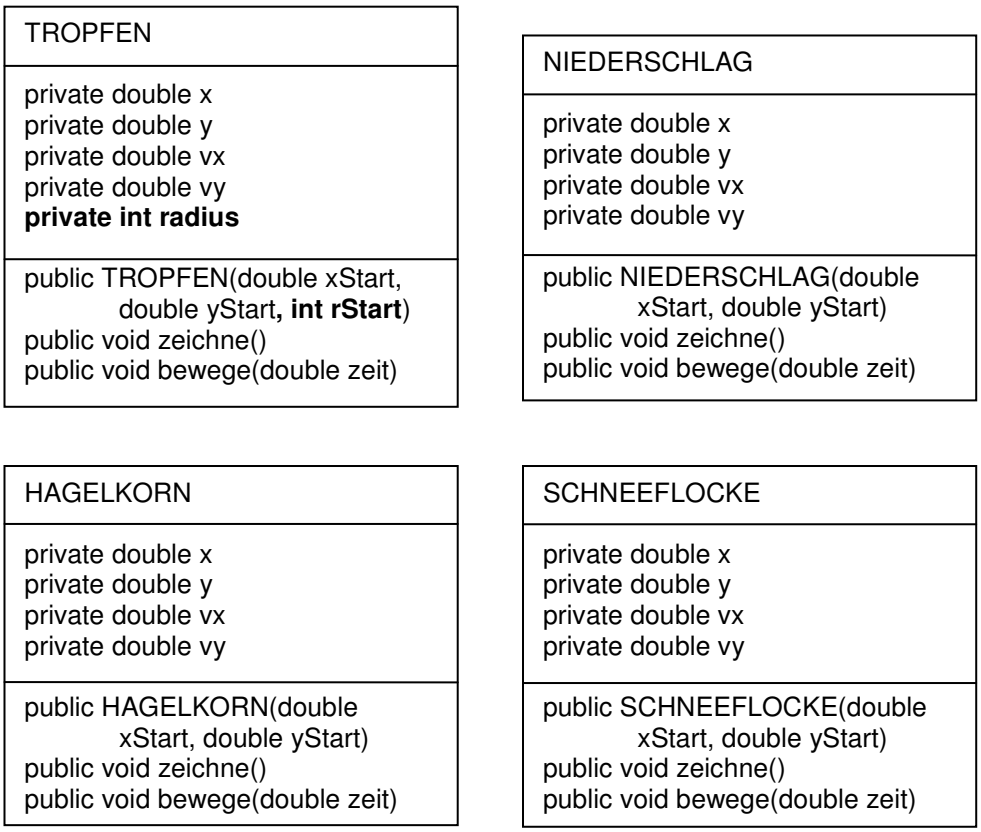
***Ausschnitt des Klassendiagramms zum BlueJ-Projekt „Petrus“***

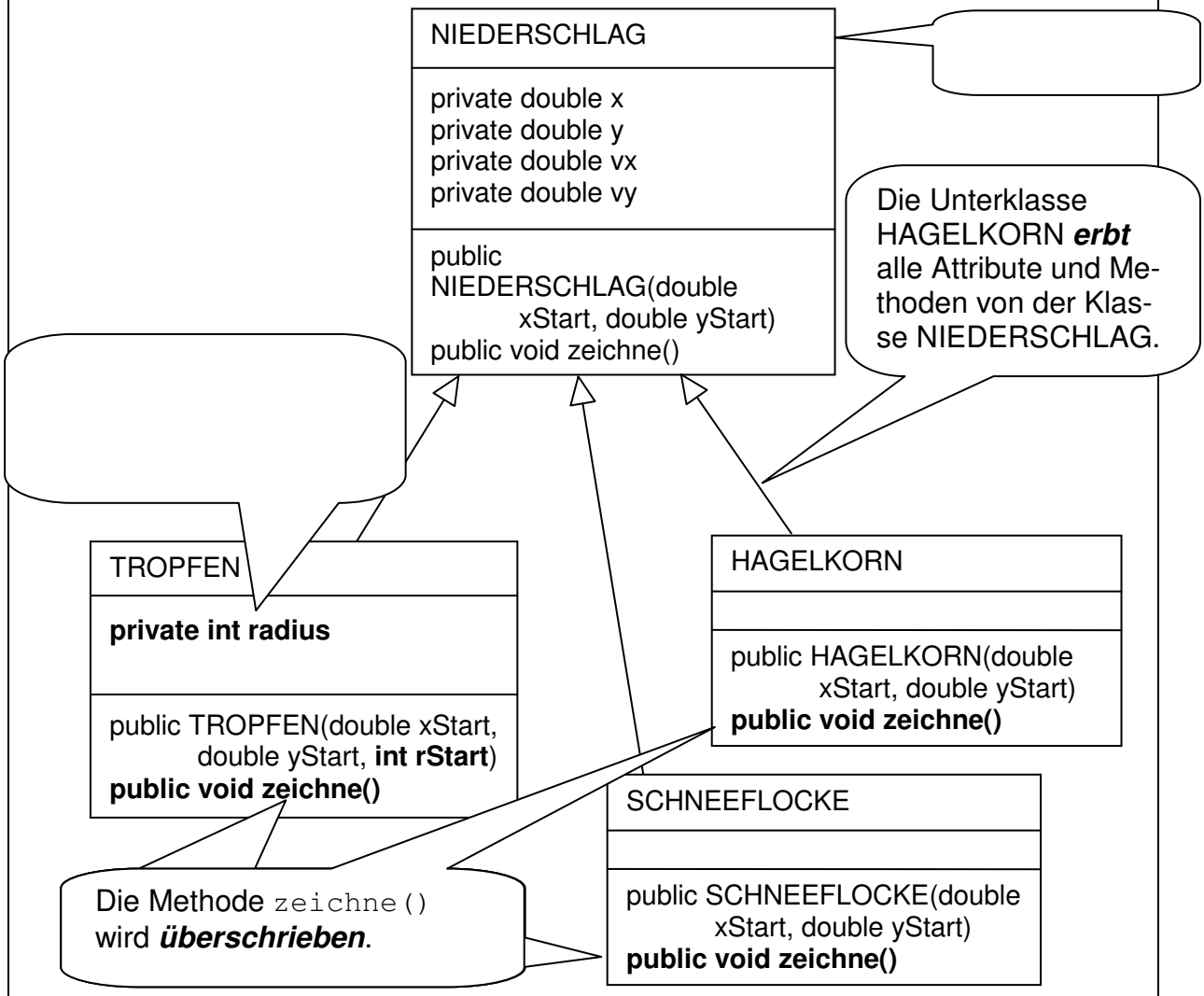



Die Klassendiagramme von TROPFEN, HAGELKORN und SCHNEEFLOCKE zeigen _____ wie NIEDERSCHLAG. TROPFEN besitzt zusätzlich ein Attribut `radius`, dem über einen weiteren Parameter _____ ein Wert zugeordnet wird.

Eine vollständige Neuentwicklung der neuen Klassen ist ineffizient und insbesondere bei Änderungen sehr fehleranfällig. Deshalb wurde das Konzept der ***Spezialisierung und Generalisierung*** entwickelt. Die Klassen TROPFEN, HAGELKORN und SCHNEEFLOCKE sind Spezialisierungen der Klasse NIEDERSCHLAG und werden deshalb als ***Unterklassen*** der ***Oberklasse*** NIEDERSCHLAG bezeichnet. In den Unterklassen werden nur die neuen Attribute und Methoden aufgeführt. Weil `zeichne()` in den Unterklassen neu implementiert wird, muss auch diese Methode jeweils genannt werden. Diese Methode der Oberklasse wird in den Unterklassen ***überschrieben***.

**Generalisierung und Spezialisierung**

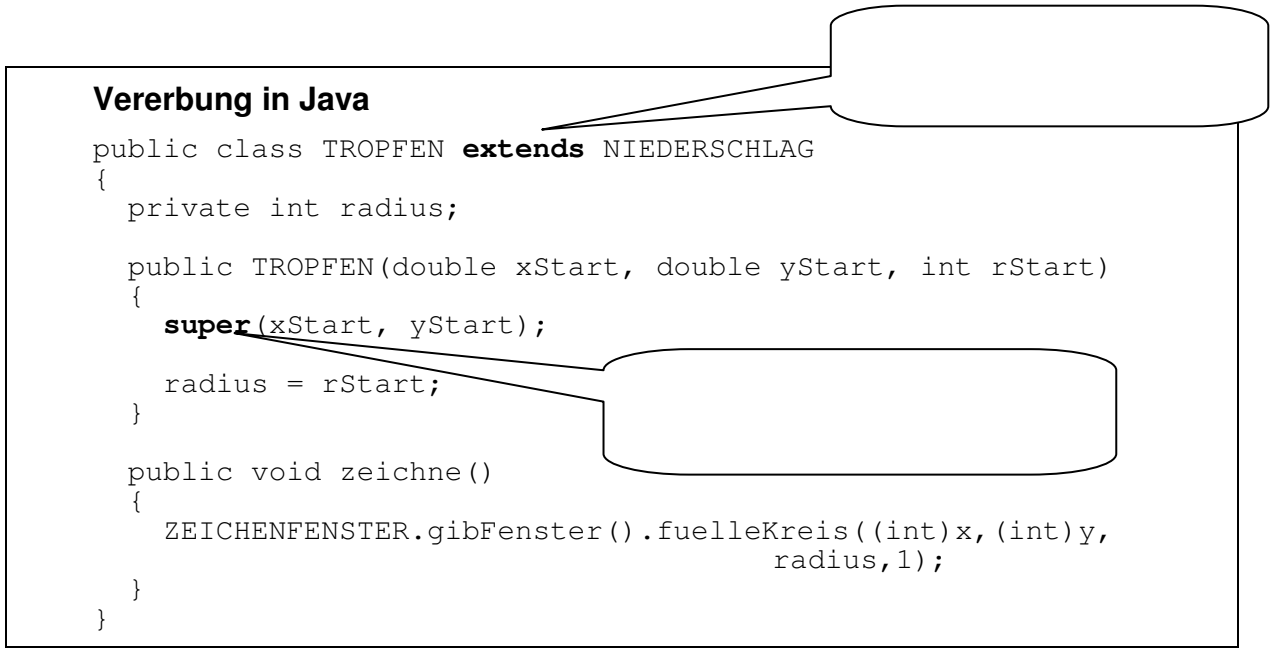
TROPFEN-Objekte, HAGELKORN-Objekte und SCHNEEFLOCKE-Objekte haben alle Attribute und Methoden eines NIEDERSCHLAG-Objektes. Sie unterscheiden sich in der Methode `zeichne()`, TROPFEN hat ein weiteres Attribut. Die objektorientierte Modellierung bietet für eine derartige Situation das Konzept der Spezialisierung/Generalisierung an:




TROPFEN, HAGELKORN und SCHNEEFLOCKE sind ***Spezialisierungen*** von NIEDERSCHLAG; NIEDERSCHLAG ist eine ***Generalisierung*** jeder der Unterklassen.

Bei der Umsetzung in Java wird die Beziehung zur Oberklasse mit dem Schlüsselwort `extends` wiedergegeben. Im Konstruktor der Unterklasse (z. B. TROPFEN) muss zunächst der Konstruktor der Oberklasse NIEDERSCHLAG mit `super()` aufgerufen werden.

**Vererbung in Java**



```
public class TROPFEN extends NIEDERSCHLAG
{
  private int radius;

  public TROPFEN(double xStart, double yStart, int rStart)
  {
    super(xStart, yStart);

    radius = rStart;
  }

  public void zeichne()
  {
    ZEICHENFENSTER.gibFenster().fuelleKreis((int)x,(int)y,
                                            radius,1);
  }
}
```

Damit die Attribute `x` und `y` in der Methode `zeichne()` verwendet werden können, kann der Sichtbarkeitsmodifikator `protected` verwendet werden, der den Zugriff der Unterklasse auf Attribute der Oberklasse ermöglicht. Da damit eine Schwächung der Datenkapselung verbunden ist, sollte der Sichtbarkeitsmodifikator protected nur gezielt eingesetzt werden.

**Polymorphismus**

Ein Attribut oder Parameter vom Datentyp einer Oberklasse kann auch eine Referenz auf ein Objekt einer Unterklasse aufnehmen. Dies nennt man ***Polymorphismus***.

Beispiel:

`fuegeHinzu(NIEDERSCHLAG elementNeu)` in der Klasse WOLKE hat einen Parameter vom Typ NIEDERSCHLAG. Auch ein TROPFEN-Objekt kann als Parameter übergeben werden:

```
TROPFEN tr = new TROPFEN(100, 200, 10);
wolke.fuegeHinzu(tr);
```

In einer Unterklasse kann eine in der Oberklasse vorhandene Methode neu definiert werden. Beim Aufruf wird dann die Methode der Klasse des ausführenden Objekts verwendet. Auch das gehört zum Polymorphismuskonzept.

Beispiel:

Die Methode `zeichne()` wird in TROPFEN, HAGELKORN und SCHNEEFLOCKE überschrieben (engl. override).

`elemente[i].zeichne()` in der Klasse WOLKE ruft die Methode `zeichne()` entweder der Klasse TROPFEN, HAGELKORN oder SCHNEEFLOCKE auf, je nachdem zu welcher Klasse `elemente[i]` gehört.