Klasse Bank entwerfen

- Mögliche Attribute:
  - ⇨ Name der Bank
  - ⇨ Konten (als Array mit vorgegebener Größe)
- Mögliche Methoden:
  - ⇨ Anlegen eines Kontos
  - ⇨ Löschen eines Kontos
  - ⇨ Einzahlen auf ein Konto
  - ⇨ Abheben von einem Konto
  - ⇨ Überweisen von einem Konto auf ein anderes

Klasse Bank: Attribute

```
public class Bank {
    private String name; // Name der Bank
    private Konto[] kontoTab;
    private int anzKonten;
...
```

Klasse Bank: Konstruktor

```
/**
 * Konstruktor fuer Objekte der Klasse Bank
 *
 * @param name Name der Bank (vorlaeufig ohne
 *             Vorbedingung)
 * @param maxAnzKonten  maximale Anzahl an
 *                      anzulegenden Konten
 */
public Bank(String name, int maxAnzKonten) {
    this.name = name;
    kontoTab = new Konto[maxAnzKonten];
    anzKonten = 0;
}
```


Anlegen Konto

```
/**
 * Anlegen eines Kontos.
 *
 * @param  kontonr  Kontonr des anzulegenden Kontos
 * @param  inhaber  Name des Kontoinhabers
 */
public void legeKontoAn(int kontonr, String inhaber) {
...
}
```

Anlegen Konto

```
/**
 * Anlegen eines Kontos. (noch ohne Pruefung)
 *
 * @param  kontonr  Kontonr des anzulegenden Kontos
 * @param  inhaber  Name des Kontoinhabers
 */
public void legeKontoAn(int kontonr, String inhaber) {
    kontoTab[anzKonten] = new Konto(kontonr, inhaber);
    anzKonten++;
}
```


Einzahlen auf ein Konto

```
/**
 * Einzahlen auf ein Konto
 *
 * @param  kontonr   zu uebergebende Kontonummer
 * @param  betrag    einzuzahlender Betrag (> 0)
 */
public void einzahlen(int kontonr, double betrag) {
// Konto mit dieser kontonr im Array suchen
// auf dieses Konto einzahlen
}
```

Zusätzliche Methode findeKonto

```
/**
 * Interne Methode findeKonto zur Vereinfachung anderer
 * Methoden
 *
 * @param  kontonr   zu uebergebende Kontonummer
 * @return  Index des gesuchten Kontos oder -1
 *
 */
private int findeKonto(int kontonr) {
    for(int i = 0; i < anzKonten; i++) {
        if(kontoTab[i].getKontonr() == kontonr)
            return i;
    }
    return -1;
}
```


Einzahlen auf ein Konto

```
/**
 * Einzahlen auf ein Konto
 *
 * @param  kontonr   zu uebergebende Kontonummer
 * @param  betrag    einzuzahlender Betrag (> 0)
 */
public void einzahlen(int kontonr, double betrag) {
    int i = findeKonto(kontonr);
    if(i >= 0)
        kontoTab[i].einzahlen(betrag);
    else
        throw new
              RuntimeException(MSG_KONTO_NICHT_VORHANDEN);
}
```

Abheben von einem Konto

```
/**
 * Abheben von einem Konto
 *
 * @param  kontonr   zu uebergebende Kontonummer
 * @param  betrag    einzuzahlender Betrag (> 0)
 */
public void abheben(int kontonr, double betrag) {
    int i = findeKonto(kontonr);
    if(i >= 0)
        kontoTab[i].abheben(betrag);
    else
        throw new
              RuntimeException(MSG_KONTO_NICHT_VORHANDEN);
}
```


Konto löschen (1. Variante)

```
/**
 * Konto loeschen, wenn die Kontonr nicht existiert
 * Loesung mit Shiften der Konto-Objekte
 * @param  kontonr   zu uebergebende Kontonummer
 *
 */
public void loescheKonto(int kontonr) {
    int i = findeKonto(kontonr);

    if (i >= 0) {
        for (int j = i; j < anzKonten-1; j++)
            kontoTab[j] = kontoTab[j+1];
        kontoTab[anzKonten-1] = null;
        anzKonten--;
     }
}
```

Konto löschen (2. Variante)

```
/**
 * Konto loeschen, wenn die Kontonr nicht existiert
 * Vertauschen mit dem letzten Eintrag
 * @param  kontonr   zu uebergebende Kontonummer
 *
 */
public void loescheKonto(int kontonr) {
    int i = findeKonto(kontonr);

    if (i >= 0) {
        kontoTab[i] = kontoTab[anzKonten-1];
        kontoTab[anzKonten-1] = null;
        anzKonten--;
    }
}
```


toString-Methode

```
/**
 * Bank-Objekt als Zeichenkette aufbereiten;
 * verwendet implizit die toString-Methode von
 * Konto
 * Erste noch ineffiziente Variante
 *
 * @return  Zeichenkette
 */
public String toString() {
    String s = "Bank: " + name + '\n';
    for(int i = 0; i < anzKonten; i++) {
        s += i + ": " + kontoTab[i] + '\n';
    }
    return s;
}
```

Klasse Bank weiterentwickeln

Noch zu erledigen:

- In der Methode `legeKontoAn` muss geprüft werden, ob bereits ein Konto mit der übergebenen Nummer existiert.
- Die Methode `ueberweisen` fehlt noch
- Die Fehlermeldungen sollten generell in Klassenkonstanten stehen.

Fehlermeldungen

```
// Fehlermeldungen
   private static final String MSG_KONTO_NICHT_VORHANDEN =
                        "Konto nicht vorhanden!";
   private static final String MSG_KONTO_ANLEGEN =
                        "Konto konnte nicht angelegt werden!";
   private static final String MSG_KONTO_VORHANDEN =
                        "Konto bereits vorhanden!";
   private static final String MSG_KONTO_VERSCHIEDEN =
                        "Die Konten muessen verschieden sein!";
   private static final String MSG_MAX_ANZ_KONTEN =
                        "Die Zahl der Konten muss > 0 sein!";
```

Anlegen Konto (bisher)

```
/**
     * Anlegen eines Kontos. (noch ohne Pruefung)
     *
     * @param  kontonr  Kontonr des anzulegenden Kontos
     * @param  inhaber  Name des Kontoinhabers
     */
    public void legeKontoAn(int kontonr, String inhaber) {
        kontoTab[anzKonten] = new Konto(kontonr, inhaber);
        anzKonten++;
    }
```


Anlegen Konto (neu)

```
    /**
     * Anlegen eines Kontos. Es ist zu prüfen, ob die
     * KontoNr bereits vergeben wurde und ob die
     * Kontotabelle bereits voll ist.
     *
     * @param  kontonr  Kontonr des anzulegenden Kontos
     * @param  inhaber  Name des Kontoinhabers
     */
    public void legeKontoAn(int kontonr, String inhaber) {
        if (findeKonto(kontonr) >= 0)
           throw new IllegalArgumentException
                                          (MSG_KONTO_VORHANDEN);
        if (anzKonten >= kontoTab.length)
           throw new IllegalArgumenException(MSG_ANLEGEN);

        kontoTab[anzKonten] = new Konto(kontonr, inhaber);
        anzKonten++;
    }
```

Anlegen Konto (besser)

```
/**
    * Anlegen eines Kontos. Es ist zu pruefen, ob die
    * KontoNr bereits vergeben wurde und ob die
    * Kontotabelle bereits voll ist.
    *
    * @param  kontonr  Kontonr des anzulegenden Kontos
    * @param  inhaber  Name des Kontoinhabers
    */
   public void legeKontoAn(int kontonr, String inhaber) {
       check(findeKonto(kontonr) < 0, MSG_KONTO_VORHANDEN);
       check(anzKonten < kontoTab.length, MSG_KONTO_ANLEGEN);

       kontoTab[anzKonten] = new Konto(kontonr, inhaber);
       anzKonten++;
   }
```


Check-Methode

```
private static void check(boolean bedingung,
                          String msg) {
   if (!bedingung)
       throw new IllegalArgumentException(msg);
}
```

Überweisen

```
/**
 * Ueberweisen von einem Konto zum anderen
 * Vorbedingungen:
 * - beide Konten muessen vorhanden sein
 * - die Kontonummern muessen verschieden sein
 *
 * @param  vonKontonr   Kontonummer des Kontos, von dem
 *                      ueberwiesen werden soll
 * @param  nachKontonr  Kontonummer des Kontos, auf das
 *                      ueberwiesen werden soll
 * @param  betrag       zu ueberweisender Betrag (> 0)
 */
public void ueberweisen(int vonKontonr,
                        int nachKontonr, double betrag) {
...
}
```


Überweisen

```
public void ueberweisen(int vonKontonr,
                        int nachKontonr,
                        double betrag) {
    check(vonKontonr != nachKontonr,
                                MSG_KONTO_VERSCHIEDEN);
    int i1 = findeKonto(vonKontonr);
    int i2 = findeKonto(nachKontonr);
    check(i1 >= 0 && i2 >= 0,
                              MSG_KONTO_NICHT_VORHANDEN);
    kontoTab[i1].abheben(betrag);
    kontoTab[i2].einzahlen(betrag);
}
```

Anwendung auf Methode einzahlen

```
public void einzahlen(int kontonr, double betrag) {
    int i = findeKonto(kontonr);
    if(i >= 0)
        kontoTab[i].einzahlen(betrag);
    else
        throw new RuntimeException(
                            MSG_KONTO_NICHT_VORHANDEN);
}
```

besser:

```
public void einzahlen(int kontonr, double betrag) {
    int i = findeKonto(kontonr);
    check(i >= 0, MSG_KONTO_NICHT_VORHANDEN);
    kontoTab[i].einzahlen(betrag);
}
```


Interaktive Testklasse für Bank

Forderungen:

- In einer Schleife soll die Schnittstelle der Klasse interaktiv aufgerufen werden können
- Nach jedem Schleifendurchlauf sollen die Inhalte des Bank-Objektes ausgegeben werden
- Aufbau analog zur KontoDialog-Klasse

BankDialog: Grobstruktur

```
public class BankDialog {
    // Attribute
    private Bank bank1;

    /**
     * Hauptschleife des Testprogramms
     */
    public void start() { ... }

    /**
     * Main-Methode zum Erzeugen des BankDialog-Objektes
     * und zum Anstarten der Testschleife
     */
    public static void main (String[] args) {
        new BankDialog().start();
    }
```


Bank-Dialog: Hauptschleife

- Struktogramm für die Hauptsteuerung

Bank-Objekt anlegen	
Solange Funktion != Ende	
	Funktion einlesen
	Funktion ausführen

Bank-Dialog: Hauptschleife

```
public void start() {
    bank1 = new Bank("Java-Bank", 10);
    int funktion = -1;

    while (funktion != ENDE) {
        try {
            funktion = einlesenFunktion();
            ausfuehrenFunktion(funktion);
        } catch (IllegalArgumentException e) {
            System.out.println("Ausnahme gefangen: " + e);
        } catch (java.util.InputMismatchException e) {
            System.out.println(e);
            input.next();
        } catch (Exception e) {
            System.out.println("Ausnahme gefangen: " + e);
            e.printStackTrace(System.out);
        }
     }
}
```


Methode einlesenFunktion()

```
// Klassenkonstanten
    private static final int ANLEGEN     = 1;
    private static final int EINZAHLEN   = 2;
    private static final int ABHEBEN     = 3;
    private static final int LOESCHEN    = 4;
    private static final int UEBERWEISEN = 5;
    private static final int ENDE        = 0;

    private int einlesenFunktion() {
        System.out.print(ANLEGEN     + ": anlegen; " +
                         EINZAHLEN   + ": einzahlen; " +
                         ABHEBEN     + ": abheben; " +
                         LOESCHEN    + ": löschen; " +
                         UEBERWEISEN + ": überweisen; " +
                         ENDE        + ": beenden -> ");

        return input.nextInt();
    }
```

Methode ausfuehrenFunktion()

```
// grobe Struktur:
private void ausfuehrenFunktion(int funktion) {
    switch (funktion) {
    case ANLEGEN:
         ...
    case ABHEBEN:
         ...
    case EINZAHLEN:
         ...
    case LOESCHEN:
         ...
    case UEBERWEISEN:
         ...
    case ENDE:
         ...
    }
}
```


Methode ausfuehrenFunktion()

```
private void ausfuehrenFunktion(int funktion) {
   int kontonr, nachKontonr;
   double betrag;

      switch (funktion) {
       case ANLEGEN:
             bank1.legeKontoAn(einlesenKontonr(),
                               einlesenInhaber());
             break;

        case ABHEBEN:
        ...
```

Hilfsmethoden

```
private int einlesenKontonr() {
    System.out.print("Kontonummer: ");
    return input.nextInt();
}

private double einlesenBetrag() {
    System.out.print("Betrag: ");
    return input.nextDouble();
}

private String einlesenInhaber() {
    System.out.print("Inhaber    : ");
    input.nextLine();
    return input.nextLine();
}
```