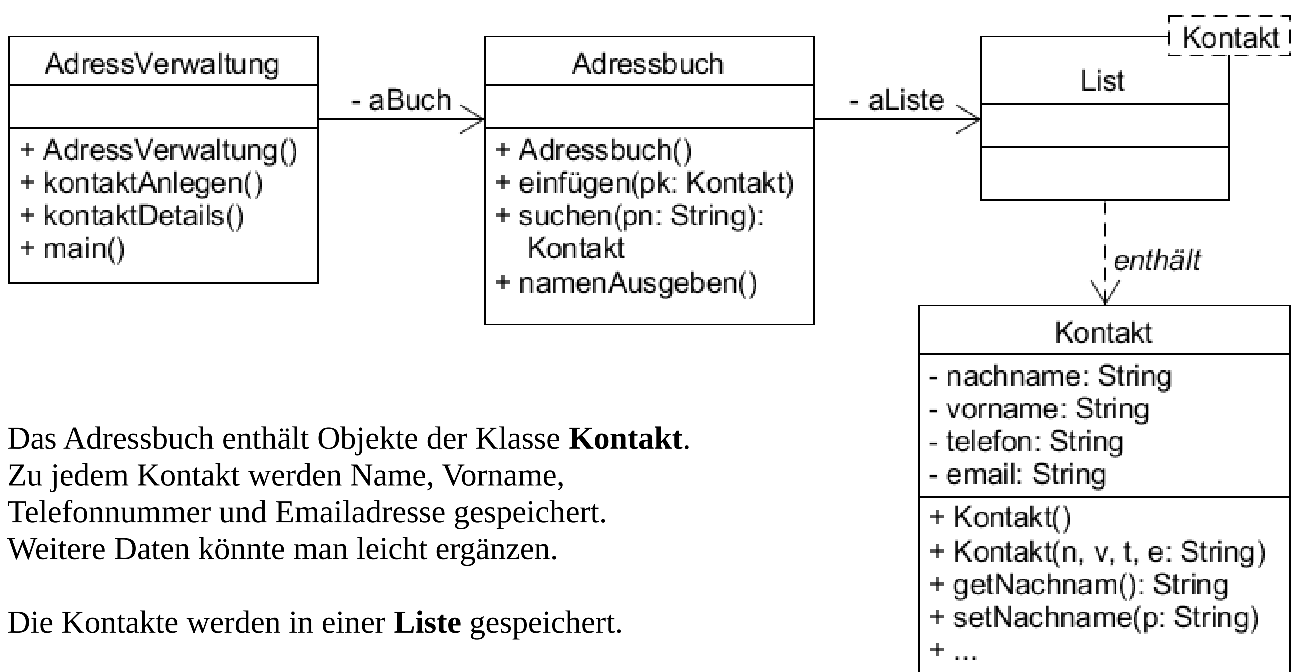
Eine typische Anwendung für eine Liste ist ein **Adressbuch**:
Ein Adressbuch enthält eine Liste von Kontakten.
Jeder Kontakt gehört zu einer Person, und speichert Namen, Adresse, Telefonnummer, Emailadresse usw.

Ein Adressbuch ist in der Regel nicht statisch, d.h. im Lauf der Zeit kommen neue Adressen hinzu, nicht mehr benötigte Kontakte werden gelöscht, und manchmal ändern sich Daten, z.B. wenn jemand umzieht.

Die Software für das Adressbuch ist im folgenden Klassendiagramm dargestellt:



Das Adressbuch enthält Objekte der Klasse **Kontakt**.
Zu jedem Kontakt werden Name, Vorname, Telefonnummer und Emailadresse gespeichert.
Weitere Daten könnte man leicht ergänzen.

Die Kontakte werden in einer **Liste** gespeichert.

Die Klasse **Adressbuch** bietet Methoden, um Kontakte in die Liste einzufügen und zu suchen. In dieser Klasse werden die Algorithmen mit der Datenstruktur Liste implementiert.

Das Adressbuch soll mithilfe der Konsole bedient werden. Diese Benutzerschnittstelle wird in der Klasse **AdressVerwaltung** (Hauptklasse) implementiert, mit einer **Ausnahme**:

Die Methode namenAusgeben() gibt die Namen aller Kontakte auf der Konsole aus.
Die Klasse AdressVerwaltung sollte keinen direkten Zugriff auf die Liste haben.
Da die Ausgabe aber Zugriff auf alle Elemente der Liste braucht, lässt sich diese Methode besser in der Klasse AdressBuch implementieren.

Aufgabe 1 (typische Abituraufgabe)

Erläutere für jede der vier Datenstrukturen Array, Queue, Stack und Liste, wie gut diese für die Anwendung „Adressbuch“ geeignet ist oder nicht (jeweils mit Angabe von Vor- und Nachteilen).

Aufgabe 2

Verwende die ausgeteilte BlueJ-Vorlage. Die Klassen Kontakt und List sind bereits fertig implementiert. Implementiere die Klasse **Adressbuch** nach den folgenden Vorgaben:

- `Adressbuch() (Konstruktor)`
  Erzeugt die (leere) Liste.

- `void einfügen(Kontakt pk)`
  Als Parameter pk wird ein bereits mit Daten gefülltes Kontakt-Objekt übergeben.
  Die Methode fügt diesen Kontakt einfach am Ende der Liste ein.
  (Das Objekt zu erzeugen und die Daten vom Benutzer zu erfragen ist Aufgabe der Hauptklasse)

- `Kontakt suchen(String pname)`
  Als Parameter wird ein Nachname übergeben. Die Methode sucht in der Liste das Kontakt-Objekt mit diesem Nachnamen und gibt es zurück. Falls kein Kontakt-Objekt mit diesem Nachnamen in der Liste vorhanden ist, wird null zurückgegeben.
  - Wir nehmen zur Vereinfachung an, dass jeder Name nur einmal in der Liste vorkommt.
  - Zur Erinnerung: Strings werden mit equals() verglichen:
    `if (str1.equals(str2)) { ... }`

- `void namenAusgeben()`
  Gibt (nur) Nachnamen und Vornamen für alle Kontakte der Liste auf der Konsole aus.
  Außerdem wird die Anzahl der Kontakte ausgegeben.

Aufgabe 3

Implementiere die Hauptklasse **AdressVerwaltung**.
Bei Schwierigkeiten mit der Konsole findest du eine Hilfestellung in der Vorlage.

- `AdressVerwaltung() (Konstruktor)`
  Erzeugt das Adressbuch-Objekt.

- `void kontaktAnlegen()`
  Erfragt die Daten für ein neues Kontakt-Objekt (Name, Vorname, Telefon, Email) vom Benutzer. Erstellt das Kontakt-Objekt und fügt es ins Adressbuch ein.

- `void kontaktDetails()`
  Bittet um die Eingabe eines Nachnamens.
  Sucht den entsprechenden Kontakt aus dem Adressbuch und gibt dessen Daten aus.
  Falls der gesuchte Name nicht enthalten ist, wird eine entsprechende Meldung ausgegeben.

- `main()`
  Zeigt ein Konsolenmenü an. Der Benutzer wählt einen Menüpunkt durch Eingabe einer Zahl. Die gewählte Funktion wird ausgeführt.
  Das wiederholt sich solange, bis der Benutzer das Programm beendet.

  Das Menü könnte folgendermaßen aussehen:
  ```
  Wählen Sie einen der folgenden Punkte
  1.  Neuer Kontakt
  2.  Kontakte auflisten
  3.  Kontaktdetails
  4.  Programm beenden
  ```

Aufgabe 4

Für die Anzeige aller Namen und für die Suche innerhalb der Liste ist es sinnvoller, wenn die Kontakte in **alphabetischer Reihenfolge** in der Liste gespeichert werden.
Geordnet werden soll die Liste nach den Nachnamen der Kontakte.

Passe die Implementierung der Klasse **Adressbuch** entsprechend an:

- `void einfügen(Kontakt pk)`
  Statt den neuen Kontakt pk einfach ans Ende der Liste zu hängen, soll er an der alphabetisch richtigen Stelle eingefügt werden.

  Bevor du die Methode implementierst, erarbeite den **Algorithmus** zunächst in Umgangsprache.

  Hilfestellung zur Implementierung: Ob ein String alphabetisch „größer" oder „kleiner" als ein anderer String ist, kann man mithilfe der compareTo-Methode herausfinden:
  `str1.compareTo(str2)` < 0 → str1 steht im Alphabet VOR str2
  `str1.compareTo(str2)` = 0 → beide stehen an der gleichen Stelle (d.h. sind gleich)
  `str1.compareTo(str2)` > 0 → str1 steht im Alphabet HINTER str2

- `Kontakt suchen(String pname)`
  Die Methode wird etwas optimiert: Die Suche wird abgebrochen, sobald sicher ist, dass der gesuchte Name nicht in der Liste enthalten sein kann.

Aufgabe 5

Begründe, warum die Klasse Adressbuch keinen direkten Zugriff auf die Datenstruktur Liste für andere Klassen erlauben sollte (also auch nicht für die Hauptklasse AdressVerwaltung).

Autor: Christian Pothmann – cpothmann.de, freigegeben unter CC BY-NC-SA 4.0, Juli 2021
Quellen: Foto Adressbuch: de.wikipedia.org, freigegeben unter CC BY 3.0