

Softwareentwicklung II (IB)

Klassenvariablen und statische Methoden

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 28.01.2020 17:34

Inhaltsverzeichnis

Klassenvariablen	2
Definition	2
Zugriff	2
Initialisierung	2
Konstanten	3
Benennung von Konstanten	3
Beispiel: Objektzähler	3
Beispiel: Seriennummern	4
Beispiel: Seriennummern in Java	4
Darstellung UML	5
Statische Methoden	5
Definition und Aufruf	5
main-Methode	5
Zugriffsmethoden	6
Statische Hilfsmethoden	7
Einschränkungen	7
Statischer Initializer	8

Klassenvariablen

Definition

- Bisher betrachtete Objektvariablen und Methoden beziehen sich ein bestimmtes Objekt (= Zielobjekt)
- **Klassenvariable**: einer ganzen Klasse zugeordnet, keinem einzelnen Objekt
- Definition einer Klassenvariablen: Wie normale Objektvariable + Modifier `static`

```
class Rational {  
    static int count;  
    ...  
}
```

Zugriff

- Zugriff mit Klassennamen statt Zielobjekt
`Rational.count = 0;`
- Klassenvariable existiert unabhängig von Objekten der Klasse
- Beispiel Mathematische Konstante: `Math.PI` = Klassenvariable `PI` in der Klasse `Math`
- Weitere Beispiele:
 - Wertebereichsgrenzen `Integer.MIN_VALUE`, `Integer.MAX_VALUE`, ...
 - Fluchtwerte `Double.POSITIVE_INFINITY`, `Double.NEGATIVE_INFINITY`, `Double.NaN`, ...

Initialisierung

- Initialisierung bei der Definition wie andere Objektvariablen

```
class Rational {  
    static int count = 0;  
    ...  
}
```

- Ohne explizite Initialisierung: Defaultwert, abhängig vom Typ
- Beispiel: `Rational.count` hätte auch ohne explizite Initialisierung den Defaultwert `0`
- Lebensdauer einer Klassenvariablen: gesamte Programmlaufzeit, unabhängig von Objekten

Konstanten

- Anwendung von Klassenvariablen: öffentliche Konstanten
- Beispiele: `Math.PI`, `Integer.MAX_VALUE`
- Werte öffentlicher Konstanten sollen sich nicht ändern: Modifier `static` und `final`
- Beispiel: Auszug aus der Definition der Klasse `Integer`:

```
class Integer {  
    static final int MAX_VALUE = 2_147_483_647;  
    static final int MIN_VALUE = -2_147_483_648;  
    ...  
}
```

- Öffentliche Konstanten müssen bei der Definition initialisiert werden (Alternative: statische Initializer)

Benennung von Konstanten

- Konvention zur Benennung von Konstanten:
 - nur Großbuchstaben
 - Wortteile mit Unterstrichen (`_`) getrennt
- Betrifft nur Konstanten, das heisst unveränderliche Klassenvariablen `static final ...`
- Beispiel: statt `MaxValue: MAX_VALUE`
- Alle anderen Namen im CamelCase, abgesehen von Typvariablen generischer Klassen und Methoden

Beispiel: Objektzähler

- Klassenvariablen für Methoden ebenso verfügbar wie Objektvariablen
- Aber: Nur ein einziges Exemplar für alle Objekte
- Beispiel: Anzahl `Rational`-Objekte mitzählen, Zähler im Konstruktor inkrementieren:

```
class Rational {  
    static int count = 0;  
    private final int numer;  
    private final int denom;  
    Rational() {  
        numer = 0;  
    }  
}
```

```
        denom = 1;
        count++;
    }
    ...
}
```

- Jeder Konstruktor inkrementiert die einzige, für alle Objekte selbe Klassenvariable
- Jederzeit Abruf der Anzahl bisher geschaffener Objekte:

```
System.out.printf("%d rationals created up to now.%n",
    Rational.count);
```

- Vor dem ersten Konstruktoraufruf: Anzahl Objekte = 0

Beispiel: Seriennummern

- Beispiel: Eindeutige Seriennummern für Objekte
- Idee:
 - Klassenvariable `nextSerial` speichert die jeweils nächste freie Seriennummer.
 - Im Konstruktor die nächste freie Seriennummer an die Objektvariable `serial` zuweisen.
 - Klassenvariable inkrementieren

Beispiel: Seriennummern in Java

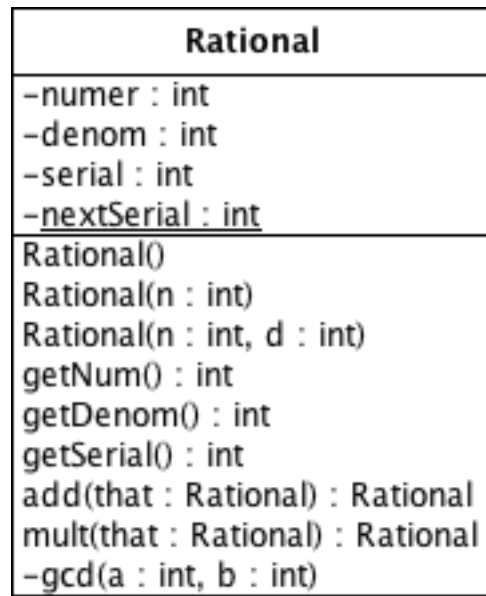
```
class Rational {
    private static int nextSerial = 0; // nächste freie Nummer
    private final int serial; // Nummer dieses Objektes
    ...
    Rational() {
        serial = nextSerial;
        nextSerial++;
    }
    int getSerial() {
        return serial;
    }
}
```

- Jedes Objekt hat „seine“ eindeutige Nummer:

```
Rational r = new Rational();
...
System.out.printf("Rational No. %d%n", r.getSerial());
```

Darstellung UML

- Statische Elemente in der UML unterstrichen
- Beispiel Seriennummern für Rationals



Rational UML mit statischem Element

Statische Methoden

Definition und Aufruf

- **Statische Methode:** richtet sich an ganze Klasse, kein bestimmtes Objekt
- Definition mit Modifier `static`

```
class Rational {
    static int getCount() {...}
}
```

- Zugriffsschutz (Modifier `private`) und Überladen wie normale Methoden
- Aufruf mit Klassennamen statt Zielobjekt

```
System.out.println(Rational.getCount());
```

main-Methode

- Schon von Anfang an benutzt: statische Methode `main` = Hauptprogramm:

```
class SomeClass {
    public static void main(String[] args) {
        ...
    }
}
```

- Vor `main` existiert noch kein Objekt: `main` muss statisch sein
- Wird beim Start eines Javaprogramms von der JVM automatisch in der Klasse gesucht, die auf der Kommandozeile genannt ist
- Beispiel: Das Kommando

```
$ java Classname
```

sucht in der Klasse `Classname` nach einer Methode `public static void main(String[] args)` und ruft diese auf

- `main` ansonsten normale Methode: kann zum Beispiel ...
 - ... überladen werden
 - ... vom Programm selbst aufgerufen werden
 - ... in mehreren verschiedenen Klassen definieren sein
 - ... mit einem anderen Ergebnistyp als `void` definiert werden
- vor `main` werden statische Initializer ausgeführt

Zugriffsmethoden

- Datenkapselung und unveränderliche Klassen betreffen auch Klassenvariablen
- `final`, Getter und Setter auch für Klassenvariablen sinnvoll
- Beispiel: Verbergen des Objektzählers hinter (statischem) Getter, verhindert unerwünschte Manipulation

```
class Rational {
    private static int count = 0;
    static int getCount() {
        return count;
    }
}
```

- Statischer Getter liefert vor dem ersten Konstruktoraufruf korrekt 0:

```
public static void main(String[] args) {
    System.out.println(Rational.getCount()); // gibt 0 aus
}
```

Statische Hilfsmethoden

- Einsatz von statischen Methoden als Hilfsmethoden, unabhängig von Objekten der Klasse
- Beispiel: Berechnung des ggT in der Klasse Rational:

```
class Rational {  
    static int gcd(int a, int b) {...}  
}
```

- Nutzbar ohne Rational-Objekte:

```
System.out.println(Rational.gcd(221, 255));
```

- Einige vordefinierte Klassen (beispielsweise Math) definieren nur statische Methoden
- Klasse selbst nebensächlich, dient nur zur Organisation einer Sammlung verwandter Methoden

Einschränkungen

- Statische Methode unterliegt Einschränkungen gegenüber normalen, nicht-statischen Methoden
 - Zugriff nur auf Klassenvariablen
 - Nur Aufruf anderer statischer Methoden
 - `this` nicht verfügbar
 - Wird statisch gebunden, nicht dynamisch
- Beispiel: Statische gcd-Methode hat keinen Zugriff auf Objektvariablen:

```
class Rational {  
    private int numer;  
    private int denom;  
  
    static int gcd() {  
        int a = numer; // Fehler - numer ist Objektvariable  
        int b = denom; // Fehler - denom ist Objektvariable  
        ...  
    }  
    ...  
}
```

Statischer Initializer

- Syntaktisch ähnlich wie (normaler) Initializer, markiert mit Modifier `static`

```
class Classname {  
    static {  
        ...  
    }  
}
```

- Wird einmal beim Laden der Klasse ausgeführt
- Erlaubt Vorbereitungen vor allen Methodenaufrufen (einschließlich `main`)
- Beispiel: Initialisierung von Klassenvariablen

```
class Rational {  
    private final static boolean debugMode;  
    static {  
        System.out.println("run in debug mode?");  
        debugMode = System.console().readLine().length() > 0;  
    }  
    public static void main(String[] args) {  
        if (debugMode) System.out.println("debug mode");  
        else System.out.println("production mode");  
    }  
}
```