Informationen: Die Beispiele sind bis zur Übung am 24.10 vorzubereiten.

Beispiel 3.1:
Gegeben sei eine Klasse *Anlage* mit Methoden *double getPreis()* und *int getKapa()*, die den Preis bzw. die Kapazität der Anlage zurückgeben. Schreiben Sie eine Klasse *AnlagenAngebot* mit Konstruktor

```
AnlagenAngebot(Anlage[] anlagenliste)
```

zur Erzeugung eines entsprechenden Anlagenangebots. Schreiben Sie weiters eine Methode

```
Anlage getBilligsteMitMinKapa(int minKapa),
```

die die günstigste Anlage zurückgibt, die mindestens *minKapa* Kapazität hat. Falls es keine Anlage mit hinreichend großer Kapazität geben sollte, soll die Methode *null* zurückgeben. Gibt es mehrere gleich billige Anlagen, so soll jene mit der höchsten Kapazität zurückgegeben werden.

Beispiel 3.2:
Ergänzen Sie die Klasse *Zeitpunkt* aus Beispiel 1.7 oder 2.1 um eine Methode

```
Zeitpunkt copy(),
```

die eine Kopie des Zeitpunkts zurückgibt, d.h. ein neues Objekt der Klasse *Zeitpunkt*, das in allen Attributwerten mit dem ursprünglichen Objekt übereinstimmt.

Erweitern Sie die Klasse *Zeitpunkt* außerdem um eine Methode

```
void setDatum(int jahr, int monat, int tag),
```

mit der Tag, Monat und Jahr des Zeitpunkts neu gesetzt werden können.

Beispiel 3.3:
Ergänzen Sie die Klasse *Termin* aus Beispiel 1.8 oder 2.2 wie folgt. Die Methode

```
Zeitpunkt getZeitpunkt()
```

sollen den Zeitpunkt des Termins zurückgeben. Die Methoden

```
void setDatum(int jahr, int monat, int tag)
void setZeitpunkt(Zeitpunkt einZeitpunkt)
```

sollen Tag, Monat und Jahr bzw. den Zeitpunkt des Termins neu setzen. Die Methode

```
Termin copy()
```

soll eine Kopie des Termins zurückgeben, d.h. ein neues Objekt der Klasse *Termin*, das in allen Attributwerten mit dem ursprünglichen Objekt übereinstimmt. Weiters soll eine Methode

```
Termin copyDeep()
```

eine Kopie des Termins zurückgeben. Die Ergebnisse von *getZeitpunkt()* für die Kopie sollen eine ***Kopie*** des Zeitpunkts des ursprünglichen Objektes liefern.

Erzeugen Sie nun ein Objekt *termin1* der Klasse *Termin*. Erzeugen Sie zwei weitere Objekte *termin2* und *termin3* der Klasse *Termin* mit Hilfe der Methoden *copy()* und *copyDeep()*. Verändern Sie das Datum von *termin1* mit Hilfe der Methode *setDatum()*. Was passiert mit den Datumsattributen von *termin2* und

termin3? Geben Sie *termin1* durch Aufruf von *setZeitpunkt()* einen neuen Zeitpunkt. Was passiert mit dem Zeitpunkt in *termin2* und *termin3*?

Beispiel 3.4:
Schreiben Sie eine Klasse *Schleifen*, in der Sie folgende Methode implementieren:

```
static void schleifen()
{
      for(int i=1;i<=3;i++)
            for(int j=1;j<=3;j++)
                  System.out.println(i+j);
}
```

Führen Sie diese Methode aus. Wieviele Zahlen werden ausgegeben? Kommen Zahlen doppelt vor? Warum (nicht)? Gehen Sie die Methode mit dem Debugger Schritt für Schritt durch, bis Sie sicher sind, die Funktionsweise der beiden verschachtelten Schleifen verstanden zu haben. Schreiben Sie die *for*-Schleifen zunächst in *while*- und anschließend in *do-while*-Schleifen um, sodass sich am Ablauf der Methode nichts ändert.

Beispiel 3.5:
Schreiben Sie in einer Klasse Ihrer Wahl eine Methode

```
static void addiere(int[] array1, int[] array2),
```

die sämtliche möglichen Additionen von Elementen in *array1* und Elementen in *array2* mittels *System.out.println()* ausgibt. Der Aufruf *addiere({1,3},{2,4,6})* soll beispielsweise die Additionen

1+2=3 1+4=5 1+6=7 3+2=5 3+4=7 3+6=9

ausgeben. Schreiben Sie die Methode einmal mit *for*-Schleifen, dann mit *while*- und anschließend mit *do-while*-Schleifen.

Beispiel 3.6:
Gegeben seien Klassen *Schueler*, *Lehrer* und *Text*. Die Klasse *Schueler* verfügt über Methoden

```
public Text schreibe(),
public Text schreibeVerbesserung(Text einText),
```

die den Schüler einen Text schreiben bzw. verbessern lässt. Diese Methoden geben den geschriebenen bzw. verbesserten Text zurück.

Die Klasse *Lehrer* verfügt über die Methode

```
public int korrigiere(Text t),
```

die den übergebenen Text korrigiert und die Anzahl der entdeckten Fehler zurückgibt.

Schreiben Sie eine Klasse *Hausuebung* mit einer statischen Methode

```
public static Text macheHausuebung(Schueler max, Lehrer laempel),
```

die den Schüler *max* einen Text schreiben lässt, der vom Lehrer *laempel* korrigiert wird. Der korrigierte Text soll anschließend solange vom Schüler verbessert und anschließend vom Lehrer korrigiert werden, bis der Lehrer keinen Fehler mehr findet. Die Methode soll den abschließenden Text zurückgeben.

Beispiel 3.7: Gegeben sei die Klasse *Filiale*, die über folgende Methoden verfügt:

```
public double getAbstand(Filiale andereFiliale)
```

gibt den Abstand zwischen der Filiale und *andereFiliale* zurück.

```
public void schliesse()
```

schließt die Filiale.

Schreiben Sie eine Klasse *Filialnetz* mit Konstruktor

```
public Filialnetz(ArrayList<Filiale> dieFilialen),
```

der ein entsprechendes Filialnetz anlegt. Weiters soll es in *Filialnetz* eine Methode

```
public void restrukturiere()
```

geben, die eine Filiale schließt und aus dem Filialnetz entfernt, die den kleinsten Abstand zu einer anderen Filiale im Filialnetz hat.

Beispiel 3.8:
Gegeben seien die Klassen *Person* und *Posting*.

Die Klasse *Person* verfügt über die folgenden Methoden:

```
public Posting getLastPosting()
```

gibt das letzte Posting der Person zurück.

```
public boolean istBefreundet(Person einePerson)
```

gibt genau dann *true* zurück, wenn die Person mit *einePerson* befreundet ist.

```
public void findeGut(Posting einPosting)
```

lässt die Person das Posting *einPosting* gut finden.

Schreiben Sie eine Klasse *SozialesNetzwerk* mit Konstruktor

```
public SozialesNetzwerk(Person[] dieUser),
```

der ein soziales Netzwerk mit den Personen in *dieUser* erstellt.

Schreiben Sie in die Klasse *SozialesNetzwerk* weiters eine Methode

```
public void sozialeInteraktion(),
```

die jede Person im sozialen Netzwerk das letzte Posting all ihrer Freunde gut finden lässt.