Informationen: Die Beispiele sind bis 16.1., 8 Uhr abzugeben bzw. bis zur Übung vorzubereiten.

Beispiel 11.1:
Schreiben Sie eine generische Klasse *Pair<S,T>*, mit einem Konstruktor

```
Pair(S left, T right)
```

und Methoden

```
S getLeft(),
T getRight().
```

Schreiben Sie eine zweite Klasse, in der Sie Objekte von verschiedenen Datentyp-Paaren anlegen und die Methoden der Klasse *Pair* testen.

Beispiel 11.2:
Gegeben seien die Klassen *Anlage*, *Bauteil* und *Prozess*. Schreiben Sie eine Klasse *Produktion* mit Konstruktor

```
public Produktion(Prozess[] dieP, Bauteil[] dieB,
      Anlage[] dieA, double[][][] prozesszeiten),
```

dem die Anlagen, Bauteile und Prozesse in der Produktion als Arrays sowie die entsprechenden Prozesszeiten übergeben werden. Dabei ist in *prozesszeiten[i][j][k]* die Prozesszeit des Prozesses *dieP[i]* für Bauteil *dieB[j]* auf Anlage *dieA[k]* hinterlegt.

Schreiben Sie weiters die folgenden Methoden:

```
public Anlage getSchnellste(Bauteil b, Prozess p)
```

gibt jene Anlage zurück, die Prozess *p* für Bauteil *b* am schnellsten bearbeiten kann.

```
public int[] getIndizesDer(Anlage a)
```

gibt die Indizes jenes Bauteils und jenes Prozesses in einem *int*-Array zurück, die Anlage *a* am schnellsten bearbeiten kann.

Beispiel 11.3:
Schreiben Sie eine *enum Monat*, wobei Sie für jeden Monat eine entsprechende Zahl (Jänner = 1, Februar = 2, usw.) sowie die deutschen als auch die englischen Bezeichnungen in jeweils einem eigenen Attribut speichern.

Beispiel 11.4:
Schreiben Sie die Klasse *Zeitpunkt* (aus Beispiel 6.1) so um, dass der Monat nicht als *int* sondern als Objekt der *enum Monat* aus Beispiel 10.3 gespeichert wird. Welche weiteren Änderungen müssen Sie vornehmen?

Beispiel 11.5:
Schreiben Sie die Klasse *Termin* (aus Beispiel 6.1 oder 9.4) so um, dass eine *NullPointerException* geworfen wird, falls im Konstruktor von *Termin* die Art des zu erstellenden Termins *null* sein sollte. Was müssen Sie beachten, wenn Sie nun in einer anderen Klasse den Konstruktor von Termin aufrufen möchten?

Beispiel 11.6:
Schreiben Sie eine Klasse *WrongDateException,* die von *Exception* erbt. Schreiben Sie den Konstruktor der Klasse *Zeitpunkt* (aus Beispiel 6.1 oder 11.3) so um, dass eine *WrongDateException* geworfen wird, wenn ein Datum erstellt wird, das nicht existiert. Berücksichtigen Sie dabei auch Schaltjahre richtig.

Beispiel 11.7:
Ergänzen Sie in die Klasse *Auftragsverwaltung* aus Beispiel 8.1 um eine Methode

```
public void speichere(String filename),
```

die die Auftragsverwaltung in einem File mit Namen *filename* speichert. Schreiben Sie außerdem eine Methode

```
public void lade(String filename),
```

die die Auftragsverwaltung aus dem File *filename* lädt, vor dem Überschreiben der aktuellen Auftragsverwaltung von dieser allerdings noch ein Backup (mit einem Filenamen Ihrer Wahl) erstellt.

Schreiben Sie in die Klasse *Auftragsverwaltung* außerdem einen zweiten Konstruktor

```
public Auftragsverwaltung(String filename),
```

der eine neue Auftragsverwaltung mit den im File *filename* gespeicherten Aufträgen lädt.
Welche Änderungen müssen Sie in den Klassen *Auftrag* und *Person* vornehmen?

Beispiel 11.8: ***[Bruch.java]***
Schreiben Sie eine Klasse *Bruch* mit Konstruktor

```
public Bruch(int z, int n),
```

der einen Bruch mit dem Zähler *z* und dem Nenner *n* erzeugt. Die Klasse soll weiters über Methoden *getZaehler()*, und *getNenner()* verfügen. Eine weitere Methode

```
public void kuerzeBruch()
```

soll den Bruch kürzen, indem Zähler und Nenner durch ihren größten gemeinsamen Teiler dividiert werden. Den größten gemeinsamen Teiler zweier positiver ganzer Zahlen *a* und *b* soll die Methode

```
public int ggT(int a, int b)
```

zurückgeben.

Beispiel 11.9: ***[SimpleList.java, Umkehren.java]***

SimpleList ist ein Interface, das eine einfache Liste von Strings beschreibt. Der Zugriff auf eine *SimpleList* ist nur mit den angegebenen Methoden *removeLast()* und *addFirst()* möglich:

```
interface SimpleList
{
      /**
       * Entfernt den letzten String aus der SimpleList
       * und liefert ihn zurück. Wenn die SimpleList leer
       * ist, wird null zurückgegeben.
       */
       public String removeLast();

       /** Fügt s als ersten String in die SimpleList ein . */
       public void addFirst(String s);
}
```

Implementieren Sie eine Klasse *Umkehren* mit einer Methode

```
static void invert(SimpleList list),
```

die die Reihenfolge der Strings in *list* umkehrt, also z.B. die Liste mit den Elementen ("a","b","c","d","e") in die Liste mit den Elementen ("e","d","c","b","a") umwandelt. Implementieren Sie zum Testen von Umkehren auch eine Klasse mit dem Interface *SimpleList*.

Beispiel 11.10:

Gegeben ist die Klasse *Filiale* mit den Methoden

```
double getEntfernung(),
int getBedarf(),
```

die die Entfernung zum Zentrallager bzw. den aktuellen Bedarf zurückgeben.

Schreiben Sie eine Klasse *Belieferung* mit einer Methode

```
List<Filiale> getBelieferung(List<Filiale> filialliste,
                double maxEntfernung,  int maxMenge),
```

die in einer Liste einen Teil der Filialen aus *filialliste* zurückgibt, die höchstens *maxEntfernung* vom Zentrallager entfernt sind, und deren Bedarf zusammen höchstens *maxMenge* ist. Die zurückgelieferte Liste soll maximal insofern sein, als dass sie um keine Filiale mit Entfernung höchstens *maxEntfernung* ergänzt werden kann, ohne *maxMenge* zu überschreiten.

Hinweis: Im allgemeinen gibt es mehrere korrekte Listen, die von der Methode zurückgegeben werden können. Z.B. können bei den im folgenden gegebenen Filialen für *maxEntfernung*=50 und *maxMenge*=100 entweder die Filialen 1,4,6 oder die Filialen 4,5,6 zurückgegeben werden.

Filiale	**1**	**2**	**3**	**4**	**5**	**6**
Entfernung	30	70	70	20	10	15
Bedarf	70	10	50	20	50	10