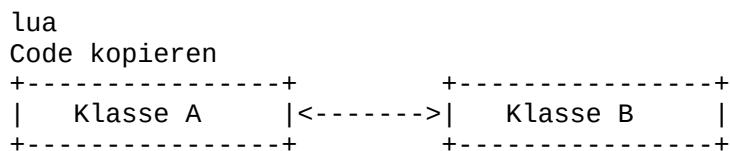


In **UML-Diagrammen** für Java repräsentieren Pfeile verschiedene Arten von Beziehungen zwischen Klassen, Interfaces oder Objekten. Die wichtigsten Pfeilarten und ihre Bedeutung werden im Folgenden erklärt, jeweils mit Beispielen in **Java-Code**.

1. Assoziation (Association)

- **Pfeiltyp:** Eine einfache Linie zwischen zwei Klassen (ohne Pfeilkopf oder optional mit Pfeil für gerichtete Assoziation).
- **Bedeutung:** Eine **Beziehung** zwischen zwei Klassen; eine Klasse verwendet die andere.
- **Richtung:** Wenn ungerichtet, können beide Klassen sich kennen; bei gerichteter Assoziation nur in eine Richtung.

UML-Diagramm:



Java-Code:

```
java
Code kopieren
class Engine {
    public void start() {
        System.out.println("Engine started.");
    }
}

class Car {
    private Engine engine; // Assoziation: Car hat eine Engine

    public Car(Engine engine) {
        this.engine = engine;
    }

    public void startCar() {
        engine.start();
    }
}
```

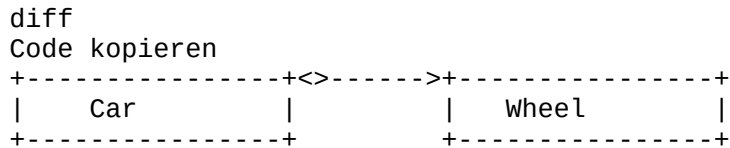
Erläuterung:

- Car kennt Engine, verwendet sie aber nur.
- Der Pfeil in UML kann gerichtet sein, falls nur Car die Engine kennt.

2. Aggregation

- **Pfeiltyp:** Eine Linie mit einem **leeren Rauten-Kopf** an der Aggregations-Seite.
- **Bedeutung:** Eine **Ganz-Teil-Beziehung** mit schwacher Bindung – das Teil kann unabhängig vom Ganzen existieren.

UML-Diagramm:



Java-Code:

```
java
Code kopieren
class Wheel {
    public void roll() {
        System.out.println("Wheel is rolling.");
    }
}

class Car {
    private Wheel[] wheels;

    public Car(Wheel[] wheels) {
        this.wheels = wheels; // Aggregation: Car hat mehrere Wheels
    }

    public void move() {
        for (Wheel wheel : wheels) {
            wheel.roll();
        }
    }
}
```

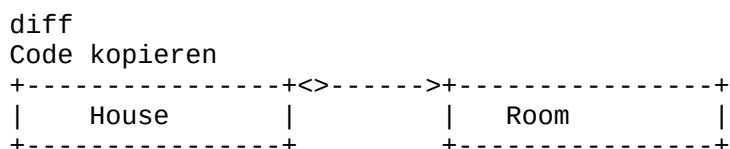
Erläuterung:

- **Aggregation** zeigt, dass `Wheel` unabhängig existieren kann.
 - `Car` hat Räder (`Wheel`), aber die Räder können auch ohne `Auto` existieren.
-

3. Komposition (Composition)

- **Pfeiltyp:** Eine Linie mit einem **gefüllten Rauten-Kopf** an der Kompositions-Seite.
- **Bedeutung:** Eine **starke Ganz-Teil-Beziehung** – das Teil kann **nicht unabhängig** vom Ganzen existieren.

UML-Diagramm:



Java-Code:

```
java
Code kopieren
class Room {
    private String name;
```

```

        public Room(String name) {
            this.name = name;
        }
    }

    class House {
        private Room room;

        public House(String roomName) {
            this.room = new Room(roomName); // Komposition: House "besitzt" Room
        }
    }

```

Erläuterung:

- Wenn ein **HOUSE** zerstört wird, existiert der **ROOM** nicht mehr.
- **Room** wird vollständig im **House** verwaltet und ist von ihm abhängig.

4. Vererbung (Inheritance/Generalization)

- **Pfeiltyp:** Eine **durchgezogene Linie** mit einem **leeren Dreieck** als Pfeilkopf.
- **Bedeutung:** Zeigt eine **is-a** Beziehung – eine Klasse erbt von einer anderen Klasse.

UML-Diagramm:

```

lua
Code kopieren
+-----+
|   Vehicle   |<|-----|   Car   |
+-----+
+-----+

```

Java-Code:

```

java
Code kopieren
class Vehicle {
    public void start() {
        System.out.println("Vehicle started.");
    }
}

class Car extends Vehicle {
    @Override
    public void start() {
        System.out.println("Car started.");
    }
}

```

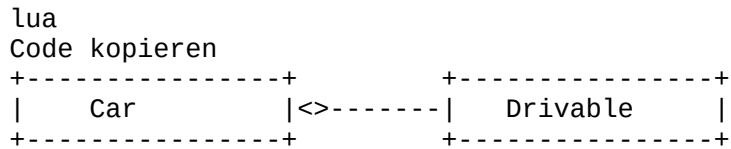
Erläuterung:

- **Car** erbt von **Vehicle** und erweitert dessen Funktionalität.

5. Implementierung (Realization)

- **Pfeiltyp:** Eine **gestrichelte Linie** mit einem **leeren Dreieck** als Pfeilkopf.
- **Bedeutung:** Eine Klasse **implementiert ein Interface**.

UML-Diagramm:



Java-Code:

```
java
Code kopieren
interface Drivable {
    void drive();
}

class Car implements Drivable {
    @Override
    public void drive() {
        System.out.println("Car is driving.");
    }
}
```

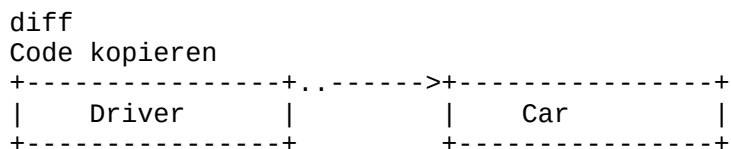
Erläuterung:

- Car implementiert das Interface Drivable und garantiert, dass die Methode drive() definiert wird.

6. Abhängigkeit (Dependency)

- **Pfeiltyp:** Eine **gestrichelte Linie** mit einem **Pfeilkopf**.
- **Bedeutung:** Eine Klasse verwendet eine andere Klasse, meist nur kurzfristig.

UML-Diagramm:



Java-Code:

```
java
Code kopieren
class Car {
    public void drive() {
        System.out.println("Car is driving.");
    }
}
```

```

class Driver {
    public void driveCar(Car car) {
        car.drive(); // Abhängigkeit: Driver nutzt Car
    }
}

```

Erläuterung:

- Driver hängt von Car ab, aber nur temporär (z.B. als Parameter).

Zusammenfassung der Pfeile:

Pfeil	Beziehung	Beschreibung
Linie	Assoziation	Eine Klasse nutzt die andere.
Linie mit leerem Rhombus	Aggregation	Ganz-Teil-Beziehung (schwach).
Linie mit gefülltem Rhombus	Komposition	Ganz-Teil-Beziehung (stark).
Linie mit leerem Dreieck	Vererbung	Eine Klasse erbt von einer anderen Klasse.
Gestrichelte Linie mit Dreieck	Implementierung	Eine Klasse implementiert ein Interface.
Gestrichelte Linie mit Pfeil	Abhängigkeit	Eine Klasse verwendet die andere kurzfristig.