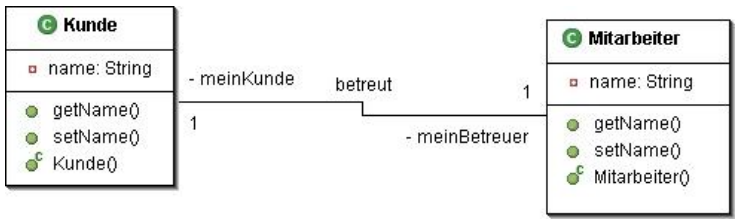
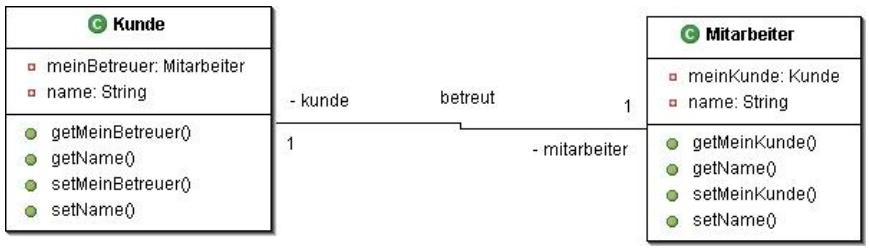
1. 1:1 Assoziation

Zwischen den Klassen Kunde und Mitarbeiter besteht eine 1:1 Assoziation. Ein Kunde wird von einem Mitarbeiter betreut. Ein Mitarbeiter betreut genau einen Kunden.



Diese Assoziation ist in beiden Richtungen navigierbar, d.h. ein Kunde weiß jederzeit über seinen Betreuer bescheid und ein Mitarbeiter kennt zu jedem Zeitpunkt seine betreute Firma. Bei der Implementierung müssen deshalb entsprechende Variablen mit den zugehörigen Methoden in den beiden Klassen vorgesehen werden.



I

```
public class Kunde {

        private String name;

        private Mitarbeiter meinBetreuer;

        public String getName() {return name;}

        public void setName(String name) {this.name = name; }

        public Mitarbeiter getMeinBetreuer() {return meinBetreuer; }

        public void setMeinBetreuer(Mitarbeiter meinBetreuer) {
                this.meinBetreuer = meinBetreuer;
        }
}
```

```
public class Mitarbeiter {

        private String name;

        private Kunde meinKunde;

        public String getName() {return name;}

        public void setName(String name) {this.name = name;}

        public Kunde getMeinKunde() { return meinKunde;      }

        public void setMeinKunde(Kunde meinKunde) {
                this.meinKunde = meinKunde;
        }
}
```

Damit nun die Bedingungen der Assoziation komfortabel umgesetzt werden können, bedient man sich der Konstruktoren.

```
public class Kunde {

        . . .

        public Kunde(Mitarbeiter m) {
                meinBetreuer = m;
                m.setMeinKunde(this);
        }

        . . .
}

public class Mitarbeiter {

        . . .

        public Mitarbeiter(Kunde k) {
                meinKunde = k;
                k.setMeinBetreuer(this);
        }
        . . .
}
```

So wird gewährleistet, dass beim Erzeugen eines Kundenobjektes – wenn das Mitarbeiterobjekt bereits existiert – die entsprechende Zuordnung gleichzeitig mit vorgenommen wird. Falls ein Kunde auch ohne Mitarbeiterzuordnung erzeugt werden soll, wrird noch ein Standard-Konstruktor benötigt. Sinngemäß gelten diese Aussagen analog für die Rückrichtung.

Die Klasse Verwaltung zeigt die Anwendung dieser Mechanismen:

```
public class Verwaltung {

        public static void main(String[] args) {
                Mitarbeiter m1 = new Mitarbeiter();
                m1.setName("Anders");

                Kunde k1 = new Kunde(m1);
                k1.setName("Schulz");

                System.out.print("Herr " + k1.getName() + " wird betreut von ");
                System.out.println("Herrn " + k1.getMeinBetreuer().getName());

                System.out.println();

                System.out.print("Herr " + m1.getName() + " betreut ");
                System.out.println("Herrn " + m1.getMeinKunde().getName());
        }
}
```

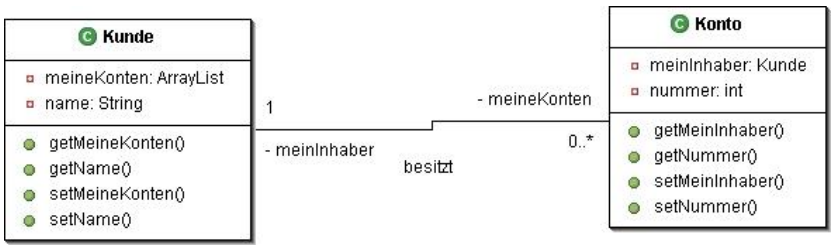
Nach der Erzeugung des Mitarbeiterobjektes, zunächst noch ohne Kundenzuordnung, wird das Kundenobjekt erzeugt. Dem Konstruktor wird der Mitarbeiter als Parameter übergeben. Der Konstruktor seinerseits erzeugt das Kundenobjekt und etabliert die 1:1 Assoziation zwischen dem Kundenobjekt und dem Mitarbeiterobjekt.

Bildschirmausgabe:

```
Herr Schulz wird betreut von Herrn Anders

Herr Anders betreut Herrn Schulz
```

2. 1 : n Assoziation
Am Beispiel der Kunde-Konto-Beziehung wird die Umsetzung einer 1:n Assoziation aufgezeigt. Ein Kunde einer Bank kann mehrere Konten besitzen. Jedes Konto ist genau einem Kontoinhaber zugeordnet. Diese Assoziation ist ebenfalls in beiden Richtungen navigierbar. Ein Kunde hat alle seine Konten in einem Container (ArrayList) gespeichert. Ein Konto besitzt das Attribut meinInhaber vom Typ Kunde zur Speicherung des Besitzers.




```
public class Konto {

        private int nummer;

        private Kunde meinInhaber;

        public int getNummer() {return nummer;}

        public void setNummer(int nummer) {this.nummer = nummer;}

        public Kunde getMeinInhaber() {return meinInhaber;}

        public void setMeinInhaber(Kunde meinInhaber) {
                this.meinInhaber = meinInhaber;
        }

        public Konto(Kunde k) {
                meinInhaber = k;
                ArrayList meinContainer = meinInhaber.getMeineKonten();
                meinContainer.add(this);
        }
}
```

```
public class Kunde {

        private String name;

        private ArrayList meineKonten;

        public String getName() {return name;}

        public void setName(String name) {this.name = name;}

        public ArrayList getMeineKonten() {return meineKonten;}

        public void setMeineKonten(ArrayList meineKonten) {
                this.meineKonten = meineKonten;
        }

        public Kunde() {meineKonten = new ArrayList();      }

}
```

Beim Anlegen eines neuen Kundenobjektes wird zunächst nur der Container für die Konten (ArrayList) initialisiert. Wird anschließend ein Kontoobjekt angelegt, so erhält der Konstruktor den Kunden als Parameter und kann so, neben dem Erzeugen des Kontenobjektes die 1 : n Assoziation zum Kundenobjekt herstellen, indem der Kontencontainer des Kunden geholt und das eben erzeugte Kontenobjekt (this) angehängt (add) wird. Zusätzlich wird natürlich auch der Kontenbesitzer gesetzt.

Die Bankklasse zeigt die Anwendung dieser Beziehung:

```
import java.util.ArrayList;
import java.util.Iterator;

public class Bank {

	public static void main(String[] args) {
		Kunde meinKunde = new Kunde();
		meinKunde.setName("Ehrlich");

		Konto k1 = new Konto(meinKunde);
		k1.setNummer(12345);

		Konto k2 = new Konto(meinKunde);
		k2.setNummer(34567);

		Konto k3 = new Konto(meinKunde);
		k3.setNummer(56789);

		System.out.println("Die Konten von Herrn " + meinKunde.getName() + ":");
		System.out.println("");
		ArrayList meinContainer = meinKunde.getMeineKonten();
		Iterator it = meinContainer.iterator();
		while (it.hasNext()) {
			int  ktoNr = ((Konto)it.next()).getNummer();
			System.out.println(ktoNr);
		}
	}
}
```

Bildschirmausgabe:

```
Die Konten von Herrn Ehrlich:

12345
34567
56789
```

Nachdem das Kundenobjekt und drei Kontenobjekte angelegt sind, werden die Kontenobjekte des Kunden mit Hilfe eines Iterators ausgegeben. Dazu wird der Iterator für den Container *meinContainer* angelegt. Mit der Methode *hasNext()* lässt sich prüfen, ob der Iterator weitere Objekte aus dem Container nehmen kann . Mit der Methode *next()* nimmt der Iterator ein Objekt aus dem Container. Da ein Container verschiedene Objekte enthalten kann, wird ein *Cast* durchgeführt, um ein Kontoobjekt zu erhalten. Um alle Objekte aus dem Container auszulesen, eignet sich eine while-Schleife. Der Zugriff auf die Kontonummer eines ausgelesenen Objektes lässt sich zum besseren Verständnis auch in zwei Zeilen schreiben:

```
Konto einKonto = (Konto)it.next();
int  ktoNr = einKonto.getNummer();
```

Aufgaben:

1. Eine Person besitzt einen Namen und kann beliebig viele Termine verwalten. Jeder Termin enthält aus eine Beschreibung und ist genau einer Person als verantwortliche Person zugeordnet. Jede Person soll einen Überblick über die verantworteten Termine besitzen. Jeder Termin enthält eine Person als Verantwortlichen. Entwerfen Sie ein Klassendiagramm mit einer Assoziation und Kardinalitäten und erstellen Sie eine lauffähige "Minianwendung" die genau diese Assoziation verwaltet. Weitere Attribute lassen Sie unberücksichtigt. Erstellen Sie eine Anwendung zum Testen der Funktionalitäten.

2. Zu einem Termin können mehrere Personen als Teilnehmer eingeladen werden. Jede Person soll laufend einen Überblick über die wahrzunehmenden Termine besitzen. Erweitern Sie das Klassendiagramm aus 1. und setzen Sie die neue Assoziation entsprechend um.