Objektorientierte Modellierung
Implementierung einer Banksoftware

Hinweis: Auf der Rückseite seht ihr auf der rechten Seite das *Implementierungsklassendiagramm* zum *Entwurfsklassendiagramm* links.

Aufgabe 1

Analysiert die Diagramme und vergleicht sie mit dem Entwurf aus dem Unterricht.

Aufgabe 2

Implementiert die Klasse `Kunde` entsprechend des *Implementierungsdiagramms*. Geht dazu so vor:

- Öffnet BlueJ und erstell ein neues Projekt. Speichert das Pojekt in eurem Laufwerk (`N:`).
- Erstellt die Klasse `Kunde` als leere Klasse, indem ihr auf Neue Klasse klickt und unten „Leere Klasse“ auswählt.
- Deklariert die Objektvariablen `name`, `geburtstag`, `adresse` und `konto` in der Klasse.
- Implementiert den *Konstruktor* der Klasse, der die Attribute initialisiert.
- Implementiert die *Getter* und *Setter*.

Aufgabe 3

Implementiert die Klasse `Konto` entsprechend des *Implementierungsdiagramms*. Geht dazu so vor:

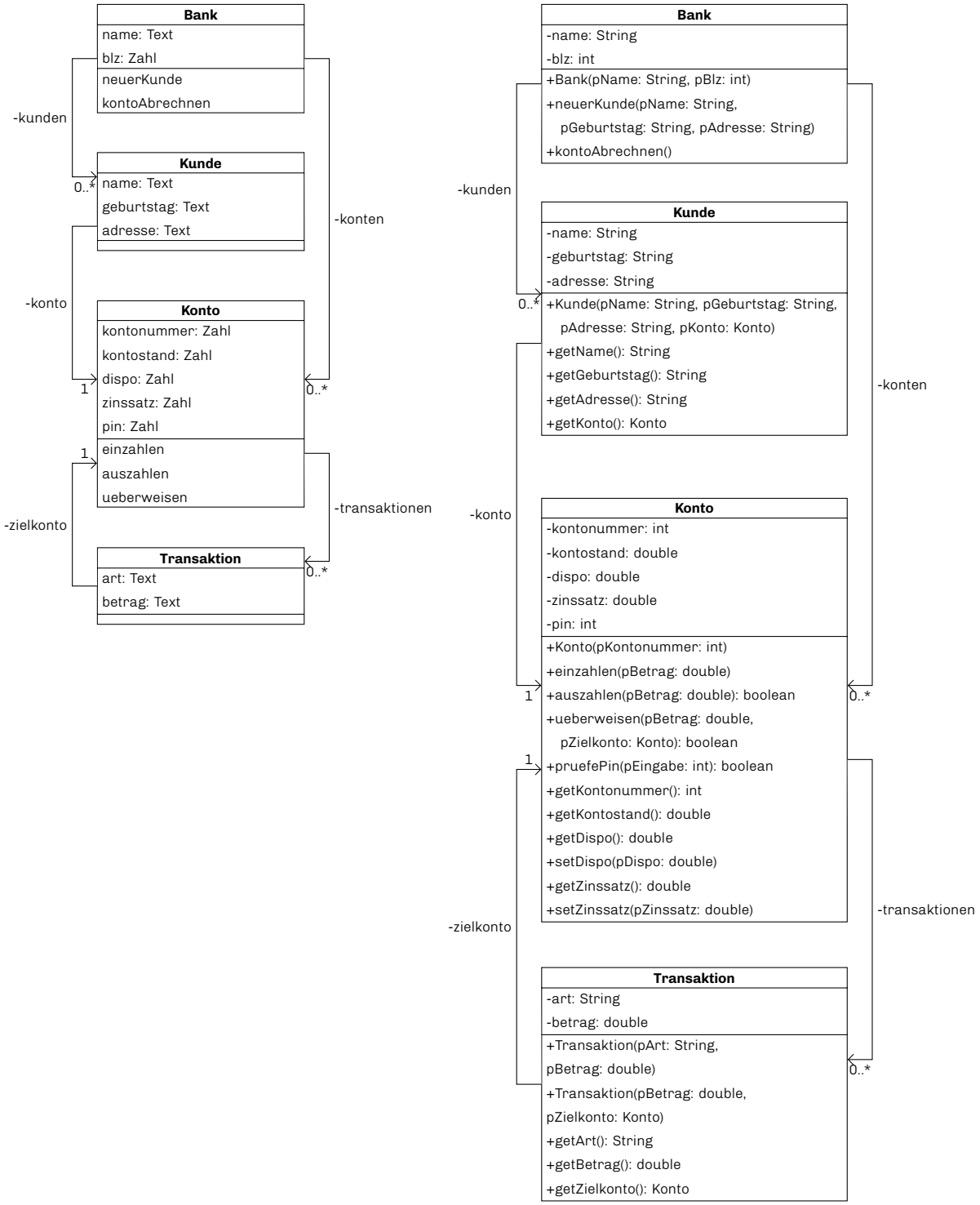
- Erstellt die Klasse `Konto` als leere Klasse wie oben.
- Deklariert die Objektvariablen der Klasse (mit Ausnahme der Transaktionen).
- Implementiert den *Konstruktor* der Klasse, der die Attribute initialisiert. Dabei sollen `kontostand`, `dispo` und `zinssatz` zunächst null sein und die vierstellige `pin` soll zufällig generiert werden. (Siehe `Random`.)
- Implementiert die *Getter* und *Setter*.
- Implementiert die Methode `einzahlen` und `auszahlen`. `auszahlen` soll `true` zurück gegeben werden, wenn die Auszahlung (unter Berücksichtigung des Dispos) erfolgreich war.
- Implementiert die Methode `ueberweisen`. Sie bekommt das Zielkonto als Parameter und ruft die Methode `einzahlen` des Zielkontos auf.

Testet die Klasse ausgiebig.

Aufgabe 4

Implementiert die Klasse `Transaktion` entspechend des *Implementierungsdiagramms*. Geht dazu ungefähr so vor, wie bei den anderen Klassen.

Modifiziert die Klasse `Konto` dann so, dass bei jeder Einzahlung, Auszahlung und Überweisung ein neues Transaktion-Objekt mit den passenden Informationen erstellt wird. Das neue Objekt soll in einer Objektvariablen „letzteTransaktion“ gespeichert werden. Ergänzt auch einen entsprechenden *Getter*.


Bank
name: Text
blz: Zahl
neuerKunde
kontoAbrechnen
-kunden
0..*
Kunde
name: Text
geburtstag: Text
adresse: Text
-konten
-konto
1
0..*
Konto
kontonummer: Zahl
kontostand: Zahl
dispo: Zahl
zinssatz: Zahl
pin: Zahl
einzahlen
auszahlen
ueberweisen
1
-transaktionen
-zielkonto
Transaktion
art: Text
betrag: Text
0..*
Bank
-name: String
-blz: int
+Bank(pName: String, pBlz: int)
+neuerKunde(pName: String,
pGeburtstag: String, pAdresse: String)
+kontoAbrechnen()
-kunden
Kunde
-name: String
-geburtstag: String
-adresse: String
0..*
+Kunde(pName: String, pGeburtstag: String,
pAdresse: String, pKonto: Konto)
+getName(): String
+getGeburtstag(): String
+getAdresse(): String
+getKonto(): Konto
-konten
-konto
Konto
-kontonummer: int
-kontostand: double
-dispo: double
-zinssatz: double
-pin: int
+Konto(pKontonummer: int)
+einzahlen(pBetrag: double)
1
+auszahlen(pBetrag: double): boolean
0..*
+ueberweisen(pBetrag: double,
pZielkonto: Konto): boolean
1
+pruefePin(pEingabe: int): boolean
+getKontonummer(): int
+getKontostand(): double
+getDispo(): double
+setDispo(pDispo: double)
+getZinssatz(): double
+setZinssatz(pZinssatz: double)
-transaktionen
-zielkonto
Transaktion
-art: String
-betrag: double
+Transaktion(pArt: String,
pBetrag: double)
+Transaktion(pBetrag: double,
pZielkonto: Konto)
+getArt(): String
+getBetrag(): double
+getZielkonto(): Konto
0..*