universität innsbruck

Records

Programmiermethodik

Lukas Kaltenbrunner, Simon Priller
Universität Innsbruck

Motivation (1)

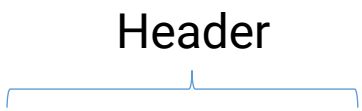
- Beispiele von Werteklassen:
  - Compound Keys bei Maps
  - DTOs (Data Transfer Objects)
  - Parameter Value Objects
  - Rückgabe von mehreren Werten
- In solchen Klassen muss typischerweise wiederholt fehleranfälliger Code implementiert werden.
  - Getter-Methoden
  - `equals`-Methode
  - `hashCode`-Methode
  - `toString`-Methode
- Bei der Modellierung dieser Klassen stehen die Daten im Fokus.

Motivation (2)

```java
public final class Point {
    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int x() { return x; }

    public int y() { return y; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Point point)) return false;
        return x == point.x && y == point.y;
    }

    @Override
    public int hashCode() {
        return Objects.hash(x, y);
    }

    @Override
    public String toString() {
        return "Point{x=%d, y=%d}".formatted(x, y);
    }
}
```

Records (1)

- Weitere Art einer Klasse mit welcher leichtgewichtige Datencontainer erstellt werden können.
- Records sind implizit `final`.
- Sie können nicht `abstract` sein.
- Die direkte Superklasse ist die abstrakte Klasse `Record`.
  - Es kann keine extends-Klausel verwendet werden!
- Die Deklaration besteht aus Name, Header und Rumpf.
- Im Header werden die Record-Komponenten angegeben.




```
public record Point(int x, int y) { }
```

Records (2)

- Der Compiler erstellt implizit folgende Members:
  - Für jede Record-Komponente wird ein privates, finales Feld (Komponentenfeld) sowie ein öffentlicher Getter mit demselben Namen und einem Rückgabetyp, welcher dem Typ der Record-Komponente entspricht, erzeugt.
  - Die Methoden equals und hashCode werden automatisch erstellt, wobei sichergestellt wird, dass zwei Records gleich sind, wenn die Komponenten die gleichen Werte haben.
  - Die Methode toString liefert einen String, welcher den Namen des Records sowie die Namen und Werte aller Komponenten aufweist.
- Der Zustand kann nicht von der API entkoppelt werden.

Konstruktoren (1)

- Die Regeln für Konstruktoren unterscheiden sich im Vergleich zu konkreten und abstrakten Klassen.
- Es wird kein default-Konstruktor bereitgestellt.
- Records haben immer einen kanonischen Konstruktor, der alle Komponenten übernimmt.
- Ein kanonische Konstruktor kann explizit als kompakter kanonischer Konstruktor implementiert werden.
  - Hierbei werden keine formalen Parameter angegeben.
  - Im Rumpf darf kein anderer Konstruktor aufgerufen, keine return-Anweisung verwendet und keine Zuweisung von Record-Komponenten durchgeführt werden.
  - Am Ende des Konstruktors wird jedem Komponentenfeld der Wert des entsprechenden formalen Parameters zugewiesen.

Konstruktoren (2)

```
public record Rational(long numerator, long denominator) {

    public Rational {
        if (denominator == 0) {
            throw new IllegalArgumentException();
        }
        if (denominator < 0) {
            denominator = -denominator;
            numerator = -numerator;
        }
        long gcd = gcd(numerator, denominator);
        numerator /= gcd;
        denominator /= gcd;
    }

    public Rational(long integer) {
        this(integer, 1);
    }

    private static long gcd(long x, long y) {
        return y == 0 ? Math.abs(x) : gcd(y, x % y);
    }
}
```

kompakter kanonischer Konstruktor

Konstruktoren (3)

- Es können weitere Konstruktoren angeboten werden.
  - Jeder nicht kanonische Konstruktor muss einen anderen Konstruktor aufrufen.
- Bei kanonischen Konstruktoren darf keine throws-Klausel angegeben werden.
- Die Sichtbarkeit von Konstruktoren muss mindestens den Zugriffsschutz der Record-Klasse aufweisen.

Members (1)

- In Record-Klassen können, abgesehen von folgenden Einschränkungen, dieselben Members wie in einer konkreten Klasse deklariert werden.
  - Es können keine Objektvariablen im Rumpf deklariert werden.
  - Der Exemplarinitialisierer kann nicht verwendet werden.
- Implizit bereitgestellte Methoden können explizit implementiert werden.
  - Die Deklaration muss genau übereinstimmen.
  - Getter, `equals` und `hashCode` sollten die Invariante einhalten.
- Invariante:
  - Gegeben sei eine Record-Klasse `record R(T1 c1, …, TN cn) {}`, `r1` ist eine Referenz auf ein Objekt von `R` und `r2` wurde durch `new R(r1.c1(), …, r1.cn())` erstellt.
  - Dann muss `r1.equals(r2)` zu `true` evaluieren.

Members (2)

```
public record Rational(long numerator, long denominator) {

    …
    public Rational multiply(Rational other) {
        return new Rational(numerator * other.numerator,
                denominator * other.denominator);
    }

    public Rational multiply(long integer) {
        return multiply(new Rational(integer));
    }
}
```

```
public class RationalApplication {

    public static void main(String[] args) {
        Rational r1 = new Rational(2);
        Rational r2 = new Rational(2, 3);
        Rational r3 = r2.multiply(3);
        System.out.println(r1.numerator());
        System.out.println(r1.denominator());
        System.out.println(r1);
        System.out.println(r1.equals(r3));
    }
}
```

Ausgabe:

```
2
1
Rational[numerator=2, denominator=1]
true
```

src/at/ac/uibk/pm/objectorientation/rectangle/RectangleApplication.java

Quellen

- Gavin Bierman: **JEP 395: Records**, besucht am 15.04.2022, https://openjdk.java.net/jeps/395
- James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith, Gavin Bierman: **The Java® Language Specification (***Java SE 17 Edition***)**, Oracle, 2021
- Michael Inden: **Der Weg zum Java-Profi: Konzepte und Techniken für die professionelle Java-Entwicklung**, dpunkt.verlag, 5. Auflage, 2021
- Christian Ullenboom: **Java ist auch eine Insel: Einführung, Ausbildung, Praxis**, Rheinwerk Verlag, 16. Auflage, 2022 (Java 17)