Vererbung: Generalisierung und Spezialisierung

Eine kleine Bank bietet drei Arten von Konten an: Girokonten, Sparkonten und Geschäftskonten. Alle drei Kontoarten haben die Methoden `Einzahlen`, `Abheben` und `KontostandGeben` sowie die Attribute `kontostand` und `kontonummer`.

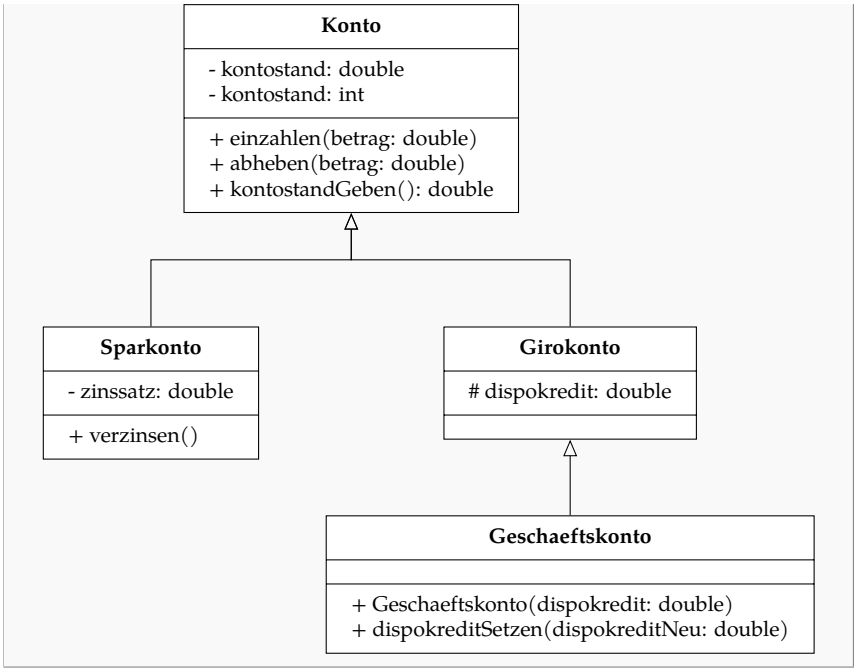
- Sparkonten haben einen Zinssatz und eine Methode `Verzinsen`, die den Jahreszins zum Guthaben addiert. Maximalbetrag beim Abheben ist der aktuelle Kontostand.
- Girokonten können um bis 2000 € überzogen werden (Dispokredit).
- Geschäftskonten haben einen variablen Dispokredit, der über die Methode `DispoKreditSetzen` festgelegt wird; der Startwert für den Dispokredit wird mit dem Konstruktor beim Einrichten des Kontos als Parameter mitgegeben.

(a) Überlege dir, welche Konten Generalisierungen bzw. Spezialisierungen anderer Konten sind. Warum ist es sinnvoll, eine Klasse Konto als oberste Klasse Generalisierungshierarchie einzuführen?

> Da alle Konten die Methoden `einzahlen()`, `abheben()` und `kontostandGeben()` sowie die Attribute `kontostand` und `kontonummer` besitzen, bietet sich deren Verwaltung in einer einzigen Klasse an. Da jede Kontoart aber auch individuelle Eigenschaften bzw. Methoden hat, muss es für jede auch eine eigene Klasse geben. Daher bietet es sich an, die Gemeinsamkeiten in eine Oberklasse `Konto` auszulagern (Generalisierung).

(b) Entwirf ein Klassendiagramm für die Klassen `Konto`, `Girokonto`, `Sparkonto`, `Geschaeftskonto`.

> Da das Geschäftskonto genauso wie das Girokonto einen Dispokredit besitzt, dieser nur anders festgelegt wird, wurde bei der Modellierung die Klasse `Geschaeftskonto` als Unterklasse der Klasse `Girokonto` umgesetzt.
>
> Die Oberklasse `Konto` wurde abstrakt modelliert, da von ihr direkt keine Objekte erzeugt werden können.
>
> Die Attribute `kontostand` und `kontonummer` in der Klasse `Konto` haben den Sichtbarkeitsmodifikator `private`, da die Unterklassen nie direkt auf die Attribute zugreifen, sondern die zur Verfügung stehenden Methoden dafür verwenden.



(c) Implementiere die Klassen in einem eigenen Projekt und teste die vorhandenen Methoden.

```java
public abstract class Konto {
  private double kontostand;
  @SuppressWarnings("unused")
  private int kontonummer;

  public Konto(int kontonummer) {
    this.kontonummer = kontonummer;
    kontostand = 0;
  }

  public void einzahlen(double betrag) {
    kontostand = kontostand + betrag;
  }

  public void abheben(double betrag) {
    // Ob abgehoben werden darf, entscheidet die Methode der
    ↪ jeweiligen Unterklasse.
    kontostand = kontostand - betrag;
  }

  public double kontostandGeben() {
    return kontostand;
  }
}
```

```java
public class Sparkonto extends Konto {
  private double zinssatz;

  public Sparkonto(int kontonummer, double zinssatz) {
    super(kontonummer);
    // Aufruf des Konstruktors der Oberklasse
```

```
9       this.zinssatz = zinssatz;
10    }
11
12    /**
13     * Überschreiben der Methode abheben() aus der Oberklasse. Ist
      ↪  genügend Geld auf
14     * dem Sparkonto...? dann darf man den gewünschten Betrag abheben,
      ↪  sonst nicht.
15     */
16    public void abheben(double betrag) {
17      if (kontostandGeben() >= betrag) {
18        super.abheben(betrag);
19      }
20    }
21
22    /**
23     * Der aktuelle Kontostand wird mit dem Zinssatz verrechnet und der
      ↪  sich daraus
24     * ergebende Betrag (= Zinsen) dem Konto gutgeschrieben.
25     */
26    public void verzinsen() {
27      einzahlen(kontostandGeben() * zinssatz);
28    }
29  }
```

```
3   public class Girokonto extends Konto {
4     /**
5      * Die Unterklasse muss auch auf das Attribut zugreifen können.
6      */
7     protected double dispokredit;
8
9     public Girokonto(int kontonummer) {
10      super(kontonummer);
11      dispokredit = 2000;
12    }
13
14    /**
15     * Die Methode abheben() kann direkt von der Oberklasse GIROKONTO
      ↪  und die
16     * Methoden einzahlen() und kontostandGeben() von der Oberklasse
      ↪  KONTO genutzt
17     * werden.
18     *
19     * Überschreiben der Methode abheben() der Klasse KONTO
20     * Ist genügend Geld auf dem Konto bzw. reicht der Dispokredit
      ↪  aus...
21     *  ...dann darf man den gewünschten Betrag abheben, sonst nicht.
22     */
23    public void abheben(double betrag) {//
24      if (kontostandGeben() + dispokredit >= betrag) {
25        super.abheben(betrag);
26      }
27    }
28
29  }
```

```
3   public class Geschaeftskonto extends Girokonto {
4
5     /**
6      * Für das Geschäftskonto wird der Dispo individuell festgelegt.
7      *
```

```
8       * @param kontonummer Die Kontonummer
9       * @param dispo Der maximale Rahme des Dispokredits.
10      */
11     public Geschaeftskonto(int kontonummer, double dispo) {
12       super(kontonummer);
13       dispokredit = dispo;
14     }
15
16     public void dispokreditSetzen(double dispokreditNeu) {
17       dispokredit = dispokreditNeu;
18     }
19   }
```