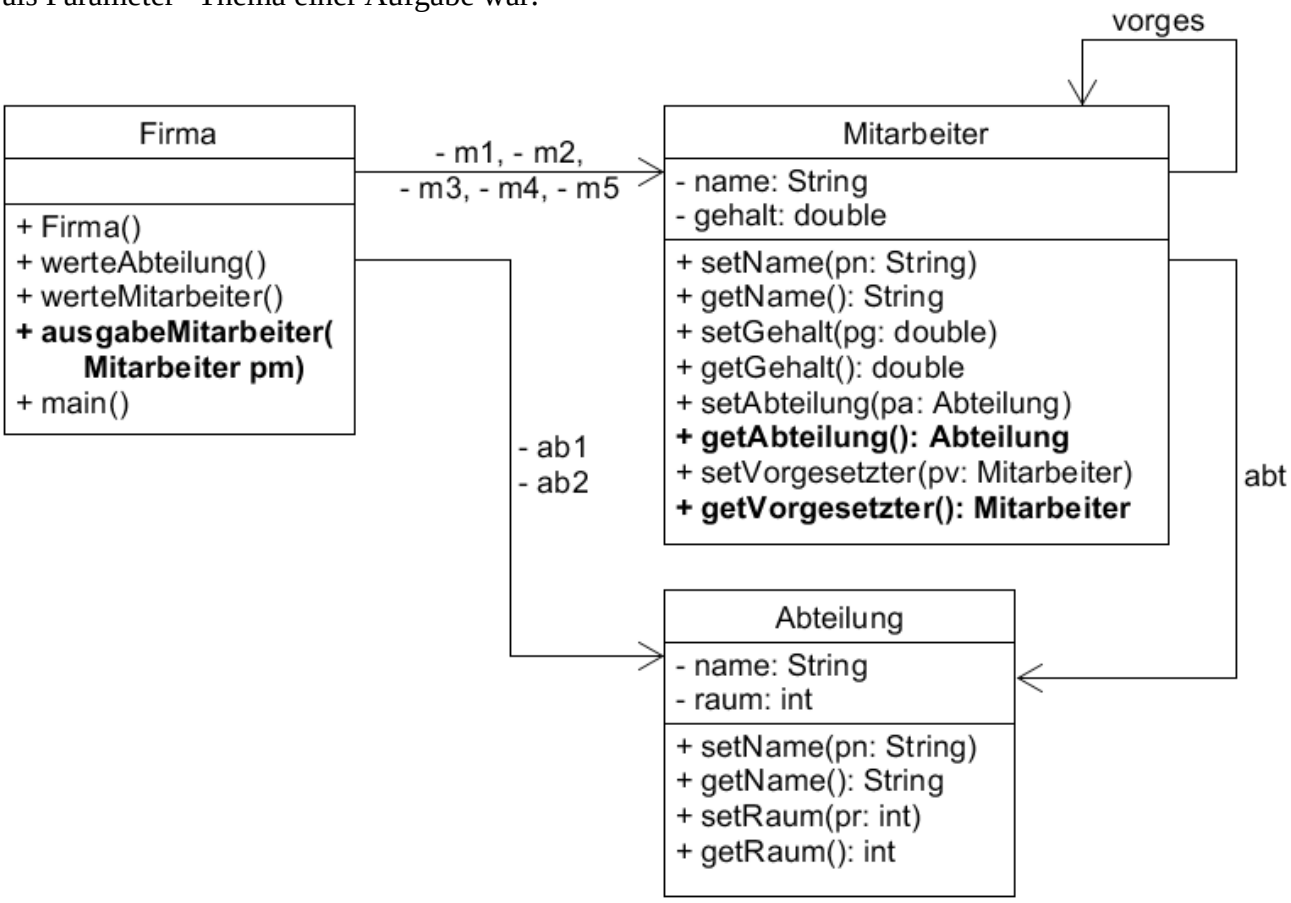
Aufgabe 1

Betrachte das Klassendiagramm zur Mitarbeiterverwaltung, die bereits im Abschnitt zu „Referenzen als Parameter“ Thema einer Aufgabe war:



Zur Erinnerung, die Mitarbeiter arbeiten in verschiedenen Abteilungen. Für jeden Mitarbeiter werden Name und Gehalt, für jede Abteilung Bezeichnung und Raumnummer festgehalten. Jedem Mitarbeiter kann eine Abteilung, sowie ein anderer Mitarbeiter als Vorgesetzter zugeordnet werden. Für alle Attribute soll es jetzt auch get-Methoden geben.

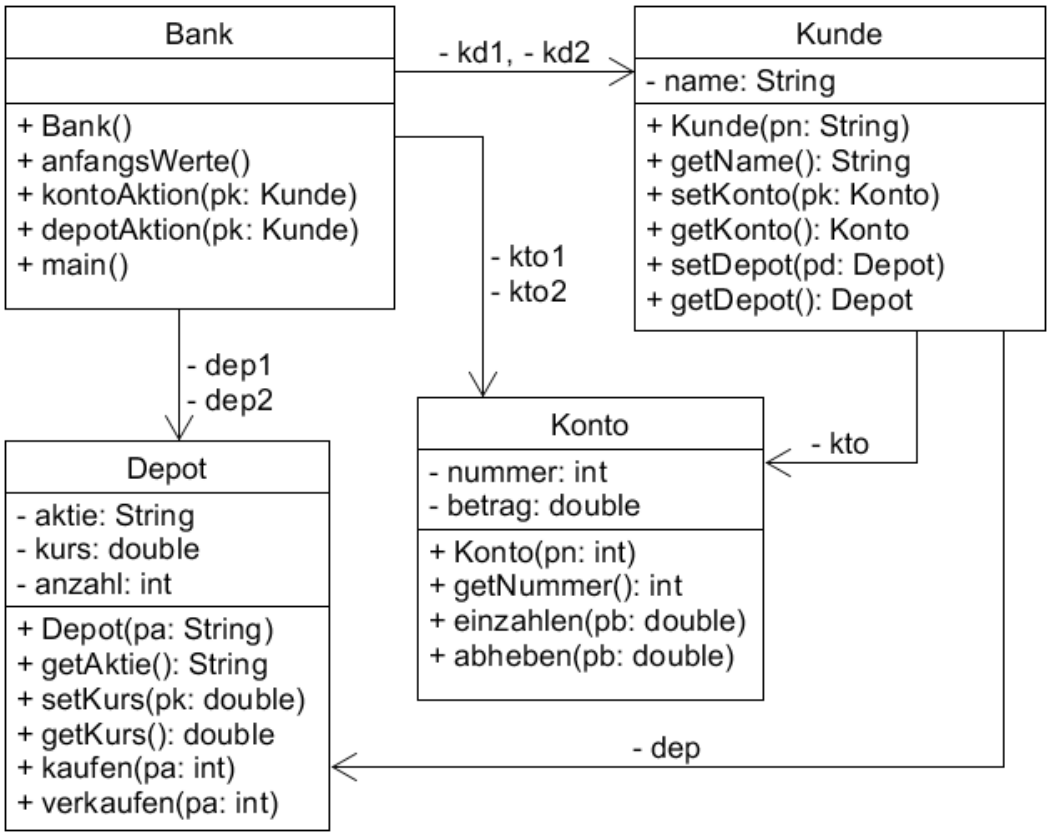
a) Nutze die ausgeteilte BlueJ-Vorlage. Füge der Klasse **Mitarbeiter** get-Methoden für alle Attribute (insbesondere auch Abteilung und Vorgesetzten) hinzu.

b) Für die Klasse **Firma** implementiere die Methode ausgabeMitarbeiter():
Sie erhält einen Mitarbeiter als Parameter und gibt dessen Name und Gehalt aus.
Sie gibt außerdem den Namen des Vorgesetzten und die Bezeichnung der Abteilung aus.
Falls der Mitarbeiter keinen Vorgesetzten bzw. keine Abteilung hat, wird jeweils eine entsprechend Meldung ausgegeben.

c) Erweitere die **main-Methode**, so dass für alle Mitarbeiter die Methode ausgabeMitarbeiter() aufgerufen wird.

Aufgabe 2

Im folgenden Klassendiagramm ist die Software für eine Bank dargestellt.
Für jeden Kunden legt die Software ein Objekt an und kann diesem Kunden jeweils ein Konto und ein Aktiendepot zuordnen.
Jedes Konto hat eine Nummer und einen Betrag (der in diesem Beispiel nicht negativ sein darf).
Ein Depot gehört zu einer bestimmten Aktie (z.B. „VW“ oder „Bosch“). Die Aktie hat einen Kurs, also ein Geldbetrag, zu dem man einen Anteil der Aktie kaufen bzw. verkaufen kann. Wenn der Kunde Anteile kauft, erhöht sich entsprechend die Anzahl, beim Verkauf verringert sie sich.



Nutze die ausgeteilte BlueJ-Vorlage. Die Klassen Konto und Depot sind bereits implementiert.

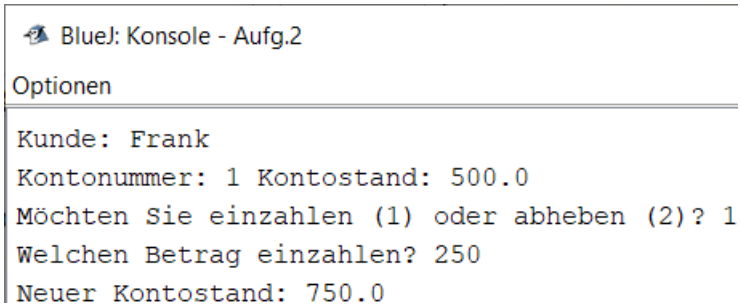
a) Implementiere die Klasse **Kunde** mit allen im Diagramm dargestellten Methoden.
Der Konstruktor setzt den Namen des Kunden.

b) Implementiere die Klasse **Bank**: Gib zunächst die Deklarationen der Referenzen an und implementiere den Konstruktor, der die Kunden-, Konto- und Depot-Objekte erzeugt.
Hinweis: der Depot-Konstruktor setzt den Namen der Aktie, der Konto-Konstruktor die Kontonummer. Namen für Kunden und Aktien sowie Kontonummern denke dir aus.

c) Implementiere die Methode **anfangsWerte()**. Diese ordnet jedem Kunden ein Konto und ein Depot zu. Außerdem setzt sie beispielmäßig Kurse für die Aktien und zahlt auf die Konten einen Anfangsbetrag, z.B. 500 € ein.

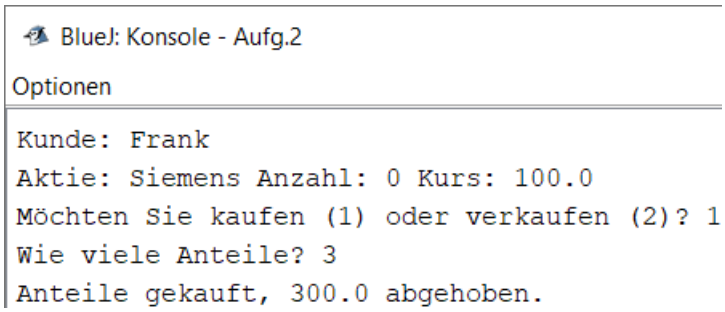
d) Implementiere die Methode **kontoAktion()**. Sie erhält eine Kunden-Referenz als Parameter. Sie gibt zunächst den Namen des Kunden und seine Kontodaten (Nummer und Stand) aus. Der Kunde soll die Wahl zwischen Ein- und Auszahlung haben und kann dann entsprechend einen Betrag eingeben, der auf das Konto eingezahlt bzw. davon abgehoben wird. Zum Schluss wird der neue Kontostand ausgegeben.

Die Methode muss folgende Sonderfälle behandeln:
Wenn die Kunden-Referenz null ist, bricht die Methode einfach ab.
Wenn der Kunde kein Konto hat, wird eine entsprechende Meldung ausgegeben.
Ebenso, wenn der einzuzahlende Betrag negativ ist oder er mehr abheben möchte als möglich.



e) Implementiere die Methode **depotAktion()**. Sie erhält eine Kunden-Referenz als Parameter. Sie gibt zunächst wieder den Kundennamen und die Daten seines Depots aus – Name der Aktie, Kurs und Anzahl der bereits gekauften Anteile.
Der Kunde hat die Wahl, Aktien zu kaufen oder zu verkaufen. Entsprechend erhöht bzw. verringert sich die Anzahl der Anteile. Außerdem wird beim Kauf der entsprechende Geldbetrag vom Konto abgehoben, beim Verkauf eingezahlt.

Ähnlich wie in Aufgabe d) muss die Methode die möglichen Sonderfälle behandeln.



f) Implementiere die **main-Methode**. Sie ruft die Methode anfangsWerte() auf und dann für die beiden Kunden je einmal eine Konto-Aktion und eine Depot-Aktion.