Inhaltsverzeichnis

1. Java Enums

`Enumerations` (Aufzählungen) sind vordefinierte **Wertelisten** und können typsicher genutzt werden.

Java 5 hat das Schlüsselwort `enum` eingeführt. Es bezeichnet einen speziellen Typ einer Klasse und erbt immer (implizit) von `java.lang.Enum`. Die offizielle Dokumentation findet sich hier → Enums in Java 17

Konstanten, die mithilfe von **Enums** definiert werden,

- machen den Code besser lesbar,
- erlauben typ-sichere Nutzung zur Compile-Time,
- listen den Wertebereich von vorneherein auf und
- vermeiden Fehler durch die Verhinderung von falschen Werten, die nicht aus der Menge der definierten stammen.

Die Aufzählungswerte haben einen **Index** (=Ordinal), beginnend mit `0`, und können automatisch von einem String in den gleichlautenden Enum-Wert umgewandelt werden.

Ein einfaches Beispiel:

```
public enum Color {
    Red,   // ordinal 0
    Green, // ordinal 1
    Blue;  // ordinal 2
}

// Die Nutzung erfolgt einfach durch:
Color blue = Color.Blue;

// Das Ordinal
int ordinal = Color.Blue.ordinal();

// oder bzgl. der Umwandlung von String in den enum Typ
Color blue = Color.valueOf("Blue");
```

Der Aufruf der von `enum` zur Verfügung gestellten Methode `ordinal()` gibt entsprechend die **Ordnungszahl** bzw. den **Positionsindex** des jeweiligen Enum-Wertes zurück.

1.1. Zusätzliche Methoden

`Enums` erlauben darüber hinaus die Implementierung von **zusätzlichen Methoden** in der Enum-Klasse.

→ **Demo 1** *in EnumsTest.java*

```
public enum Color {
    Red,
    Green,
    Brown,
    Unknown;

    public String getPalette() {
        return "RGB";
    }
}
```

1.2. Enums in Switch-Ausdrücken

Ein weiterer Vorteil ist die bequeme Nutzung der enum Konstanten in `switch` Statements:

→ **Demo 2** *in EnumsTest.java*

```
public enum ColorPalette {
    RGB, CYMK;
}

switch (color) {
    case RGB  -> { /* ... do stuff ...       */ }
    case CYMK -> { /* ... do other stuff ... */ }
}
```

Ergänzung zur Nutzung vom **logischen UND** in den Unit-Tests → siehe `module-operators`

1.3. Zusätzliche Datenfelder

Da Enums "normale" Klassen sind, lassen sie sich mit **Datenfeldern** erweitern. Dazu müssen lediglich die Datenfelder selbst sowie ein oder mehrere **Konstruktoren** erzeugt werden, um die (ebenfalls konstanten) Werte für die Enum-Konstanten zu definieren.

Dazu ein Beispiel:

→ **Demo 3** *in EnumsTest.java*

```
public enum FoodGroup {
```

```
  Fruchtsaefte("Fruchtsäfte"),
  Suesswaren("Süßwaren"),
  Sonstiges("Sonstiges");

  private final String label;

  FoodGroup(String label) {
    this.label = label;
  }
}
```

Demo:

Die Unit-Tests zur **Demonstration** finden sich hier:

```
src/test/java/de/dhbw/EnumsDemoTests.java
```

Übungen:

Nutze folgendes Package für die **Unit-Tests**:

```
src/test/java/de/dhbw/EnumsExerciseTests.java
```

Die im Test benutzten **Implementierungen** sind im Package:

```
src/main/java/de/dhbw/exercises/*.java
```

Übung 1:

Ändere die Methode `mix` in der Enumeration `Color`, sodass sie anstatt des Konditional-Ausdruckes `if-elseif-else` nun den `switch` Ausdruck nutzt, um die Mischfarben zurückzugeben.

Übung 2:

Erweitere die vorbereitete Enumeration `de.dhbw.enums.exercises.RgbColor` mit drei Attributen `r` (red), `g` (green) und `b` (blue), jeweils mit Datentyp `int`, sodass jede Enum-Konstante direkt den RGB-Wert erhält.

Nutze die Methode `getRGB()`, um die jeweils zugeordneten RGB-Werte in einem Unit-Test zu prüfen.