CARL VON OSSIETZKY universität OLDENBURG

Sommersemester 2020

Objektorientierte Modellierung und Programmierung

Dr. C. Schönberg

CARL VON OSSIETZKY universität OLDENBURG

Einführung Modelle und Modellierung

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- Abstrakte Datentypen
- Wiederholung Klassen und Objekte
- Modellbegriff
  - statisch
  - dynamisch
- Modellierung versus Programmierung

CARL VON OSSIETZKY universität OLDENBURG

Motivation

CARL VON OSSIETZKY universität OLDENBURG

Bisher

- Grundlegende Algorithmen und Datenstrukturen
  - Such- und Sortieralgorithmen
  - Felder (Arrays)
  - Listen, Stacks, Queues
  - Bäume (Binärbäume, AVL-Bäume), Graphen
  - Klassen und Objekte
- Einfache Programme
  - < 5 Klassen
  - < 1.000 LoC

CARL VON OSSIETZKY universität OLDENBURG

In der Realität

- Grundlegende Algorithmen und Datenstrukturen
- Komplexe Algorithmen und Datenstrukturen
  - komplexe Klassenstrukturen
  - komplexe Probleme, z.B. TSP
  - Heuristiken
- Komplexe Programme
  - > 1.000 Klassen
  - > 1.000.000 LoC

CARL VON OSSIETZKY universität OLDENBURG

Herausforderungen

- Umgang mit Komplexität
  - komplexe Probleme
    - analysieren
    - formalisieren
    - Lösungsansätze entwerfen
  - komplexe Programme
    - entwerfen
    - analysieren
    - korrigieren
  - komplexe Algorithmen
    - anwenden
    - entwerfen

CARL VON OSSIETZKY universität OLDENBURG

Herausforderungen (2)

- Viele Schritte vom Problem zur Lösung
  - Vision
  - Spezifikation, Konzept
  - Entwurf
  - Programmierung einzelner Teile
  - Testen
  - Zusammensetzen der Teile
  - Testen
  - Einführung, Deployment
  - Betrieb, Wartung
  - Anpassungen
  - Testen
  - …

CARL VON OSSIETZKY universität OLDENBURG

Handwerkszeug

- Abstraktion
- Vorgehensweisen (z.B. Divide-and-Conquer)
- Konstrukte der Programmiersprache
  - Vererbung
  - Schnittstellen
  - Generics
  - APIs
- Patterns

CARL VON OSSIETZKY universität OLDENBURG

Wiederholung

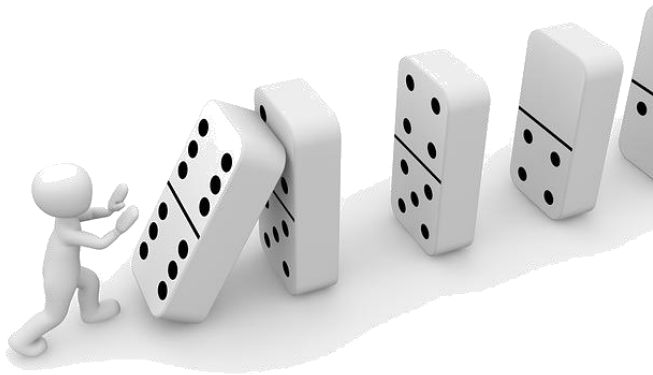
Klassen, Objekte, Abstrakte Datentypen

CARL VON OSSIETZKY universität OLDENBURG

Klassen

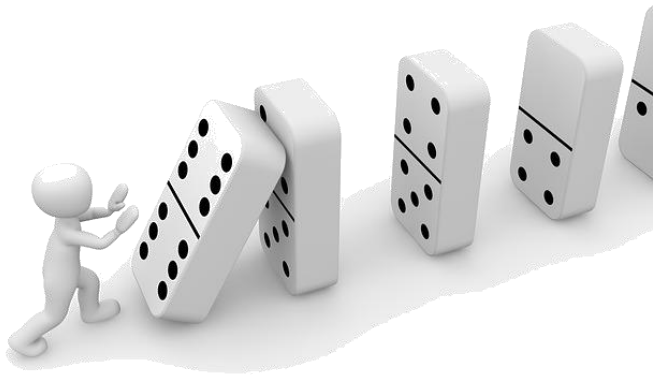
- Bauplan für Objekte mit gleichen/ähnlichen Eigenschaften
- Definiert
  - Attribute
  - Konstruktoren
  - Methoden

Beispiel: Klasse



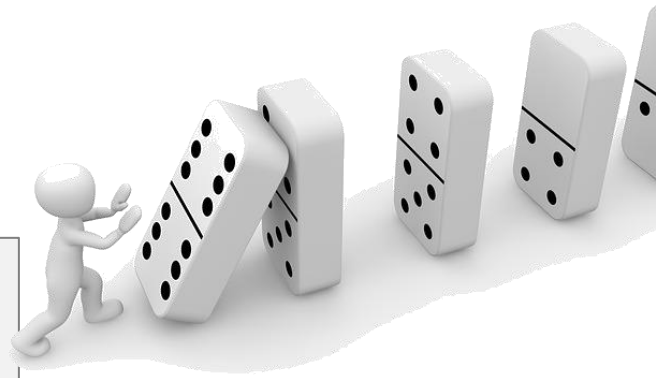
```
public class Domino {

    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() {…}

}
```

Beispiel: Klasse (2)



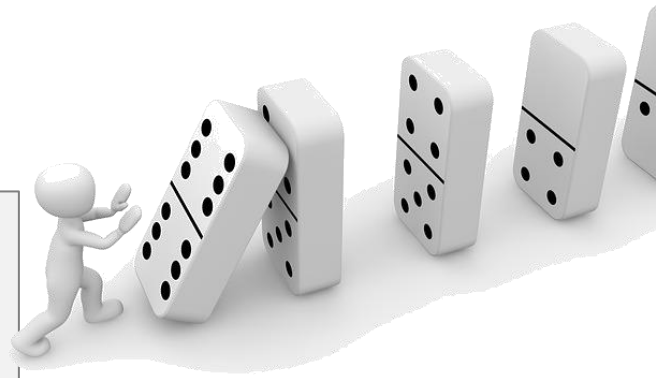
```
public Domino(int dots) {
    this.dots = dots;
}

public int getDots() {
    return dots;
}

public boolean isFallen() {
    return fallen;
}
```

Beispiel: Klasse (3)



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

public void push() {
    fallen = true;
    System.out.println(dots + " topples.");
    if (neighbor != null) {
        neighbor.push();
    }
}
```

Beispiel: Klasse (3)



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

public void push() {
    if (!fallen) {
        fallen = true;
        System.out.println(dots + " topples.");
        if (neighbor != null) {
            neighbor.push();
        }
    }
}
```

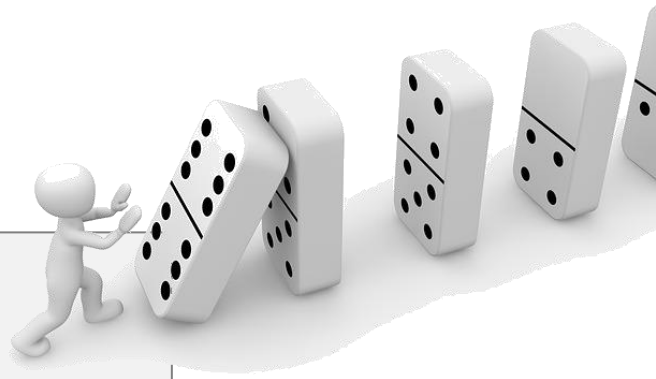
CARL VON OSSIETZKY universität OLDENBURG

Objekte

- Instanzen von Klassen
- Enthalten konkrete Daten, gespeichert in der von der Klasse vorgegebenen Struktur

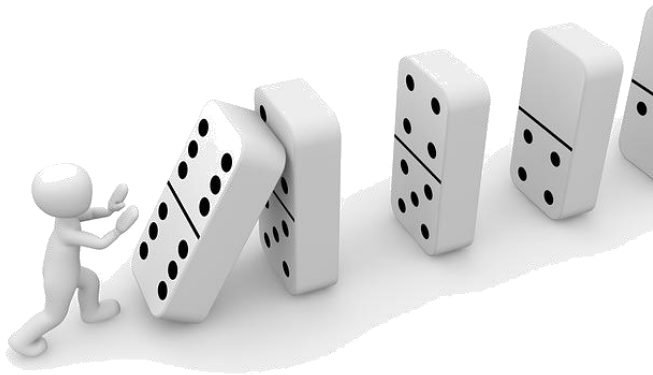
CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Objekte



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

first.place(second);
second.place(third);
third.place(fourth);
first.push();

System.out.println("Domino-Effect: " + fourth.isFallen());
```

Beispiel: Objekte (2)



```
12 topples.
8 topples.
9 topples.
4 topples.
Domino-Effect: true
```

CARL VON OSSIETZKY universität OLDENBURG

Abstrakte Datentypen

- Daten und Operatoren
  - Zugriff auf die Daten nur über die Operatoren
    → Kapselung
- Beispiele
  - Listen (**`add`**, **`insert`**, **`get`**, **`remove`**, …)
  - Stacks (**`push`**, **`peek`**, **`pop`**)
  - Queues (**`enqueue`**, **`dequeue`**)
  - Binärbäume (**`setLeft`**, **`setRight`**, **`traverse`**, **`find`**, **`remove`**, …)
- Umsetzung in Java
  - Klassen: Definition der Datenstruktur und der Operatoren
  - Objekte: Konkrete Instanzen der Klassen

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Liste (Hilfsklasse)

```
class ListItem {

    private String data;
    private ListItem next;

    public ListItem(String value)     { data = value; }

    public String getData()           { return data; }
    public void setData(String value) { data = value; }
    public ListItem getNext()         { return next; }
    public void setNext(ListItem ref) { next = ref; }
}
```

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Liste (Klasse)

```
public class LinkedList {

    private ListItem head;

    public void add(String data) {
        // …
    }

    public String get(int index) {
         // …
    }

    public void remove(String data) {
         // …
    }
}
```

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Liste (Objekt)

```
LinkedList list = new LinkedList();

list.add("1");
list.add("2");
list.add("3");

list.remove("2");

System.out.println(list.get(0));
System.out.println(list.get(1));
```

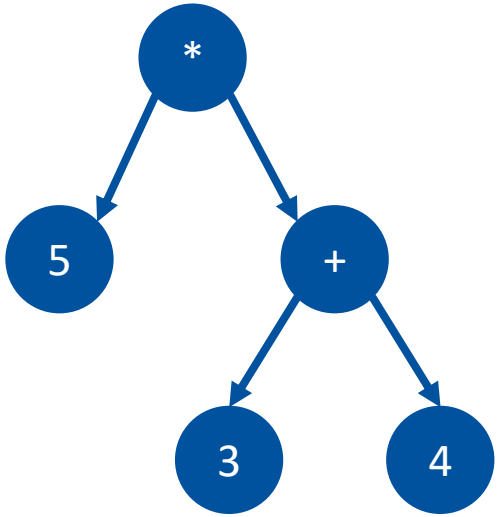
CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Baum (Klasse)

```
public class BinTree {

    private String data;
    private BinTree left;
    private BinTree right;

    public BinTree() { }
    public BinTree(String data) { … }

    public String getData() { … }
    public void setData(String data) { … }
    public BinTree getLeft() { … }
    public void setLeft(BinTree left) { … }
    public BinTree getRight() { … }
    public void setRight(BinTree right) { … }

    public boolean find(String data) {
        // …
    }
}
```

Beispiel: Baum (Objekt)

```
BinTree tree = new BinTree("*");

BinTree left = new BinTree("5");
tree.setLeft(left);

BinTree right = new BinTree("+");
tree.setRight(right);

BinTree right_left = new BinTree("3");
right.setLeft(right_left);

BinTree right_right = new BinTree("4");
right.setRight(right_right);
```



CARL VON OSSIETZKY universität OLDENBURG

Modelle

CARL VON OSSIETZKY universität OLDENBURG

Motivation

Implementierung der Klasse

```
public class Domino {

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

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

	public int getDots() {
		return dots;
	}

	public boolean isFallen() {
		return fallen;
	}

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

	public void push() {
		if (!fallen) {
			fallen = true;
			System.out.println(dots + " topples.");
			if (neighbor != null) {
				neighbor.push();
			}
		}
	}
}
```

Motivation

Weiterentwicklung der Klasse

```
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() {









	}
}
```

Motivation

Verwendung der Klasse

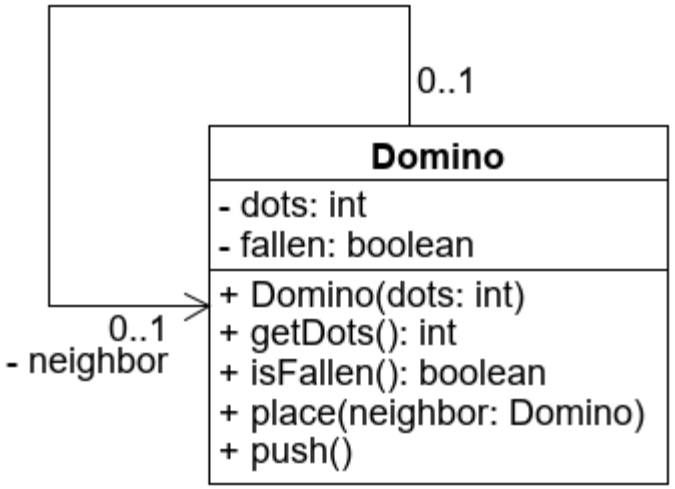
```
public class Domino {



      public Domino(int dots) {

      }

      public int getDots() {

      }

      public boolean isFallen() {

      }

      public void place(Domino neighbor) {


      }

      public void push() {









      }
}
```

CARL VON OSSIETZKY universität OLDENBURG

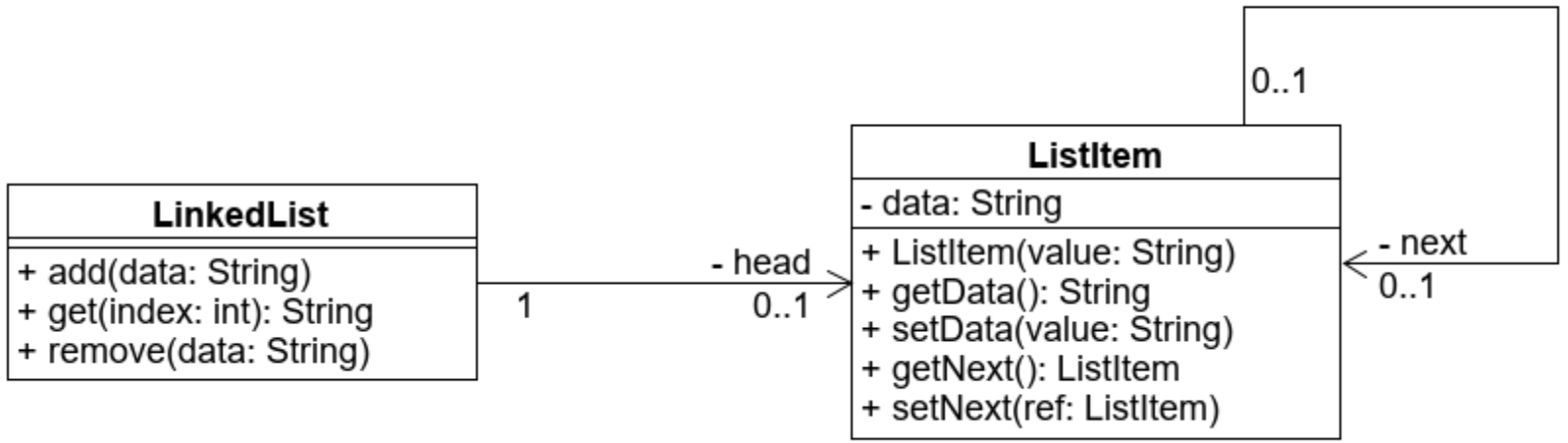
Motivation (2)

- Relevante Informationen für die Verwendung der Klasse:
  - Schnittstellen (öffentliche Methoden)
  - Interaktion mit anderen Klassen (Methodensignaturen)
- Relevante Informationen für die Analyse, Korrektur oder Weitereinwicklung der Klasse:
  - Schnittstellen (öffentliche Methoden)
  - Hilfsmethoden (nicht-öffentliche Methoden)
  - Interaktion mit anderen Klassen (Methodensignaturen)
  - Struktur (Attribute)
  - ggf. Details (Methodenimplementierung)

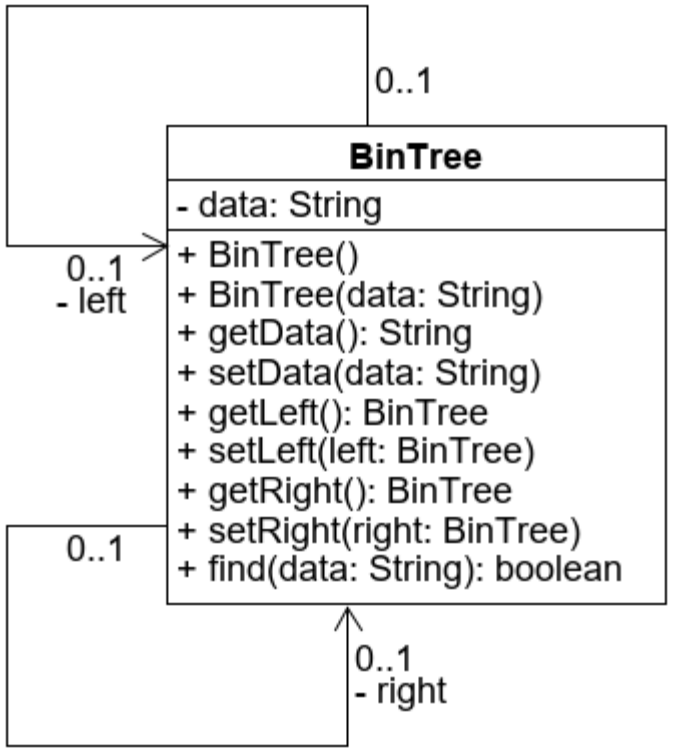
Beispiel: Domino Modell



Beispiel: Listen Modell



Beispiel: Baum Modell

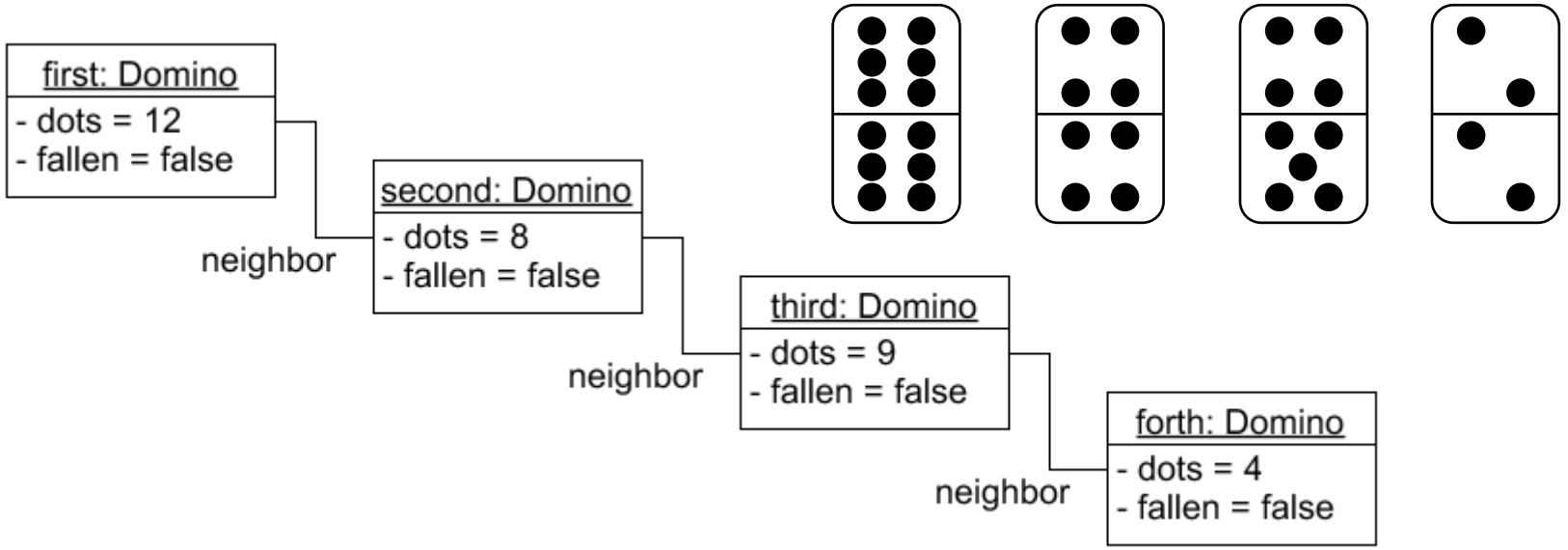


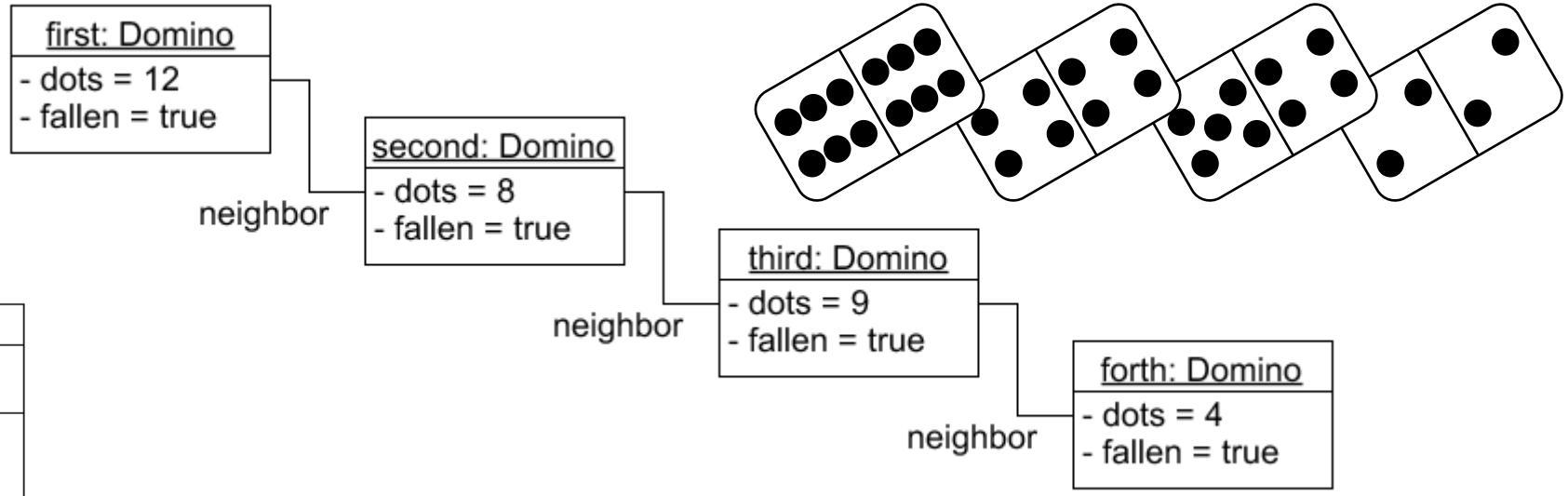
Beschreibung: Modell

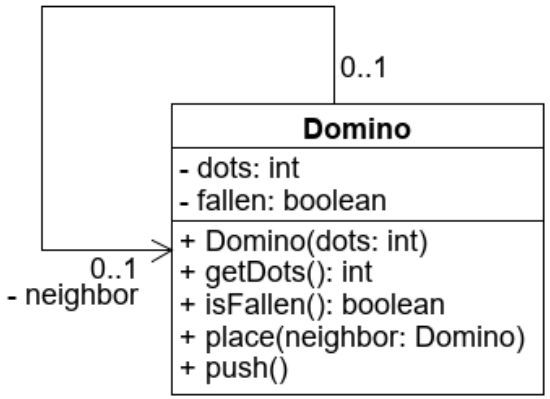
- Abstraktion von der Implementierung
  - Beherrschung von Komplexität
  - Verlust von Informationen
  - zielgerichtet
- Unabhängig von der Programmiersprache
  - verschiedene Implementierungen möglich
- Enthält Daten, die für das Verständnis (Verwendung, Analyse) relevant sind
  - Schnittstellen
  - Signaturen
  - Hilfsmethoden
  - Attribute
  - Werte

→ es kann abhängig vom Zweck verschiedene Modelle mit unterschiedlichen Abstraktionsstufen geben

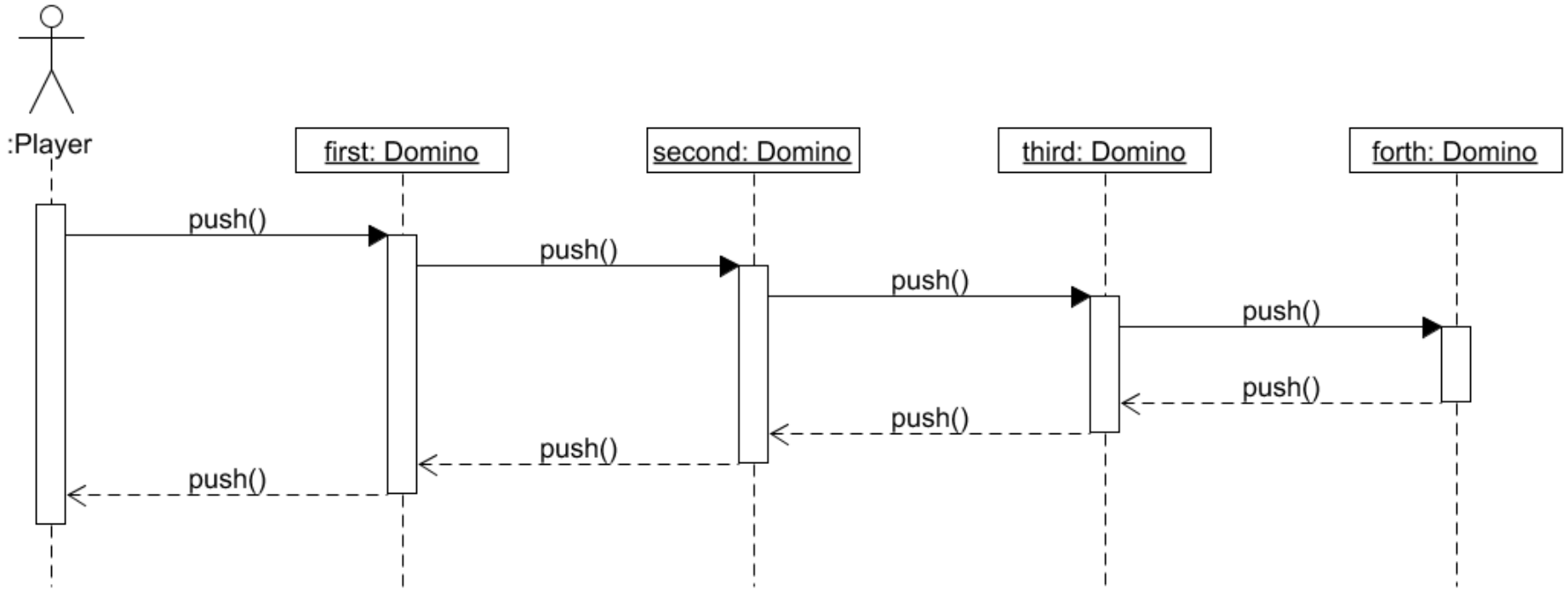
Beispiel: Domino Modell (Objekte)

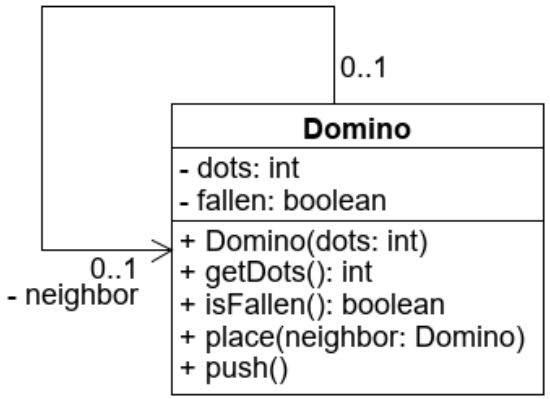






Beispiel: Domino Modell (Ablauf)





Statisches vs. Dynamisches Modell

- Statische Modelle beschreiben einen konkreten Zustand zu einem konkreten Zeitpunkt
  - ➢ hier
- Dynamische Modelle beschreiben Veränderungen, Entwicklungen und Verhalten
  - ➢ Softwaretechnik I

CARL VON OSSIETZKY universität OLDENBURG

Verwendung von Modellen

- Dokumentation
  - welche Klassen gibt es?
  - wie hängen diese Klassen zusammen?
  - was können diese Klassen?
  - wie müssen diese Klassen verwendet werden?
- Planung
  - welche Klassen werden gebraucht?
  - wie müssen diese Klassen zusammenhängen?
  - was müssen diese Klassen können?
  - wie sollen diese Klassen verwendet werden?
- Erst planen, dann programmieren!
- Planung wird schrittweise konkreter!

CARL VON OSSIETZKY universität OLDENBURG

Modellierung und Programmierung

- Modellierung ist
  - abstrakt und fokussiert, sie konzentriert sich auf das für einen Aspekt Wesentliche
  - vielseitig, unterschiedliche Modelle können unterschiedliche Aspekte darstellen
- Programmierung ist
  - konkret und ausführbar, jede Funktionalität muss bis ins Detail beschrieben (implementiert) sein
    - Ausnahme: Pseudocode
  - umfassend, alle Aspekte müssen gleichzeitig behandelt werden
- Code und Modell sind eng miteinander verwandt

Modellierung und Programmierung (2)

- Implementierung kann (in Teilen) aus Modellen generiert werden
- Modelle können (in Teilen) aus Implementierung generiert werden
- Roundtrip (state of the art):
  1. Modellierung (erster Entwurf)
     → Code-Generierung
  2. Programmierung (Anpassung des Codes)
     → automatische Anpassung der Modelle
  3. Verfeinerung der Modellierung
     → automatische Anpassung des Codes

  ➢ weiter bei 2.

CARL VON OSSIETZKY universität OLDENBURG

Ausblick

CARL VON OSSIETZKY universität OLDENBURG

Inhalte

- Modellierung von statischen (Klassen, Objekte) Zusammenhängen
- Objektorientierter Entwurf
- Fortgeschrittene Programmier-/Modellierkonzepte
  - Vererbung
  - Polymorphie
  - Generics
  - …
- GUIs
- Parallelität
- Lösungsstrategien
- Programmierparadigmen

CARL VON OSSIETZKY universität OLDENBURG

Lernziele OMP

- Analyse von Problemen
- Erstellung von Lösungskonzepten (UML Modelle)
- Umsetzung von Lösungskonzepten (Implementierung der UML Modelle)
- Kenntnis der nötigen Modellier- und Programmierkonzepte für Lösungskonzepte zu komplexen Problemen
- Analyse und Korrektur von komplexen Modellen und Programmen

CARL VON OSSIETZKY universität OLDENBURG

Literatur

- Heide Balzert: *Lehrbuch der Objektmodellierung: Analyse und Entwurf mit der UML 2*. Spektrum Verlag, 2011.
- Christian Ullenboom: *Java ist auch eine Insel: das umfassende Handbuch*. Galileo Press, 2017.
- Christian Ullenboom: *Java SE 9 Standard-Bibliothek: das Handbuch für Java-Entwickler*. Galileo Press, 2017.

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- Abstrakte Datentypen
- Wiederholung Klassen und Objekte
- Modellbegriff
  - statisch
  - dynamisch
- Modellierung versus Programmierung