Inhalt der Vorlesung

1. Klassen und Objekte
2. Vererbung
3. Enums, Wrapper und Autoboxing
4. Interfaces
5. Generics
6. Exceptions
7. Polymorphismus
8. Grafische Oberflächen
9. Streams und Lambda Expressions
10. Leichtgewichtige Prozesse – Threads

Kapitel 3: Enums, Wrapper und Autoboxing

Inhalt

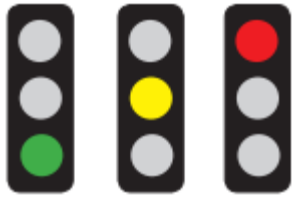
Lernziele

[LZ 3.1] Erläutern können, was ein Java Enum ist und wie es definiert wird
[LZ 3.2] Das Wrapper-Konzept in Java erklären und anwenden können
[LZ 3.3] Autoboxing und Autounboxing erklären können

3. Enums, Wrapper und Autoboxing

3.1 Einführung in Enums

Aufzählungstypen helfen, Quellcode lesbarer und wartbarer zu gestalten. Seit Java 5 gibt es auch für Java ein Konzept zur Definition von Aufzählungstypen. Am Beispiel einer Ampel soll die Syntax eingeführt werden.



Eine Ampel in Deutschland arbeitet typischerweise folgenden Zyklus ab:
ROT – ROT+GELB – GRÜN – GELB – ROT usf.
Zur Umsetzung der Steuer-Software bietet sich ein Aufzählungstyp mit folgenden Werten an: INAKTIV, ZEIGT_ROT, ZEIGT_ROT_GELB, ZEIGT_GRUEN, ZEIGT_GELB, wobei INAKTIV einen Zustand andeutet, in dem die Ampel den Phasenzyklus nicht durchläuft.
In Java kann obiger Ansatz folgendermaßen umgesetzt werden:

```
enum AmpelZustand {
     INAKTIV, ZEIGT_ROT, ZEIGT_ROT_GELB, ZEIGT_GRUEN, ZEIGT_GELB
}
AmpelZustand zustand = AmpelZustand.INAKTIV; // Ampel ausgeschaltet
```

Ein Java-„enum“ kann als eine Klasse aufgefasst werden, die die Aufzählungswerte als konstante statische Attribute öffentlich zugänglich macht.

Die Variable „zustand“ aus dem letzten Beispiel ist eine Referenz auf ein enum-Objekt vom Typ AmpelZustand. Solche Enum-Objekte bieten folgende Standard-Methoden an:

- name(): liefert den Namen des Enum-Wertes. Bsp.: zustand.name() liefert „INAKTIV“
- ordinal(): liefert die Indexposition des Enum-Wertes. Bsp: zustand.ordinal() liefert 0, da der Compiler die Werte bei 0 beginnend fortlaufend durchnummeriert

Jedes Enum bietet eine statische Methode values(), die ein Array der enum-Objekte zurückgibt. Damit lässt sich sehr einfach eine Iteration über die Werte realisieren:

```
for (AmpelZustand z : AmpelZustand.values())
      System.out.printf("name: %s, index: %d", z.name(), z.ordinal());
```

Darüber hinaus ist es auch möglich, enums um eigene Methoden zu erweitern:

```
public enum Enum {
    WERT1, WERT2;

    public String toString() {
        return "Enum(" + name() + ")";
    }
    public static void main(String[] args) {
        Enum e1 = WERT1;
        System.out.println("e1 = " + e1);
    }
}
```

Weiterhin werden die Methoden toString und equals (von Object geerbt) für Enums überschrieben:
- toString liefert den Wert von name(), also den symbolischen Namen
- equals() führt einen Inhaltsvergleich durch

Entsprechend haben folgende Zeilen die gleiche Wirkung:

```
System.out.println(obst1.equals(obst2));
System.out.println(obst1.ordinal() == obst2.ordinal());
```

Aus Effizienzgründen (Speicherplatz, Performance) gibt es in Java neben den Objekten auch Instanzen primitiver Typen wie char, int, usf. In gewissen Situationen ist es jedoch erforderlich, Instanzen primitiver Typen in ein Objekt umzuwandeln, wie im folgenden Beispiel:

```
int x = 3, y = 5;
System.out.println("x = " + x + ", y = " + new Integer(y).toString());
```

`println` benötigt einen `String` als Parameter. Der +-Operator wandelt entsprechend den Wert von x in eine Objektinstanz vom Typ `Integer` um und ruft `toString` auf. Für y ist dieser Vorgang explizit ausprogrammiert.
Java stellt für solche Fälle folgende sogenannte Wrapper-Klassen zur Verfügung, die primitive Werte in eine Objekthülle „einwickeln“. Für die Typen boolean, byte, char, double float, long, short und void gibt es gleichnamige Klassen, die jedoch gross geschrieben werden. Für char bzw. int heißen die Wrapper-Klassen Character bzw. Integer.

Wrapper-Instanzen werden über Konstruktoren erzeugt, denen jeweils der primitive Wert übergeben wird (s. Beispiel oben). Das Auslesen des eingewickelten Wertes gelingt mit Hilfe von Methoden namens `xxxValue()`, wobei xxx für den Namen des primitiven Typs steht. Beispielsweise kann ein Character-Wrapper cWrapper so ausgelesen werden:

```
char c = cWrapper.charValue();
```

Ab Java 5 werden die Umwandlungen primitiver Werte in Wrapper-Objekte und wieder zurück bei Bedarf automatisch vom Compiler durchgeführt. Die automatische Umwandlung eines primitiven Wertes in ein Wrapper-Objekt bezeichnet man als **Autoboxing**, die Gegenrichtung als **Autounboxing**:

```
char c = 'a';
Character cWrapper = c;  // Autoboxing: c wird in ein Character-Objekt eingebettet
char c2 = cWrapper;      // Autounboxing: char-Wert aus cWrapper wird in c2 gespeichert
```

Hinweis: Sehr wichtig sind Wrapper und Autoboxing/Autounboxing im Zusammenhang mit parametrisierbaren Klassen (Generics), siehe Kapitel 5.