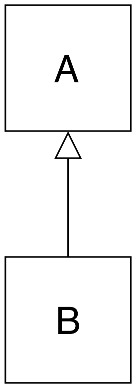
2.2 Class diagrams in Java

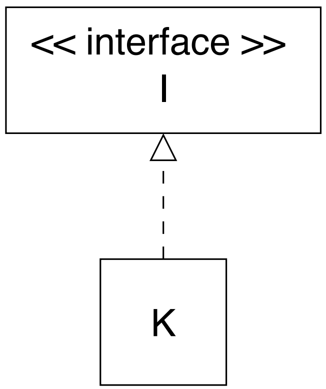
- The static information of a class diagram can be translated directly into Java.
- The code skeleton has no implemented methods.

2.2.1 Declaring Classes and Interfaces

UML	Java
K	`class K {...}`
K {abstract}	`abstract class K {...}`
<< interface >> I	`interface I {...}`

UML





Java

```
class A {...}

class B extends A {...}
```

```
interface I {...}

class K implements I {...}
```

2.2.2 Declaring Attributes

UML	**Java**
attribute:Type	JavaType attribute;

The standard types of UML are translated as follows to Java.

UML	**Java**
Boolean	boolean
Integer	int
Real	float or double
String	String

2.2.3 Declaring Methods

UML	**Java**
op(x: Type)	void op(JavaType x) {...}
op(x: Type): ResType	JavaResType op(JavaType x) {...}
op() {abstract}	abstract void op();
K(x: Type)	K(JavaType x) {...} // constructor

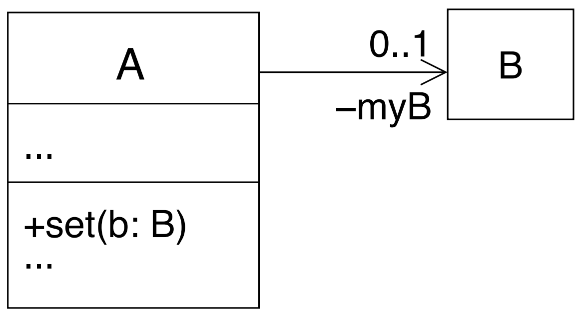
2.2.4 Defining Access Rights

UML	**Java**	
-	private	//accessible within the class
$\sharp$	protected	//accessible in subclasses and in the package
$+$	public	//accessible outside the class
$\sim$		//java default: accessible in the package

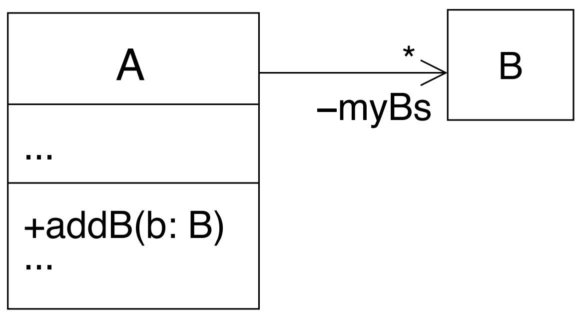
2.2.5 Defining Directed Associations

UML

Java



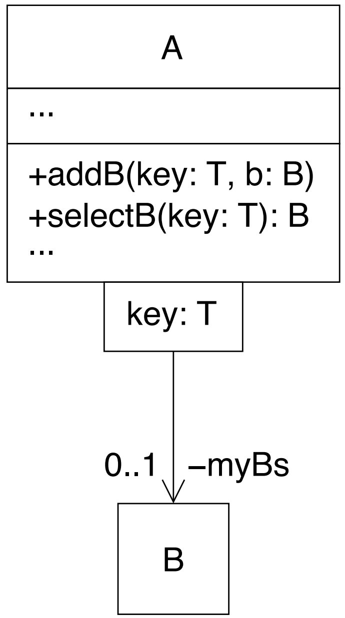

```
class A {
  private B myB; //Referenzattribut
  ...
  public void set(B b) {
    myB = b;
  }
  ...
}
```



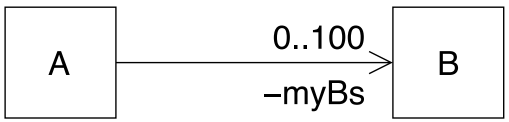

```
//Set = Interface für Mengen
//HashSet = Implementierung v. Set

import java.util.*;

class A {
  private Set<B> myBs = new HashSet<B>();
  ...
  public void addB(B b) {
    myBs.add(b);
  }
  ...
}
```

UML

Java

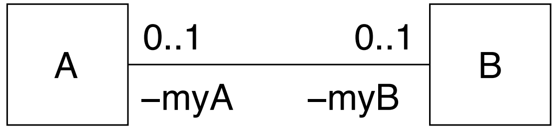



```
//Map = Interface für Schlüssel/Element-Paare
import java.util.*;
class A {
  private Map<T,B> myBs = new HashMap<T,B>();
  ...
  public void addB(T key, B b) {
    myBs.put(key, b);
  }
  public B selectB(T key) {
    return myBs.get(key);
  }
  ...
}
```




```
class A {
  private B[] myBs = new B[100];
  ...
}
```

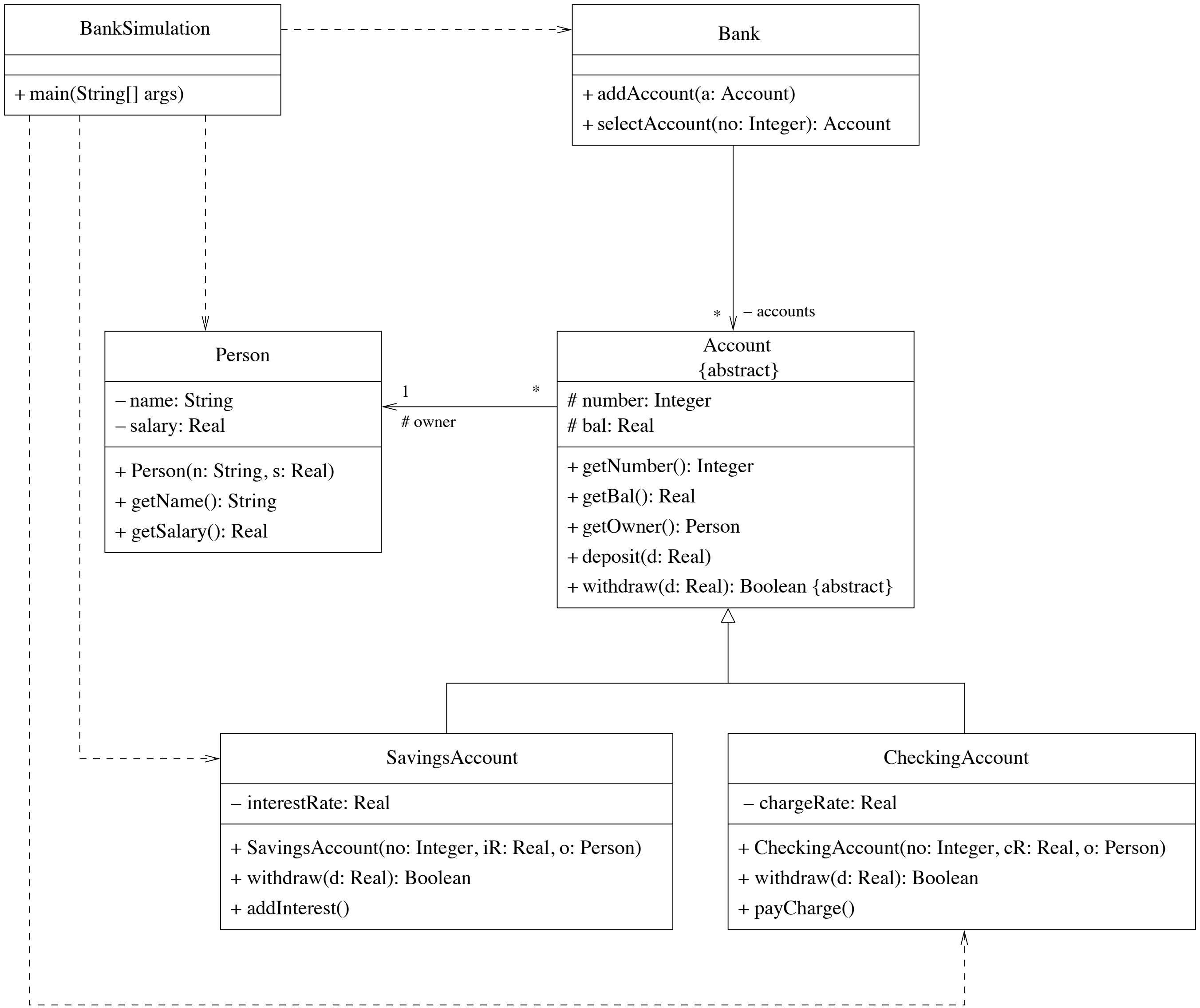
2.2.6 Defining Bidirectional Associations




```
class A {
  private B myB;

  public B getMyB() {
    return myB;
  }

  public void relate(B b) {
    myB = b;
    myB.setMyA(this);

  }
  public void unrelate() {
    myB.unset();
    myB = null;
  }
}
```

```
class B {
  private A myA;

  public A getMyA() {
    return myA;
  }
  void setMyA(A a) {
    myA = a;
  }
  void unsetMyA() {
    myA = null;
  }
}
```

2.2.7 Example (Class Diagram)



Example (Code Skeleton)


```
class BankSimulation {
    public static void main(String[] args) {
        // to be filled
    }
}
import java.util.*;
class Bank {
    private Set<Account> accounts = new HashSet<Account>();
    public void addAccount(Account a) {
        // to be filled
    }
    public Account selectAccount(int no) {
        // to be filled
    }
}
class Person {
    private String name;
    private double salary;
    public Person(String n, double s) {
        // to be filled
    }
    public String getName() {
        // to be filled
    }
    public double getSalary() {
        // to be filled
    }
}
```

```
abstract class Account {
    protected int number;
    protected double bal;
    protected Person owner;

    public int getNumber() {
        // to be filled
    }
    public double getBal() {
        // to be filled
    }
    public Person getOwner() {
        // to be filled
    }
    public void deposit(double d) {
        // to be filled
    }
    public abstract boolean withdraw(double d);
}
class SavingsAccount extends Account {
    private double interestRate;
    public SavingsAccount(int no,double iR,Person o) {
        // to be filled }
    public boolean withdraw(double d) {
        // to be filled  }
    public void addInterest() {
        // to be filled  }
}
class CheckingAccount extends Account {...}
```

2.3 Modelling of Dynamic Behaviour

Techniques

- *Interaction diagrams:*
  describe the communication and cooperation of *several* objects.
- *State diagrams:*
  describe the behaviour of *one* object of a certain class at runtime.
- *Activity diagrams:*
  describe (possibly parallel) traces of activities.

2.3.1 States and Events

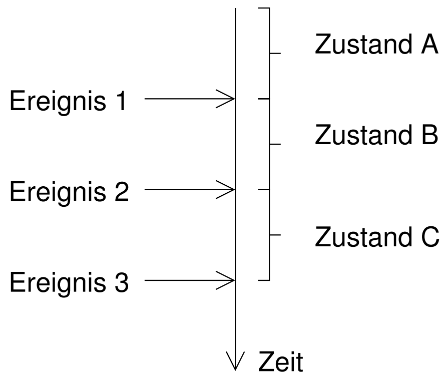
States

- A state is a situation during the lifetime of an object during which some condition is satisfied:
  - The object is performing some activity (do activity).
  - The object is waiting for some event to trigger a change in state.
- When the object satisfies the conditions for a state, the state is said to be active.
- In general, an object is associated with a set of active states.
- A state machine specifies the sequence of states that an object may go through during its lifetime.

Notation

- States are represented graphically by a rectangle with rounded corners.
- It may optionally have a name (a string).

Event = something which happens at a certain time point.



Event Kinds

- Signal event (e.g., pushing a button, open a door)
- Call event (calling a function, e.g., `myKonto.einzahlen(1000)`)
- Change event (e.g., **when** (temperature < 0))
- Time event (e.g., **after**(5 sec))
- Completion event (e.g., downloading a file)

Note:
Events have a duration (in contrast to states)!

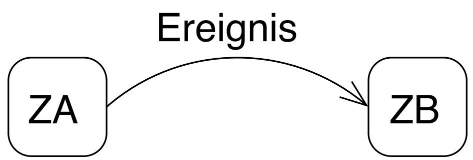
2.3.2 Flat State Diagrams

Directed graph with

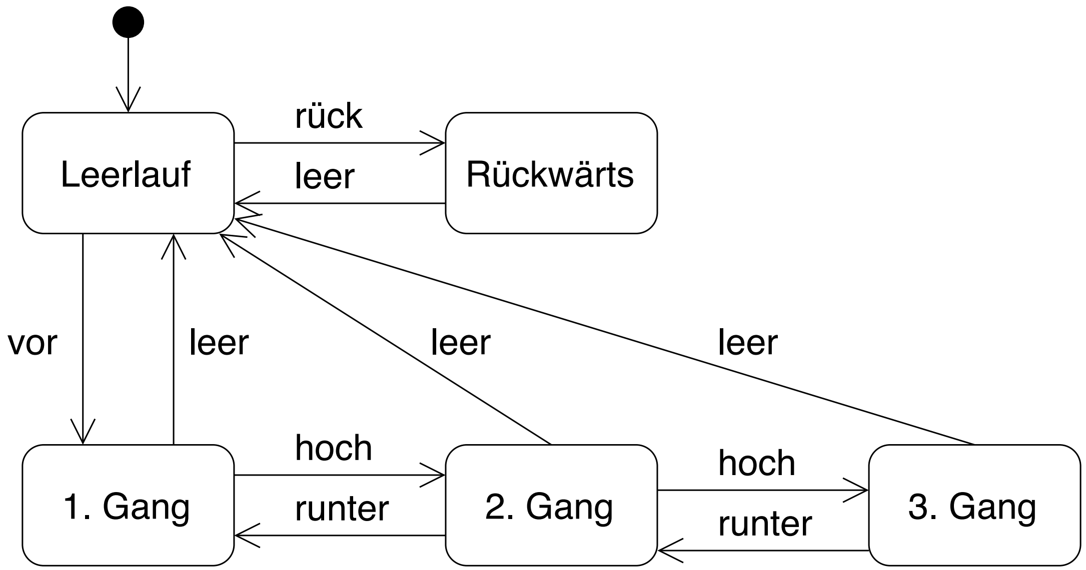
- nodes = states
- arcs = transitions

Transition

describes an event-induced change from the *source* state ZA to the *target* state ZB.



Example: Automatic Gearbox



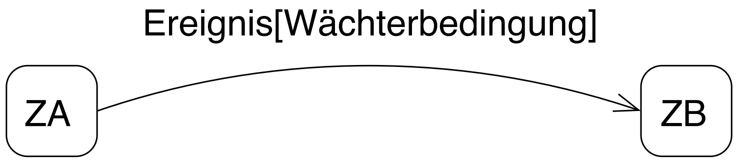
Remarks

- A state with no transitions for a certain event ignores such an event.
- The symbol • denotes the initial state ("pseudo state").
- The symbol ⊙ denotes a final state (destruction of the object or termination of an activity).

Guards

- A condition (boolean statement) can be used as a guard for a transition.
- The transition fires if the event occurs and the condition holds.

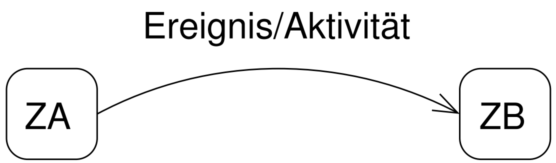
Syntax:



Activity

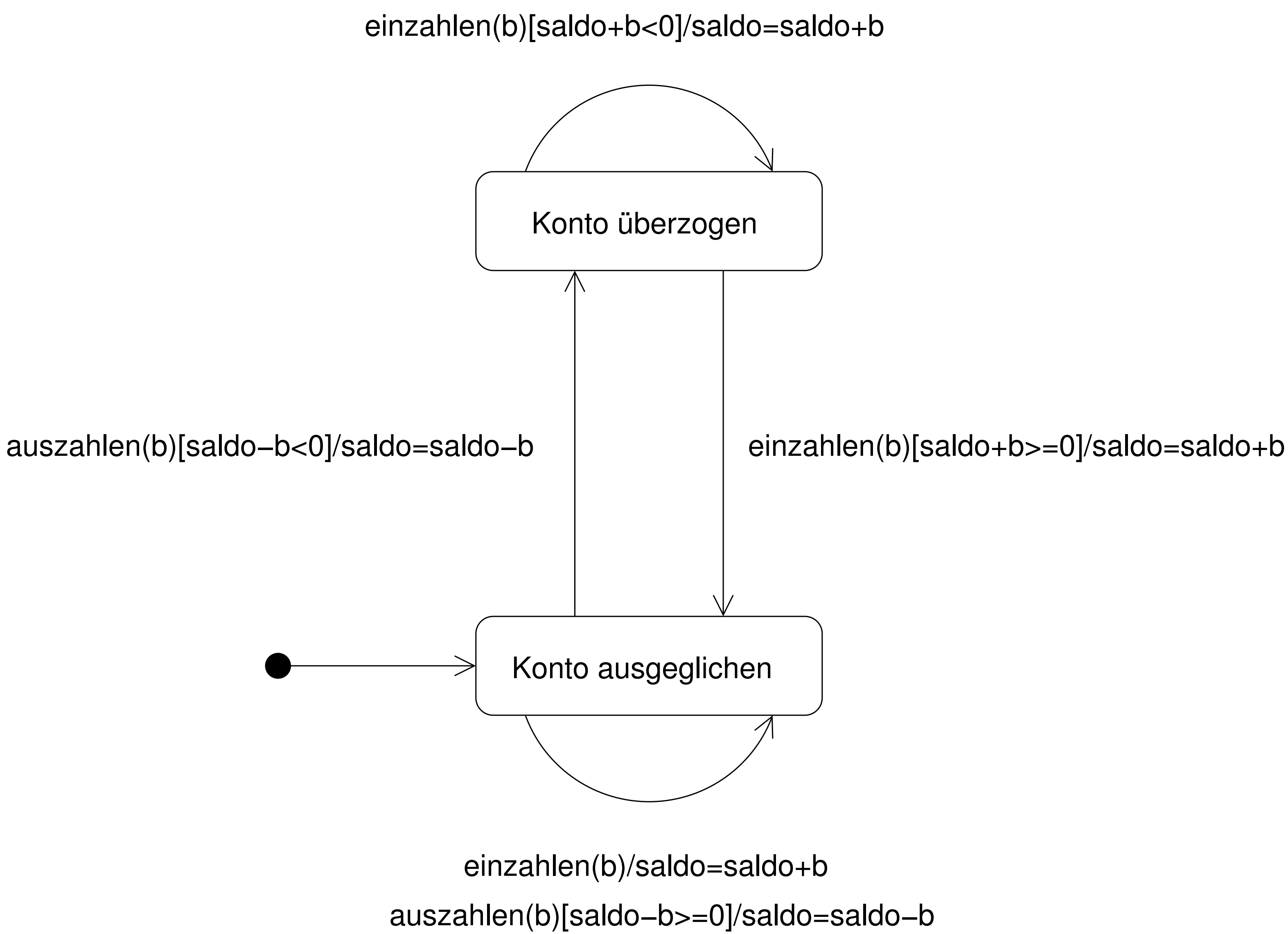
- Action (which may need time).
- Can be a response to an event.

Syntax:

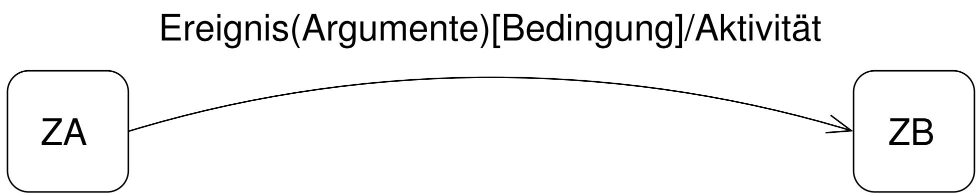


Note:
An activity on the transition arc cannot be interrupted by an event.

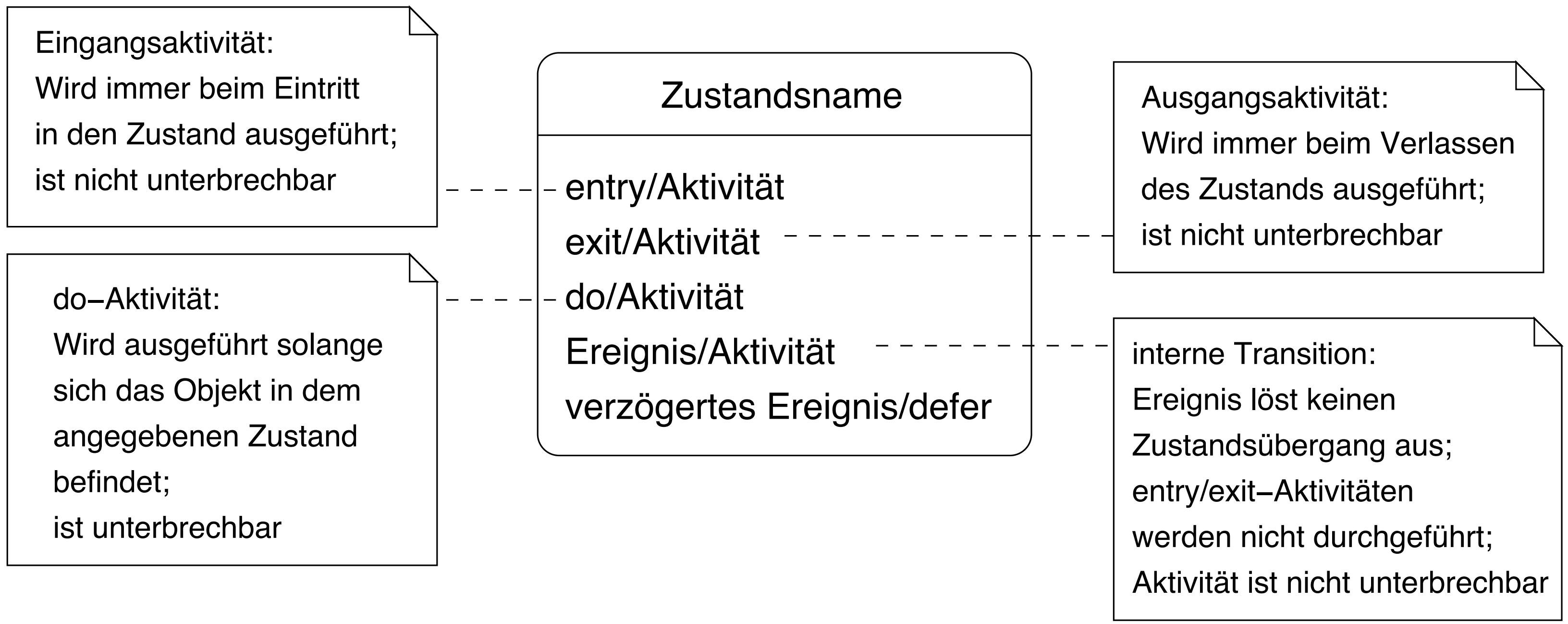
Example:


einzahlen(b)[saldo+b<0]/saldo=saldo+b
Konto überzogen
auszahlen(b)[saldo−b<0]/saldo=saldo−b
einzahlen(b)[saldo+b>=0]/saldo=saldo+b
Konto ausgeglichen
einzahlen(b)/saldo=saldo+b
auszahlen(b)[saldo−b>=0]/saldo=saldo−b

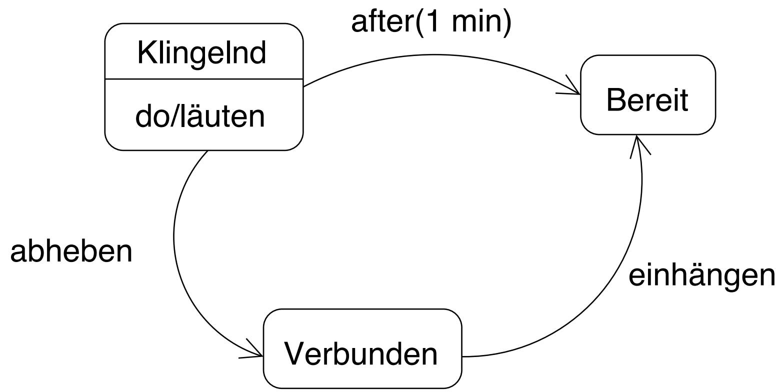
General Transition Syntax



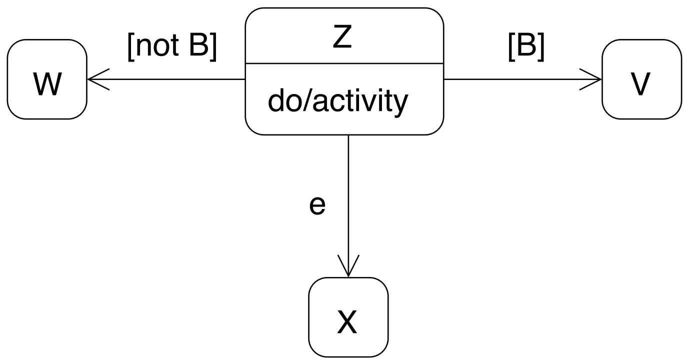
General State Syntax



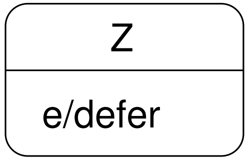
Example: Telephone



Note:
A (perhaps conditional) completion event arises if the do activity terminates on its own.

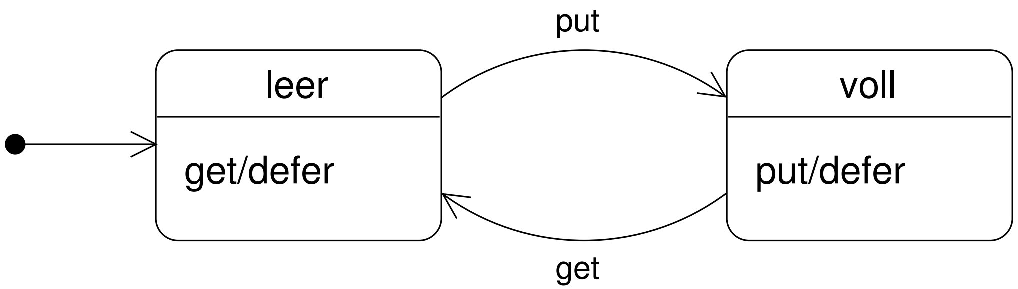


Delayed Event



An event e is stored and processed later in some state which can handle e, if the event e arises before in a state Z which has no outgoing e transitions.

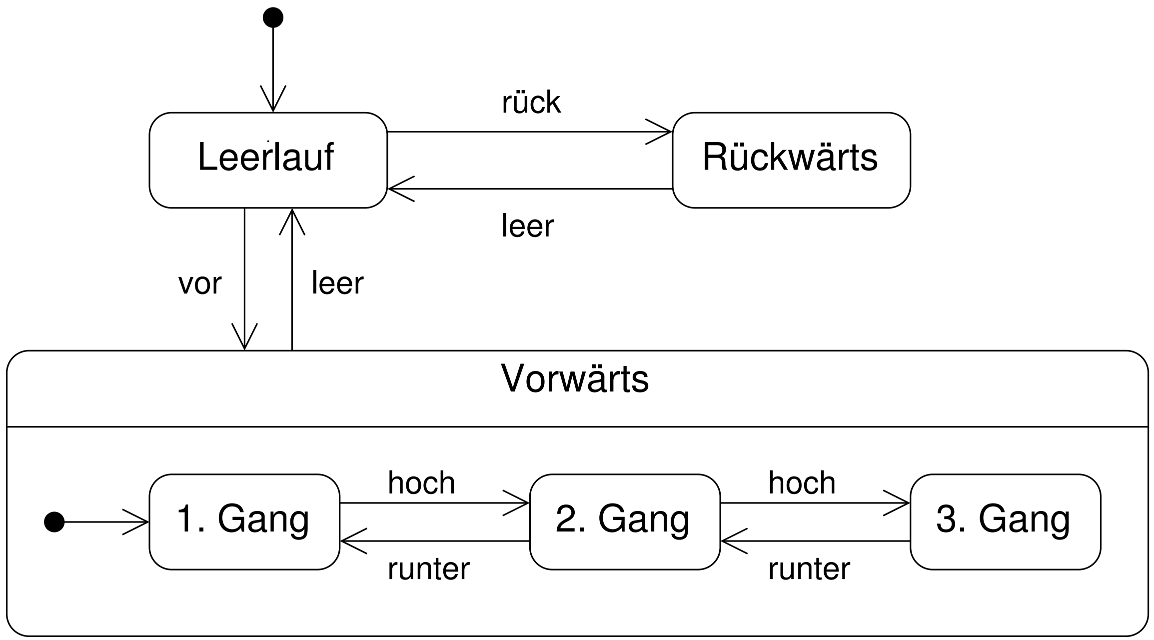
Example: Buffer with one element



2.3.3 Hierarchical State Diagrams

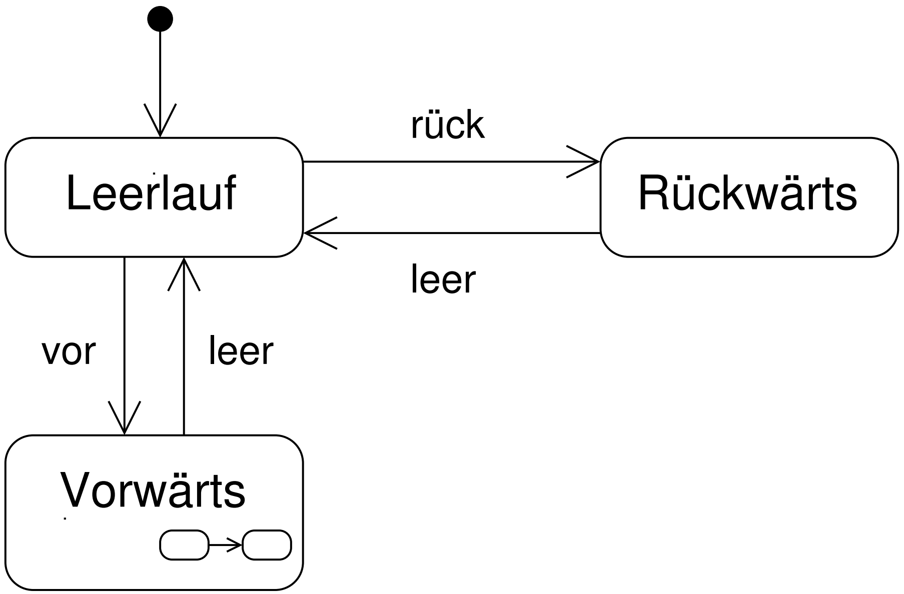
A state can be refined into substates.

1. Sequential Substates



- A transition in a superstate (Vorwärts) refers to a transition in the initial state of the nested diagram (1. Gang)
- A transition from a superstate refers to a transition from a contained substate.

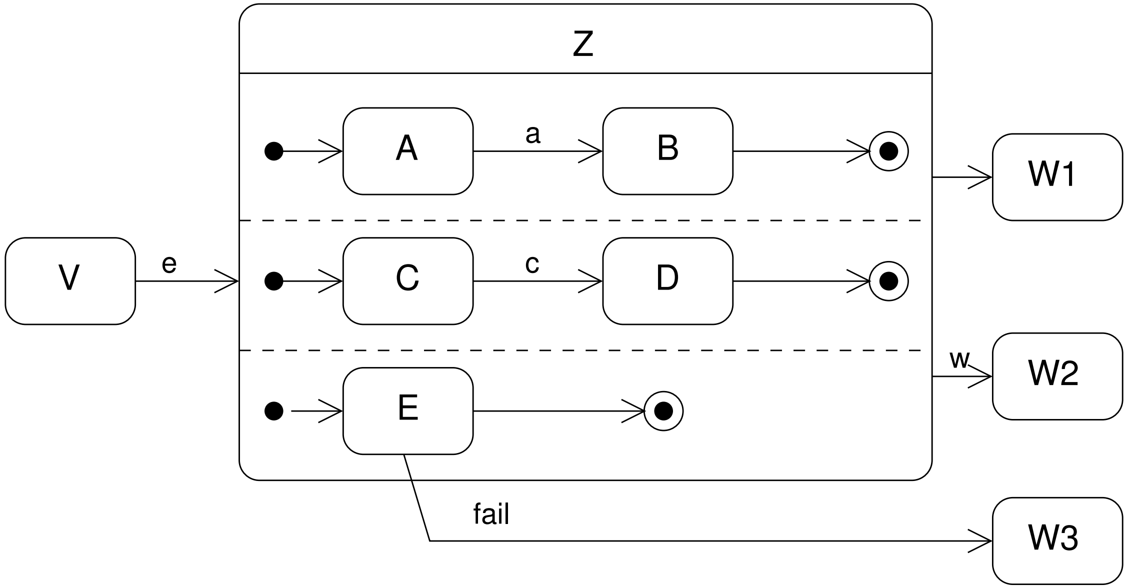
Abstract representation of a complex state:



Remark

Complex states can be equipped with *entry* and *exit* points.

2. Parallel Substates



- An object is in several states at the same time. Entering the superstate means therefore that the object is in the initial state of each region.
- The superstate will be exited if one reaches the final state in each single region, or there is a direct outgoing transition from a substate, or there is an outgoing transition from the superstate originating from an explicit event.

2.3.4 Activity Diagrams

Can be used to describe the behaviour of

- business processes
- use cases
- operations and processes

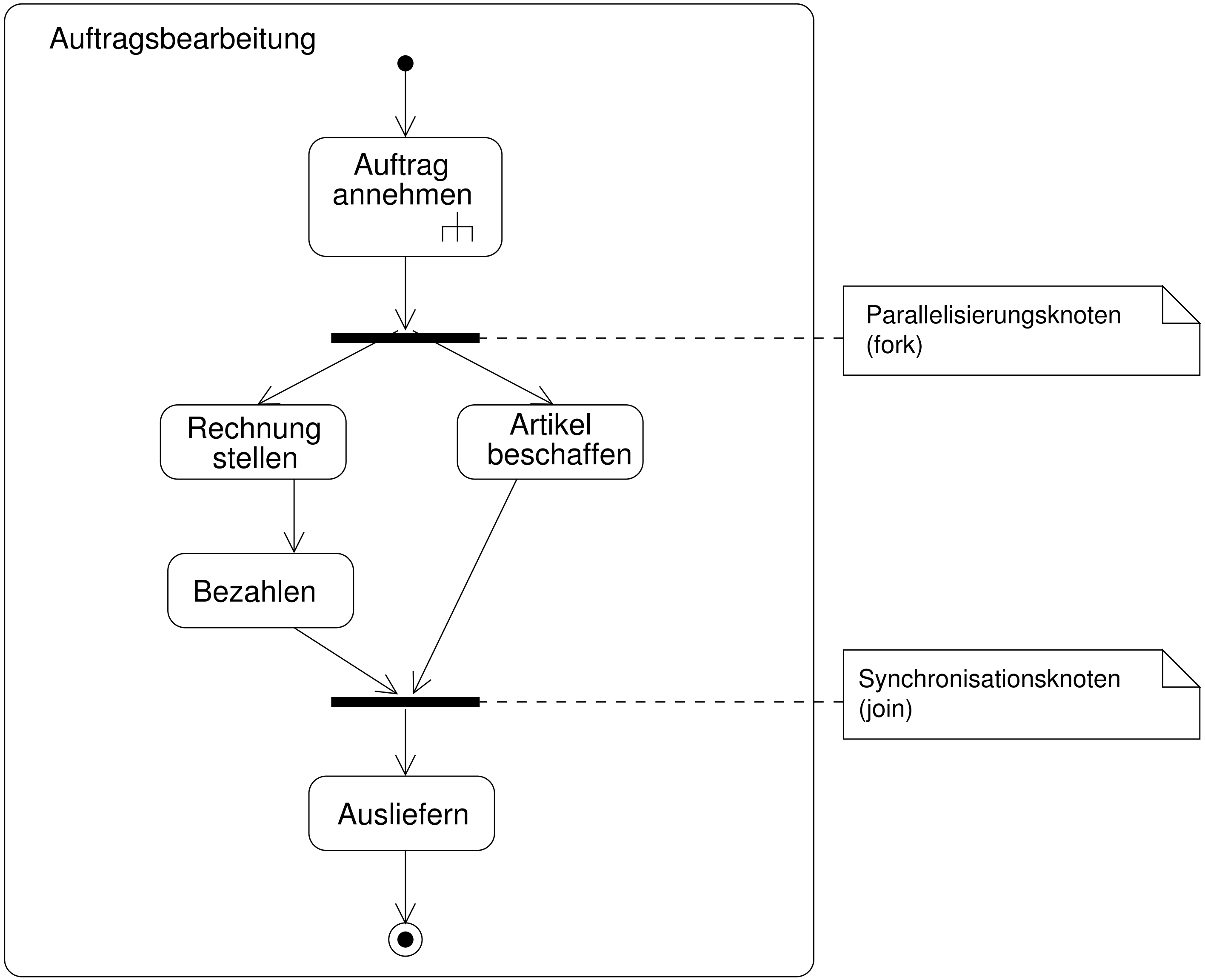
An activity diagram is a directed graph which contains

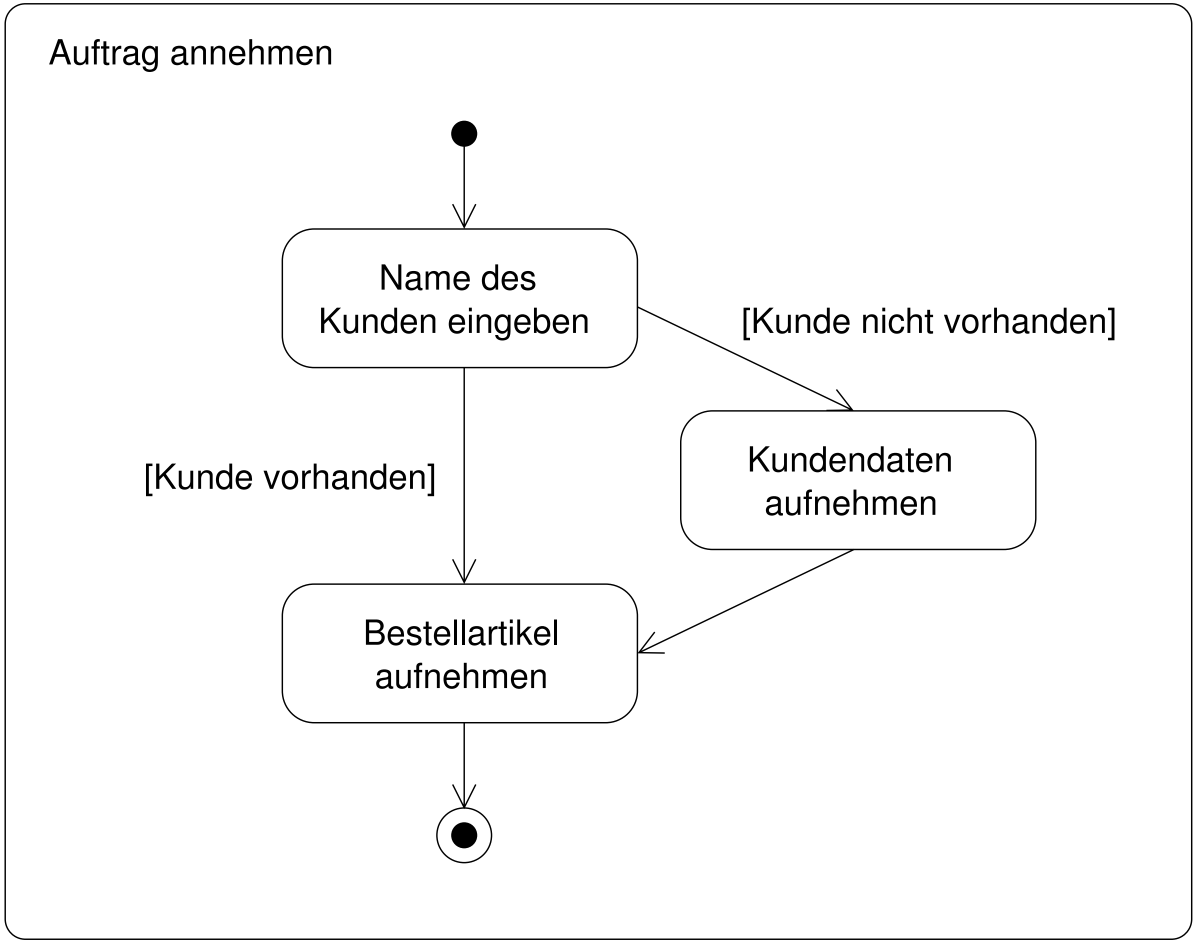
- *activity nodes*: describe actions, control structures and data.
- *activity arcs*: connect activity nodes, inducing therefore traces.

Remark

Activities which are expressed as entry, exit and do activities with respect to a state can be modelled by activity diagrams in a more precise way.

Example: Business process "Order Processing"





Bemerkung

Case distinctions can be modelled by *decision nodes*.

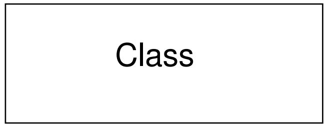
Conclusion of Section 2.3

- State diagrams describe the behaviour of each object from a certain class during its lifetime.
- A transition refers to the state change caused by an event.
- Events are in contrast to states (and activities) timeless.
- We distinguish between five different kinds of events.
- Events can be guarded and an event can be followed by an activity.
- State diagrams can have a hierarchical structure.
  We distinguish between
  - sequential substates
  - parallel substates
- Activity diagrams describe traces of activities.

2.4 Meta Modelling

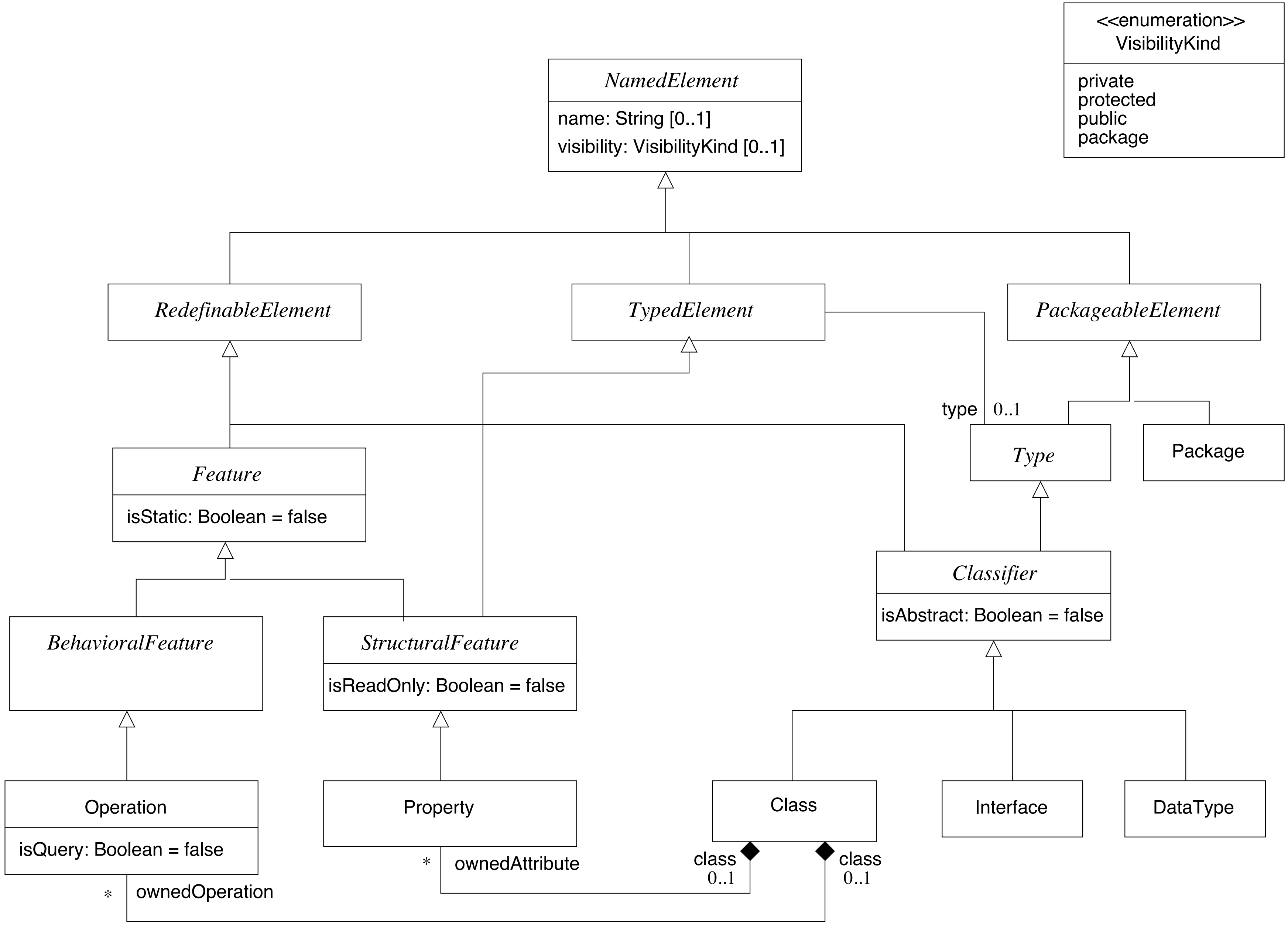
All concepts which are used in an UML model (e.g., class, operation, state, activity, ...) can be described by a class model.

Example: A metaclass has classes as instances

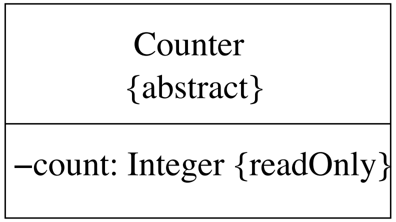


- The meta model specifies all valid UML models.
- This gives us
  - a tool for checking the syntactical correctness of UML models
  - a basis for generic formats (XMI)
- The meta model can be extended to business modelling, web engineering, ...

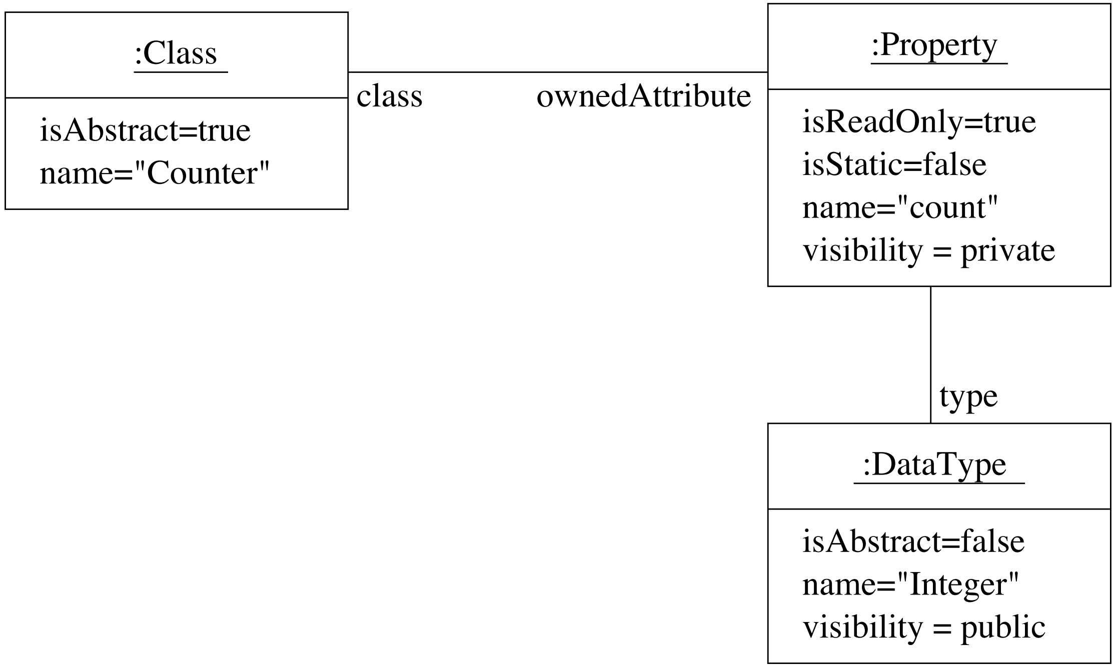
A section of an UML meta model



A possible application of the meta model



Darstellung der Klasse Counter als Instanzendiagramm des Metamodells



Meta model for state diagrams

(UML 2.2 Superstructure Specification)

Package BehaviorStateMachines