**PR2 – Formular für Lesenotizen**

**SS2021**

|  |  |  |  |
| --- | --- | --- | --- |
| Nachname  Lushaj | Vorname  Detijon | Matrikelnummer  1630149 | Abgabedatum:  21.03.21 |

**Instanzmethode toString**

**public** **class** S {

String name;

**int** nummer;

**public** String toString() {

**return** name + "(" + nummer + ")";

}

}

**L.2.3 Konstruktoren**

**Konstruktor:** Initialisiert den Zustand eines neuen Objekts.

**Standardkonstruktor**

Java automatisch ein Standard Konstruktor bereitgestellt, der alle Attribute auf 0 initialisiert.

**L.2.3.6 Objekte mit Startzustand ungleich 0, 0.0, ...**

Es ist nicht immer die beste Idee ist, die Initialisierung direkt mit der Deklaration zu erledigen. Denken Sie immer auch an die Möglichkeit, die Initialisierung in Konstruktoren separat zu programmieren.

**public** **class** Studi {

**int** semester = 1;

String name;

**public** Studi(String initialName) {

name = initialName; }

**public** Studi() {

name = "NN";

}

...

}

**L.2.4 Kohärente(zusammenhängend) Schnittstellen**

• Die Schnittstelle einer Klasse ist kohärent.

– Ihre Methoden hängen logisch zusammen und passen zueinander.

– Minimal und vollständig: Es soll keine Methode fehlen oder zu viel sein.

• Inkohärente Klassen sind in mehrere kohärenten Klassen zu zerlegen.

**Kapseln:** Implementierungsdetails vor Clients verbergen.

Kapselung erzwingt Abstraktion.

* Trennung der externen Sicht (Verhalten) von der internen Sicht (Zustand)
* Schützt die Integrität der Objektdaten.

Vorteile:

* Schützt ein Objekt vor unerwünschtem Zugriff
  + Beispiel: Verhinderung eines unerlaubten Zugriffs auf einen Kontostand.
* Abstraktion zwischen Objekten und Klienten
* Man kann die Implementierung einer Klasse später ändern
  + Beispiel: Loc könnte später in Polarkoordinaten (r, θ) umgeschrieben werden, unter Beibehaltung der alten Methoden mit ihren alten Typen
* Man kann den Objektzustand überwachen (sog. Invarianten)
  + Beispiel: Erlaubt sind nur Konten mit Kontostand ≥ 0
  + Beispiel: Erlaubt sind nur Dates mit einem Monat zwischen 1 und 12.
* Kapselung ist eine mögliche Umsetzung des Geheimnisprinzips (vgl. PR1).

**Privates Attribut:** Ein Attribut, das von außerhalb der Klasse nicht unmittelbar zugreifbar ist.

**L Verdeckung von Variablen**

**this:** Schlüsselwort, das auf den impliziten Parameter verweist.

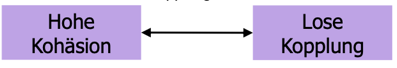
* this ist eine Variable, die immer auf das Objekt referenziert, dessen Methode aufgerufen wird.

**Shadowing / Verdeckung**: Zwei Variablen gleichen Namens im gleichen Gültigkeitsbereich.

**Verkettung von Konstruktoren** public Loc() { **this(0, 0); // calls (x, y) constructor**}

**Kohäsion ("Zusammenhang"):** beschreibt, inwiefern eine Klasse genau eine Aufgabe bzw. ein Konzept repräsentiert.

Jede Klasse soll eine, maximal zwei Verantwortlichkeiten besitzen.



**Kopplung**: Grad der Abhängigkeit einer Klasse von einer anderen

**Bündelung von Daten und Operationen:**   
Daten und die Methoden, die diese Daten benötigen, sollten in derselben Klasse stehen.

**L.2.6 Entwurfsprinzipien und Verantwortung**

Wir haben bisher **drei wichtige Entwurfsprinzipien** kennen gelernt.

* Eine Klasse soll eine **hohe innere Kohäsion** besitzen, d. h. sie repräsentiert genau ein Konzept.
* **Zwischen Klassen** soll eine **lose Kopplung** herrschen.
* **Daten und die Methoden**, die diese Daten benötigen, sollten **in derselben Klasse stehen**.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**CRC-Karte (Class-Responsibility-Collaboration-Karte):**

Eine Karteikarte mit Namen, Verantwortlichkeiten und einer Liste von kooperierenden Klassen.

**L.2.8 Zugriffsschutz für Attribute**

• Attribute sind immer **private**

– von außen gibt es keinen direkten Zugriff auf die Attribute

– der Zugriff auf die Attribute erfolgt nur über **public**-Methoden

• Attribute, die nach Konstruktor nicht mehr geändert werden, werden **final** deklariert:   
class Aktie { private final String isin; ...

**Konstruktoren überladen**

* Sie unterscheiden sich anhand ihrer Parameterstruktur (Anzahl der Parameter mit ihren Datentypen). Die Konstruktoren müssen unterschiedliche Signaturen haben.

**Methoden-Signatur:** Besteht aus Methodenname, Anzahl der Parameter und Reihenfolge der Parametertypen.

**Überladen (overload):** Definition mehrerer Methoden gleichen Namens mit unterschiedlicher Signatur.

public Loc(Loc loc) {

x = loc.getX();

y = loc.y;

}

Ein Bild, das Text, Uhr enthält.

Automatisch generierte Beschreibung

**Spezielle Konstruktoren**

**Copy-Konstruktor**: Konstruktor, der ein vorhandenes Objekt als Vorlage für die Initialisierung verwendet.

• Ein Copy-Konstruktor hat ein Objekt derselben Klasse als Parameter

• Dieser Konstruktor erzeugt ein neues Objekt (mit neuem Speicherplatz) als Kopie.

**Kopier-Varianten**

• Objekte können wiederum Objekte enthalten

public class Einwohner {

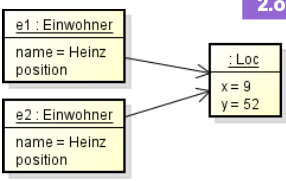
private String name;

private Loc position;

public Einwohner(String name, Loc position) {

this.name= name;

this.position= position; }



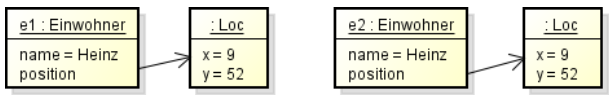
**Flaches Kopieren**: einfache Attribute werden kopiert, Referenzen werden kopiert (keine Erzeugung neuer Objekte).

public Einwohner(Einwohner e) {

name= e.name;

position= e.position; }

**Tiefes Kopieren**: einfache Attribute werden kopiert, Objekte werden neu erzeugt und inhaltlich kopiert.

public Einwohner(Einwohner e) {

name= e.name;

position= new Loc(e.position);

}

Einwohner e1= new Einwohner("Heinz", new Loc(9,52)); Einwohner e2= new Einwohner(e1);

• Unterschied: Verhalten bei Manipulation der Position von e1

– Flache Kopie e2: zieht mit e1 an neue Position um

– Tiefe Kopie e2: behält alte Position bei

public void setMorgen() {

if (monat == 2) {

if (istSchaltjahr() == true && tag == 29 ) {

tag = 1;

monat++;

} else if (istSchaltjahr() == false && tag == 28) {

tag = 1;

monat++;

} else {

tag++;

}

} else if (tag < 30) {

tag++;

} else if (tag == 31 && monat == 12) {

monat = 1;

tag = 1;

jahr++;

} else {

if (tag == 31 && (monat == 1 || monat == 3 || monat == 5 || monat == 7 || monat == 8 || monat == 10)) {

tag = 1;

monat++;

} else if (monat == 1 || monat == 3 || monat == 5 || monat == 7 || monat == 8 || monat == 10 || monat == 12) {

tag++;

}

if (tag == 30 && ( monat == 4 || monat == 6 || monat == 9 || monat == 11 )) {

tag = 1;

monat++;

}

}

}

}

**Besonderheit String**

• Hätten wir das hier schreiben müssen? name= new String(e.name); NEIN

• Da **Strings immutable (unveränderbar**), ist es unerheblich, ob man eine Kopie anlegt.

**Methoden überladen**

• Es können nicht nur Konstruktoren, sondern beliebige Methoden überladen werden.

public class Datum {

int tag;

int monat;

int jahr;

StringBuilder sb = new StringBuilder();

Formatter formatter = new Formatter(System.out,new Locale("de", "DE"));

public boolean istSchaltjahr() {

return jahr%4 == 0 && (jahr%100 != 0 || jahr%400 == 0);

}

public String getDeutscheSchreibung() {

return String.format("%02d", this.tag) + "."+ String.format(ss, this.monat) + "."+ this.jahr;

}

public String getAmerikanischeSchreibung() {

return String.format("%02d", this.tag) + "/"+ String.format(ss, this.monat) + "/"+ this.jahr;

}