1. Aufgabe

Die Klassen `Kfz` und `Pkw` sollen so modifiziert werden, dass der Auf- und Abbau von Objekten verfolgt werden kann. Außerdem ist die Klassenhierarchie durch eine weitere Klasse `Lkw` zu ergänzen.

- Ändern Sie die Klassen `Kfz` und `Pkw` so, dass der jeweilige Konstruktor die Meldung ausgibt:

```
"Ich baue ein Objekt vom Typ ... auf."
```

Definieren Sie für die Klassen `Kfz` und `Pkw` auch einen Destruktor, der die Meldung ausgibt:

```
"Ich zerstöre ein Objekt vom Typ ...."
```

- Definieren Sie dann die von `Kfz` abgeleitete Klasse `Lkw` mit den nebenstehenden Datenelementen, einem Konstruktor und einem Destruktor sowie den nebenstehend angegebenen zusätzlichen Methoden.
- Implementieren Sie den Konstruktor für die Klasse `Lkw`, der wieder eine passende Meldung ausgibt. Verwenden Sie Basisinitialisierer für die Initialisierung der Datenelemente von `Kfz`.
- Definieren Sie den Destruktor von `Lkw`, der ebenfalls eine passende Meldung ausgibt, und die zusätzlichen Methoden für Lkw.
- Zum Testen vereinbaren Sie in der `main`-Funktion zunächst ein Objekt vom Typ `Lkw` und lassen es anzeigen. Auf Wunsch des Anwenders wird auch je ein Objekt vom Typ `Pkw` und `Kfz` angelegt und angezeigt.

Verfolgen Sie den Auf- und Abbau der verschiedenen Objekte bzw. Teilobjekte.

2. Aufgabe

Von der bereits definierten Klasse `Konto` sollen zwei Klassen – `GiroKonto` und `SparKonto` – abgeleitet werden. Die Klasse `GiroKonto` enthält zusätzlich ein Überziehungslimit und einen Soll-Zinssatz. Die Klasse `SparKonto` enthält neben den Elementen der Basisklasse nur einen Haben-Zinssatz.

- Definieren Sie beide Klassen mit Konstruktoren, die für jeden Parameter einen Default-Wert vorsehen, mit Zugriffsmethoden und mit je einer Methode `display()` für die Ausgabe auf dem Bildschirm.
- Testen Sie die neuen Klassen, indem Sie Objekte vom Typ `GiroKonto` und `SparKonto` in der Vereinbarung initialisieren und ausgeben lassen. Anschließend ändern Sie im Dialog ein Spar- und ein Giro-Konto und lassen die Werte wieder anzeigen.

3. Aufgabe
Für eine Supermarktkette ist ein automatisches Registrierkassensystem zu entwickeln. Alle Produkte werden durch einen Barcode und ihre Bezeichnung erfaßt. Lebensmittel werden entweder fest verpackt oder je nach Gewicht verkauft. Fest verpackte Lebensmittel haben einen Festpreis. Für Lebensmittel, die abgewogen werden, ergibt sich der Preis aus dem aktuellen Gewicht und dem Kilopreis.
Zunächst werden nur die Klassen zur Darstellung der Produkte entwickelt. Diese sind hierarchisch organisiert. Als gemeinsame Basisklasse dient eine Klasse `Product`, die die allgemeinen Informationen über ein Produkt (Barcode, Bezeichnung) enthält.

- Die Klasse `Product` besitzt zwei Datenelemente zum Speichern des Bar-Codes vom Typ `long` und der Bezeichnung des Produkts. Es soll ein Konstruktor mit Parametern für beide Datenelemente definiert werden. Die Parameter sind mit Default-Werten zu versehen, so dass auch ein Default-Konstruktor zur Verfügung steht.
Neben den Zugriffsmethoden wie `setCode(), getCode()` usw. sollen auch die Methoden `scanner()` und `printer()` zur Verfügung gestellt werden. Zu Testzwecken geben diese die Daten eines Produkts auf dem Bildschirm aus bzw. lesen die Daten eines Produkts im Dialog ein.
- Im nächsten Schritt werden die Spezialisierungen der Klasse `Product` entwickelt. Definieren Sie zwei von der Klasse `Product` abgeleitete Klassen `PackedFood` und `UnpackedFood`. Zusätzlich zu den Daten eines Produkts soll die Klasse `PackedFood` einen Stückpreis und die Klasse `Unpacked-Food` ein Gewicht und einen Kilopreis als Datenelemente besitzen.
In jeder der beiden Klassen ist ein Konstruktor mit Parametern für jedes Datenelement mit Default-Werten zu definieren. Dabei sind Basis- und Elementinitialisierer zu verwenden.
Definieren Sie die erforderlichen Zugriffsmethoden für die neuen Datenelemente. Redefinieren Sie außerdem die Methoden `scanner()` und `printer()`, so dass die neuen Datenelemente mit berücksichtigt werden.
- Testen Sie die verschiedenen Klassen in einer `main`-Funktion, die je zwei Objekte vom Typ `Product`, `PackedFood` und `UnpackedFood` anlegt. Ein Objekt jeden Typs wird in der Definition vollständig initialisiert. Das andere Objekt wird mit dem Default-Konstruktor aufgebaut. Testen Sie die `get-` und `set`-Methoden sowie die Methode `scanner(),` und lassen Sie die Produkte auf dem Bildschirm anzeigen.

Zu Aufgabe 1:

Zusätzliche Datenelemente:
Anzahl Achsen int
Ladekapazität double

Zusätzliche Methoden:
void setAchsen(int a);
int getAchsen() const;
void setKapazitaet(double cp);

int getKapazität() const;
void display() const;