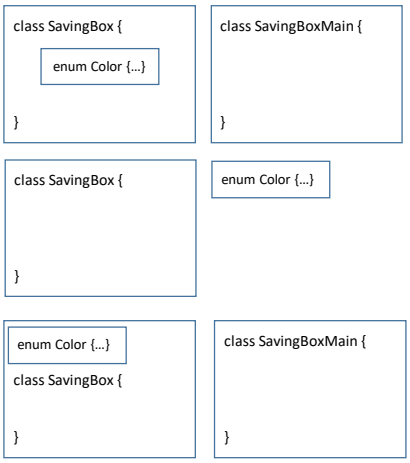
- Innerhalb einer Klasse
  - Zugriff aus der gleichen Klasse über **<EnumTyp>**

    `Color.RED`
  - Zugriff aus anderen Klassen über **<Klasse>.<EnumTyp>**

    `SavingBox.Color.RED`
- In eigener Datei
  - Dateiname entspricht **<EnumTyp>.java**
  - Zugriff über **<EnumTyp>**

    `Color.RED`
- Außerhalb einer (beliebigen) Klasse
  - Zugriff über **<EnumTyp>**

    `Color.RED`






## Methoden auf Enums

```
static EnumType[] values()
```

➢liefert ein Array der Enumwerte.

```
static EnumType valueOf(String value)
```

➢liefert den Enumtyp zum gegeben Wert.

```
int ordinal()
```

➢liefert die Position des Werts innerhalb der Definition des Enums. Die erste Position beginnt mit 0.

```
String toString()
```

➢Liefert den Namen einer Konstante entsprechend zur Definition.

## Übung

Gegeben ist folgender Enum

```
enum Color {PINK, YELLOW, RED, BLUE, GREY}
```

Mit welcher Anweisung erhalten Sie zum letzten Wert folgende Informationen:

```
// Letzte Farbe in Enum als String
String lastColorAsString =

// Letzte Farbe im Enum als Color-Objekt
Color lastColorAsObject =

// Ordinalzahl der Farbe RED =
int redColorOrdinal =

// Der String Yellow als Colorobjekt
Color yellowStringAsObject =
```



## Erweiterungen von Enums

- Enums entsprechen einer verkürzten Klassendefinition.
- Die Enumwerte entsprechen dabei den Konstanten.
- Als (stark) vereinfachte Analogie:

```
class Color {
    private static final Color RED = new Color();
    Private static final Color BLUE = new Color();
    …
}
```

- Enums können zusätzlich alle anderen Elemente einer Klasse beinhalten wie beispielsweise Objektvariablen, Konstruktoren und Methoden.

## Beispiel: Enum für Wochentage

```
enum Day { MON(false), TUE(false), WED(false), THU(false),
        FRI(false), SAT(true), SUN(true);

        // Wochenende?
        private final boolean weekend;

        // Konstruktor
        Day (boolean w) {
                weekend = w;
        }
        // Abfrage, ob es sich um Wochenende handelt
        boolean isWeekend() {
                return weekend;
        }
}
```



## Beispiel: Enum für Wochentage

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

        Day day1 = Day.SAT;
        System.out.println(day1.isWeekend());

        Day day2 = Day.MON;
        System.out.println(day2.isWeekend());

}
```

## Zusammenfassung

- Mit **static** deklarierte Variablen gehören allen Objekten zu einer Klasse. Sie werden als Klassenvariablen bezeichnet.
- Mit **static** und **final** deklarierte Variablen (und nur diese!) bezeichnet man als Konstanten.
- Konstanten gehören der Klasse (nicht einzelnen Objekten), benötigen bei Ihrer Initialisierung einen Wert und sind unveränderlich.
- Mit **static** deklarierte Methoden gehören allen Objekten. Sie werden als Klassenmethoden bezeichnet. Klassenmethoden dürfen nicht auf Objektvariablen zugreifen.
- Enums sind Referenztypen wie alle anderen Klassen auch. Sie definieren fachlich motivierte festgelegte Wertebereiche.
- Ein Enum kann zusätzlich einen Konstruktor, Objektvariablen sowie Methoden enthalten.