1. Aufgabe

Aufgabenstellung

Beschreiben Sie bei den folgenden Java- Methoden, um was für einen Typ es sich handelt, und welche Parameter die Methoden verwendet.

```
public class Methoden {
      private static int objektAnzahl;
      private int objektNr;
      public Methoden(){objektAnzahl++; objektNr=objektAnzahl;}
      public void drucke() {System.out.println(this.getClass().getName());}
      public boolean letztesObjekt () {boolean status =
                                    objektAnzahl==objektNr; return status;}
      public int zaehleZeichen(String einString) {return
                                                  einString.length();}
      public static int getObjektanzahl() {return objektAnzahl;}
      public String verkette(String einString, String zweiString) {return
                                                  einString + zweiString;}
}
```

Zeigen Sie in welchem Kontext (statisch, nicht statisch) diesen Methoden wie aufgerufen werden können.

2. Aufgabe

Erklären Sie den Unterschied zwischen aktuellen und formalen Parametern. Welchen Sinn macht diese Unterscheidung?

3. Aufgabe - Parameterübergabe

Betrachten Sie die folgende Java-Methode:

```
public void tausche(int a, int b)
{
   int x;
   x = a;
   a = b;
   b = x;
}
```

Funktioniert diese Implementierung? Begründen Sie Ihre Antwort. Beweisen Sie diese durch ein Testprogramm.

4. Aufgabe – Prozeduren und Funktionen

Finden Sie zu jedem Aufgabenpunkt ein geeignetes Beispiel und erstellen Sie (falls möglich) die Funktion/Prozedur in einer Klasse. Testen Sie die Methoden dieser Klasse.

- Funktion mit zwei Eingabeparametern.
  Konzept eines Beispiels: *ergebnis = addiere (eingabe1, eingabe2);*
- Prozedur mit je einem Eingabeparameter für die unterschiedlichen Parameterübergabemechanismen – *by value, by reference*. Der *by reference* Eingabeparameter, kann auch zur Ausgabe des Ergebnisses verwendet werden.
  Konzept eines Beispiels: *void addiere(eingabe, einAusgabe);*
- Prozedur mit zwei Ein/Ausgabeparametern.
  Konzept eines Beispiels: *tausche(einAus1, einAus2);*

Hinweis: Sie benötigen eventuell weitere (Hilfs-)Klassen.

5. Aufgabe – Methoden, Parameterübergabe

Das Programm Zählerverwaltung (Projekt Zähler – siehe Vorlesung) soll um einen Gesamtkilometerzähler erweitert werden. Anforderungen:

1.	Der Gesamtkilometer-Zähler soll ebenfalls digital angezeigt werden.
2.	Im Gegensatz zum Tageskilometer-Zähler kann der Gesamtkilometer-Zähler nur einmal auf Null gesetzt werden (Voreinstellung).
3.	Wird der Zählerstand erhöht, dann ist er bei beiden Zählern zu erhöhen.
4.	Der Stand beider Zähler soll zusammen angezeigt werden können.

Erweitern Sie das bisherige Java-Programm entsprechend der neuen Funktionalität. Ändern Sie die Fachkonzept-Klasse *Zähler* nicht! Sie benötigen allerdings zwei Objekte dieser Klasse: Tageskilometerzähler und Gesamtkilometerzähler.

Möglicher Ein-/Ausgabe-Dialog des Programms:

```
anzeigen(1), reset Tg-Zähler(2), erhöhe alle Zähler(3), ende(0) --> 1
>> Tageszähler: 0, Gesamtzähler: 0
anzeigen(1), reset Tg-Zähler(2), erhöhe alle Zähler(3), ende(0) --> 3
anzeigen(1), reset Tg-Zähler(2), erhöhe alle Zähler(3), ende(0) --> 3
anzeigen(1), reset Tg-Zähler(2), erhöhe alle Zähler(3), ende(0) --> 1
>> Tageszähler: 2, Gesamtzähler: 2
anzeigen(1), reset Tg-Zähler(2), erhöhe alle Zähler(3), ende(0) --> 2
anzeigen(1), reset Tg-Zähler(2), erhöhe alle Zähler(3), ende(0) --> 1
>> Tageszähler: 0, Gesamtzähler: 2
anzeigen(1), reset Tg-Zähler(2), erhöhe alle Zähler(3), ende(0) --> 3
anzeigen(1), reset Tg-Zähler(2), erhöhe alle Zähler(3), ende(0) --> 1
>> Tageszähler: 1, Gesamtzähler: 3
anzeigen(1), reset Tg-Zähler(2), erhöhe alle Zähler(3), ende(0) -->
```

6. Aufgabe – Methoden, Parameterübergabe

Um die Pegelstände von Flüssen anzeigen zu können, sollen Digitalzähler verwendet werden, die folgende Anforderungen erfüllen:

1.	Die Zähler können Aufwärts und Abwärts gezählt werden.
2.	Alle Zähler sind mit Null zu initialisieren.
3.	Die Zähler können jeweils um ± 1 oder um ± 5 verändert werden.
4.	Es kann immer nur ein Pegel verändert werden.

Modifizieren Sie die bisherige Fachkonzeptklasse *Zähler* entsprechend und gestalten Sie ein geeignetes Konsolprogramm mit zwei Pegelstands-Anzeigern.

Möglicher Ein-/Ausgabe-Dialog des Programms:

```
anzeigen(1), ändere(2), ende(0) --> 1
>> Pegel1: 0, Pegel2: 0
anzeigen(1), ändere(2), ende(0) --> 2
Pegel? (1,2)> 2
Delta(-1,1,-5,5)> 5
anzeigen(1), ändere(2), ende(0) --> 2
Pegel? (1,2)> 1
Delta(-1,1,-5,5)> 5
anzeigen(1), ändere(2), ende(0) --> 2
Pegel? (1,2)> 1
Delta(-1,1,-5,5)> -1
anzeigen(1), ändere(2), ende(0) --> 1
>> Pegel1: 4, Pegel2: 5
anzeigen(1), ändere(2), ende(0) -->
```

7. Aufgabe – Fachkonzeptklasse mit komplexen Kontext

Erstellen Sie eine Fachkonzeptklasse **Tank**, die folgendes leistet:

- Ein Tank besitzt eine Soll-Füllhöhe und eine Ist-Füllhöhe (in Litern).
- Ein Tank kann um eine bestimmte Menge gefüllt werden.
- Wird beim Füllen die Soll-Füllhöhe überschritten, meldet der Tank eine Überschreitung der Soll-Füllhöhe (Rückgabe der Überschussmenge).
- Ein Tank kann um eine bestimmte Menge geleert werden.
- Wird beim Leeren die Höhe 0 unterschritten, meldet der Tank dies (Rückgabe der Fehlmenge).

Schreiben Sie ein entsprechendes Konsolprogramm um einen Tank zu betreiben. Es soll möglich sein, bestimmte Mengen einzufüllen/zu entnehmen und die jeweilige Ist-Füllhöhe anzuzeigen. Unter- / Überschreitungen werden als Statusmeldung ausgegeben. Verwenden Sie eine Sollkapazität von 1000 Liter.

Möglicher Ein-/Ausgabe-Dialog des Programms:

```
anzeigen(a), bedienen(b), ende(e)   -->a
>> Iststand: 0, Kapazität: 1000
anzeigen(a), bedienen(b), ende(e)   -->b
Menge (positiv=füllen, negativ=leeren)> 300
anzeigen(a), bedienen(b), ende(e)   -->a
>> Iststand: 300, Kapazität: 1000
anzeigen(a), bedienen(b), ende(e)   -->b
Menge (positiv=füllen, negativ=leeren)> 800
>> Überschussmenge: 100
anzeigen(a), bedienen(b), ende(e)   -->a
>> Iststand: 1000, Kapazität: 1000
anzeigen(a), bedienen(b), ende(e)   -->b
Menge (positiv=füllen, negativ=leeren)> -400
anzeigen(a), bedienen(b), ende(e)   -->a
>> Iststand: 600, Kapazität: 1000
anzeigen(a), bedienen(b), ende(e)   -->b
Menge (positiv=füllen, negativ=leeren)> -800
>> Fehlmenge: 200
anzeigen(a), bedienen(b), ende(e)   -->a
>> Iststand: 0, Kapazität: 1000
anzeigen(a), bedienen(b), ende(e)   -->
```

8. Aufgabe – überladene Methoden

Schreiben Sie eine Klasse »Werkzeug«, die mehrere Methoden »plus« zur Verfügung stellt. Eine Variante addiert zwei **int**-Zahlen, eine zweite Variante addiert zwei **double**-Zahlen, eine dritte Variante verkettet zwei Zeichenketten.

Testen Sie diese Klasse

Beachten Sie die korrekten Parameterübergabemechanismen und erläutern Sie die verwendeten Übergabetypen.

9. Aufgabe – Beziehung zwischen Klassen

Mit der KundenUndAuftragsverwaltung (siehe Vorlesung) sollen auch die Rechnungen erstellen werden. Folgende Anforderungen müssen dafür zusätzlich erfüllt sein:

- Der aktuelle volle MWST-Satz muss bei der Klasse Auftrag gespeichert werden können.
- Der MWST-Satz muss geschrieben und gelesen werden können.
- Die Auftragsnummer soll (mit 1 beginnend) automatisch gesetzt werden

a. Realisieren Sie die `KundenUndAuftragsverwaltung` als Anwendung mit main-Prozedur. Dazu gehören die zwei eigenständigen Klassen `Kunde` und `Auftrag`.
b. Erweitern Sie das Programmsystem `KundenUndAuftragsverwaltung` so, dass eine Verbindung von einem Auftrags-Objekt zu einem Kunden-Objekt hergestellt werden kann.
c. Erweitern Sie die Klasse `Auftrag` um die obigen Anforderungen.
d. Legen Sie zwei Kunden-Objekte mit jeweils einem Auftrag an und erstellen Sie für beide Aufträge eine Rechnung. Ergänzen Sie dazu die Klasse `Auftrag` um eine Methode `erstelleRechnung`. Holen Sie über die Verbindung zum Kunden den Firmennamen und die Firmenadresse. Übersetzen Sie das Java-Programm und führen Sie es aus.

Hinweise:

- Für die Klasse `Kunde` reichen Attribute für `Name` und `Auftrag` mit den entsprechenden get-/set-Methoden.
- Zur Klasse `Auftrag` gehören u.a. die Attribute `Auftragsnummer`, `Titel` und `Preis` mit den entsprechenden get-/set-Methoden.