Reflexion

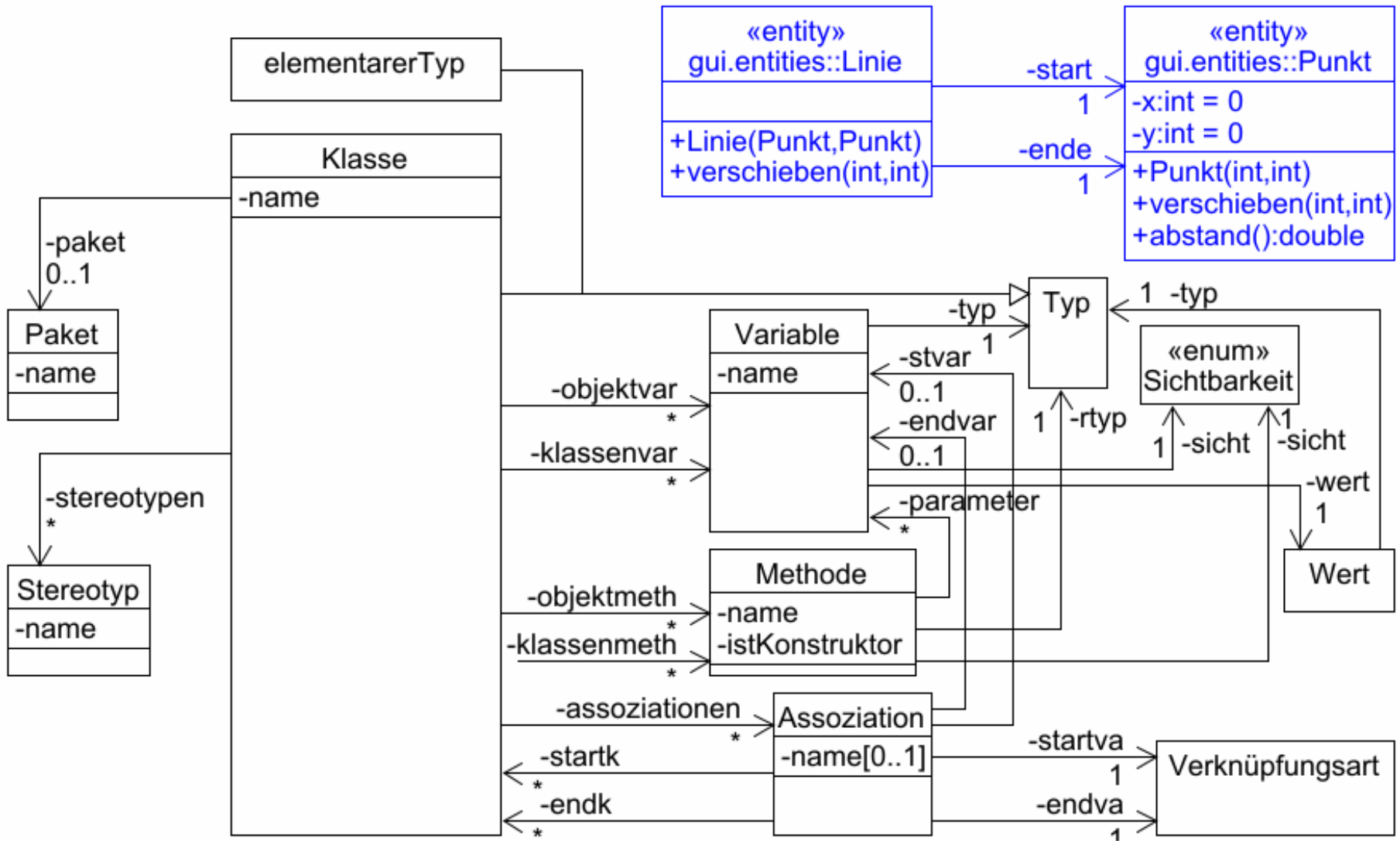
HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- auch Introspektion, engl. reflection
- Reflexion erlaubt in Java Meta-Programmierung; Klassen selbst als Objekte nutzen
- Paket `java.lang.reflect`
- Zu jeder Klasse gibt es ein Klassenobjekt der Klasse `Class`

```
Class cl1 = String.class;
try { // oder
  Class cl2 = Class.forName("java.lang.String");
} catch (ClassNotFoundException e) {
  System.out.println("gibs nich")
}
```

- Metamodell beschreibt alle möglichen Modelle, Reflexion basiert auf so einem Meta-Modell

Vom Klassendiagramm zum Metamodell



Nutzung von Class

- Klassenobjekt erlaubt
  - Abfrage von Variablen, Konstruktoren, Methoden (mit Sichtbarkeiten, Typen, Parametern, Exceptions,…)
  - Abfrage und Änderung von Objektwerten
  - Nutzung von Konstruktoren und Methoden
  - Abfrage von Oberklassen
  - …

- Insgesamt können damit zur Laufzeit beliebige Klassen analysiert und genutzt werden
- Zentrale Reflection-Klassen: Field, Constructor, Method, Modifier

Exemplarische Methoden der Klasse Class

Annotation[]	getDeclaredAnnotations() Returns all annotations that are directly present on this element.
Constructor[]	getDeclaredConstructors() Returns an array of Constructor objects reflecting all the constructors declared by the class represented by this Class object.
Field[]	getDeclaredFields() Returns an array of Field objects reflecting all fields declared by the class represented by this Class object.
Method[]	getDeclaredMethods() Returns an array of Method objects reflecting all the methods declared by the class or interface represented by this Class object.
Class[]	getInterfaces() Determines the interfaces implemented by the class or interface represented by this object.
int	getModifiers() Returns the Java language modifiers for this class or interface, encoded in an integer.
boolean	isArray() Determines if this Class object represents an array class.
T	newInstance() Creates a new instance of the class represented by this Class object.

http://docs.oracle.com/javase/7/docs/api/java/lang/Class.html

Einsatzbereiche von Reflexion

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Eingesetzt zur Entwicklung von Entwicklungswerkzeugen
  - Debugger
  - Class Browser, UML-Diagrammableitung
  - GUI Builder
  - IDEs wie z.B. Eclipse und Netbeans
- Eingesetzt in Frameworks
  - Einheitliche Klassenbehandlung ohne Interfaces
  - Finden bestimmter Methoden (z. B. getXXX) und Eigenschaften
  - Meist Verwaltung unterschiedlicher Klassen
  - Tests privater Methoden
- Oft verknüpft mit anderen fortgeschrittenen Ansätzen (ClassLoader, Annotationen, Bytecode-Manipulation)

Ungewöhnliches Beispiel (1/2)

- Praktisch nicht nutzbare Klasse

```
package model;

public class Heimlich {
  private int x;

  private Heimlich(){  }

  private void ausgeben(){
    System.out.println("x ist "+x);
  }
}
```

- Hinweis: Folgender Ansatz geht nicht immer, hängt von Security-Einstellungen ab

Ungewöhnliches Beispiel (2/2)

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
public static void main(String[] s) throws Exception{
  Class cl = Class.forName("model.Heimlich");
  Constructor[] cons = cl.getDeclaredConstructors();
  Constructor con = cons[0];
  System.out.println(con);
  if(!con.isAccessible()) // evtl. SecurityManager-Exception
    con.setAccessible(true);
  Object[] parameter = {};
  Heimlich h = (Heimlich) con.newInstance(parameter);
  System.out.println("" + h);
  Field f=cl.getDeclaredFields()[0];
  f.setAccessible(true);
  f.set(h, 42);
  Method m=cl.getDeclaredMethods()[0];
  m.setAccessible(true);
  m.invoke(h, parameter);
}
```

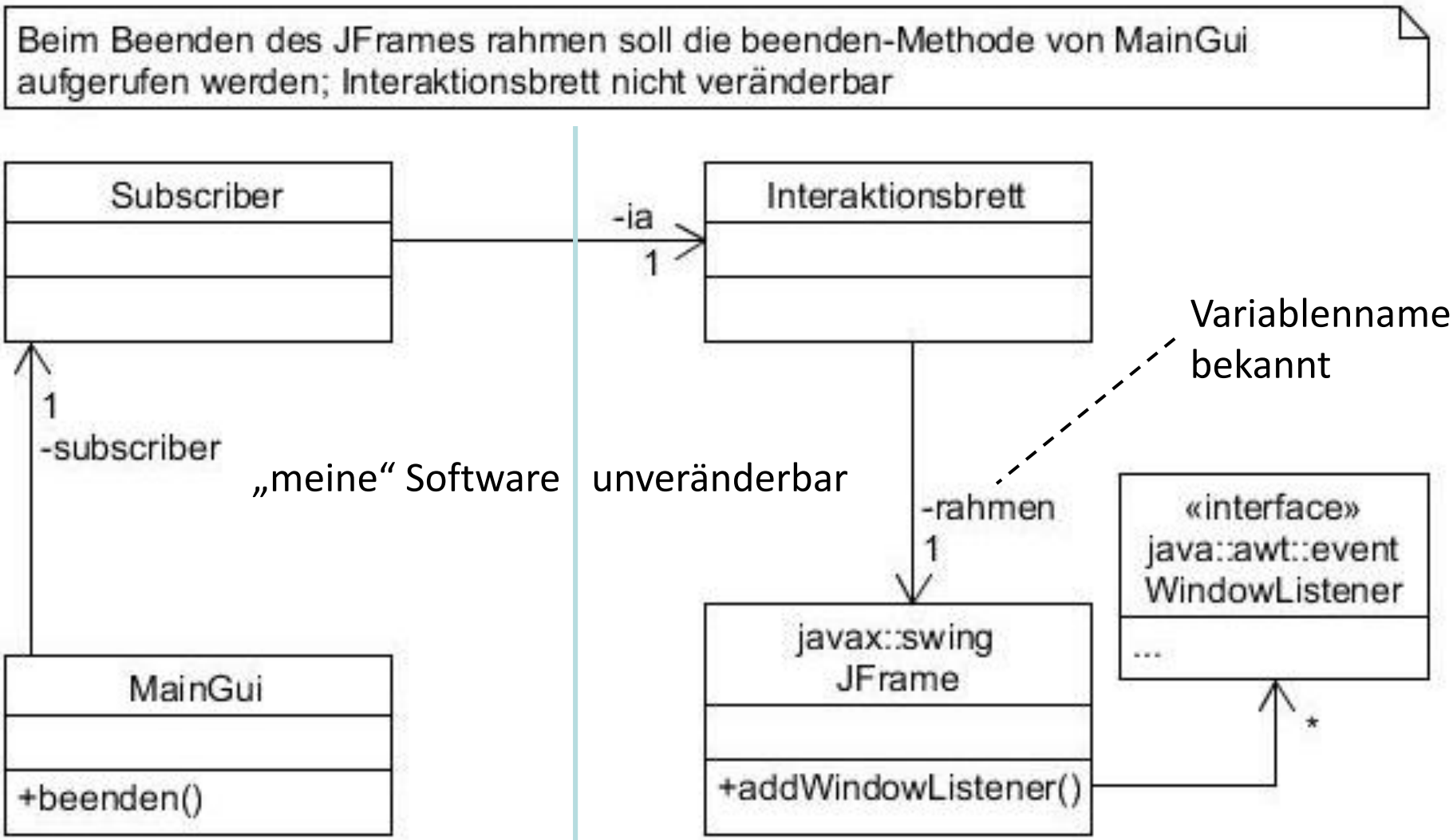
```
private model.Heimlich()
model.Heimlich@19821f
x ist 42
```

Anwendung von Reflexion

- Gegeben Klasse Interaktionsbrett, die nicht verändert werden darf, z. B. aus externer Bibliothek
- Interaktionsbett bietet Knopf zum Schließen des Programms über einen JFrame (Swing) an, der nur in einer privaten Exemplarvariablen rahmen steht

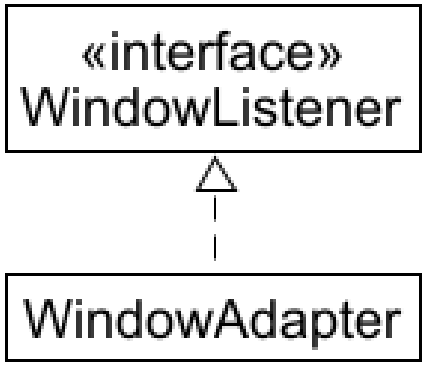
- Schließen erfolgt mit System.exit(0)
- Generell ok, aber haben Nutzer von Interaktionsbrett offene Ressourcen, werden diese ggfls. nicht geschlossen
- Ansatz: Nutze Reflexion, um auf JFrame zuzugreifen und ergänze Methode, die zusätzlich beim Schließen aufgerufen wird
- Bekannt: JFrame führt bei Fenster-Events in Sammlung zugeordneter WindowListener-Objekte passende Methoden aus (Observer-Oberservable)

Ausgangssituation (Skizze)

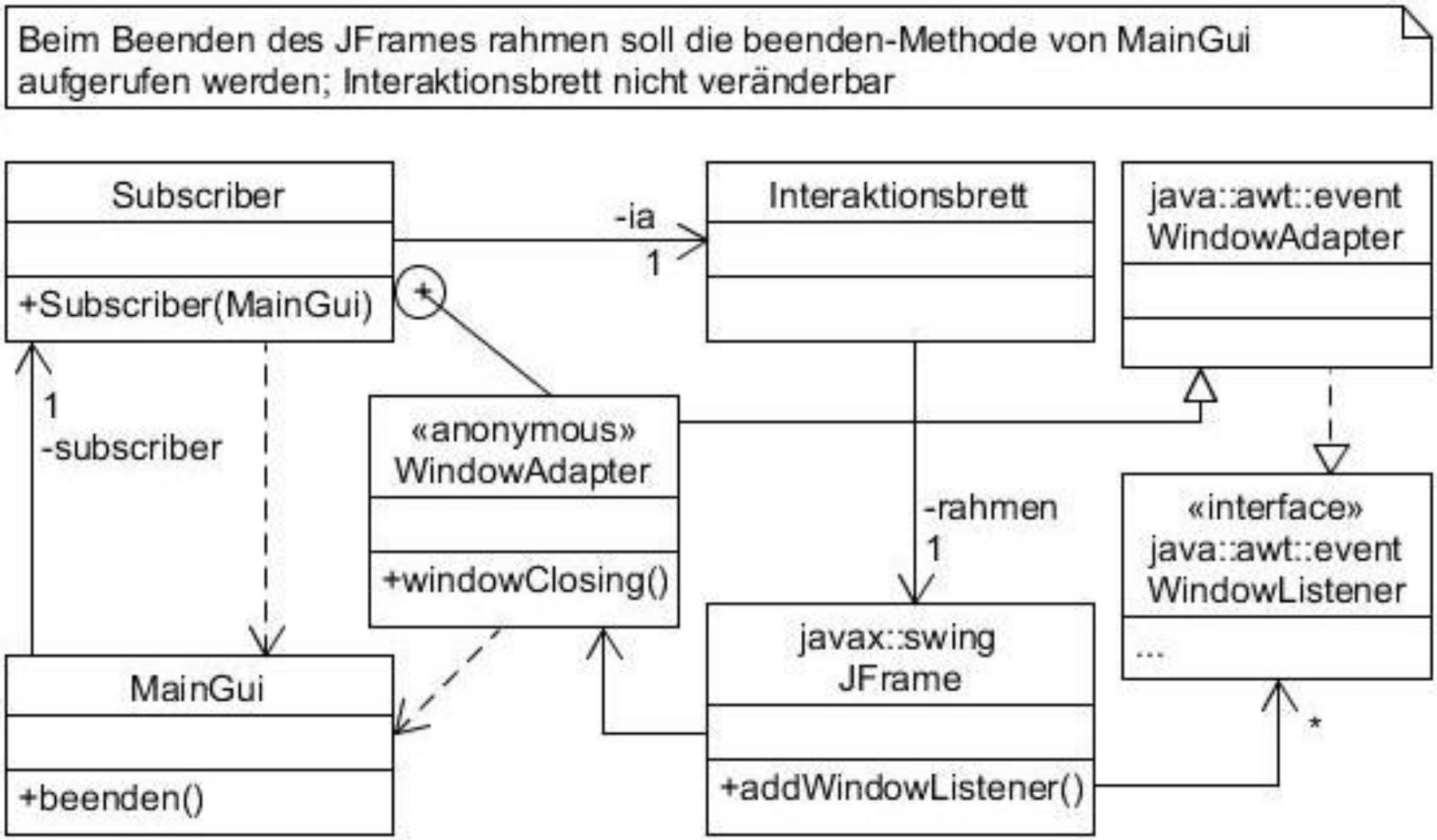


Umsetzung in Klasse Subscriber

```
public Subscriber(MainGui gui){ // gui über Ende informieren
  this.ia = new Interaktionsbrett();
  try {
    Field feld = ia.getClass().getDeclaredField("rahmen");
    feld.setAccessible(true);
    JFrame frame = (JFrame) feld.get(this.ia);
    frame.addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
          gui.beenden();
        }
    });
  } catch (NoSuchFieldException | SecurityException
      | IllegalArgumentException | IllegalAccessException e1) {
    e1.printStackTrace();
  }
}
```

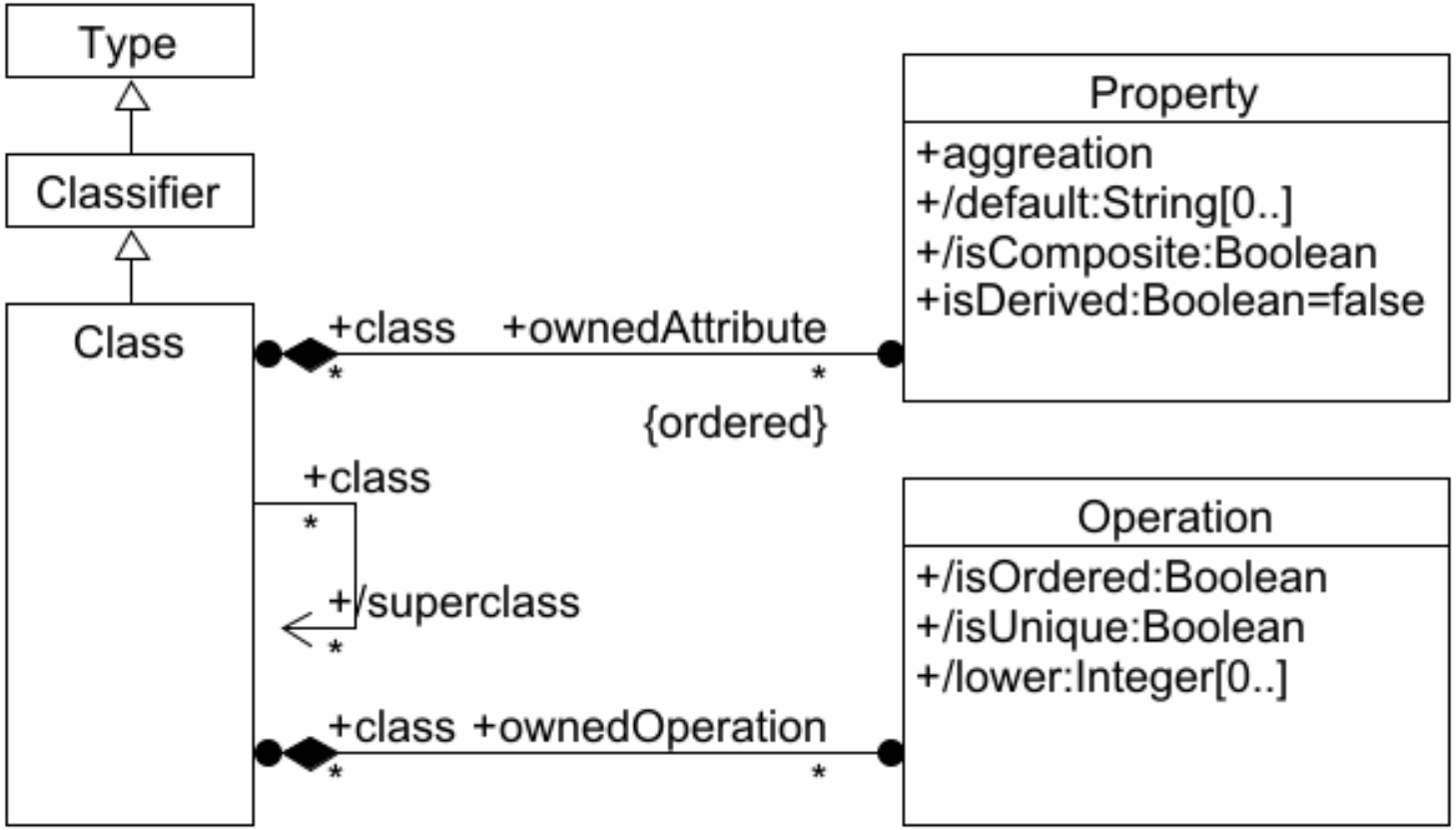


Ergebnis (Skizze, Grenze der UML-Darstellung)



Metamodellierung in der UML (kleiner Ausschnitt)

- Modell zur Spezifikation und Semantik-Definition nicht zur Implementierung



- angelehnt an Meta Object Facility (MOF), v2.5.1, S. 27, https://www.omg.org/spec/MOF

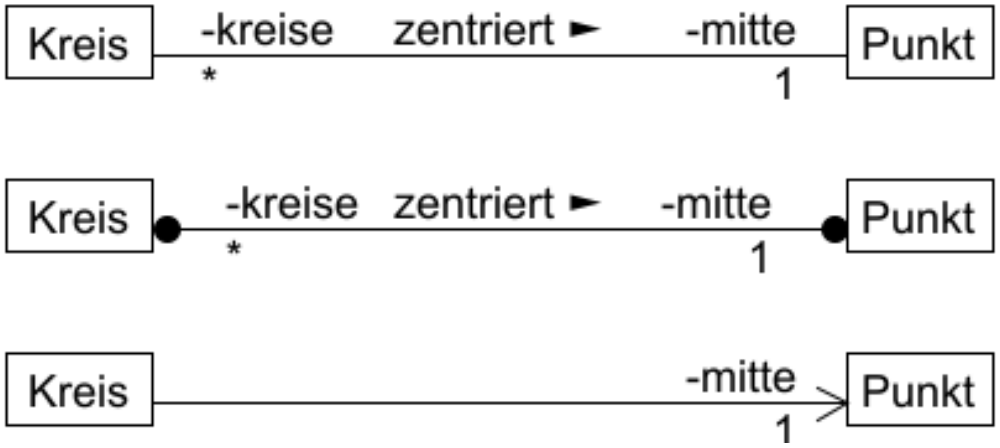
„Punkt“ genauer

```
class zentriert {
  Collection<Kreis> kreise
  Punkt mitte
```

```
class Kreis {
  Punkt mitte

class Punkt{
  Collection<Kreis> kreise
```

```
class Kreis {
  private Punkt mitte
  // genauer: Punkt fehlt
```



- typischerweise nur in Analysemodellen genutzt, da bei Implementierung Assoziationen nie zu eigenen Klassen führen
- beschreibt wem gehört Beziehung
- da wir Realisierungsmodelle machen, ignorieren

Java - Ausblick

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

```
-javaagent:F:\kleukersSEU\gsdet\Sequencediagrammer.jar
```

- Java liegt als Byte-Code vor
- JVM erkennt am Anfang welche Klassen zum Start benötigt werden und lädt deren Byte-Code
- hier kann bereits eingegriffen werden, der Byte-Code kann analysiert und verändert werden, dazu wird JVM ein (oder mehrere) Java-Agent übergeben
- Beispiel: Rufe am Anfang und Ende der Methodenausführung Methode eines Protokollierer auf, nutze Aufrufe zur Erstellung eines Sequenzdiagramms
- erkennt JVM zur Laufzeit, dass weitere Klasse benötigt wird, wird dies geladen und Agent kann wieder eingreifen
- Java-Sicherheitsmechanismen können Reflexion und Agents verhindern bzw. einschränken (starke Einschränkung durch Module)