



Programmierungsmethodik 1

Programmiertechnik

Statische "Dinge",
Aufzählungstypen

- 27.04.2015
 - mehrere kleine Typos korrigiert
 - Folie zum statischen Import

- Konstruktoren
- Sichtbarkeit
- UML
- Unveränderliche Klassen

- Entwickeln Sie folgende Klassen: Mobiltelefon, Display
- Mobiltelefon:
 - Typ (String)
 - Display
 - Anrufen (Ausgabe "Hallo?")
 - Eigenschaft beim Erzeugen gesetzt
- Display
 - Größe (Fließkommazahl)
 - Ausgeben eines Textes
 - Eigenschaft beim Erzeugen gesetzt
- Zeichnen Sie erst ein Klassendiagramm
- Schreiben Sie dann den Quellcode für die Klassen
- Erzeugen Sie mindestens eine Mobiltelefon-Instanz

Ausblick für heute

- Funktionalität wird nicht von einem Objekt, sondern von einer Klasse (gemeinsam für alle Objekte, unabhängig von den einzelnen Objekten) angeboten
- Wir repräsentieren eine (endliche) Menge von Dingen mit einem Namen (z.B. Montag, ..., Sonntag)

- Statische Objektvariablen
- Statische Methoden
- Aufzählungstypen

Statische Objektvariablen

Definition

- bisher betrachtete Objektvariablen und Methoden beziehen sich auf ein bestimmtes Objekt (= Zielobjekt)
- statische Variable = Klassenvariable
 - der Klasse zugeordnet, keinem einzelnen Objekt zugeordnet
- Definition einer Klassenvariablen
 - wie normale Objektvariable
 - zusätzlich Modifier `static`
- Beispiel

```
public class Bruch {  
    private static int objektZaehler;  
    ...  
}
```

Initialisierung

- bei der Definition wie andere Objektvariablen

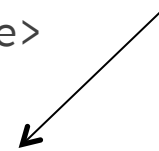
```
public class Ding {  
    private static int objektZaehler;  
    ...  
}
```

- ohne explizite Initialisierung
 - Defaultwert abhängig vom Typ
- Beispiel: objektZaehler
 - hätte auch ohne explizite Initialisierung den Defaultwert 0
- Lebensdauer einer Klassenvariablen
 - gesamte Programmlaufzeit
 - unabhängig von Objekten

Zugriff

- mit Klassennamen statt Zielobjekt
- Syntax für den Zugriff auf eine Klassenvariable:
`<Klassenname>.<Klassenvariablenname>`
- Beispiele:
 - `System.out.println(Ding.objektZaehler);`
 - `Math.PI` ist Klassenvariable `PI` der Klasse `Math`
 - `Integer.MAX_VALUE` ist Klassenvariable `MAX_VALUE` der Klasse `Integer`
- eine Klassenvariable existiert unabhängig von Objekten der Klasse!

geht natürlich
nur, wenn
objektzaehler
public sichtbar ist



Anwendung von Klassenvariablen: öffentliche Konstanten

- Werte öffentlicher Konstanten dürfen sich nicht ändern
 - Modifier `static` und `final`
- Beispiel
 - Auszug aus der Definition von Integer:

```
public class Integer {  
    public static final int MAX_VALUE = 2147483647;  
    public static final int MIN_VALUE = -2147483648;  
    ...  
}
```
- öffentliche Konstanten werden bei der Deklaration initialisiert
- Konvention zur Benennung von Konstanten
 - GROSS_BUCHSTABEN
 - Wortteile mit Unterstrichen (`_`) getrennt (anders als normale Variablen)

- Klassenvariablen sind in Methoden ebenso verwendbar wie Objektvariablen
- aber: es existiert nur ein einziges Exemplar für alle Objekte
- Beispiel
 - Generisches Ding mit eindeutiger Seriennummer

Ding mit Seriennummer



```
/**
 * Kontainerklasse für Dinge, die bei der Erzeugung automatisch
 * eine eindeutige
 * Seriennummer bekommen.
 */
public class Ding {
    /**
     * Statischer Zaehler.
     */
    private static int objektZaehler = 0;

    /**
     * Eindeutige Seriennummer.
     */
    private int seriennummer;

    /**
     * Default Konstruktor.
     */
    public Ding() {
        objektZaehler++;
        seriennummer = objektZaehler;
    }

    ...
}
```

- Geben Sie die Deklaration und Initialisierung einer Konstante für eine obere Grenze für Fließkommazahlen (Wert 23.42) an.

Statische Methoden

- Statische Methoden
 - auch Klassenmethoden
 - beziehen sich auf die ganze Klasse, nicht auf ein bestimmtes Objekt
 - existieren unabhängig von Objekten (wie Klassenvariablen)
- Definition wie normale Methode mit Modifier `static`

```
public static int getObjectZaehler() {  
    return objektZaehler;  
}
```
- Zugriffsschutz
 - Modifier `public/private` und Überladen wie normale Methoden
- Aufruf mit Klassennamen statt Zielobjekt
 - objektunabhängig
 - `System.out.println(Ding.getObjectZaehler());`

- ohne Zielobjektangabe kein Objektbezug vorhanden
 - `this` nicht verfügbar
 - nur Zugriff auf (statische) Klassenvariablen
 - nur Aufruf anderer (statischer) Klassenmethoden
- die Methode wird statisch gebunden, nicht dynamisch
 - der Compiler kann den Methodenaufruf fest implementieren
 - die JVM braucht zur Laufzeit keine Auswahlentscheidung bzgl. eines Objekts mehr zu treffen (siehe später: dynamische Bindung)

```
private int zaehler; // Objektvariable, nicht statisch

public static int getZaehler() {
    return zaehler; // Fehler, Objektvariable nicht instanziiert
}
```

- Einsatz von statischen Methoden oft als Hilfsmethoden, die unabhängig von bestimmten Objekten sind
- Beispiel
 - öffentliche GGT-Methode der Klasse Bruch:

```
public class Bruch {  
    public static int berechneGgt(int n, int m){  
        ...  
    }  
}
```
- kann auch beliebig ohne Bruch-Objekt benutzt werden:

```
System.out.println(Bruch. berechneGgt(221, 255));
```

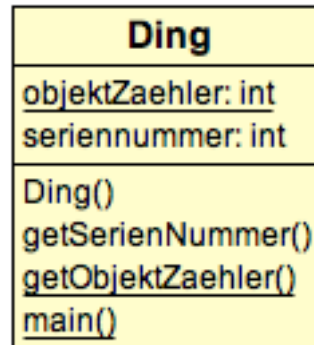
- einige vordefinierte Klassen definieren nur statische Methoden
 - beispielsweise Math
- Klasse selbst ist dabei nebensächlich
 - dient nur zur Organisation einer Sammlung verwandter Methoden

- schon von Anfang an benutzt:
 - statische Methode `main` = Hauptprogramm
- vor dem Aufruf von `main` existiert noch kein Objekt
 - `main` muss statisch sein
- Programmstart (jedes Java-Programm!)
 - Suchen von `main`
 - durch die JVM
 - in der Klasse, die auf der Kommandozeile genannt ist
 - Ausführen der Methode

Einstiegsmethode `main()`

- Beispiel
 - Kommando `$ java classname` sucht
 - in der Klasse `classname`
 - nach der Methode `public static void main(String[] args) {...}`
- `main` ist ansonsten normale Methode
- `main` kann ...
 - überladen werden
 - vom Programm selbst aufgerufen werden
 - in mehreren verschiedenen Klassen definieren sein
 - beliebige Objekte erzeugen

- statische Elemente werden in der UML-Darstellung unterstrichen
- Beispiel
 - Klasse Ding



- beim Aufruf statischer Methoden muss Klasse mit angegeben werden
 - ebenso Konstanten
- Beispiel:

```
double sinWinkel = Math.sin(2*Math.PI);
```
- manchmal wünschenswert: Methodenaufruf ohne Klassenname
- Lösung: statischer Import (gleiche Stelle im Code wie "normaler Import")

```
import static java.lang.Math.*;
```

dann:

```
double sinWinkel = sin(2*PI);
```

- Schreiben Sie eine Klasse Zeitausgabe, die mit der statischen Methode gibaus die aktuelle Zeit auf der Konsole ausgibt.
- Außerdem soll bei der Ausgabe die Anzahl der Methodenaufrufe mit angegeben werden.
- Hinweis: Die aktuelle Zeit bekommen Sie unter Java 8 z.B. so:

```
LocalTime zeit = LocalTime.now();
```

- das zeit-Objekt können Sie direkt mit `System.out.println(...)` ausgeben
- Beispiel

<code>Zeitausgabe.gibAus();</code>	1: 09:16:47.811
<code>Zeitausgabe.gibAus();</code>	2: 09:16:47.821
<code>Zeitausgabe.gibAus();</code>	3: 09:16:47.821

Aufzählungstypen

Idee

- oft wird eine Sammlung von diskreten konstanten Werten gebraucht
 - weder Zahlen noch Wahrheitswerte
- Beispiele
 - rot, grün, blau
 - Mo, Di, Mi, Do, Fr, Sa, So
 - weiblich/männlich
 - Schachfiguren
- Lösungsansatz: Codierung der Wertemengen als Zahlen oder Wahrheitswerte
 - technisch möglich
 - aber logisch willkürlich oder gar irreführend

- Lösung: Aufzählungstypen
 - engl. enumeration types oder Enums
 - erlauben spezielle Typdefinitionen
 - jeweils Aufzählung diskreter Werten
- Aufzählungstypen sind Referenztypen
 - Aufnahme in Collections möglich (\Rightarrow siehe später)

Definition

- Syntax

```
enum <Aufzählungstypname>{<Konstantenliste>}
```

- für den Aufzählungstypnamen gelten die gleichen Namenskonventionen wie bei Klassennamen
- in der <Konstantenliste> werden die Enum-Werte durch Komma getrennt
⇒ als Konstanten per Konvention groß geschrieben
- Beispiele

```
enum Farbe { ROT, GRUEN, BLAU }
```

```
enum Wochentag { MO, DI, MI, DO, FR, SA, SO }
```

```
enum Geschlecht { M, W }
```

```
enum Schachfigur {BAUER, TURM, PFERD, LAEUFER, DAME, KOENIG }
```

Anwendung

- Aufzählungstypen sind gleichberechtigt mit anderen Typen
- Zugriff auf einen Enum-Wert (Aufzählungsliteral):
`<Aufzählungstypname>.<Enumwert>`
- Beispiel
 - Definition einer *Variablen*
`Farbe farbe;`
 - Zuweisen eines Wertes:
`farbe = Farbe.ROT;`
 - *Vergleich* eines Wertes:
`if (farbe == Farbe.BLAU) {`
`...`

Switch-Anweisungen

- Aufzählungstypen können in `switch`-Anweisungen verwendet werden
- dann ohne Aufzählungstypnamen

```
Wochentag heute = ...;
switch (heute) {
    case S0:
        System.out.println("Relaxen!");
        break;
    case SA:
        System.out.println("Aufräumen!");
        break;
    default:
        System.out.println("Studieren!");
}
```


- Definition eines Aufzählungstyps
 - entspricht spezielle Klassendefinition
 - Enum-Werte = Klassenvariablen

- Beispiel: Definition

```
enum Farbe {ROT, GRUEN, BLAU}
```

- ist in etwa äquivalent zu

```
class Farbe{  
    static final Farbe ROT = new Farbe ();  
    static final Farbe GRUEN = new Farbe ();  
    static final Farbe BLAU = new Farbe ();  
}
```

- Klassenvariable `ROT` der Klasse `Farbe` liefert ein existierendes `Farbe`-Objekt:

```
Farbe farbe = Farbe.Rot;
```

- neue Enumobjekte können nicht erzeugt werden

- Definieren Sie einen Aufzählungstyp `HochschulPerson`, für Personen an einer Hochschule
 - Studierender (24)
 - Professor (45)
 - Mitarbeiter (41)
- Deklarieren und initialisieren Sie eine Variable von dem Typ
- Schreiben Sie eine statische Methode `printDurchschnittsalter`, die das Durchschnittsalter (in Klammern) für eine `HochschulPerson` auf der Konsole ausgibt. Verwenden Sie dazu `switch`.

- Aufzählungstyp durch Klasse realisiert
- daher können zusätzlich Methoden definiert werden
- Beispiel

```
enum Wochentag {  
    MO, DI, MI, DO, FR, SA, SO;  
    boolean istWochenende(){  
        return this == SA || this == SO;  
    }  
}
```

- Aufruf mit Enum-Wert vom Typ `Day` als Zielobjekt:

```
Wochentag heute = Wochentag.MO;  
if ( heute.istWochenende() ){  
    ...  
}
```

- liefert Enumwert mit dem Namen s
`static <Aufzählungstypname> valueOf(String s)`
- liefert Array mit allen Enumwerten
`static <Aufzählungstypname>[] values()`
- Index dieses Enumwertes gemäß Definitionsreihenfolge
 - erster Wert mit Index 0
`int ordinal()`

- für die Verwendung in mehreren Klassen ist auch eine Definition eines Aufzählungstyps in einer eigenen Datei möglich
 - wie bei jeder anderen Klasse
 - Datei *Wochentag.java*

```
public enum Wochentag {  
    ...  
}
```
- ansonsten Zugriff über die Klasse, in der der Aufzählungstyp definiert ist
 - in Methoden der eigenen Klasse nicht nötig
 - Klassendatei *Zeit.java*:

```
Zeit.Wochentag heute = Zeit.Wochentag.MO;
```

- Statische Objektvariablen
- Statische Methoden
- Aufzählungstypen