CARL VON OSSIETZKY universität OLDENBURG

Sommersemester 2020

Objektorientierte Modellierung und Programmierung

Dr. C. Schönberg

CARL VON OSSIETZKY universität OLDENBURG

Klassen und Objekte

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

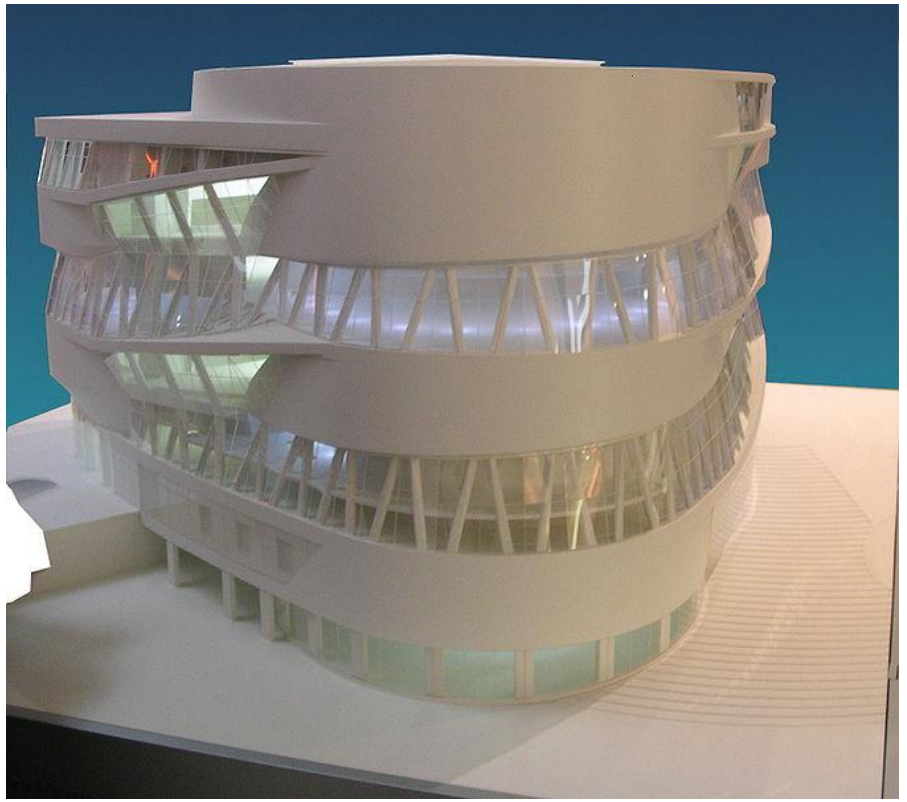
- Klassen und Objekte
  - in Java
  - in der UML
- Attribute, Methoden, Signaturen, Konstruktoren, Konstanten
- Lebenszyklus von Objekten
- Objektdiagramme
- Klassendiagramme

CARL VON OSSIETZKY universität OLDENBURG

Modelle

Architektur

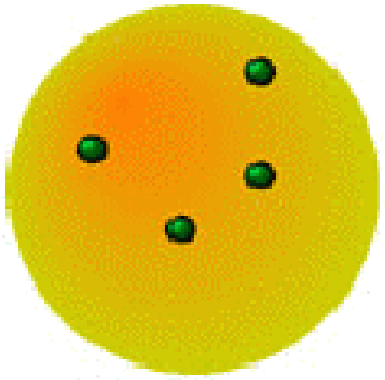
- Beschreibung von Arbeitsanweisungen zur Erstellung von Bauwerken oder Gebäudeteilen
- Frühzeitige Visualisierung von Aussehen und Funktionalität geplanter Bauwerke



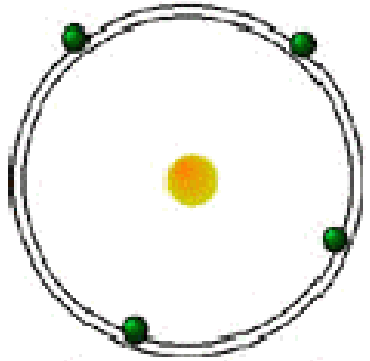
Norbert Schnitzler:
https://de.wikipedia.org/wiki/Datei:Mercedes_Museum_Modell_3.jpg

Physik

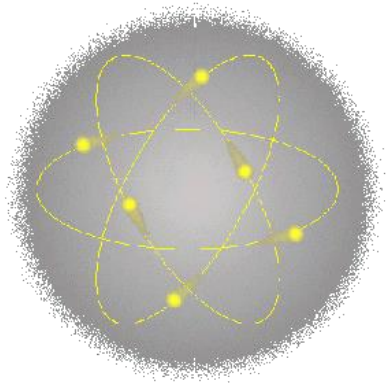
- Beschreibung beobachtbarer Natur-Erscheinungen
- Deutung und Erklärung physikalischer Gesetze
- Vorhersage über den Ablauf neuer Experimente



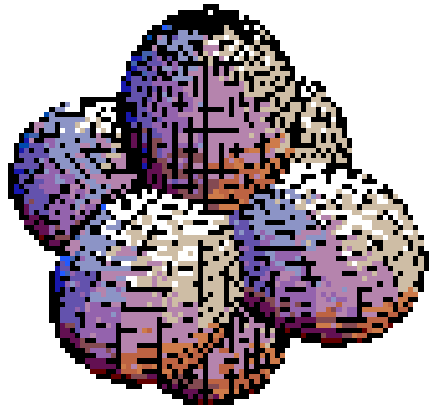
Atommodell
nach Thomson



Atommodell
nach Rutherford



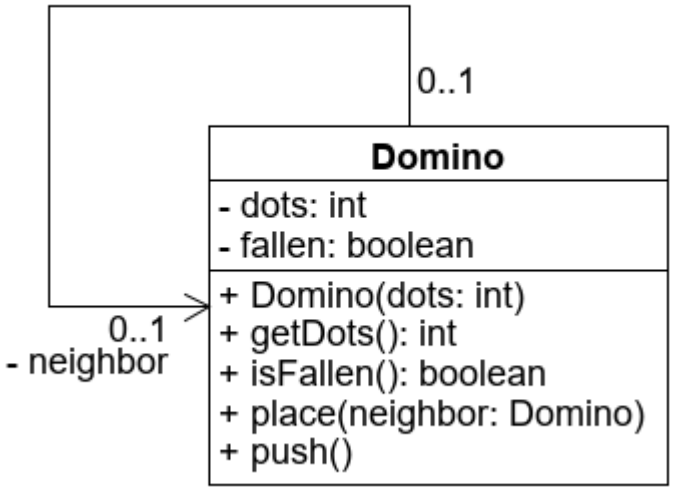
Atommodell
nach Bohr

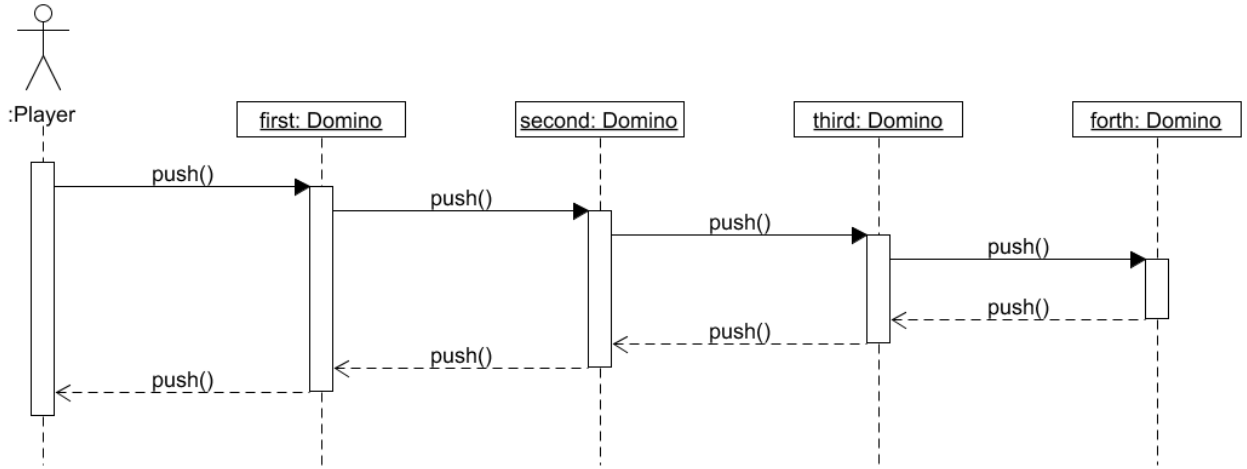


Orbital-
Modell

Softwaretechnik

- Erhebung und Beschreibung der Anforderungen an Softwaresysteme
- Beschreibung (Spezifikation) von Softwaresystemen





Verwendung von Modellen

- Abbildungsfunktion von Modellen
  - Nachbild: Beschreibung eines Ausschnitts einer Realität
  - Vorbild: Beschreibung eines zu erstellenden Systems
- Anwendung von Modellen
  - Beschreibung und Erklärung von Systemen
  - Dokumentation von Systemen
  - Gewinn von Erkenntnissen über Systeme
  - Erhebung von Systemzusammenhängen
  - Kommunikation über komplexe Systemzusammenhänge

Definition: Modell

- „Ein Modell ist ein zielgerichtetes Abbild eines Systems, das
  - zum einen ähnliche Beobachtungen und Aussagen ermöglicht wie dieses System und
  - zum anderen diese Realität durch Abstraktion auf die jeweils problembezogenen relevanten Aspekte vereinfacht.“

[Apostel, 1960; Troitzsch, 1990]

CARL VON OSSIETZKY universität OLDENBURG

Objektorientierte Modellierung

Objektorientierung

- Grundidee
  - Objekte kapseln Zustand und Verhalten
- Objekte
  - sind eindeutig identifizierbar
  - besitzen Eigenschaften (innerer Zustand, Attribute)
  - besitzen Verhalten (Methoden, Operationen)
  - kapseln ihren Zustand, der nur durch Botschaften von außen zugreifbar ist (Aufruf von Methoden)

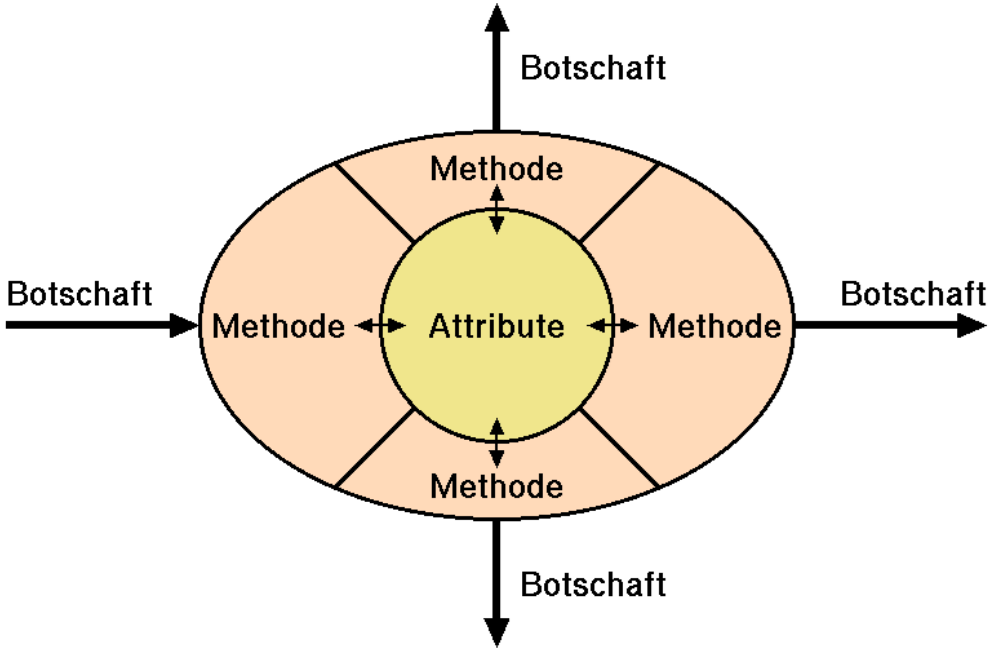


Abb. entnommen aus [Partsch, 1998, S. 130]

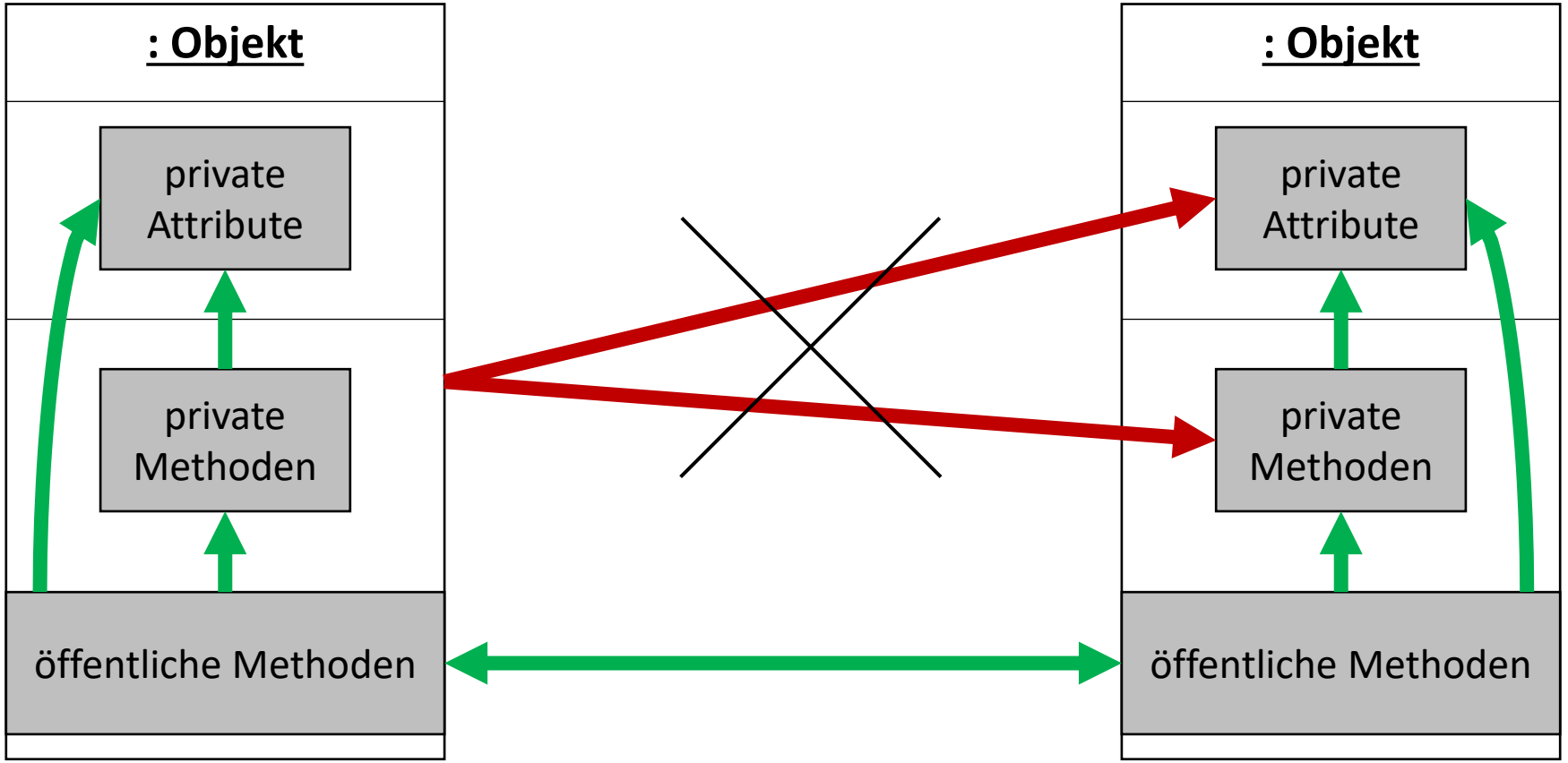
CARL VON OSSIETZKY universität OLDENBURG

Objektidentität

- Jedes Objekt besitzt einen eindeutigen Objektbezeichner (OID)
- OID wird bei Instanziierung des Objekts vergeben
- OID ist unveränderlich
- OID ist unabhängig vom aktuellen Objektzustand

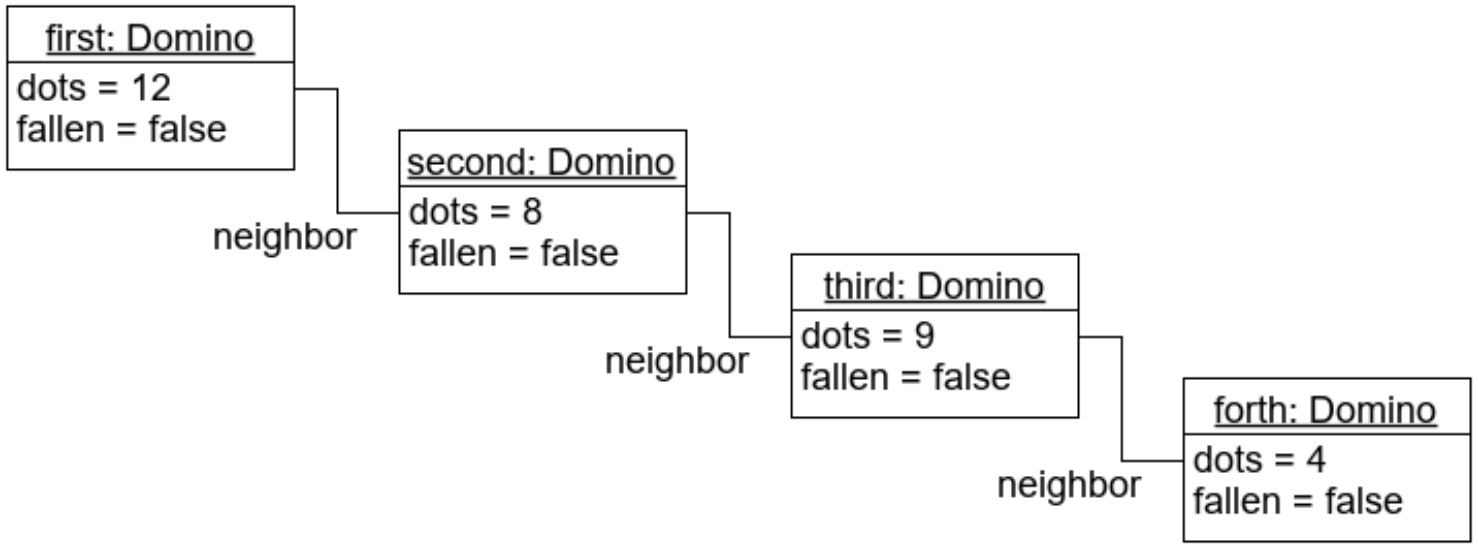
Objektinteraktion (Kapselung)

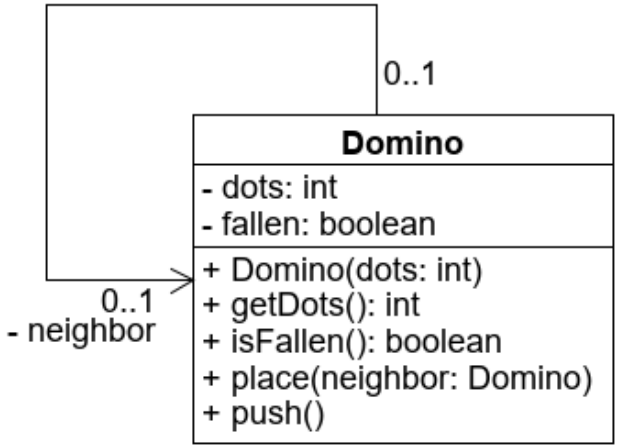
- Objekte kapseln Zustand und Verhalten
- Objekte interagieren miteinander durch Botschaften



Klassen

- Fassen gleichartige Objekte zusammen





Klassen und Objekte

- Jedes Objekt ist Instanz genau einer Klasse (in Java)
- Objekte bleiben während ihrer Lebenszeit Instanz derselben Klasse (in Java)
- Klassen legen für ihre Instanzen fest:
  - Eigenschaften/Zustand
    - Attributbezeichner und Wertebereich
  - Verhalten
    - Schnittstellen und Implementierung
- Objekte legen fest:
  - Eigenschaften/Zustand
    - konkrete Werte
  - Verhalten
    - konkrete Ausführung auf aktuellem Zustand

```
private int dots;
private boolean fallen;
private Domino neighbor;
```

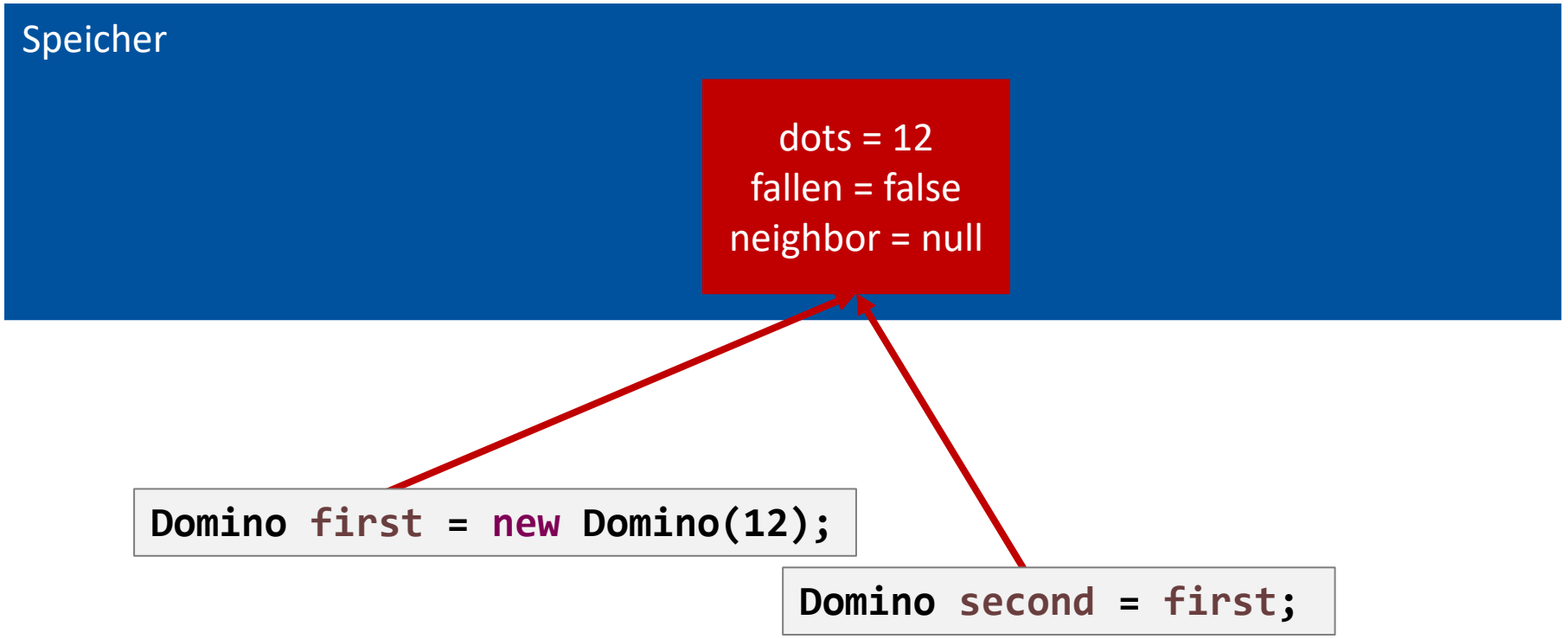
```
public void place(Domino neighbor) {
    this.neighbor = neighbor;
    fallen = false;
}
```

```
Domino first = new Domino();
first.setDots(12);
Domino second = new Domino(3);
first.setNeighbor(second);
```

CARL VON OSSIETZKY universität OLDENBURG

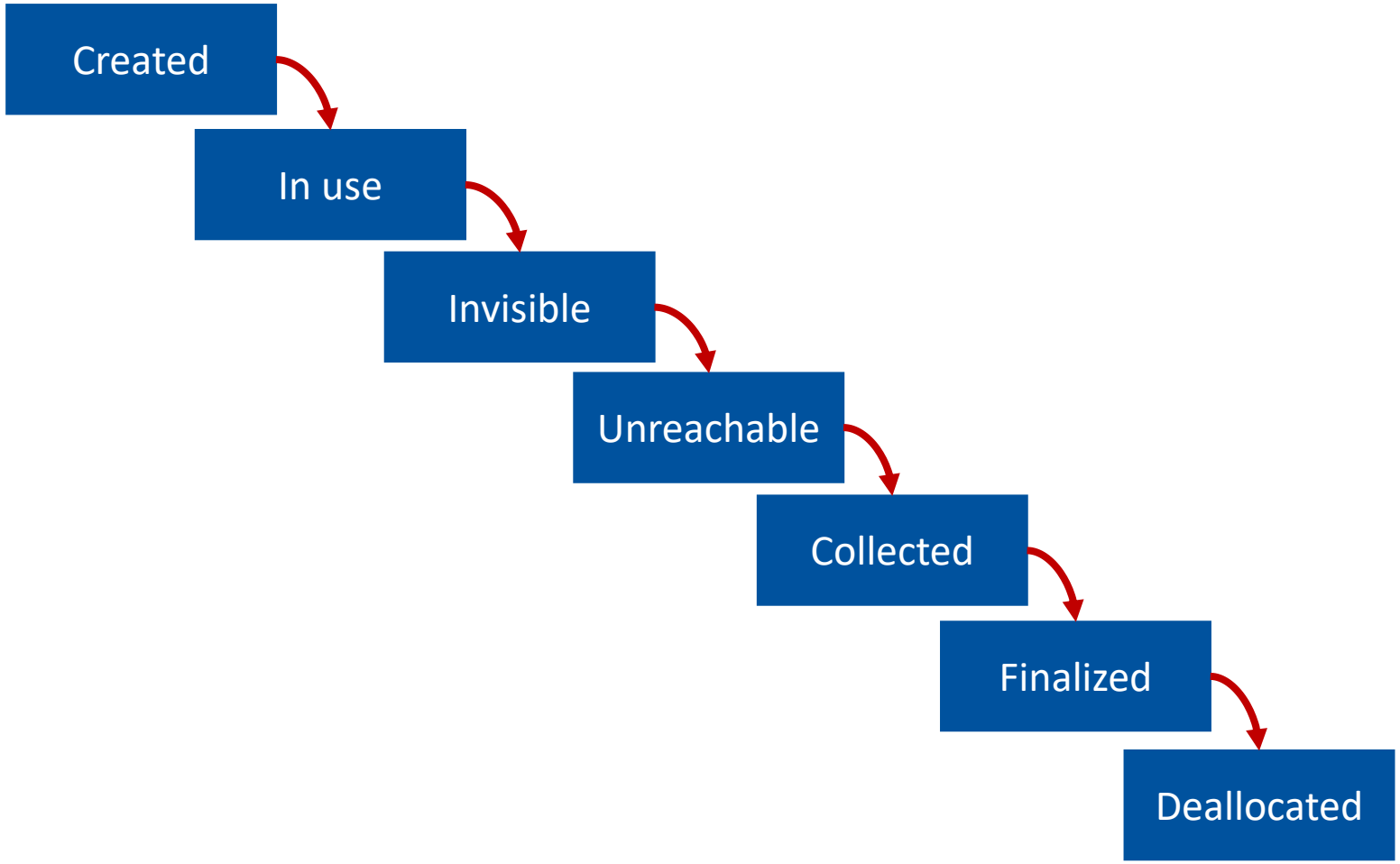
Lebenszyklus von Objekten

Objekte und Objektvariablen



`first` ist die Objektvariable,
die auf das mit **`new Domino(12)`** erzeugte Objekt verweist.

Objekt-Lebenszyklus in Java



Objekt-Lebenszyklus in Java (2)

- Created
  - Speicher für das Objekt wird bereitgestellt („allocated“)
  - das Objekt wird ggf. einer Variablen zugewiesen
- In use
  - solange die Objektvariable erreichbar ist, ist es „in use“
- Invisible / Unreachable
  - die Objektvariable ist nicht mehr erreichbar
- Collected
  - das Objekt wurde vom Garbage Collector eingesammelt
- Finalized
  - Freigabe von Ressourcen
- Deallocated
  - Speicher für das Objekt wird freigegeben („de-allocated“)

Garbage Collection: Einfacher Ansatz

- Referenzen auf Speicherbereich zählen
- Zuweisung zu einer Variablen → Zähler erhöhen

```
Domino second = new Domino(8);
```

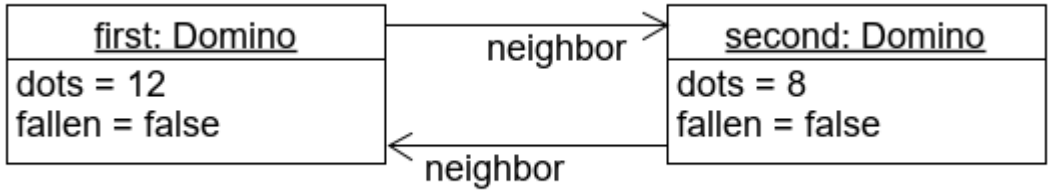
```
first.neighbor = second;
```

- Zuweisung löschen oder ändern → Zähler verringern

```
second = null;
```

```
first.neighbor = third;
```

- Problem: zyklische Referenzen

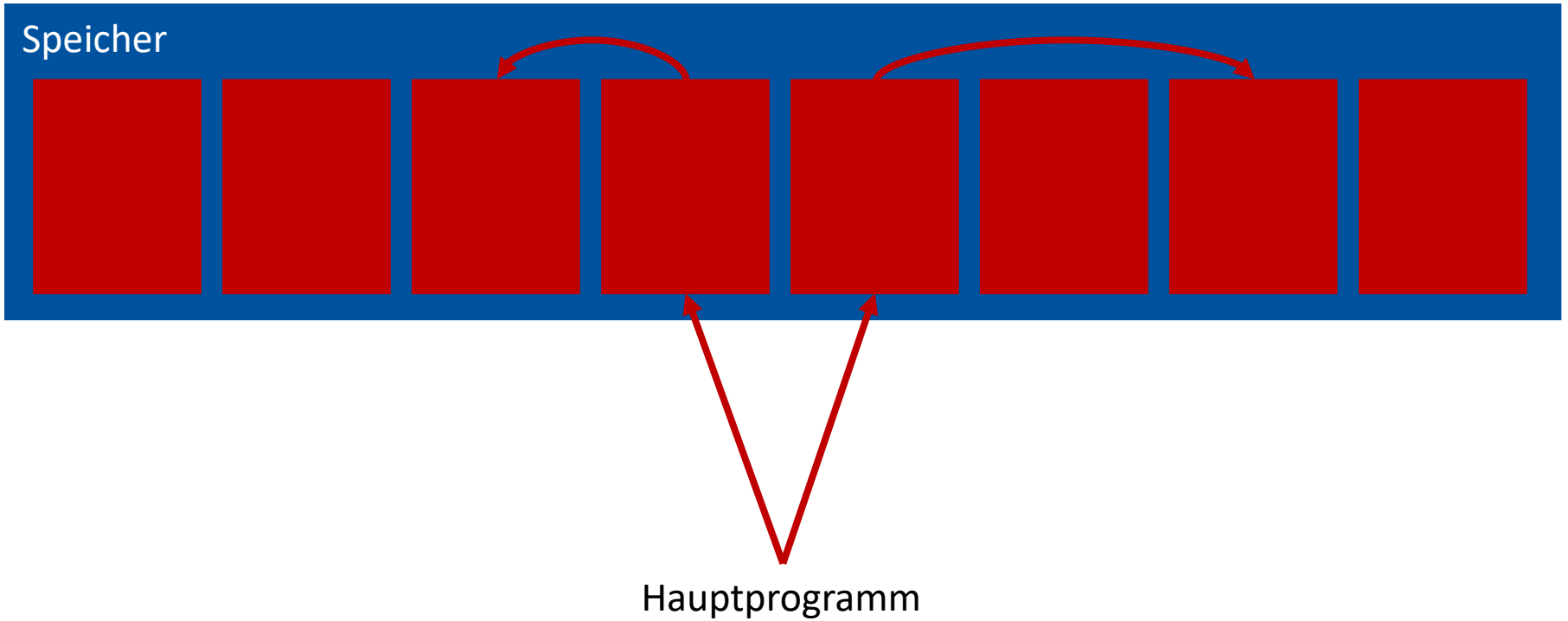


CARL VON OSSIETZKY universität OLDENBURG

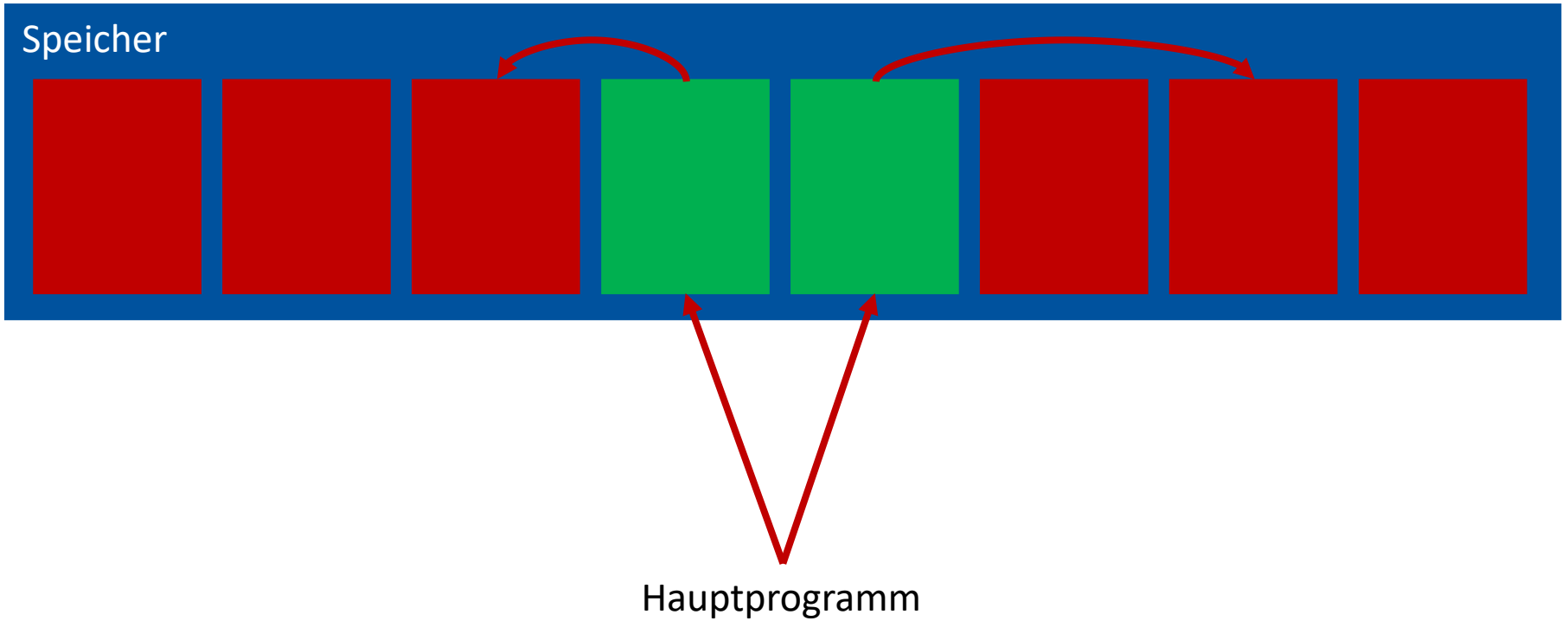
Garbage Collection: Java

- Wird automatisch im Hintergrund ausgeführt
- Programmausführung hält an während der GC („stop-the-world“)
- Mark-and-Compact
  - Markiere alle vom Hauptprogramm aus referenzierten Objekte (mark)
  - Markiere alle von diesen Objekten aus referenzierten Objekte (mark)
  - Wiederhole, bis keine weiteren Objekte mehr erreicht werden
  - Prüfe alle Objekte im Speicher und lösche die, die nicht markiert sind (sweep)
  - Verschiebe alle verbliebenen Objekte an den Anfang des Speichers (compact)

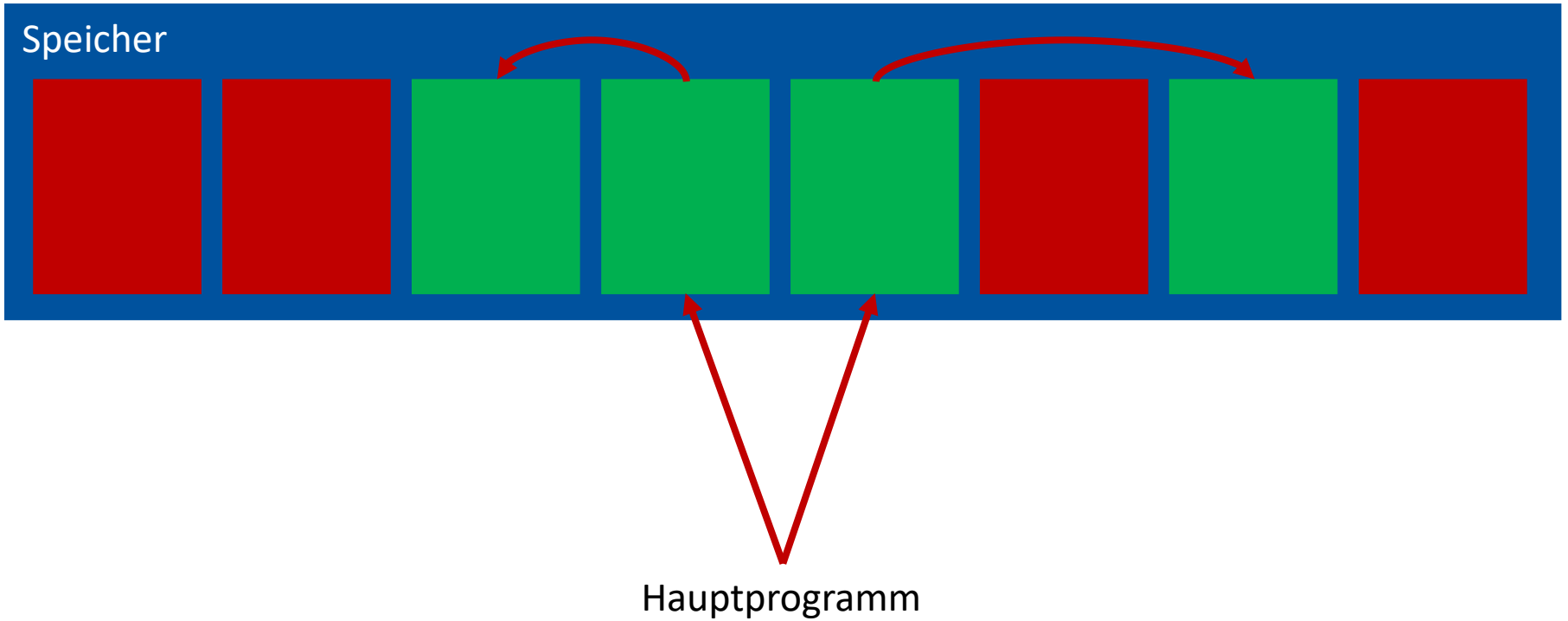
Beispiel: Garbage Collection



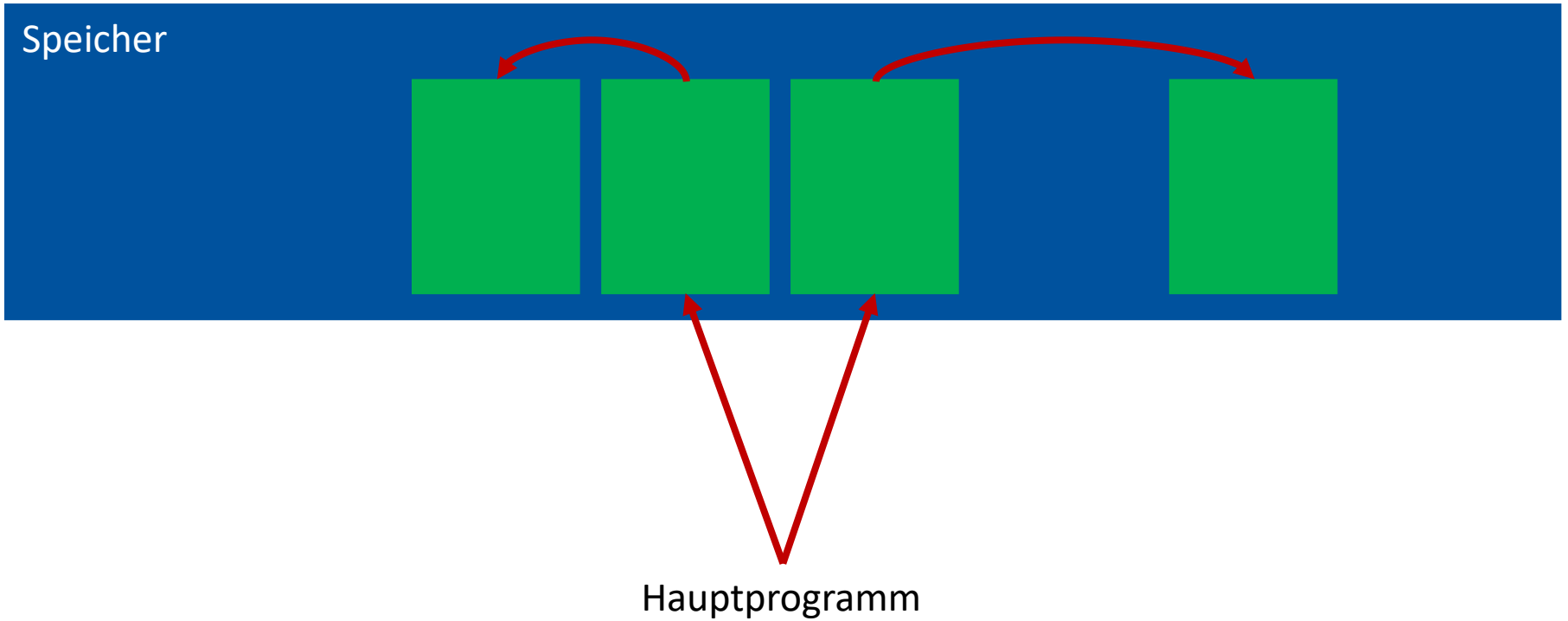
Beispiel: Garbage Collection (mark)



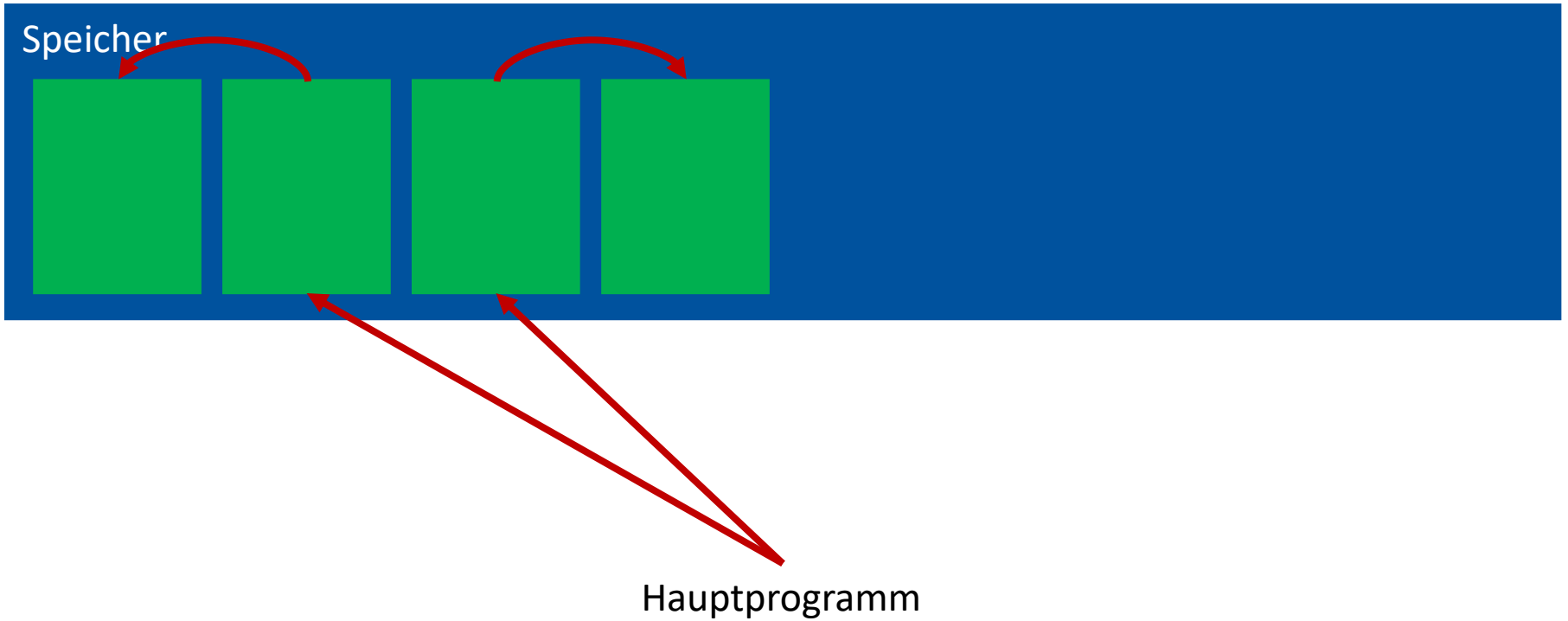
Beispiel: Garbage Collection (mark)



Beispiel: Garbage Collection (sweep)



Beispiel: Garbage Collection (compact)

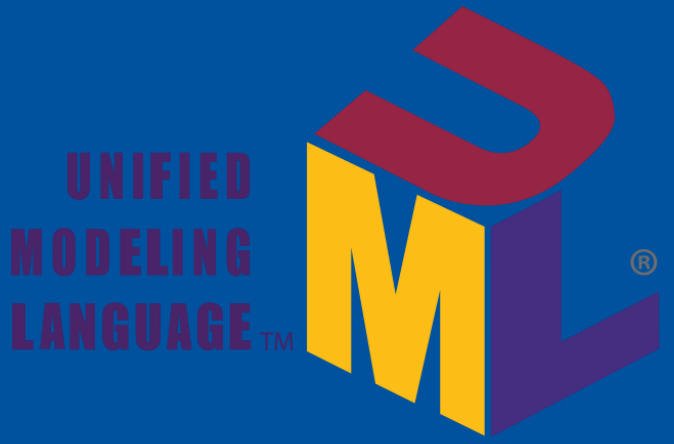


CARL VON OSSIETZKY universität OLDENBURG

Garbage Collection: Java (2)

- Generational Garbage Collection
- Annahme
  - Es gibt viele kleine Objekte mit sehr kurzer Lebensdauer
  - Es gibt wenige große Objekte mit langer Lebensdauer
- Ansatz
  - Aufteilung des Speichers in Bereiche Jung und Alt
  - Schnelle Variante Mark-and-Copy für den Bereich Jung
    - hohe Volatilität → oft aufgerufen
  - Gründliche Variante Mark-and-Compact für den Bereich Alt
    - selten aufgerufen
    - Speicherbereichsoptimierung (compact) wichtig

CARL VON OSSIETZKY universität OLDENBURG



Unified Modeling Language (UML)

http://www.omg.org/spec/UML/2.5/PDF

CARL VON OSSIETZKY universität OLDENBURG

Zielsetzung der UML

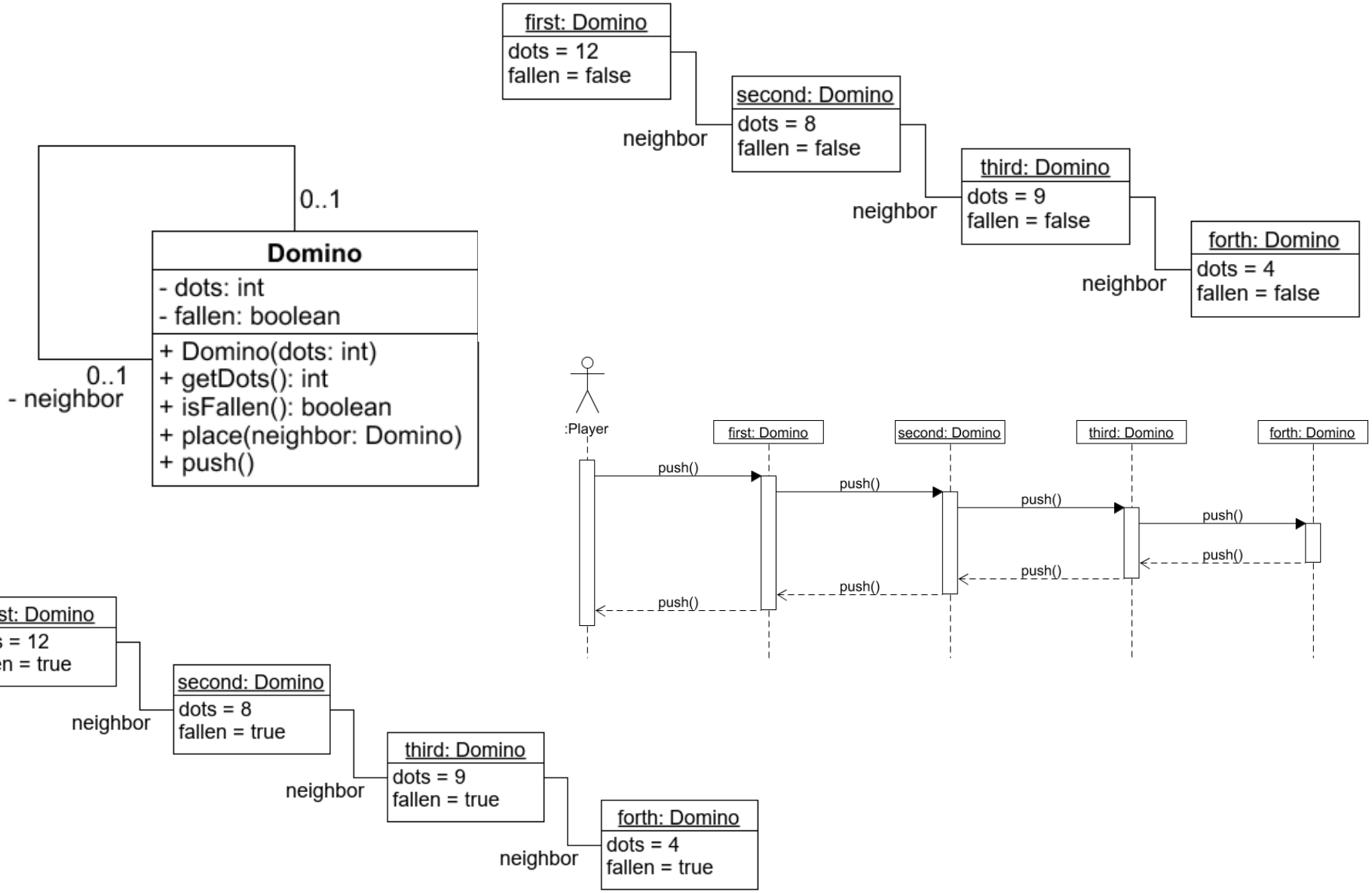
- Lingua Franca der Software-Modellierung
- Bereitstellung standardisierter Beschreibungsmittel zur objektorientierten Modellierung von Softwaresystemen
  - Spezifikation, Entwurf, Planung
  - Konstruktion
  - Visualisierung
  - Dokumentation

CARL VON OSSIETZKY universität OLDENBURG

Multi-Perspektivische Modellierung

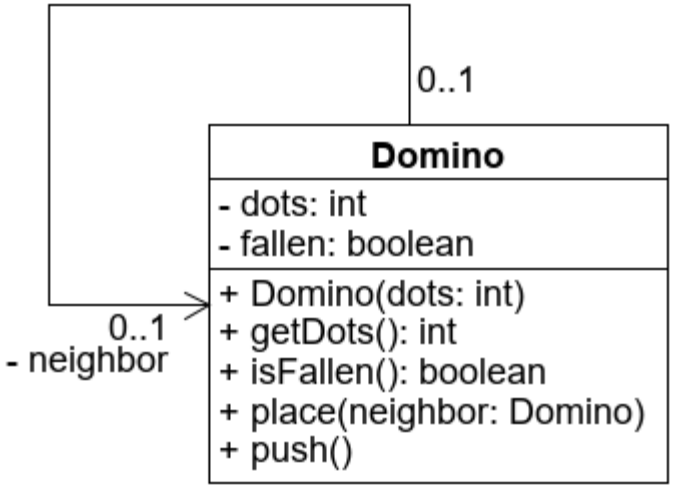
- Softwaresysteme und Anforderungen an Softwaresysteme werden aus verschiedenen Sichten beschrieben
  - Hervorheben bestimmter Systemaspekte
  - Ausblenden anderer Systemaspekte
- Sichten-orientierte Modellierung ermöglicht die Strukturierung und Zerlegung komplexer Modelle in kleinere, überschaubare Teilmodelle
- Eine vollständige Darstellung aller Systemaspekte erlaubt nur die gemeinsame Betrachtung aller Sichten

Beispiel: Perspektiven

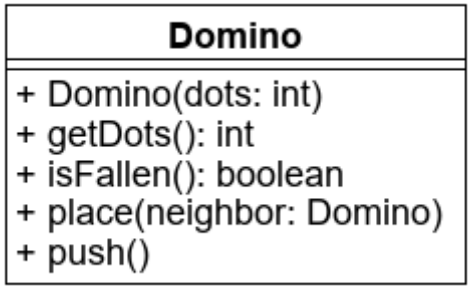


Beispiel: Perspektiven (2)

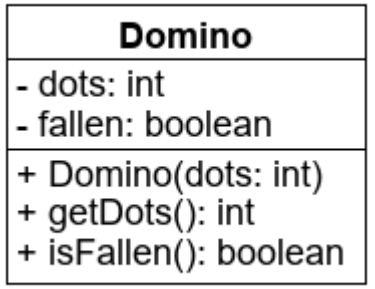
Weiterentwicklung



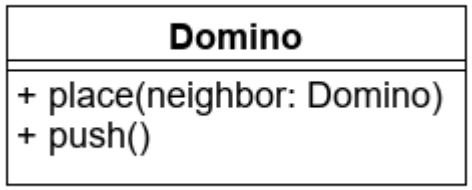
Verwendung



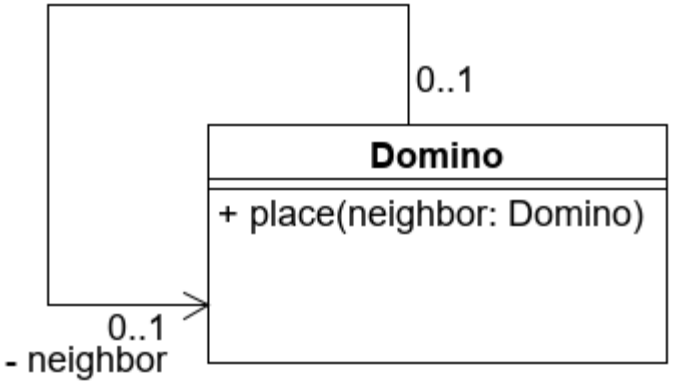
Datenhaltung



Verhalten



Interaktion



CARL VON OSSIETZKY universität OLDENBURG

Sichten

- Die statische Sicht betont Dinge (Objekte) und deren Beziehungen innerhalb von Softwaresystemen und Organisationen
- Die dynamische Sicht betont das zeitliche und logische Verhalten von Softwaresystemen und Organisationen

Sprachen der UML 2

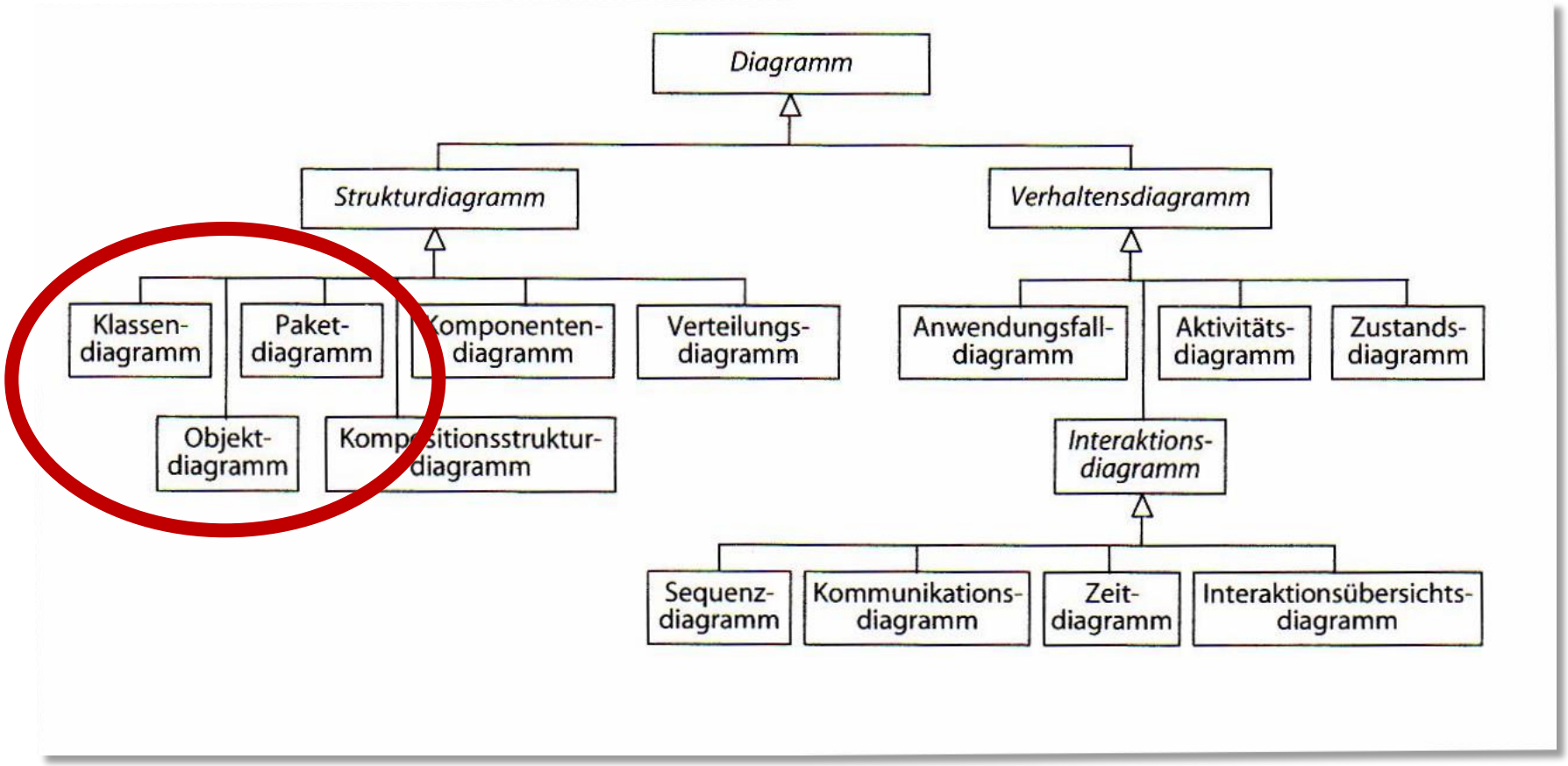
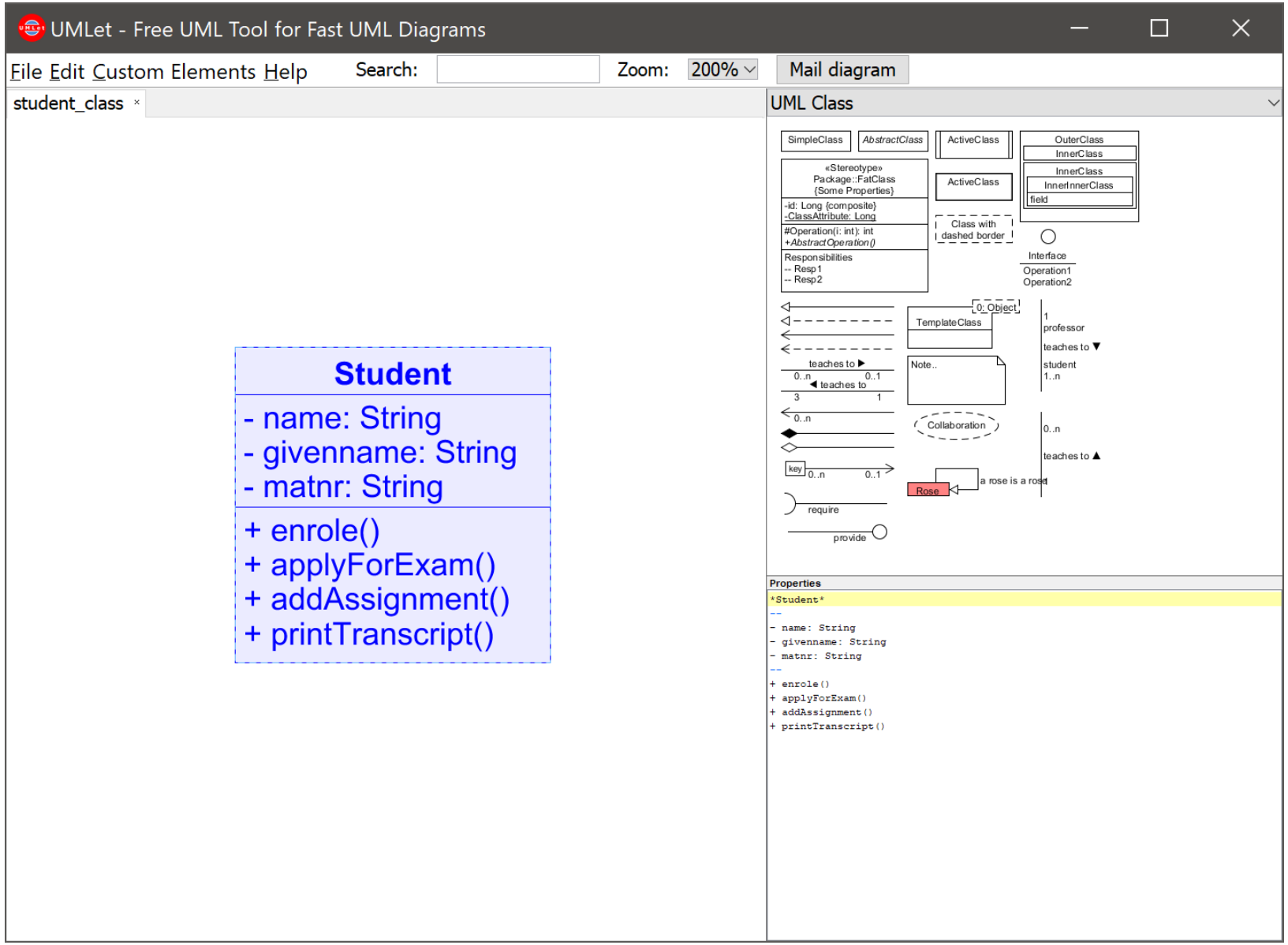


Abb. entnommen aus [Hitz et al., 2005, S. 8]

Modellierungswerkzeug

- UMLet
  - http://www.umlet.com/
  - Einfaches Werkzeug für verschiedene UML Sprachen
  - Reines Zeichenwerkzeug
    - keine Syntax-/Semantikprüfung
    - kaum Codeimport/-export
- Alternativen
  - Visio
  - Pen and Paper
- Mächtigere, aber komplexere Werkzeuge
  - IBM Rational Software Architect (RSA)
  - Visual Paradigm
  - UML Designer

UMLet Oberfläche



CARL VON OSSIETZKY universität OLDENBURG

Objektdiagramme

Objekte

Objekte in der Realität

Objekte in Java

```
Domino first  = new Domino(12);
Domino second = new Domino(8);
Domino third  = new Domino(9);
Domino fourth = new Domino(4);
```

Objekte in der UML

first: Domino
dots = 12 fallen = false

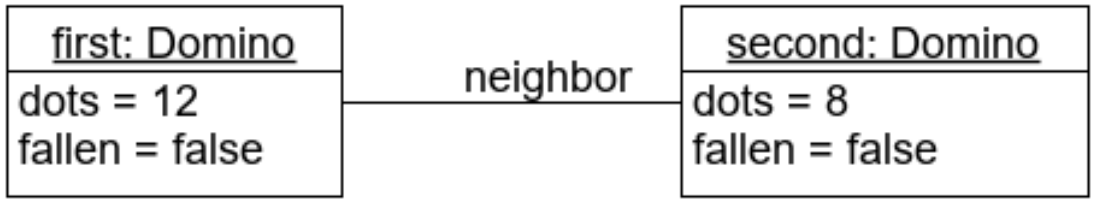
second: Domino
dots = 8 fallen = false

third: Domino
dots = 9 fallen = false

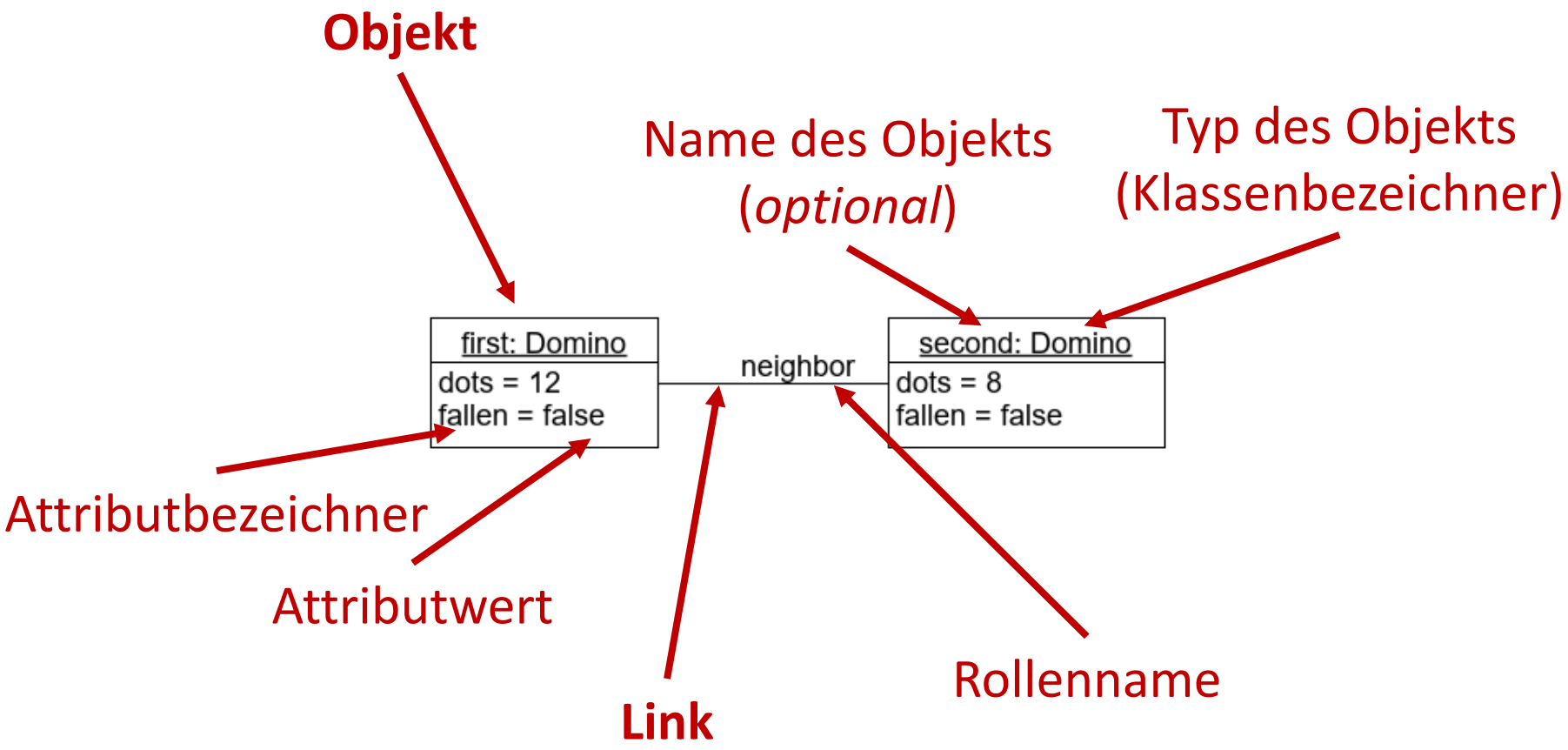
forth: Domino
dots = 4 fallen = false

Beziehungen zwischen Objekten

```
first.place(second);
```

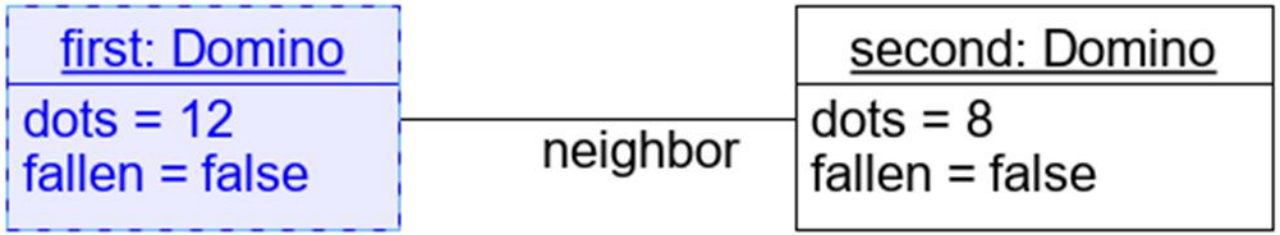


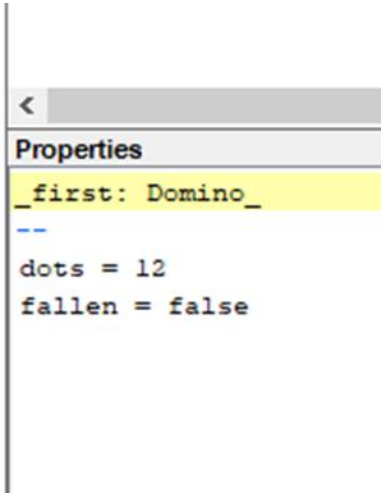
Aufbau von Objektdiagrammen



Modellierung mit UMLet

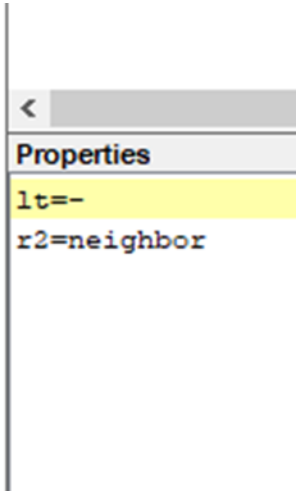
```
_underlined text_
-- (Trennlinie)
```





Modellierung mit UMLet (2)

`lt=-` *(durchgezogene Linie)*



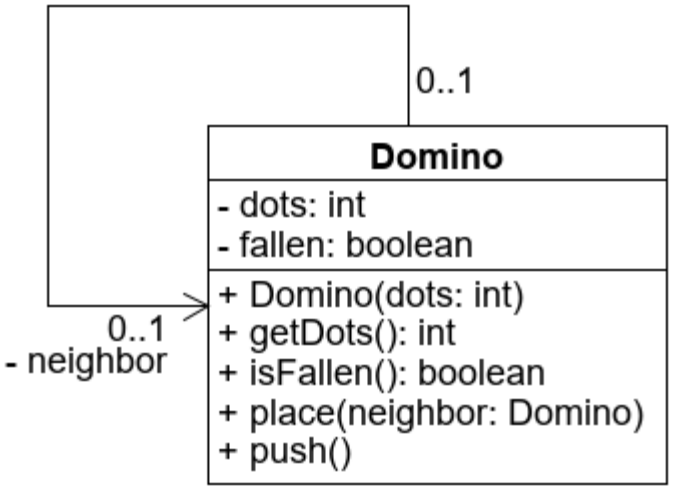
CARL VON OSSIETZKY universität OLDENBURG

Klassendiagramme

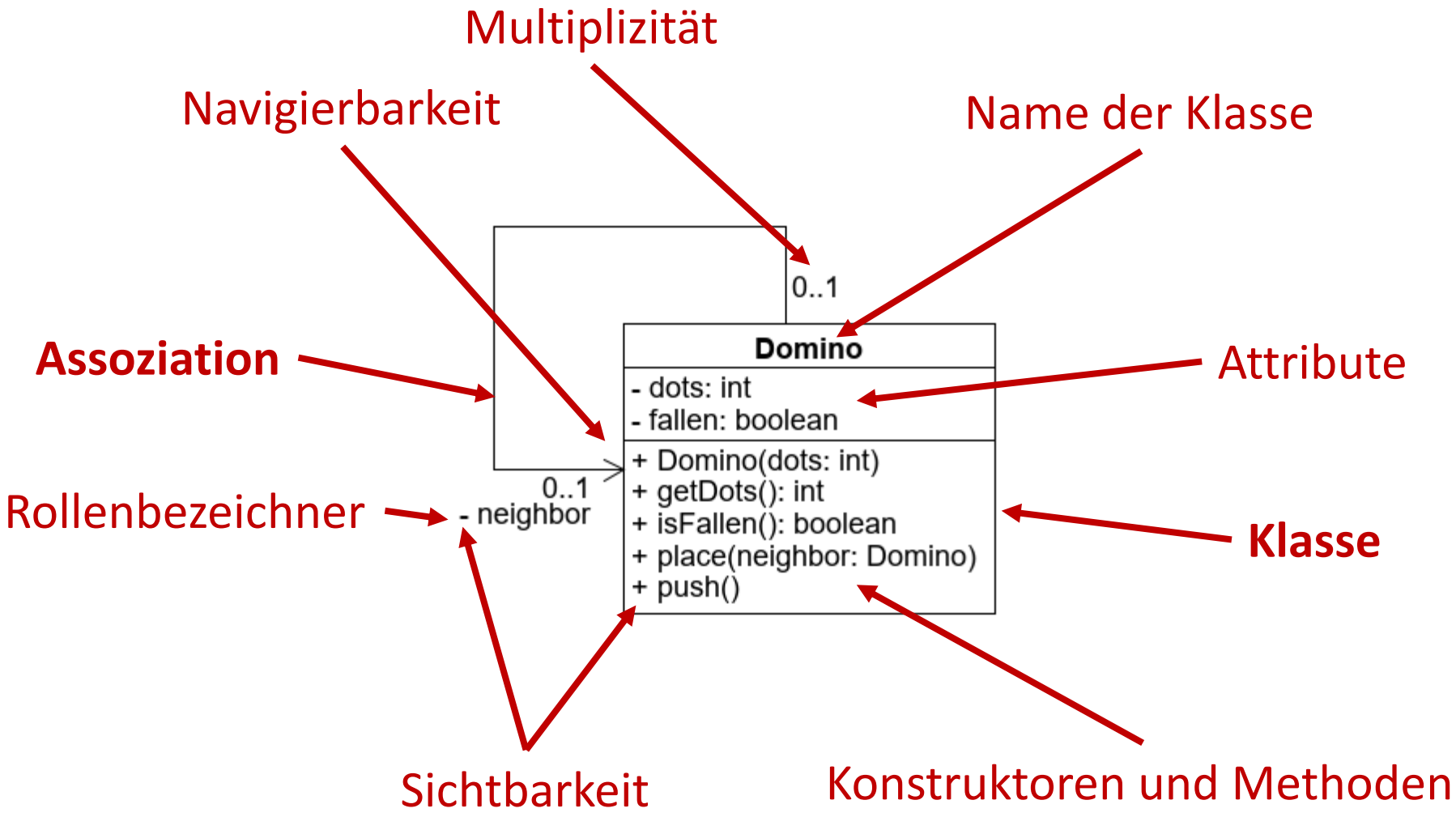
Klassen

```
public class Domino {

    private int dots;
    private boolean fallen;
    private Domino neighbor;

    public Domino(int dots) {…}

    public int getDots() {…}

    public boolean isFallen() {…}

    public void place(Domino neighbor) {…}

    public void push() {…}

}
```

bold text

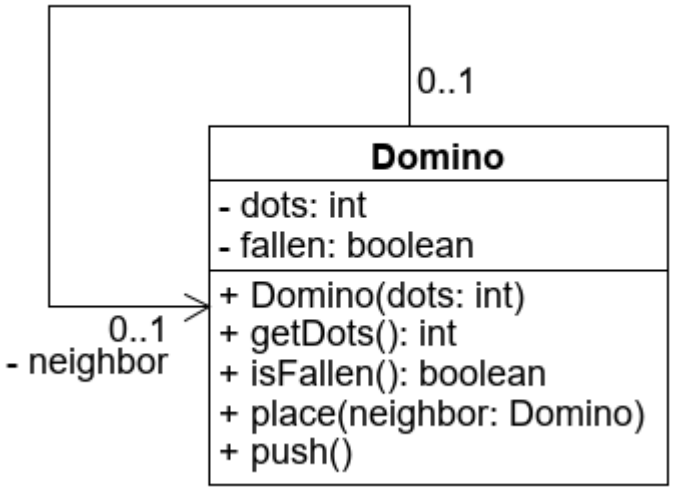


Aufbau von Klassendiagrammen



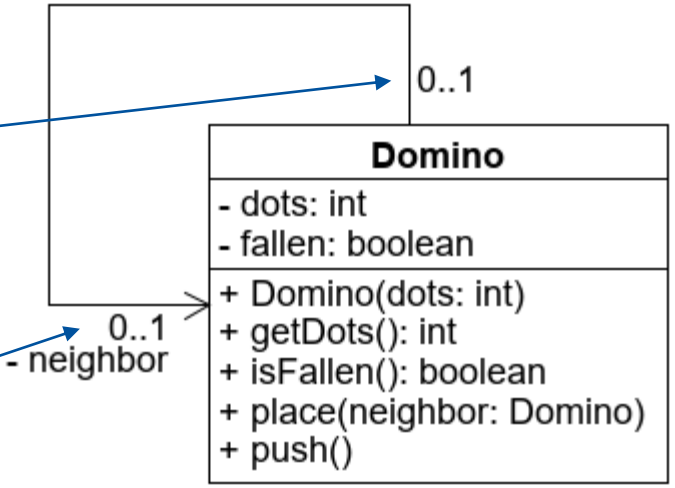
Zugriffsrechte

- Die Sichtbarkeit von Attributen, Konstruktoren und Methoden wird über das erste Zeichen angegeben
- **`public`**: **+**
  - Zugriff aus beliebigen Klassen
- **`protected`**: **#**
  - Zugriff nur aus der gleichen Klasse und aus deren Unterklassen (→ später)
- **`package`**: **~**
  - Zugriff nur aus dem gleichen Paket (→ später)
- **`private`**: **-**
  - Zugriff nur aus der gleichen Klasse

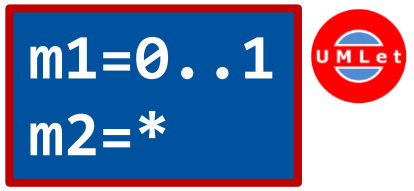


Beziehungen zwischen Klassen

- Assoziationen
  - Rollenbezeichner entsprechen den Attributnamen in der Implementierung
- Multiplizitäten
  - mögliche Anzahl von Instanzen als Quelle der Beziehung (steht am Ausgangspunkt der Assoziation)
    - *wie viele Vorgänger kann es geben?*
  - mögliche Anzahl von Instanzen als Ziel der Beziehung (steht am Endpunkt der Assoziation)
    - *wie viele Nachbarn kann es geben?*
- Navigierbarkeit
  - Zugriffsrichtung zwischen Klassen



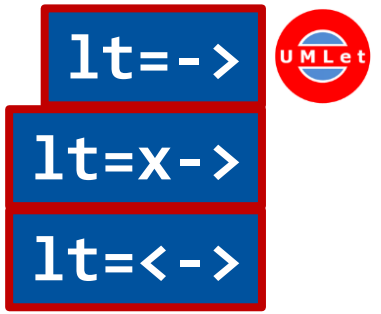
Multiplizitäten

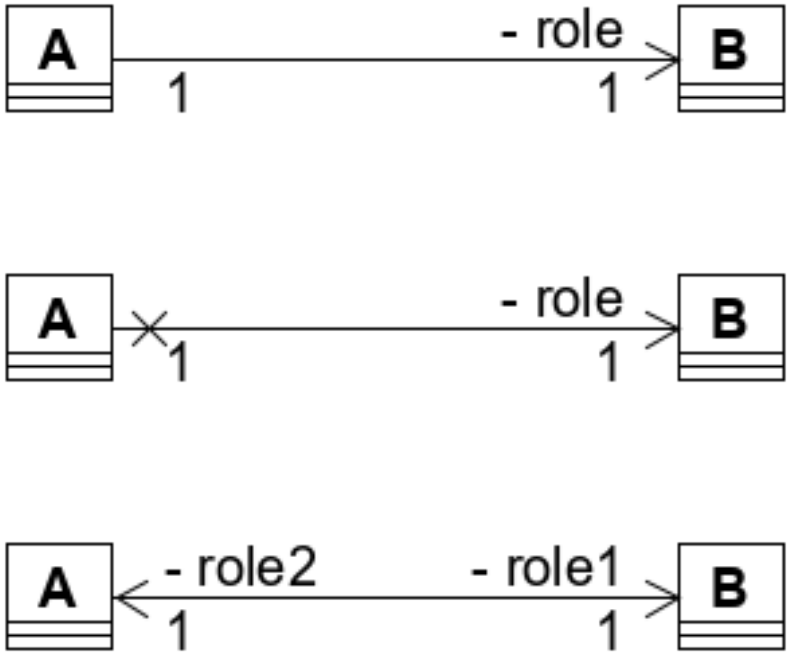


- Mögliche Werte
  - Konstante (z.B. 2)
  - Wertebereich (z.B. 0..2)
  - unbeschränkt (0..* oder *)
  - unbeschränkter Wertebereich (z.B. 2..*)
  - optional (0..1)
- Implementierung von mehrwertigen Elementen z.B. über Liste oder Array
- Auch für Attribute, Parameter und Rückgabetypen möglich
  - **`givenName: String[1..*]`**
  - Standardwert **`[1]`** ist optional
    - **`age: int[1] <-> age: int`**

Navigierbarkeit

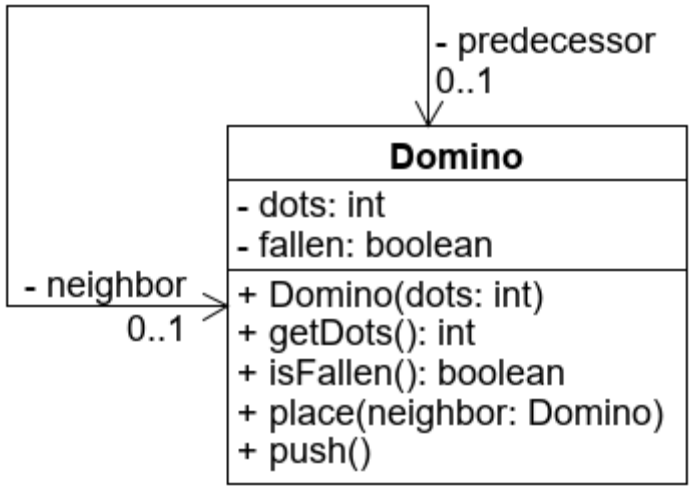
- Navigierbarkeit definiert die Zugriffsmöglichkeiten zwischen Klassen bzw. zwischen ihren Objekten
- Objekte vom Typ **A** können über ihr **`role`**-Attribut auf Objekte vom Typ **B** zugreifen
- Zugriffsrichtung durch Pfeil
- Zugriffsverbot durch X
- Wechselseitiger Zugriff möglich
- Bei vorgeschriebenem Zugriff immer Rollenbezeichner angeben

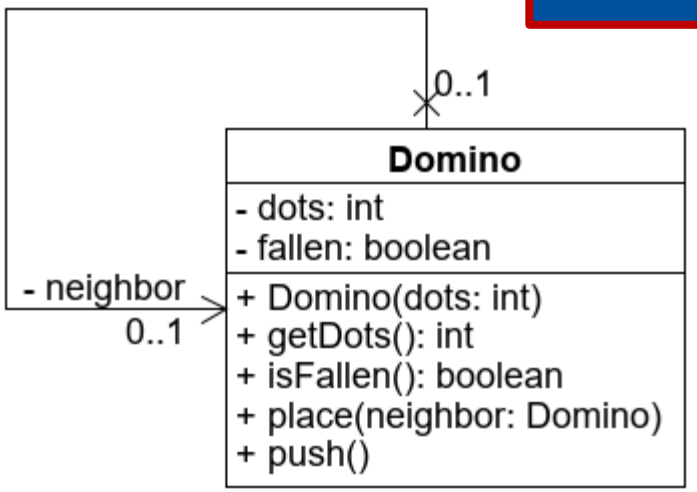




Beispiel: Rollen und Navigierbarkeit

```
r1=- predecessor
r2=- neighbor
```

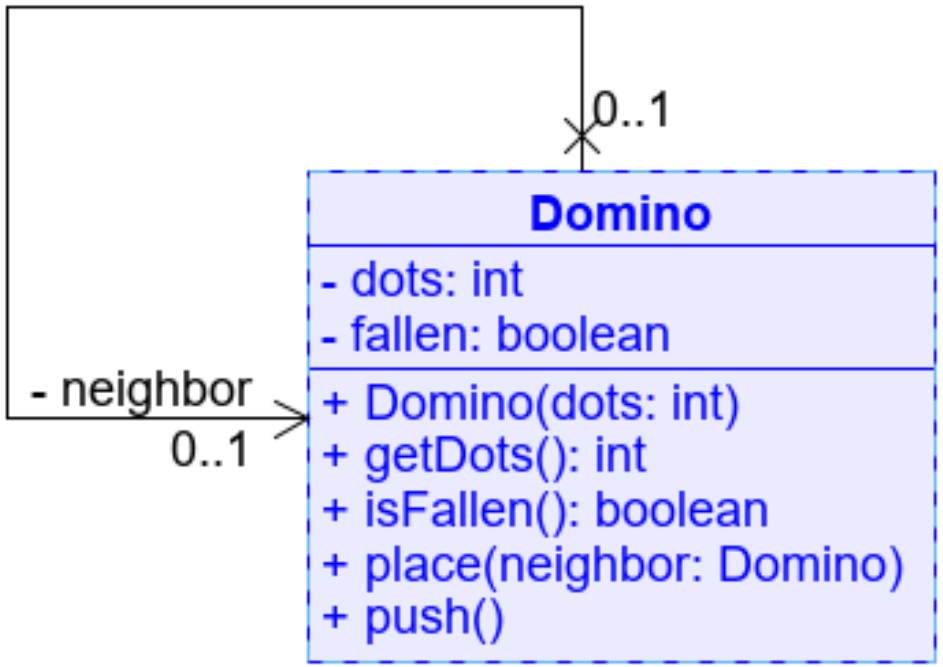





```java
public class Domino {
    private int dots;
    private boolean fallen;
    private Domino neighbor;
    private Domino predecessor;
}
```

```java
public class Domino {
    private int dots;
    private boolean fallen;
    private Domino neighbor;
    private Domino predecessor;
}
```

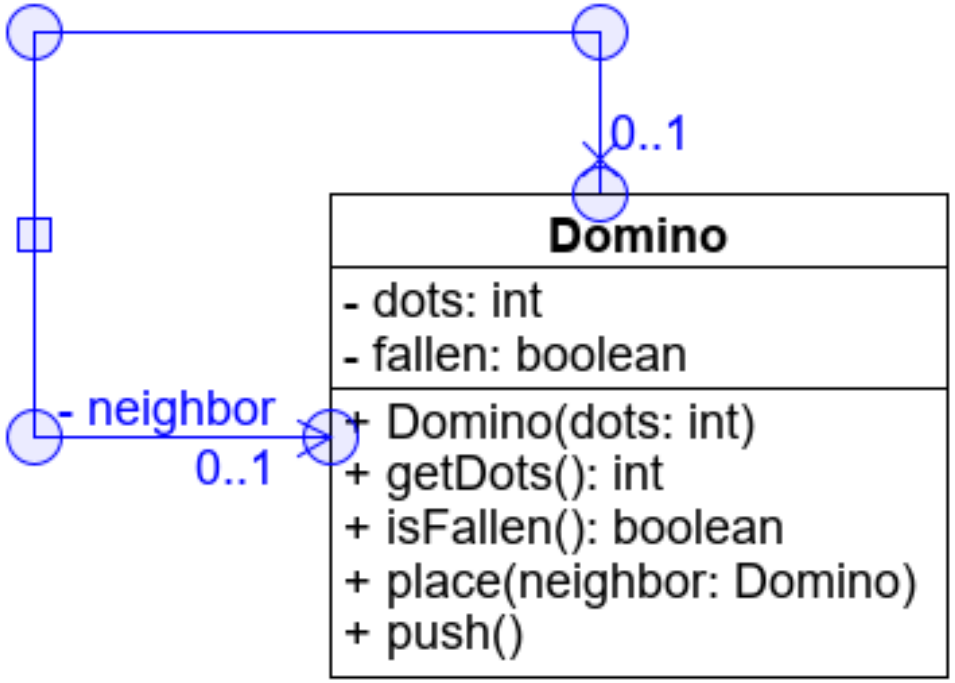
Modellierung mit UMLet

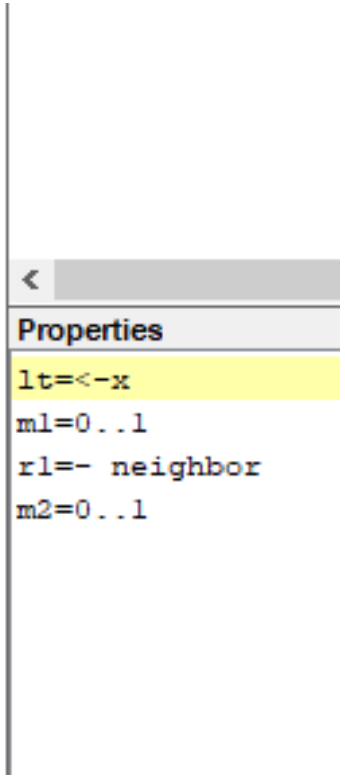


Properties

```
*Domino*
--
- dots: int
- fallen: boolean
--
+ Domino(dots: int)
+ getDots(): int
+ isFallen(): boolean
+ place(neighbor: Domino)
+ push()
```

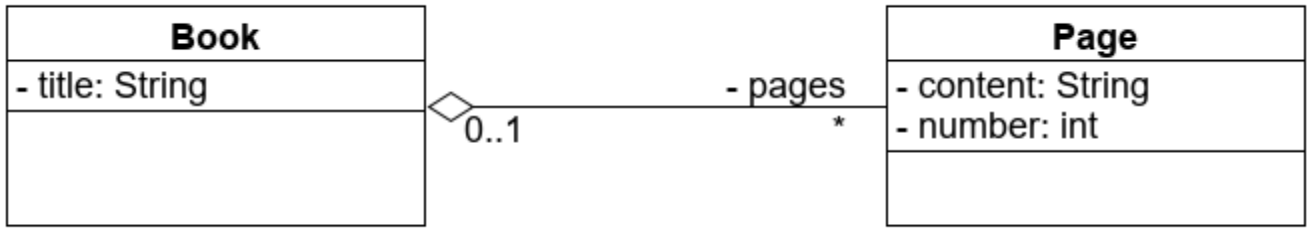
Modellierung mit UMLet (2)





Aggregation

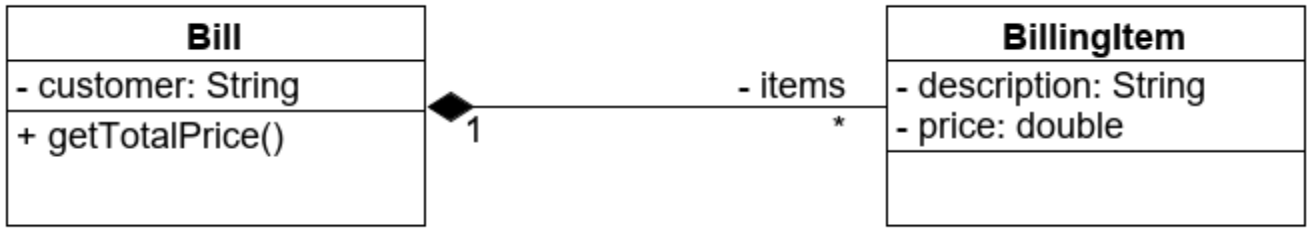
lt=<<<<-



Aggregationen sind spezielle Assoziationen zur Verdeutlichung von „Teil-Ganzes-Beziehungen“.

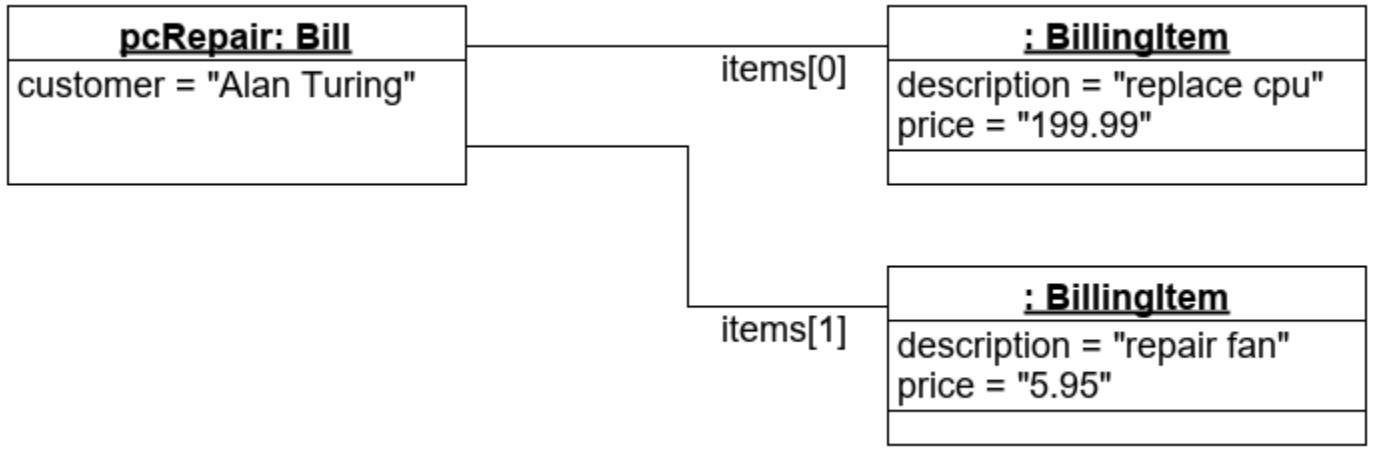
Komposition

lt=<<<<<-



Kompositionen sind spezielle Aggregationen, bei denen die Existenz der Komponenten an die Existenz des Aggregats gekoppelt ist.
Außerdem gehört jede Komponente zu höchstens einem Aggregat (strong ownership).

Komposition: Objektdiagramm



CARL VON OSSIETZKY universität OLDENBURG

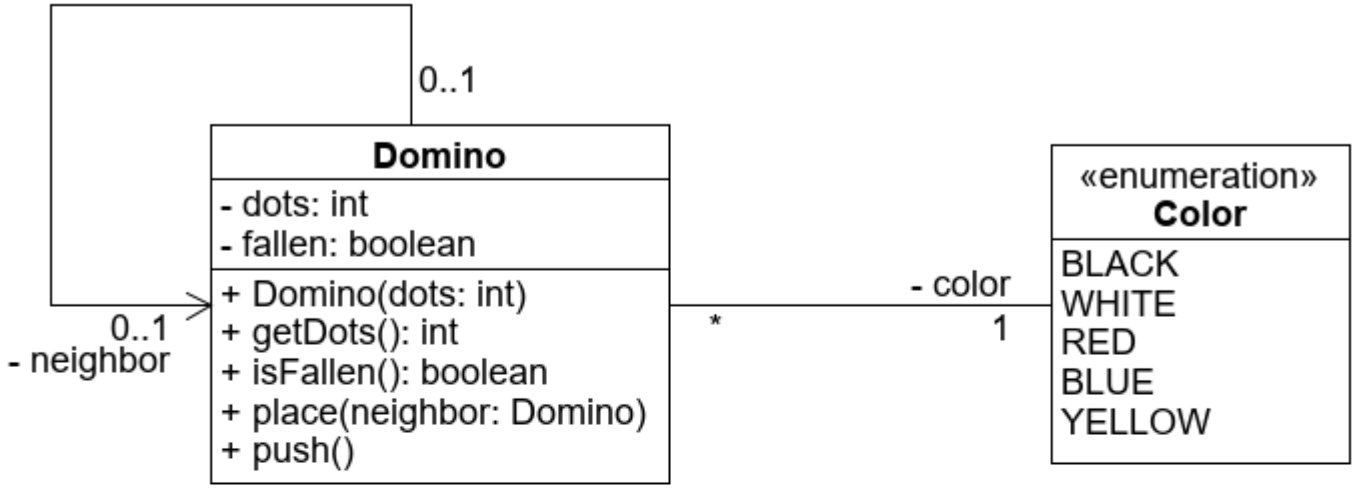
Aufzählungstypen

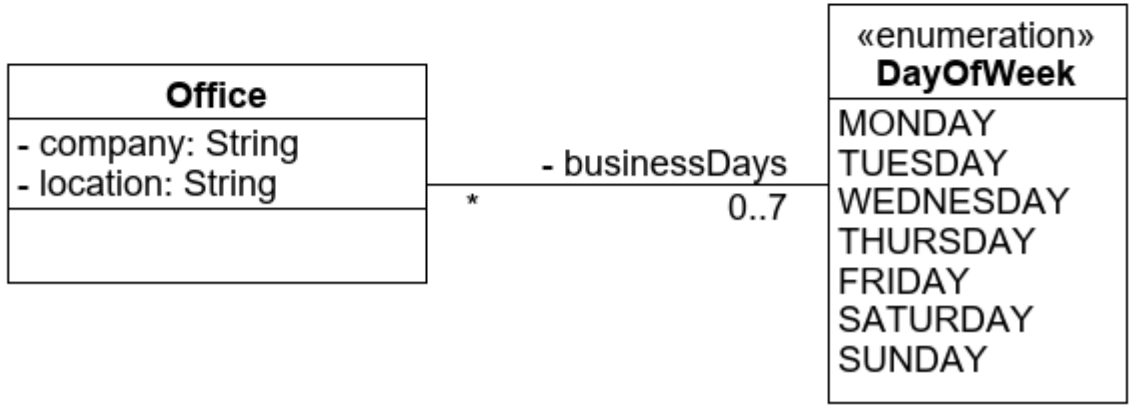
- Wie modelliert/implementiert man einen Datentyp, der aus einer (kleinen) Liste von vordefinierten Werten besteht?
  - Dominosteine sollen eine Farbe (schwarz, weiß, rot, blau oder gelb) haben
  - Dominosteine sollen verschiedene Zustände (stehend, fallend, umgefallen/frei liegend, umgefallen/blockiert) haben
  - das Büro einer Firma kann Öffnungszeiten (Montag – Sonntag) angeben
- Früher oft über Integer-Konstanten gelöst
  - fehleranfällig (ungültige Werte möglich)
  - schlecht erweiterbar (keine Aufzählung aller gültigen Werte möglich)
- Heute über Aufzählungstypen (Enumeration) gelöst

Beispiel: Aufzählungstypen

```
public enum Color {

    BLACK,
    WHITE,
    RED,
    BLUE,
    YELLOW;

}
```

```
public enum DayOfWeek {

    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY;

}
```





CARL VON OSSIETZKY universität OLDENBURG

Klassenattribute und -methoden

- Statische oder Klassenattribute sind keinem konkreten Objekt zugeordnet
- Sie gehören stattdessen zu ihrer Klasse
- Wird der Wert eines statischen Attributs in einem Objekt (=Instanz der Klasse) geändert, so ändert sich auch der Wert bei allen anderen Instanzen dieser Klasse
- Statische oder Klassenmethoden können unabhängig von einer Objektinstanz ausgeführt werden

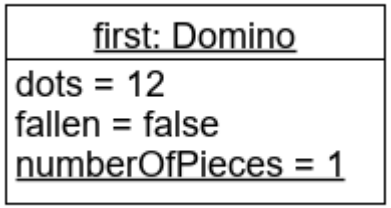
CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Klassenattribute und -methoden

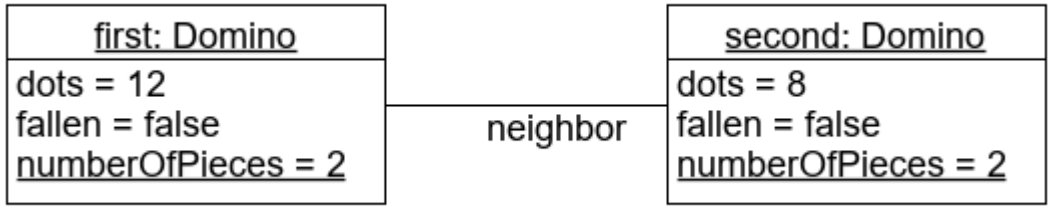
```
public class Domino {
    …
    private static int numberOfPieces = 0;

    public Domino(int dots) {
        this.dots = dots;
        numberOfPieces++;
    }

    public static int getNumberOfPieces() {
        return numberOfPieces;
    }

    …

    public static void main(String[] args) {
        Domino first = new Domino(12);
        …
        System.out.println(Domino.getNumberOfPieces());
    }
}
```

Beispiel: Klassenattribute und -methoden (2)

```
Domino first = new Domino(12);
```




```
Domino second = new Domino(8);
first.place(second);
```



Konstanten

- Konstanten sind Variablen oder Attribute, denen genau einmal ein Wert zugewiesen werden kann
- Klassenkonstanten sind statische Attribute, denen genau einmal ein Wert zugewiesen werden kann
  - oft werden Klassenkonstanten auch einfach als Konstanten bezeichnet
- In Java: Schlüsselwort **`final`**

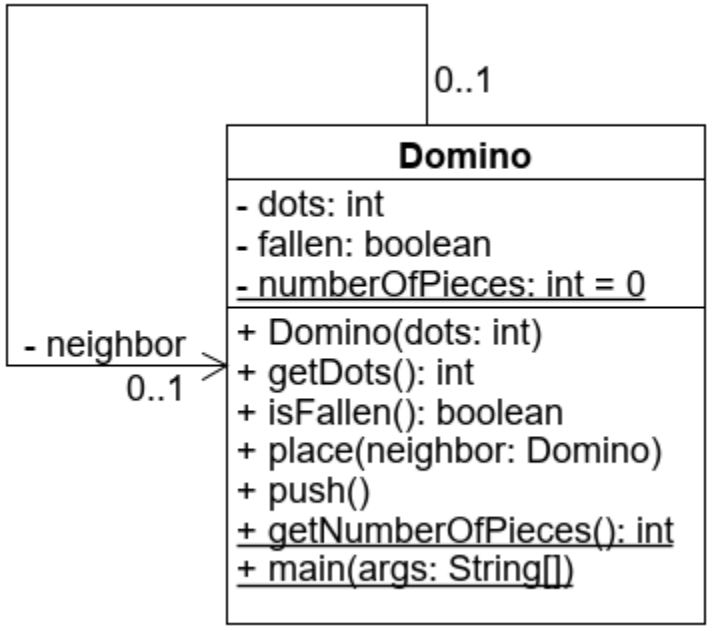
```
private static final int MAX_DOTS = 12;
```

- In der UML: Zusatz **{readOnly}**

- MAX_DOTS: int = 12 {readOnly}

Klassenattribute und -methoden in der UML

underlined text

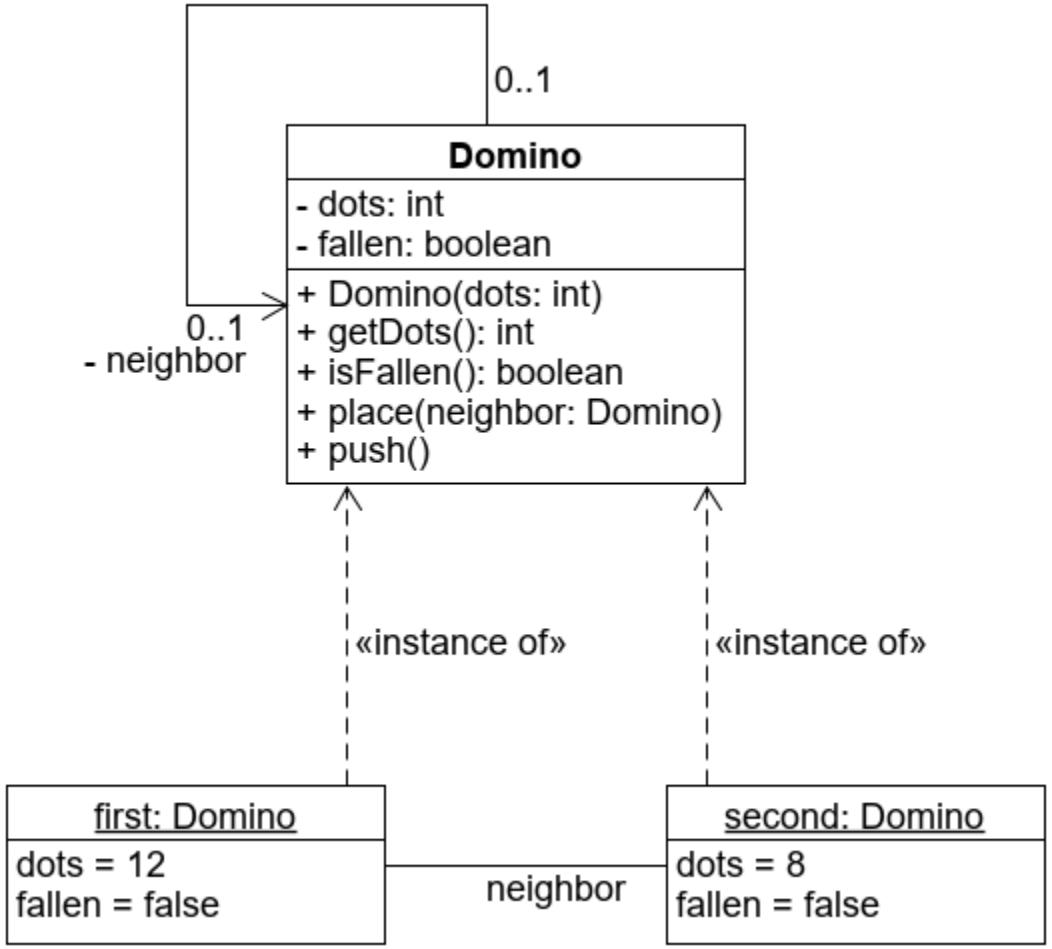


CARL VON OSSIETZKY universität OLDENBURG

Namenskonventionen für Konstanten und enums

- Konstanten werden in Java in Großbuchstaben geschrieben, Wörter werden durch Unterstriche getrennt
  - Beispiel: **`MAX_DOTS`**
- Aufzählungstypen werden in Java wie Klassen behandelt, ihre Werte wie Konstanten
  - Beispiel: **`enum Color { RED, BLUE, YELLOW }`**
- In anderen Sprachen (z.B. in C#) gelten andere Konventionen
- In der UML gibt es **keine** Namenskonventionen
  - bei einer Zielsprache: Verwendung der Konventionen der Zielsprache
  - bei unbekannter oder bei mehreren Zielsprachen: Verwendung von generischen und sprachunabhängigen Konventionen

Beispiel: Objekt- und Klassendiagramme



CARL VON OSSIETZKY universität OLDENBURG

Objekt- und Klassendiagramme

- Objektdiagramme werden in der UML 2 als Teil der Klassendiagramme aufgefasst
- Instanz-Ebene
  - Objektdiagramme beschreiben statische Zusammenhänge auf der Ebene einzelner, konkreter Dinge
- Schema-Ebene
  - Klassendiagramme beschreiben statische Zusammenhänge unabhängig von Details konkreter Objekte, auf der Ebene mehrerer gleichartiger Dinge

CARL VON OSSIETZKY universität OLDENBURG

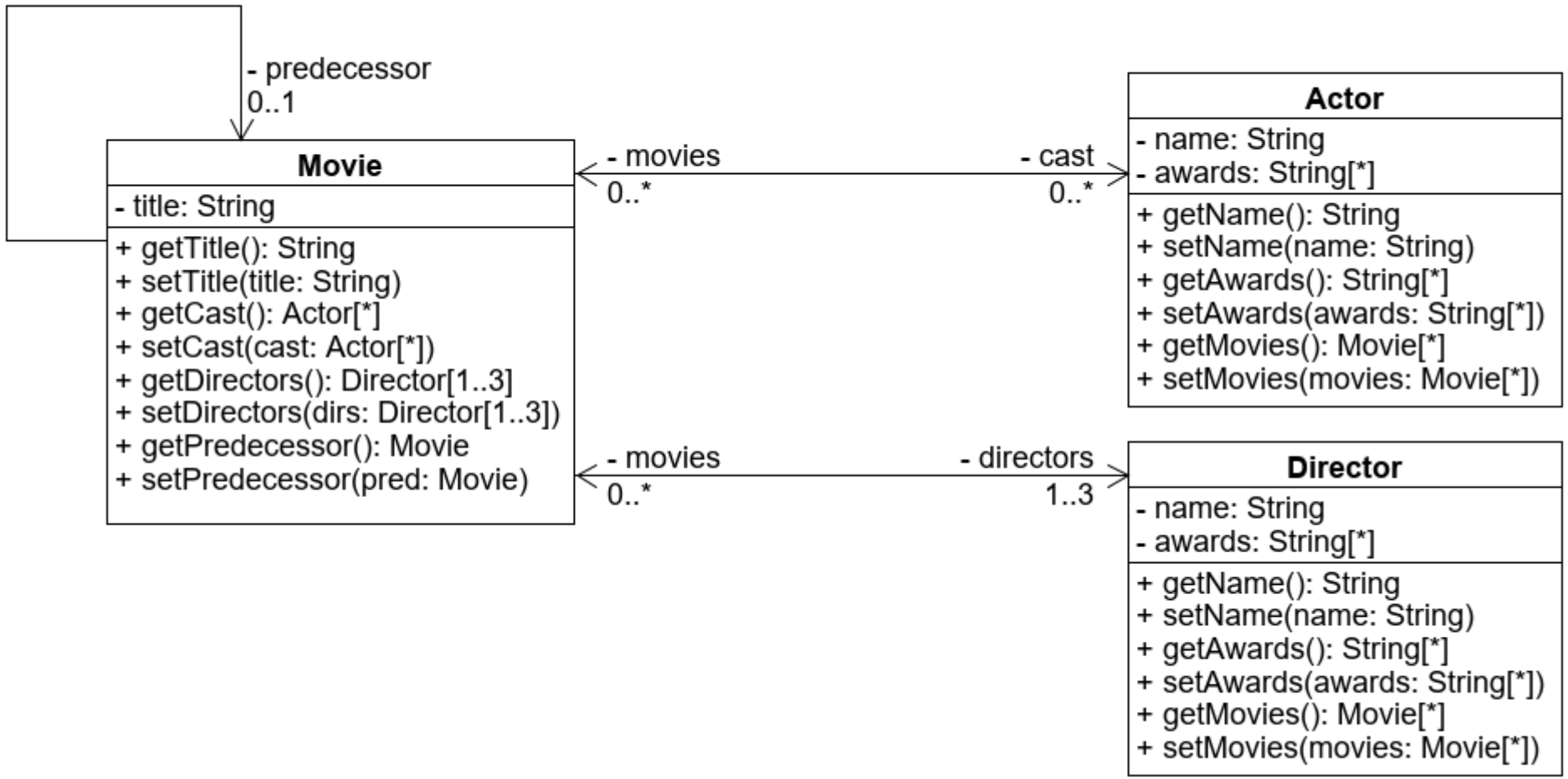
Anwendung

CARL VON OSSIETZKY universität OLDENBURG

Film-Datenbank

- Filme haben
  - einen Titel
  - Darsteller
  - Regisseure
  - ggf. einen Vorgänger (bei Filmserien)
- Schauspieler und Regisseure haben
  - einen Namen
  - eine Filmographie
  - ggf. Auszeichnungen

Film-Datenbank: Modell



Film-Datenbank: Implementierung

```
public class Movie {

    private String title;
    private Actor[] cast;
    private Director[] directors;
    private Movie predecessor;

    public String getTitle() {
        return title;
    }

    …

}
```

```
public class Actor {

    private String name;
    private String[] awards;
    private Movie[] movies;

    public String getName() {
        return name;
    }

    …
}
```

```
public class Director {

    private String name;
    private String[] awards;
    private Movie[] movies;

    public String getName() {
        return name;
    }

    …
}
```

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- Klassen und Objekte
  - in Java
  - in der UML
- Attribute, Methoden, Signaturen, Konstruktoren, Konstanten
- Lebenszyklus von Objekten
- Objektdiagramme
- Klassendiagramme